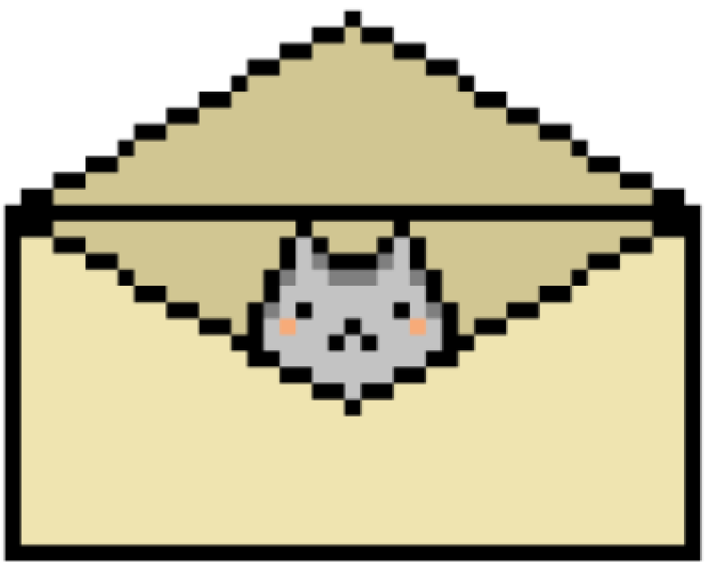
MessageCat

A simplistic Internet Messaging system created by Nathan Baines.



Analysis **6**
Problem identification 6
Why is this problem amenable to a computational approach? 6
Preconditions 7
Stakeholders 7
Data Protection Act 8
Research – Whatsapp 9
Research – Telegram 12
Research – Discord 13
Research - Snapchat 14
Features 15
Limitations 16
Requirements 17
Success criteria 18
Design **19**
Decomposition 19
Server 19
Database 19
MySQLHandler 21
MessageStore 22
MessageQueue 23
KeyStore 23
Database 24
Server functions 24
Server 25
Handler 26
Server process flow diagram 27
Build targets 29
Client application 30
General process 30
Networker service and Connection handler 31
Usability 34
UI 34
LoadingActivity 34
CreateNewUserActivity 36
MainActivity 38
InviteToChatActivity 43
Notifications 45
Local storage 46
Main process to connection handler process communication 47
Algorithms 48
Queue Data Structure 48
RSA Encryption Algorithm 48

SHA-256 Hashing Algorithm 50
Overall code structure 50
com.nathcat.RSA 50
EncryptedObject 50
KeyPair 51
PrivateKey 51
PublicKey 52
RSA 52
com.nathcat.messagecat_database 52
Result (enum) 52
MySQLHandler 52
MessageQueue 53
MessageStore 54
KeyStore 54
ExpirationManager extends Thread 55
Database 55
com.nathcat.messagecat_database_entities 55
User 55
Chat 56
ChatInvite 56
FriendRequest 56
Friendship 57
Message 57
com.nathcat.messagecat_server 57
Queue 57
Queue.Node 57
QueueManager extends Thread 58
RequestType (enum) 58
ListenRule 59
Handler 59
ConnectionHandler extends Handler 60
Server 62
com.nathcat.messagecat_client 63
AutoStartService extends BroadcastReceiver 63
LoadingActivity 63
NewUserActivity 63
NetworkerService extends Service 64
ConnectionHandler extends Handler 65
ListenRuleCallbackHandler extends Thread 66
MainActivity 67
Other… 68
Testing 69
RSA 69
Database 69

AddUser() 69
AddFriendship() 70
AddFriendRequest() 70
AddChat() 70
AddChatInvite() 71
GetUser() 71
GetFriendship() 72
GetFriendRequests() 72
GetChat() 73
GetChatInvite() 73
Server 74
Client application 76
End-user testing 78
Privacy policy 79
Development timeline 80
Implementation **81**
Database entities 81
User 81
Friendship 82
FriendRequest 82
ChatInvite 83
Chat 84
Message 84
RSA Asymmetric Encryption system 85
PublicKey 85
PrivateKey 86
EncryptedObject 86
KeyPair 88
RSA 91
Testing 92
ByteChunk 93
EncryptedObject 94
KeyPair 97
Secondary testing 99
Database 99
MySQLHandler 99
KeyStore 110
Queue 113
MessageQueue 115
MessageStore 117
Database 119
ExpirationManager 125
Testing 126
AddUser() 126

AddFriendship() 127
AddFriendRequest() 128
AddChat() 128
AddChatInvite() 129
GetUser() 129
GetFriendship() 131
GetFriendRequests() 131
GetChat() 132
GetChatInvite() 132
Connection timeout issue 133
Server 135
ListenRule 135
Handler 137
ConnectionHandler 140
QueueManager 152
Server 154
Server main method 157
Testing 157
Client application 166
ConnectionHandler (client) 167
NetworkerService 170
AutoStartService 178
LoadingActivity 178
NewUserActivity 181
MainActivity 188
ChatsFragment 201
FriendsFragment 203
FindUserFragment 206
InvitationsFragment 208
InviteToChatActivity 212
MessagingFragment 218
Testing 225
LoadingActivity 225
NewUserActivity 227
MainActivity 228
FindPeopleFragment 229
InvitationsFragment 229
InviteToChatActivity 229
FriendsFragment 230
ChatsFragment 230
MessagingFragment 230
Notifications 231
Evaluation **232**
Android 13 232

End-user testing 234
Final evaluation 236
Evaluation of success criteria 236
Code review 238
Usability 238
Limitations 239
Future development plan 240
Summary 240

Analysis

Problem identification

In this project I will develop an internet messaging system. Such a system should allow users to send text based messages between each other in real time over the internet, providing fast and efficient communication across long or short distances. The modern world requires such fast communication because things can change very quickly, for example, leaders of countries must stay up to date with things happening around the world, market traders must communicate very quickly to ensure they get the best profit from their investments, and emergency services need to be able to communicate with people in danger to ensure that they get help as quickly as possible.

An application such as this should be efficient and portable, as such the clear candidate is a mobile application. I will likely use Android for this since I have more experience with it, and Android occupies the majority of the market share of devices when compared to IOS. A server device will also be required, to act as the centre point of communication between devices. While storing all data on a single server device is a security risk, it does offer some benefits, primarily it allows for a much simpler networking structure to be implemented during development, although the server will likely have to make use of concurrent processing methods to ensure it offers the most efficient experience for many devices at once.

It must be noted that there are a number of similar applications which already solve this problem, however the aim of this project is to create an application much simpler than those, which is purely focussed on messaging. A lot of internet messaging applications have developed other features alongside their messaging system, the goal of this application is purely to focus on the messaging aspect in order to provide the simplest experience possible.

Why is this problem amenable to a computational approach?

For most of modern history humans used physical methods of transportation to communicate with each other, generally through the mail, communicating using the written word, this is an analogue approach. The main issue with this, and many other analogue approaches, is that it is slow, it may take multiple weeks or months for a letter to reach is destination, depending on the distance between the source and destination, and where that destination is, for example a war torn country might be less able to receive such communication, since its links with other countries will likely be damaged by

said war. Innovations in transportation have significantly improved this system's efficiency, but it is still rather inefficient when compared with digital methods, which brings us to my point.

With the invention of the internet and digital computers / media, we have a means for much more efficient communication, by transforming human language into binary data via the use of character sets and peripheral devices, we can communicate information across networks to anywhere in the world which has access to such information, and with the invention of satellite technology, the list of places that can access this information is very large.

There are a lot of programs which already take advantage of the internet to allow human communication, some of which I will look at in more detail later on in this document. My goal is to create a simple Android application which replicates this functionality, in a simplified manner. The app will simply allow mass communication, and nothing else, since this is all that is required from such an application.

Preconditions

Due to the nature of this application there are certain features or processes which must be present. For example, there must be a centralised server device which acts as a common connection point for all client devices. Since my plan is to develop an Android application, this adds a number of other preconditions:

- Networking operations must be performed in a separate process from the UI thread.
- The application must be lightweight enough to function effectively in a mobile environment.
- Some data should be stored on the device itself.
    - Authentication data for example, storing this locally will mean that the user only has to login to the application once, and they will then be remembered.

And of course the previously mentioned precondition:

- A centralised server device which acts as a central connection point, and includes a database of some description.

Furthermore, as much as possible, certain aspects of the application should be reusable, for maintenance as well as convenience in future development of the system. Different parts of the system should be separated into different Java packages, to differentiate source code used for different purposes, rather than having it all bundled up together. This would also reduce compilation time and the size of the final compiled application, since there would be less source code to compile.

Stakeholders

Stakeholders are an important part of any project, as they are generally the ones using the application, their input is vital to the success of the development process in producing an application which functions appropriately.

In this project, I have two stakeholders, one who is a programmer such as myself, and the other is a general user, they do not have any experience with the actual inner workings of an application, as opposed to my first stakeholder.

Following are two interviews I conducted with my two stakeholders.

General user:
Q: What other messaging applications have you used?
A: Discord, WhatsApp, and Snapchat are the ones I have used the most.

Q: What makes you want a new application, rather than using one of the already existing ones?
A: These applications have a lot of extra features which I find unnecessary and in some cases confusing, so I would like an application which simplifies the process of communication, extra features are not needed, simply the messaging.

Q: An application like this would have to hold a large amount of data about its users, how do you feel about that?
A: Having my data held by an organisation does make me anxious, in order to be able to put my trust in an application I would want to know how my data is being stored and where it is being transferred.

Q: Do you have any particular thoughts about what you might want the application to look like?
A: Preferably the application would look similar to other messaging applications like WhatsApp, to make the transition between them easier.

Programmer:
Q: What measures would an organisation have to take to convince you that your data is safe?
A: The most important measure is encryption, not symmetric encryption because that's too simple to break. They would have to implement some kind of asymmetric encryption for communication between devices. Data that is stored on a server would also have to be encrypted to some extent, hashing on the passwords for example.

Q: What level of maintainability would you expect in the source code of an application such as this?
A: The code should be as maintainable as possible, comments to explain things, and a modular, layered design, similar to that of network protocols.

Following this interview we can see that the Stakeholder is looking for a simple, secure application. It needs only to allow the user to message others, no other functions are necessary to them. This is very useful information as it places a clear restriction on the features that should be available in the application, which will simplify development of the application, and make the final application simpler to use (as is the point of this application).

However, this application will inevitably store an amount of information about its users, so we should discuss the implications this has under the Data Protection Act.

Data Protection Act

Any organisation which collects data about people has a responsibility to ensure that this data is stored securely and in a way which respects the user's privacy, in accordance with the data protection act. Given the problem this project concerns itself with, it is inevitable that we will end up storing some amount of data about users of the application, as such I should attempt to store this data in a way that

reflects the principles laid out in the data protection act, especially if I plan to deploy this application to the public.

The DPA states that data should not be stored for longer than is necessary, so we should provide a way for users to delete their data from the server, this means that they will not be able to use the application without creating new data, so they should be made aware of this.

The DPA also states that the data stored about users should not be excessive, hence we should determine the data that is absolutely required for the application to function safely and securely, and this is the data that will be stored by the application.

Data will not be transferred to third parties, so the point stating that data should not be transferred to organisations in countries that do not offer a similar level of protection is not necessarily relevant at this time. However the point stating that data should be stored securely is absolutely relevant. We should design the database system in such a way that only the application and authorised individuals should have direct access to the data it contains, this will be covered in the design section of this project. Furthermore, passwords will be hashed before being sent to the server for checking, communications between client devices and the server will be encrypted by a connection specific key pair generated when the connection is started, and messages (which may contain sensitive information), will be separately encrypted using a key pair specific to the chat the message was sent to, these processes will also be covered in more detail in the design phase of the application.

Research – Whatsapp

Whatsapp is one of the most popular Internet messaging applications in the world, it is owned by Meta platforms (previously Facebook). It was developed to solve the annoyance of having to pay for each individual message, which is the case for SMS messaging through mobile network providers.

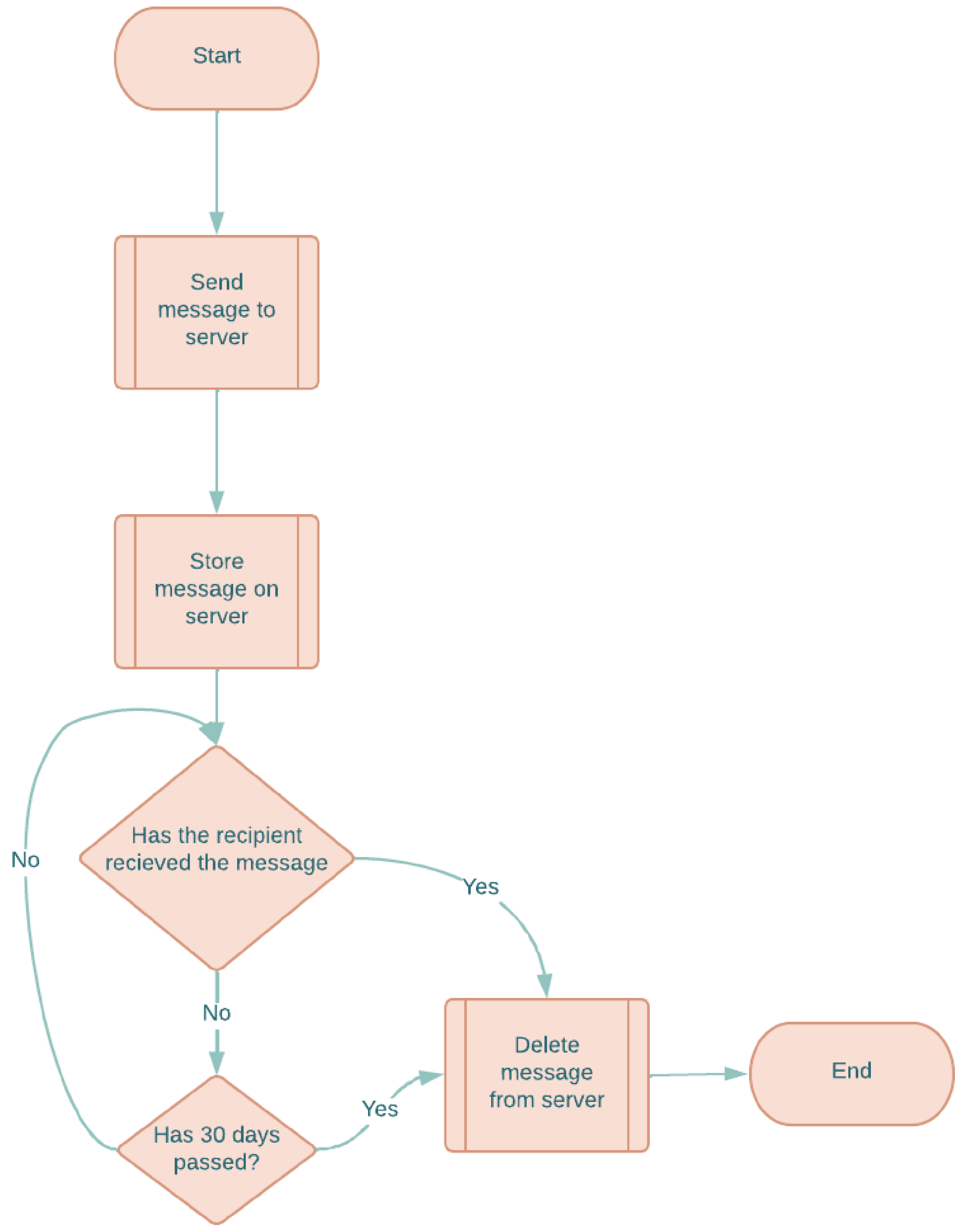
Whatsapp's user interface is well designed and easy to use, furthermore all chats on the application are end-to-end encrypted, which is something I aim to implement in my application. But returning to the interface, the message entry box and send button are clear, and it is clear who messages are from, here is an image of the user interface:

Ignoring the… questionable messages… You can see that the message entry box is clear, with a placeholder informing the user what it is for, buttons at the top and labels with information about the other person in the chat, and the send button is clearly marked, next to the message entry box.

The main features of Whatsapp are:

- A Centralised IM system
- End-to-end encryption
- Voice-over-Internet-Protocol (VoIP)

VoIP is a protocol which allows users to communicate over the internet with their voices, this takes the form of voice calling and video calling (although the transfer of video is an entirely different protocol). The protocol by which Whatsapp transmits messages between users is also very interesting, a flow chart describing this process follows:



This is an interesting approach, the main benefit of this is that it reduces the resources used on the server, as messages are stored locally on devices as opposed to being stored on the server, the server acts as a sort of post office, if you will. Although conversely this will increase the complexity in making this application cross-platform, while Whatsapp has found a way to do this, it seems that this design would make this process more difficult than it needs to be.

Another existing IM application is Signal. I include Whatsapp and Signal in the same section because they are very similar in their design and functionality. Furthermore, both applications use Open

Whisper Systems to implement end-to-end encryption on their platforms. This is absolutely a priority feature in my application, however these services have been criticised for their use of such secure encryption. While this may seem counter-intuitive, especially with the increased awareness of data protection in the modern world, it does make sense. This will be covered in more detail later.

Research – Telegram

Telegram is another IM service. Like Whatsapp, they use Phone numbers to create user accounts, and these are verified by SMS.

Telegram has a number of interesting features, some of which may be outside the scope of this project, but are certainly potential future developments. For example, they have a “secret chat” feature, which uses client-to-client communication. The platform also allows users to share their live location for a set period of time, this is a very interesting feature which is present in another IM application called “SnapChat”, while this poses a number of data safety and protection issues, it is still an interesting feature and is well worth considering for development and implementation in my application. Another feature in “SnapChat” is disappearing messages, this could be very useful as it will significantly reduce the resources required by the server.

Telegram also uses a cloud-based architecture, which is much more suited to a cross platform service than Whatsapp and Signal’s architectures. While it may be more resource heavy on the server side, this may be a more suitable choice for my application, given that a priority is easy integration with as many platforms as possible.

The platform has also provided infrastructure for developers to create “bots” on their platform, which are able to talk to users. This is a feature that is also present on yet another IM application called “Discord”.

A feature that is present on all of the platforms I have researched here is the ability to form “group chats”, this could be difficult to implement, but is certainly worth considering, although I find it unlikely that this will be part of the primary development points.

Something that all of these platforms have been criticised for is their use of end-to-end encryption, this is because of criminal activity being organised/conducted on these platforms, this is an ongoing issue and as far as I can see there is not a clear solution. For example, Telegram suffered massive criticism when it was discovered that the terrorist group ISIS had a message channel on their platform. The channel was later removed, but the point remains that such strong encryption makes it difficult to perform moderation and ensure that the platform is not being used for darker purposes. A potential solution to this is to create an Artificial Intelligence application that is trained to recognise potential criminal activity, and flags messages for review by a real person. This would be very difficult and time consuming to develop, so it is not part of the scope of this project, but I thought I should mention it anyway as it is certainly a feature that I should consider for the future if I was to take this application further.

Research – Discord

Discord is another very popular IM and VoIP application, it allows users to communicate over text, voice, or video, and users communities called “servers”, this is an interesting format which could be difficult to implement. It is similar to group chats in other applications like Whatsapp and Signal, although these are the main form of communication in Discord, whereas in Whatsapp and Signal, one-to-one communication is more a priority than group communications.

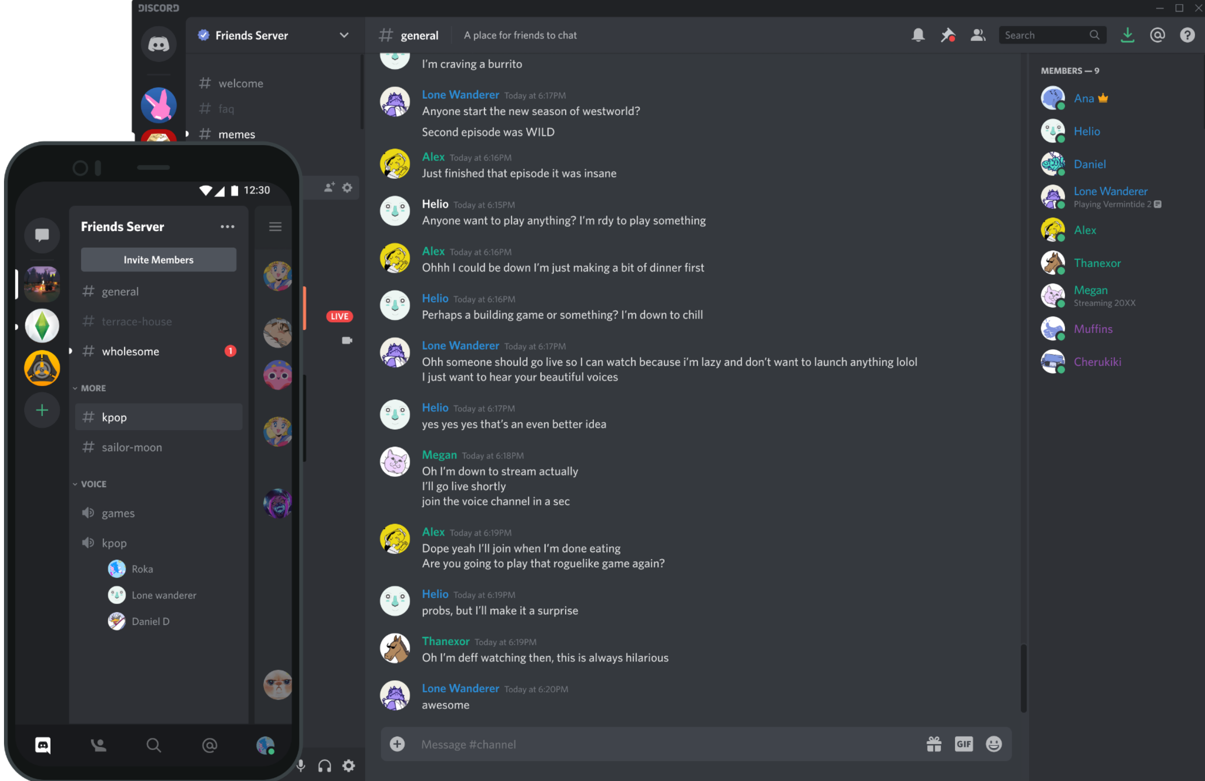
Discord has a number of APIs which are designed for developers, for example, their “bot” API is very popular for developing programs which automatically manage and moderate servers, and their GameBridge API allows developers to directly integrate Discord into their games, Rocket League for example makes use of this API.

Furthermore, the way the monetized the application is very interesting. The added the option for users to subscribe to Discord Nitro, which gives them access to custom emojis and stickers, they can also boost servers, with enough boosts, users in a server will gain various benefits within that server, higher audio quality, or animated icons for example.

Like the other platforms researched here, Discord has been criticised for abuse and bullying behaviour, and illegal activity being performed or organised on their platform. The platform is very popular with alt-right and far-right extremists, and was used to plan the Unite The Right rally in Charlottesville, Virginia on August 12 2017. Two days after the U.S Capitol Attack in January 2021, Discord deleted a pro-Donald Trump server that was found to have been involved in encouraging insurrection and violence that may have led to this tragic event.

Moving on to something less tragic, Discord’s architecture appears to be loosely based around Internet Relay Chat (IRC) protocol, although instead of physical hardware, the so called servers are simply database entities. IRC is a very interesting protocol, it was created by a Finnish IT professional by the name of Jarkko Oikarinen and is designed to facilitate group communication as well as one-on-one private communication. In researching IRC, I have decided that I don’t plan to use a strict implementation of the protocol, but I believe it could give some ideas on an architecture for group based communication as well as one-on-one private communication. The way that Discord represents its servers as database entities is an interesting idea, storing them centrally would allow them to be publically joinable, or at least be found by anyone, this is an interesting idea, although given that the application I am developing in this project has more of a focus on private communication I’m not sure it is necessary to implement chats in this way.

Furthermore, Discord’s user interface is very well designed:



In both the mobile and desktop applications, the so called "servers" are located in a strip on the left, channels within that server are located in an expandable side menu, and the chat window clearly and compactly displays messages along with the name and profile picture of the user that sent them. A list of users in the server is located to the right of the chat window. Some features of this design may be ideal for use in my application.

Research - Snapchat

Snapchat is another internet messaging application, but it contains a significant number of other features as well. The stakeholder specifically named this application as one of the ones they used the most, so it is worth researching, as it may offer some interesting features.

Snapchat allows communication in a number of formats, it allows text messages (called chats), pictures and videos (called snaps), voice messages, multimedia messages, and voice or video calls. There are a lot of different formats, some of these might not be immediately necessary but generally it seems that they are designed to fit everyone's communication style, i.e. I have found that people often prefer one form of communication over another, and as such people have the freedom to choose how they communicate with each other.

Snapchat also offers a lot of other features besides this general communication, for example, rather than profile pictures they use avatars, which users can customise to look like themselves (or however they want), and they have access to a large wardrobe of cosmetic items to choose from. There is also a feature called snap maps, which tracks the real world location of your friends on the app (provided they have consented to this).

A lot of these features are extra complications to the primary aim of the system, which is for communicating with friends and loved ones, however they do offer a very well crafted user experience over simpler apps like WhatsApp. The stakeholder specifically requested an app which can be used simply for secure messaging, however there is no reason why some of these features cannot be designated for future development.

I think the most key feature which I could incorporate into this application would be the real time location tracking, while this does imply more ethical and legal issues into the development of the application, it is a very interesting feature and would offer an improved user experience.

Snapchat is also very well known for its extensive privacy features, it will notify users in a chat if a screenshot is taken of the chat, you will be notified if someone saves something you send to a chat, and messages will disappear 24 hours after they are viewed by the person you sent them to (unless settings are set otherwise). These are very effective privacy features and could be fairly complex to implement, so I don't think I will be implementing similar features here, but they are worth mentioning as Snapchat's privacy features are incredibly well built and it would be a good idea to view them as potential future developments.

Features

Based on my research into existing solutions and the information I gathered in the interview with my stakeholder, I have enough information to compile a list of features that should be implemented into the application during development.

Some of the features that were observed in other applications may not be included in this application because they would be determined as "unnecessary features" by the stakeholder, however there are some quality of life or customisation features that I will not include in this application that might be viable candidates for implementation in future development.

Clearly the primary feature is a text-based messaging system, without this the application fails to meet its most basic requirement from the stakeholder. Here we are defining a text-based messaging system as a system which allows users to communicate messages between two or more devices. How the application will actually do this in practice is something that I will cover in the design phase of this project.

Next, the app should include a contact management system, such a system would allow users to add people to a "friends list", from which they can choose people to add to a chat. That chat could be one-on-one, or with multiple people. This is necessary because it restricts who users can chat with to people who also want to chat to them, if they don't want to chat with them, they can simply not accept their friend request, in which case they cannot be added to any chats.

The app should also include a page where users can manage their invitations, these could be friend requests or invitations to a chat. This will allow users to choose what they are a part of and who they can talk to, an essential feature for an application concerned with social networking.

Other features that could be included are an automated moderation system, which would monitor the content of user's messages and report suspicious activity for judgement by a human team. Profile

pictures, which is simply a customisation option and does not necessarily provide any extra functionality to the base purpose of the application. VoIP voice calling, which would provide another communication option to the application, but is not specifically requested by the stakeholder, and the complexity of such a feature is not something that I would feel comfortable dealing with in this project.

Limitations

Given the essential features of this project, there are a number of limitations imposed upon devices that will be using the end product.

Firstly, any device that uses the application, or intends to use it, will require access to the internet, since the application transfers data over the internet to the server and other devices, it will require access to the internet to be able to make these transfers. This also brings us onto the next requirement which is specific to an Android application. In order to perform networking operations the application will require permission from the operating system and the user to access the internet, furthermore android requires that applications perform networking in a separate process to the one in which the UI is being processed. This places restrictions on the way this part of the system can be designed, and will undoubtedly complicate some areas of processing on the UI thread. The reason android requires this is because networking can take a long time in some cases, so performing these operations on the UI thread will cause the UI to be unresponsive for some time, this is a very fair reason, so the complications it brings are generally worth it.

Another limitation is the computation resources available to the application, end-to-end encryption can be fairly computationally expensive so it is likely that a CPU with reasonable performance is required, generally modern mobile devices will have sufficient performance and computational resources for such a task, although they are limited compared to a desktop computer. The device should also have enough memory to be able to handle 2048-bit integers, as this is generally a good standard for encryption. Finally, the device should have enough secondary storage to store all of the data that the application must keep on the device. This might include information about chats the user is a part of and data required to authenticate the user when opening the application.

Although the most basic requirements are a device running the Android OS, and the hardware required for networking, namely some form of network interface card, and the drivers necessary to operate the hardware required by the application.

Some of these limitations limit the devices the application can be used by, and some place a restriction on the way the application can be designed, as such we should consider all of them during the design of this application.

Further limitations might be the fact that I have not planned to implement the ability to remove friends, or block people. These are fairly key features of social networking applications, however I do not necessarily see the need for them in this application, given the small scope. If I planned to fully release this application to the public I would consider implementing such features, as they might not be particularly complex, however in order to reduce the complexity of the solution I will develop here I will avoid implementing these features, although they are candidates for future development points.

Requirements

Following is a list of conditions that must be met in order for the application to function properly on a device. These are things that are absolutely necessary, not things that will simply improve the performance of the application, if these conditions are not met by a device, the application will not be able to function on that device.

- Android OS - Required to run an Android application. Also an OS of any kind is required for a computational device to function.
- Internet permission granted by the user / OS - Allows the application to make use of the device's internet access. The hardware side would be handled by the OS, but the application can make API or SDK calls that require network access.
- Concurrent processing design - Maximises efficiency, and is required to perform networking operations in android applications.
- Enough memory to handle 2048-bit integers + the memory required by the rest of the application - Required for sufficiently secure end-to-end encryption system to function.
- Enough secondary storage to store the application and all the data it requires to run effectively - If this is not met, the device will not be able to be installed on the device, or at the very least it will not function properly.

And a list of requirements specified by the stakeholder.

- The application must be able to send messages.

This is fairly self explanatory, these messages should be text based and be transferred through the server between client applications. This is surely the most basic requirement of a text based communication system such as this.

- The application should have a contact management system.

This implies that the application should provide an interface to find other users, and allow the user to "add" them to their contacts list by sending them a friend request. If the other user accepts this request, the two users will be added to each other's contact list by creating two friendship records in the database. Users can also decline friend requests, but at the moment I have no plans to implement a way for users to remove friends, although this would be fairly simple to implement, I don't see any requirement for it at the present moment.

- The application should provide an interface to create chats between users.

This will take the form of a separate activity, or page of the application called *InviteToChatActivity*, which will allow users to send chat invites to other users. These chat invites can be for an existing chat that the user is already a part of, or for a new chat which can be created through this part of the application.

- The application should provide an interface to send friend requests / invites to chats

This is covered by the previous two requirements.

Success criteria

In order for the end product produced by this project to be considered successful, it must meet certain success criteria which are derived from the essential features and limitations of the solution.

- The application should allow users to communicate text based messages through the internet.

This is the most basic requirement of the solution, without this the application will be relatively pointless. We will test that this criteria is met by installing the application on multiple devices and attempting to send messages between them using the application. If the messages can be read as they were typed in, and the application says they are from the correct user, then this criteria will have been met. These messages should be sent through chats, which are effectively a way to group multiple users together to allow them to send messages between each other.

- Minimal data is stored on the client device

This suggests that the application should be as lightweight as possible, while still functioning effectively. Only data that is private to the user should be stored on the client device, such as private keys attached to chats that they are a part of, authentication data, and data about the chats they are a part of. This data is essential to the effective function of the application and as such should be stored on the client device, other information can be stored on the server, and retrieved by clients as it is required. The average size of Android applications is 14.6 MB, so I will aim for this as a target.

- End-to-end encryption between clients and the server, and between clients

This implies that communication between a client device and the server should be encrypted, likely using a key set generated at the point the client connects to the server. Encryption between clients is slightly more obscure.

If we use a cloud based architecture, i.e. clients connect to the server only, then clients will not communicate directly with clients. When a message is sent to a chat it is encrypted using the chat's specific key set, which will be generated when the chat is created. This means that when the message is sent to the server it will have been encrypted twice, using two separate key sets.

- Scalable server design

The server should be able to handle a high volume of traffic, perhaps the best way to do this would be through a concurrent system design. I will test that this criteria is met by deploying the application to a number of people and asking them to use the application at the same time. If the function of the client application is unaffected by the high volume then this criteria will have been met. We can also monitor the behaviour of the server for any anomalies that occur due to the high traffic.

- Scalable client application

The meaning of this is slightly less clear, here I am not necessarily talking about a client application which performs well under high traffic, since the client application will not necessarily have to endure such high traffic. Instead I simply mean that the application should be designed and implemented in such a way that does not inhibit potential future developments, so the code should be readable and

maintainable, and should have “space” for future features, i.e. the design should facilitate future developments.

Design

Decomposition

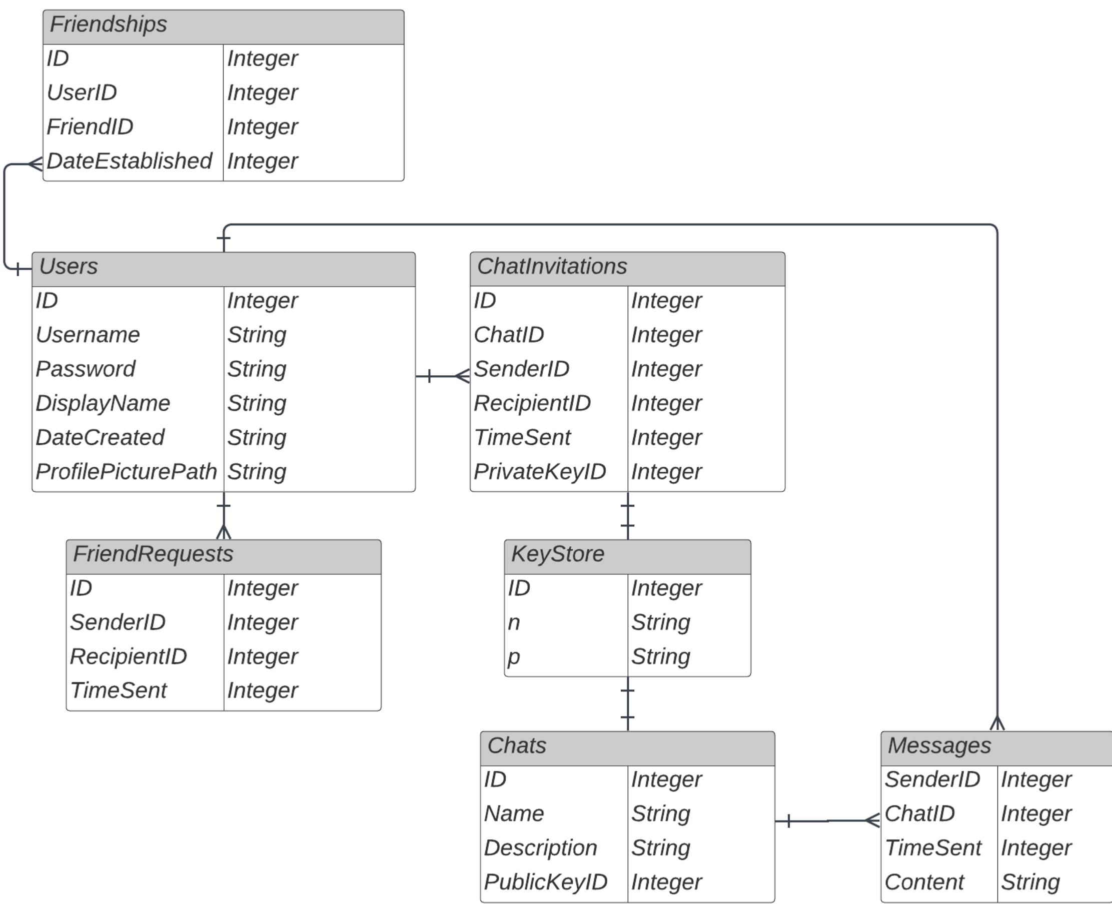
Clearly there will be two separate parts of this application which will work together to produce the desired functionality in the final product. Those two parts are the server, and the client application. These two parts can then be broken down further into even smaller parts which work together to produce those larger parts. In this section of the project we will discuss each component of the application.

Server

The server is the central point of the application, all client devices will connect to the server and perform operations through the server

Database

The Database will store all information relevant to the application, the primary database structure will be a *MySQL* database, with some extra structures for different bits of data. Following is an entity relationship diagram which details the contents and structure of the database.



This structure has been derived from the primary functions of the system. We will need to store users in the database since client applications will need to authenticate themselves with the server upon connection, and there will be no way to do this if user data is stored only on the client device. The client should store its user data so it can authenticate itself, but the server should store data for *all* users.

The server will also store friendships, these are links between two users who are *friends*, the application will request these links to display them on the application, so that the user can invite people to chats. Friend requests will also be stored on the server, so that the user can choose to accept or decline them, these must be stored on the server since they are subject to frequent change, and a client application may need to get the friends of another user, for whatever reason. To facilitate this, this data must be stored centrally on the server.

Information about all chats should also be stored on the server, since client applications will need to request this when they are joining a chat. So logically they must be stored centrally, otherwise the client will not be able to get this information and it won't know anything about the chat it is joining. Likewise, chat invitations must also be stored centrally since these will be subject to frequent change, and are effectively a form of communication between clients.

Encryption keys and messages will be stored differently to the rest of the data given above, since they must be handled in a different way. Keys will be stored in a hash table which links the *PublicKeyID* field of the Chats table to the index of the key in the hash table, this will be determined by the hash value of the key object, this will significantly improve the efficiency of requesting a key from the server, since it can be requested with a constant time complexity, rather than searching through a potentially very large database. The number of messages in a chat at any given time will be capped at 10, and all the messages in a chat will be requested when it is opened, so rather than storing all individual messages in a database table, we will store them in *MessageQueue* objects, which will in turn be stored in a hash table where the index is the *ChatID* of the chat a *MessageQueue* is linked to. Again this is just a much more efficient design than the use of a database table.

In order to protect the privacy of this sensitive data, the database will be closed off from direct outside access, to access the data it contains, an application not on the local network would have to go through the main server application, performing the authentication process and then only being able to select certain data in certain ways which the server permits. You will be able to access the database if you are connected to the local network, primarily for debugging and maintenance purposes, but devices outside the local network will only be able to access it through the server. Also on the topic of security, when a client sends a chat invitation, they must send the chat's private key along with it, since the accepting client will require it in order to use the chat. It is a security risk to store these on the server for an extended period, so we should implement a process to delete them after a set period, say 30 days. We could also delete the entire chat invitation at that point, and potentially friend requests.

As such we could have the following pseudocode implementation of the database.

MySQLHandler

This class requires extra configuration data which will be stored in a file under the path *./Assets/MySQL_Config.json*, as the name suggests, this is a JSON file and should be written in the following format:

```
{
    "connection_url": <string>,  // This is the JDBC link to connect to the server
    "username": <string>,        // This is the username the server should use to connect to the MySQL database
    "password": <string>         // This is the password the connection should be authenticated with
}
```

```
FUNCTION getConnection(connection_url, username, password)  // Get a connection to the MySQL server using the given
parameters
FUNCTION loadJSONFile(path)                                 // Loads the JSON file at path

CLASS MySQLHandler
    VAR connection  // Represents a connection between the program and the MySQL database
    VAR config      // Stores the current MySQL configuration

    FUNCTION MySQLHandler()
        // Get the MySQL configuration
        config = GetMySQLConfig()

        // Start the connection to the MySQL server
        StartConnection()
    END FUNCTION

    FUNCTION GetMySQLConfig()
        // Returns the configuration of the MySQL server as given by the JSON config file
        RETURN loadJSONFile("Assets/MySQL_Config.json")
    END FUNCTION

    FUNCTION StartConnection()
        // Get a connection to the MySQL server
        connection = getConnection(config.get("connection_url"), config.get("username"), config.get("password"))
    END FUNCTION

    FUNCTION Select(query)
        // Perform a Select statemenet on the database (or another query which produces a result set)
        VAR stmt = connection.createStatement()
        stmt.execute(query)

        RETURN stmt.getResultSet()
    END FUNCTION

    FUNCTION Update(query)
        // Perform an Update statement on the database (or another query which does NOT produce a result set)
        VAR stmt = connection.createStatement()
        stmt.execute(query)
    END FUNCTION

    FUNCTION GetUserByID(int UserID)                        // Get a user by their ID in the database
    FUNCTION GetUserByUsername(String Username)             // Get a username by their username in the
database
    FUNCTION GetUserByDisplayName(String DisplayName)       // Get users whose display names match
DisplayName + "%"
    FUNCTION GetFriendshipByID(int FriendshipID)            // Get a friendship by its ID in the database
    FUNCTION GetFriendshipByUserID(int UserID)              // Get friendships by their user ID in the
table
```

```
    FUNCTION GetFriendshipByUserIDAndFriendID(int UserID, int FriendID) // Get a friendship by its user ID and friend
ID in the table
    FUNCTION GetFriendRequestsByRecipientID(int RecipientID)            // Get friend requests by their recipient ID
    FUNCTION GetFriendRequestsBySenderID(int SenderID)                  // Get friend request by their sender ID
    FUNCTION DeleteFriendRequest(int FriendRequestID)                   // Delete a friend request from the table
where the ID is FriendRequestID
    FUNCTION GetChatByID(int ChatID)                                    // Get a chat by its ID in the database
    FUNCTION GetChatByPublicKeyID(int PublicKeyID)                      // Get a chat by its public key ID
    FUNCTION GetChatInviteByID(int ChatInviteID)                        // Get a chat invite by its ID
    FUNCTION GetChatInvitesByRecipientID(int RecipientID)               // Get chat invites by their recipient ID
    FUNCTION GetChatInvitesBySenderID(int SenderID)                     // Get chat invites by their sender ID
    FUNCTION DeleteChatInvite(int ChatInviteID)                         // Delete a chat invite where ID ==
ChatInviteID
    FUNCTION AddUser(User user)                                         // Add a user to the database
    FUNCTION AddFriendship(Friendship friendship)                       // Add a friendship to the database
    FUNCTION AddFriendRequest(FriendRequest friendRequest)              // Add a friend request to the database
    FUNCTION AddChat(Chat chat)                                         // Add a chat to the database
    FUNCTION AddChatInvite(ChatInvite chatInvite)                       // Add a chat invite to the database
END CLASS
```

MessageStore

```
FUNCTION CreateNewHashTable()              // Function which creates a new hash table
FUNCTION LoadHashTableFromFile(path)       // Function which loads a hash table from a file at path
FUNCTION WriteHashTableToFile(path, table) // Function which writes a hash table to a file at path
FUNCTION Revert(data)                      // Revert data to its original state (undo changes)

CLASS MessageStore
    VAR data  // Hash table which stores message queues

    FUNCTION MessageStore()
        // If the data file exists, read the hash table from the file, or create a new hash table and write it to the
file location
        IF "Assets/data/MessageStore.bin" exists THEN
            data = ReadFromFile()
        ELSE
            data = CreateNewHashTable()
            WriteToFile()
        END IF
    END FUNCTION

    FUNCTION ReadFromFile()  // Read the hash table from the file
        RETURN LoadHashTableFromFile("Assets/data/MessageStore.bin")
    END FUNCTION

    FUNCTION WriteToFile()  // Write the hash table to the file
        WriteHashTableToFile("Assets/data/MessageStore.bin", data)
    END FUNCTION

    FUNCTION GetMessageQueue(id)  // Get a message queue from the hash table
        RETURN data.get(id)
    END FUNCTION

    FUNCTION AddMessageQueue(queue)  // Add a message queue to the hash table
        data.put(queue.id, queue)
        WriteToFile()

        IF write failed THEN
            Revert(data)
            RETURN failed
        ELSE
            RETURN success
    END FUNCTION
```

```
    FUNCTION RemoveMessageQueue(id)  // Remove a message queue from the hash table
        data.remove(id)
        WriteToFile()

        IF write failed THEN
            Revert(data)
            RETURN failed
        ELSE
            RETURN success
    END FUNCTION
END CLASS
```

MessageQueue

```
CLASS MessageQueue
    VAR ChatID  // ID of the chat this message queue is linked to
    VAR data    // The queue containing the messages in this chat

    FUNCTION MessageQueue(ChatID)
        ChatID = ChatID    // Assign the chat id
        data = Queue(10)  // Create a new queue of maximum size 10
    END FUNCTION

    FUNCTION Push(message)
        data.Push(message)
    END FUNCTION

    FUNCTION Pop()
        RETURN data.Pop()
    END FUNCTION

    FUNCTION Get(i)
        RETURN data.Get(i)
    END FUNCTION

    FUNCTION GetJSONString()
        VAR result = StringArray(10)  // Create a string array of size 10
        FOR i = 0, i < 10
            IF data.Get(i) == null
                continue
            END IF

            result[i]  = data.Get(i).GetJSONObject().toJSONString();
        END FOR

        RETURN result
    END FUNCTION
END CLASS
```

KeyStore

```
FUNCTION openFile(path)        // Open a file at path
FUNCTION createNewHashTable()  // Create a new hash table

CLASS KeyStore
    VAR data      // Hash table containing keys
    VAR dataFile  // The file which contains the hash table

    FUNCTION KeyStore()
        dataFile = openFile("Assets/Data/KeyStore.bin")  // Open the data file

        // Check if the file contains the hash table.
```

```
        // If not we should create a new one.
        IF dataFile contains hash table THEN
            data = ReadFromFile()
        ELSE
            data = createNewHashTable()
            WriteToFile()
        END IF
    END FUNCTION

    FUNCTION ReadFromFile()
        RETURN readFile(dataFile)
    END FUNCTION

    FUNCTION WriteToFile()
        writeFile(dataFile, data)
    END FUNCTION

    FUNCTION GetKeyPair(keyID)
    FUNCTION AddKeyPair(pair)
    FUNCTION RemoveKeyPair(keyID)
END CLASS
```

Database

```
CLASS Database
    VAR MySQLHandler
    VAR MessageStore
    VAR KeyStore

    // This class effectively just a wrapper class.
    // All methods implemented in this class essentially just call
    // methods in the other database systems.
    //
    // To reduce the size of this pseudocode I won't show them here,
    // because there isn't much point anyway.
END CLASS
```

The *Database* class is effectively a wrapper class that simplifies the rest of the program's access to the various database systems by providing a single unit to access them through, and determining which system a request should be directed to.

Server functions

The primary use of the server will be to retrieve data from the database that the application needs for whatever reason. Following is an overview of the data that will be stored in the database.

- User data
- Chats
- Friendships
- Invitations (to chats or friend requests)
- Messages
- Public encryption keys for chats

There may also be a number of ways that users can request such information, for example a user could be found by searching for their display name, username, or their user ID.

The client application will also need to show notifications when certain events happen, since the server is the central point of the entire application, it will know when these events happen, so it should

have some way of notifying the relevant client application when those events happen. This could be done using a *listen rule* system. Client applications can register so called listen rules with certain conditions, when a client makes a request the server will check all registered listen rules to see if the request meets any of the rules' conditions. If they are, then the request is forwarded to the client application that registered the rule and the request is then handled as normal. This adds a few extra functions to the server.

- Adding listen rules
- Removing listen rules

The server will consist of a number of parts which work together to create a concurrent server program which handles user requests efficiently. These main parts are:

- *Server* class
  - Responsible for listening for and accepting incoming connections from client devices.
  - Passes accepted connections to a queue.
- *ConnectionHandler* class
  - When given a connection, this class will handle requests that come through this connection in a separate thread, until the connection is closed.
- *QueueManager* class
  - Checks if the queue has any new connections in it.
  - If it does it will look for a free connection handler and pass the first connection in the queue to the free handler.
  - Or it will wait until a handler is free.

From this we could interpret the following pseudo code implementation

Server

This class will take extra configuration data stored under *Assets/Server_Config.json*:

```
{
    "port": <string (castable to int)>,          // The port the server should accept connections on
    "maxThreadCount": <string (castable to int)>  // The maximum number of handlers that should be created by the
server
}
```

```
FUNCTION openFile(path)      // Open a file at the given path
FUNCTION loadJSONFile(file)  // Load the contents of a JSON file

CLASS Server
    VAR port                            // The port on which this server will accept connections
    VAR maxThreadCount                  // The maximum number of handler processes
    VAR connectionHandlerPool           // Array of connection handlers
    VAR connectionHandlerQueueManager   // The queue manager process for the connection handler pool
    VAR db                              // Database instance
    VAR listenRules                     // List of currently active listen rules registered by client devices

    FUNCTION main()
        VAR server        // Server instance
        VAR serverSocket  // Server socket which will accept connections

        WHILE (true)
            // Accept a connection and then push it to the queue manager's queue
            VAR connection = serverSocket.accept()
```

```
            connectionHandlerQueueManager.queue.Push(connection)
        END WHILE
    END FUNCTION

    FUNCTION Server()
        VAR config = GetConfigFile()  // Get the configuration file for the server

        // Assign values from the configuration data
        port = config.get("port")
        maxThreadCount = config.get("maxThreadCount")
    END FUNCTION

    FUNCTION GetConfigFile()
        VAR configFile = openFile("Assets/Server_Config.json")
        RETURN loadJSONFile(configFile)
    END FUNCTION
END CLASS
```

Handler

I should clarify that this is intended to be a parent class, not an implementation of the actual connection handler, that will come later and will inherit from this class.

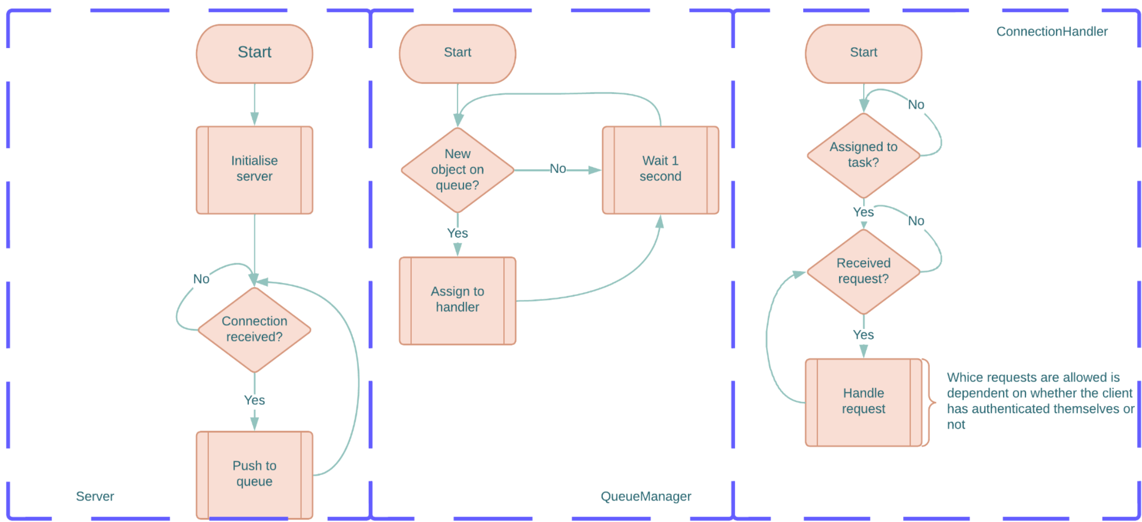
The reason for this class is that it will simplify the process of adding other handler types later if they are needed.

```
CLASS Handler
    VAR socket          // The main communication TCP/IP socket
    VAR threadNum       // The thread number, used in debug messages
    VAR className       // The name of this class (used in debug messages)
    VAR keyPair         // This handler's encryption key pair
    VAR clientKeyPair   // The client's encryption key pair
    VAR busy            // Determines if the handler is busy or not
    VAR server          // The current server instance
    VAR queueObject     // The object supplied to this handler by the queue
    VAR authenticated   // Determines if the connected client is authenticated
    VAR lrSocket        // TCP/IP socket used for communicating listen rule triggers

    FUNCTION Handler(socket, threadNum, className)  // Constructor method
    FUNCTION InitializeIO()                          // Initialize the I/O streams for the main socket
    FUNCTION StopHandler()                           // Pauses this handler's execution process
    FUNCTION DebugLog(message)                       // Output a message with this handler's signature

    FUNCTION Send(obj)                               // Send an object over the main socket
        socket.sendObject(obj)
    END FUNCTION

    FUNCTION LrSend(obj)                             // Send an object over the listen rule socket
        lrSocket.sendObject(obj)
    END FUNCTION

    FUNCTION Receive()                               // Receive an object over the main socket
        RETURN socket.readObject()
    END FUNCTION

    FUNCTION Close()                                 // Called when connection is closed
        socket.close()
        lrSocket.close()

        // Remove listen rules created by this handler from the listen rules list on the server
        VAR emptyPass = false
```

```
        WHILE (!emptyPass)
            emptyPass = true

            FOR i in 0 to server.listenRules.size()
                IF (server.listenRules.get(i).handler == this) THEN
                    server.listenRules.remove(i)
                    emptyPass = false
                    break
                END IF
            END FOR
        END WHILE
    END FUNCTION
END CLASS
```

Server process flow diagram

Following is a flow chart which illustrates the server process. Each box is a single process.



The *ConnectionHandler* will contain a number of methods, each of which handling a different type of request. These requests will require different information to complete, these methods will assume that this data is present in the request body, but will cause errors when this data is not present. This way the server can catch these errors and respond appropriately to the client. The possible types of requests are listed below, with the data they require in the form of a JSON object, which is how they should be sent to the server, and the data they return (all methods may also return null in certain circumstances, this may be if the data is not found, or an error occurs), the type of request will be stored under the key “type”:

- Authenticate
    - “Data”: User database entity containing the authentication data for the user.
        - *String*: “failed” - Returned if authentication fails for any reason
        - *User* - The full user data that the client just authenticated, returned if authentication is successful.
- Get User
    - “data” : User database entity containing the data specified by the selector field.

- “Selector”: “id”, “username”, or “displayName”, this is the field in the data object which contains the information the server should search by to find the requested user data.
    - *User* - Returned if selector is “id” or “username”, since there will only ever be 0 or 1 result for this.
    - *User[]* - Returned if selector is “displayName”, since this field allows duplicate values.
- Get Friendship
  - “Data”: Friendship database entity containing the data specified by the selector field.
  - “Selector”: “id”, “userID”, or “userID&FriendID”, this is the field/s in the data object which contains the information the server needs to find the requested friendship record.
    - *Friendship* - Returned if selector is “id” or “userID&FriendID”, since these fields are unique to singular records.
    - *Friendship[]* - Returned if selector is “userID”, since this field can be duplicated across multiple records.
- Get Friend Requests
  - “Data”: The friend request database entity containing the data required by the server to search for friend requests.
  - “Selector”: “senderID” or “recipientID”.
    - *FriendRequest[]*
- Get Chat
  - “Data”: The Chat database entity containing the ID of the chat to retrieve.
    - *Chat*
- Get Chat Invite
  - “Data”: The chat invite database entity containing the required data.
  - “Selector”: “id”, “senderID”, or “recipientID”
    - *ChatInvite*: Returned if selector is “id”
    - *ChatInvite[]*: Returned if selector is “senderID” or “recipientID”
- Get Public Key
  - “Data”: Integer object containing the ID of the key in the keystore.
    - *KeyPair*
- Get Message Queue
  - “Data”: Integer object containing the ID of the message queue in the message store.
    - *MessageQueue*
- Add User
  - “Data”: The user database entity containing the user data to add to the database.
    - *User*: Returns the full data of the user that was just added to the database
- Add Chat
  - “Data”: The chat database entity containing the chat data to add to the database.
    - *Chat*: Returns the full data of the chat that was just added to the database.
- Add Listen Rule
  - “Data”: The listen rule to add to the server.
    - *Integer*: Returns the ID of the listen rule
- Remove Listen Rule
  - “Data”: The ID of the listen rule to remove from the server.

        - *String*: "done" or "failed", depending on whether or not the operation was successful
- Accept Friend Request
    - "Data": The friend request database entity containing information about the friend request to accept.
        - *String*: "done" or "failed", depending on whether or not the operation was successful
- Decline Friend Request
    - "Data": The friend request database entity containing information about the friend request to decline.
        - *String*: "done" or "failed", depending on whether or not the operation was successful
- Accept Chat Invite
    - "Data": The chat invite database entity containing information about the chat invite to accept.
        - *KeyPair*: The private key of the chat that the client just accepted an invitation for
- Decline Chat Invite
    - "Data": The chat invite database entity containing information about the chat invite to decline.
        - *String*: "done" or "failed", depending on whether or not the operation was successful
- Send Message
    - "Data": The message database entity to add to the database.
        - *String*: "done"
- Send Friend Request
    - "Data": The friend request database entity to add to the database.
        - *String*: "done" or "failed", depending on whether or not the operation was successful
- Send Chat Invite
    - "Data": The chat invite database entity to add to the database.
    - "keyPair": The private key for the chat this chat invite refers to.
        - *String*: "done" or "failed", depending on whether or not the operation was successful

It will also be responsible for handling the handshake process at the start of a new connection. This process will involve generating a new encryption key pair, and receiving the client's generated encryption key pair, and starting the separate listen rule socket. This process effectively establishes all necessary communication links between the client and the server, and enables all communication to be encrypted.

Build targets

So we have covered the contents of the server program, let us look now how it should be built.

The client application may require access to some parts of the server, not necessarily to use them but to be aware of the types and classes involved. To do this we should build the server into a library JAR file, which I will call *ServerLib.jar*.

Of course there should also be an executable build, which will also be included as a JAR file called *MessageCatServer.jar*, which has the main class / method specified as *com.nathcat.messagecat_server.Server.main(String[] args)*.

The RSA implementation will also be packaged into a library JAR file called *RSA.jar*.

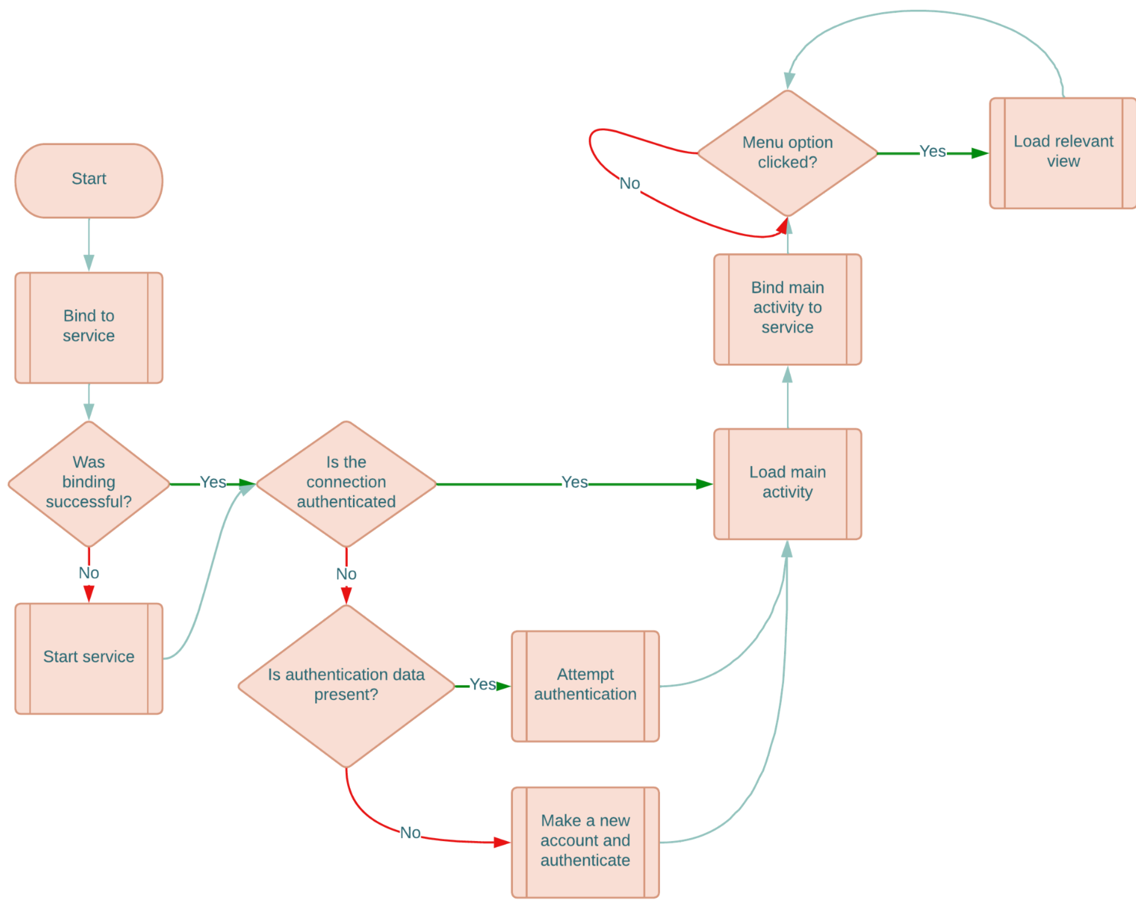
Client application

The client application is the front end of this application, this is the part of the system that the users will interact with. This will take the form of an Android application.

The main process in an Android application is the UI thread, this is where all UI related operations must be performed, and it is the process which is launched when the app is opened by the user. Android requires that applications perform networking operations *on a separate process*, the reason for this is that networking operations can often be slow, which will block the operation of the UI thread, leading to an unresponsive UI. Handling networking processes on a separate process avoids this issue by having the networking operations block a different process. So we will need to create a separate process when the application is launched, this process should wait until it is required for a networking task, this can be accomplished by using the Android SDK *Handler* class, we can derive from this class and override its *handleMessage(Message)* method to perform networking tasks in a separate process, as required. We could improve this further by creating a foreground *service* which provides an interface between the client application and the *Handler*. The client application can bind to this *Service* and make requests through there, the *Service* would handle communicating with the *Handler*, this offers a layer of abstraction to the client application's source code and a more layered approach which is generally more maintainable. To clarify, the *Handler* would perform a networking task given to it by the *Service*, and then call a *callback* function, which will perform some operation with the data that was received from the server. The callback function will be executed within the *Handler* process however, so it should be written with this in mind.

General process

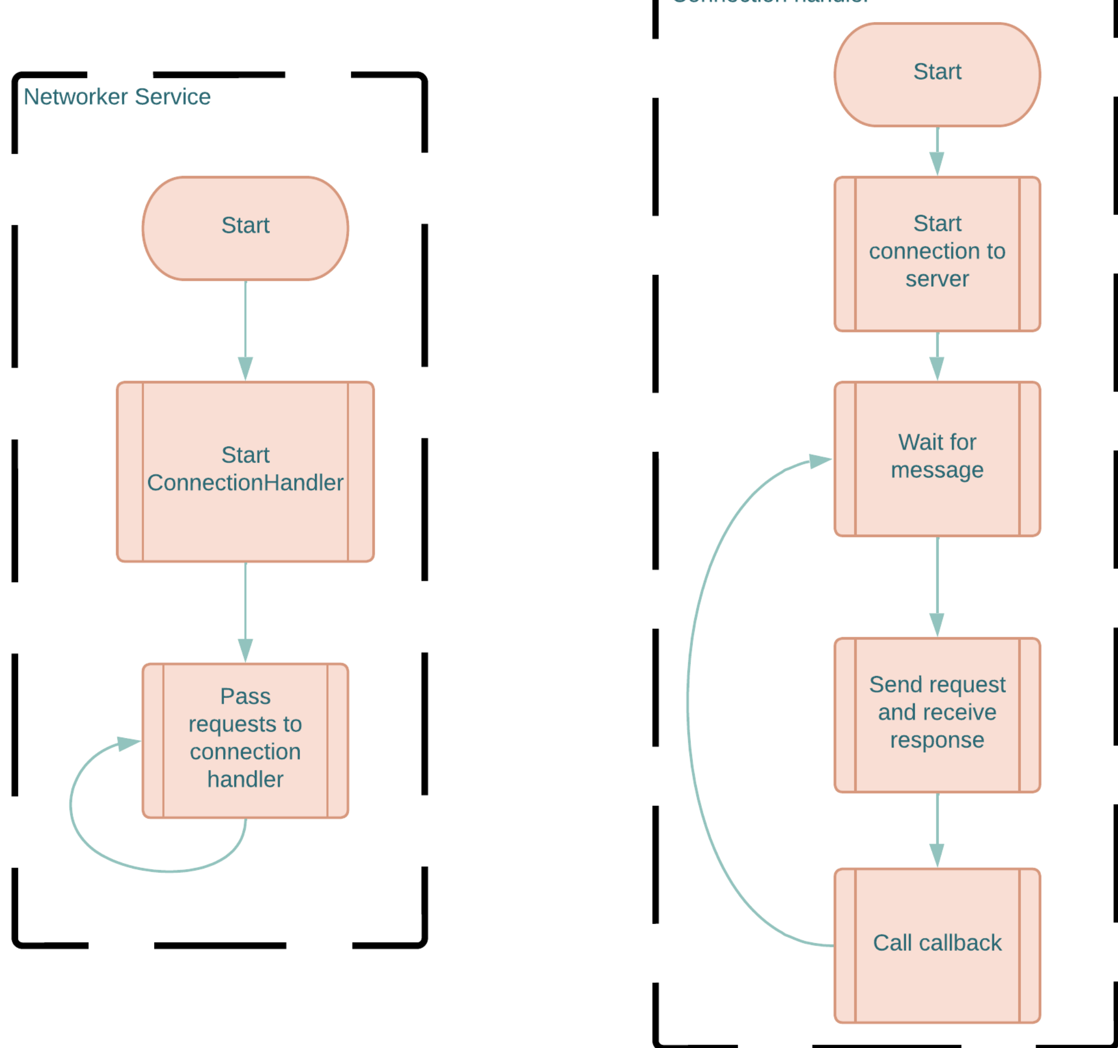
Following is a flow chart illustrating the *general* process the application will follow, particularly in the startup phase. There will be different parts of the application but this is the general process the overall application will follow.



Networker service and Connection handler

These are the parts of the application that should run persistently in the background and will be responsible for handling communication between the client application and the server. They are relatively simple processes in practice but they may be difficult to implement, although the Android

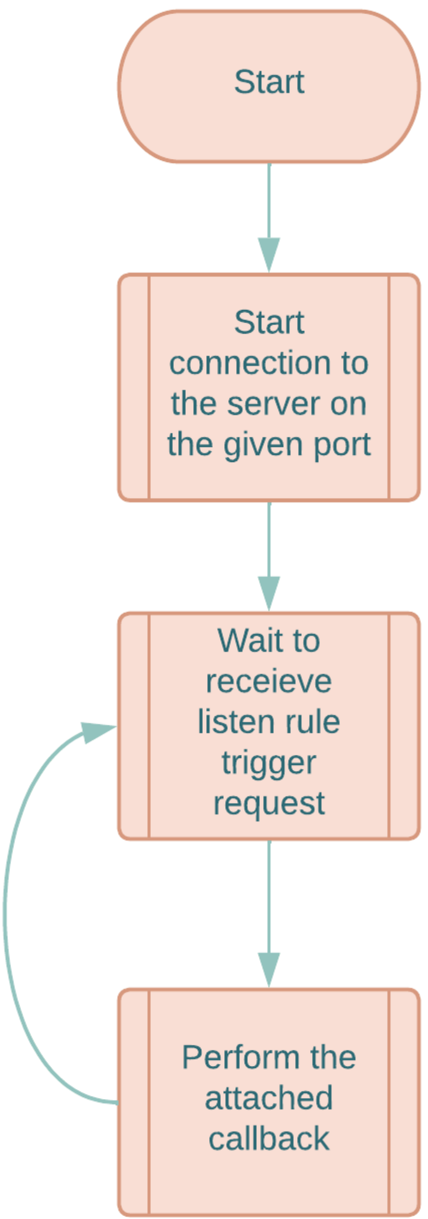
SDK provides ways for us to implement such a system.



The application will also create a number of listen rules when the application starts that will show notifications when the following events occur:

- The user is sent a message through one of their chats
- The user is sent a friend request
- The user is sent a chat invite

Following is a flow chart illustrating the listen rule handler process.



This is a comparatively simple process, it does not need to be particularly complex since its only job is to execute callbacks when a listen rule is triggered on the server. The listen rule trigger request is a copy of the request that triggered the listen rule, along with information about the listen rule that this request triggered.

This service should be started when the device turns on, or if the service has not yet started it should be started by the application, so it will have to test if the service is already running, and we will likely have to use a *BroadcastReceiver* to get the service to start when the device has finished booting up.

Usability
With so many people using applications nowadays, they must always be designed with usability features to ensure that anyone can use them. The application should be simple in its design and be clear in how it is to function.

Given that the concept of this application is relatively simple, I don't think it would be necessary to create an explicit tutorial on how to use the application, but we should ensure that the design is intuitive and similar in ways to other messaging applications, to ease the transition and make it simpler to use.

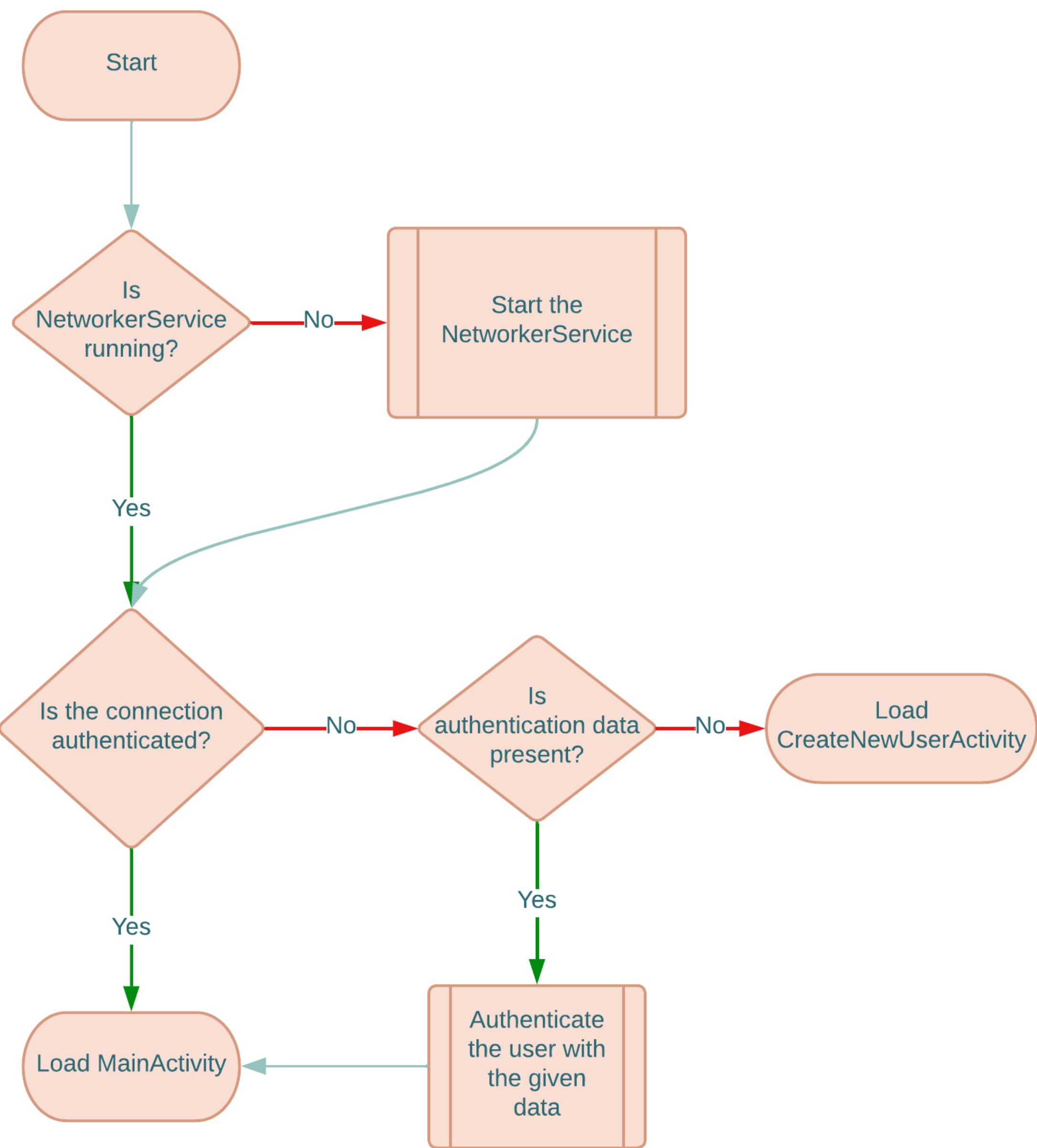
There is the option of creating a website which provides helpful instructions on how to use the application, perhaps some kind of forum, but this is not part of the scope of this project. Each page of the application should have a clear function, and the way to use them should be clear from the way it is designed as well. We can also use text hints and descriptions to show what each component on the page does.

UI
The user interface is the part of the application that the user will interact with, as such it should be well designed and user friendly. UI operations in Android will operate in a thread separate to the connection handler, so we should design the interface with the knowledge that there may be periods where the interface is forced to wait to receive certain information before it can be displayed.

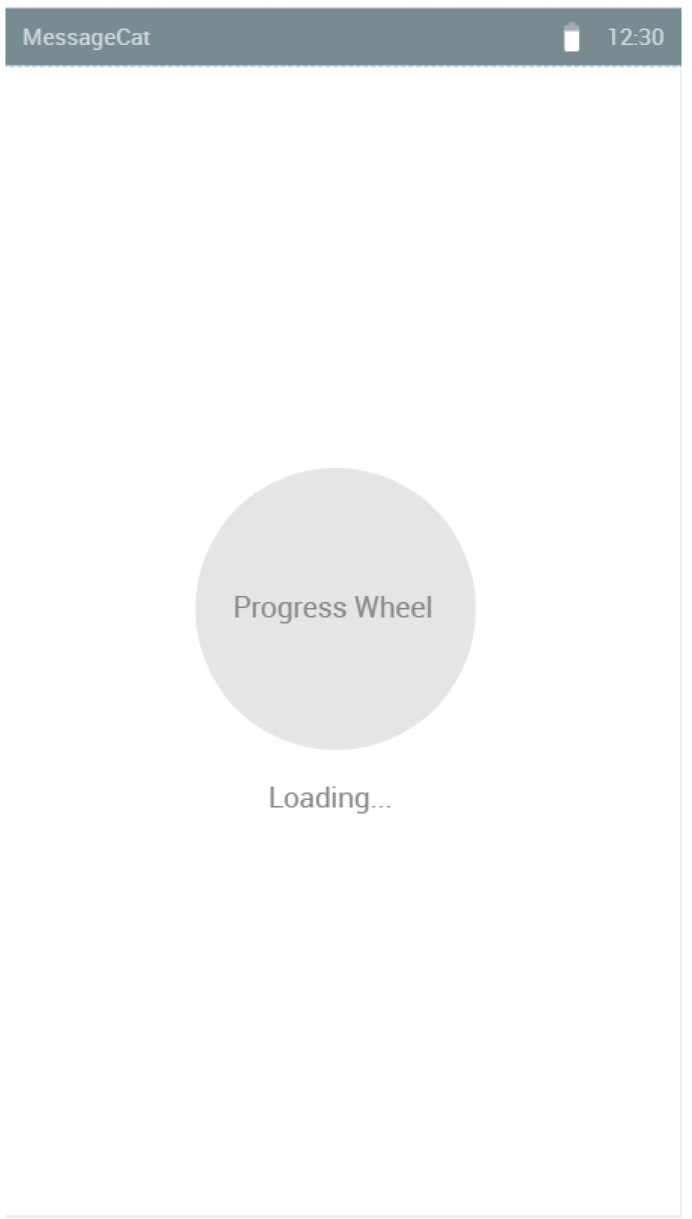
The UI for this application can be separated into 7 distinct pages:

LoadingActivity

This page is the starting point for the application, it will be responsible for ensuring that all required parts of the application are active and accessible before loading the main application page.



And a wireframe design of this page.



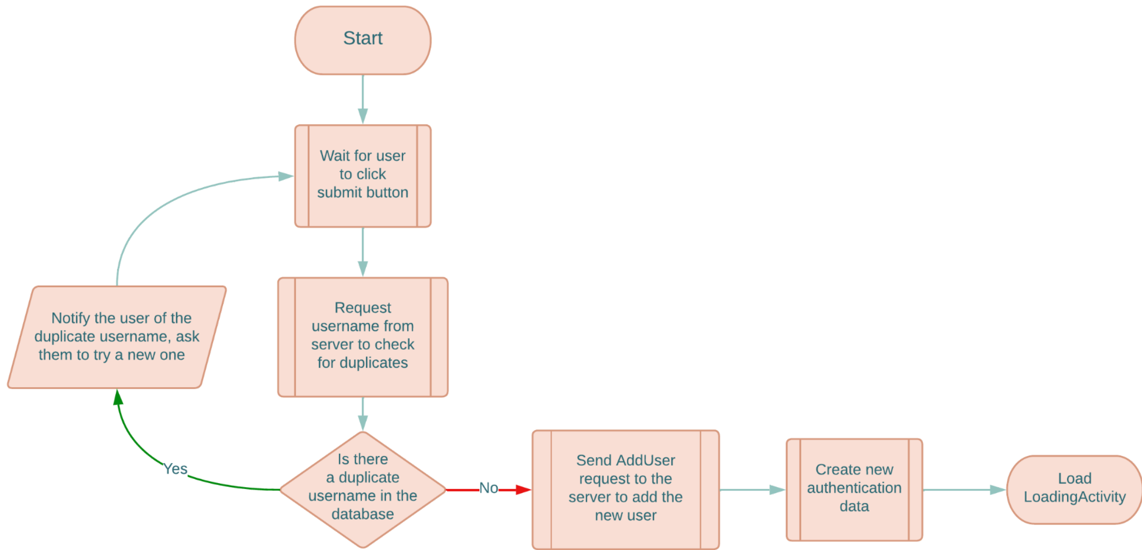
CreateNewUserActivity

This page will be loaded if there is no valid user that can be authenticated on the device. It will allow users to create an account which the application will save authentication details for so it can login automatically whenever the networker service is started.

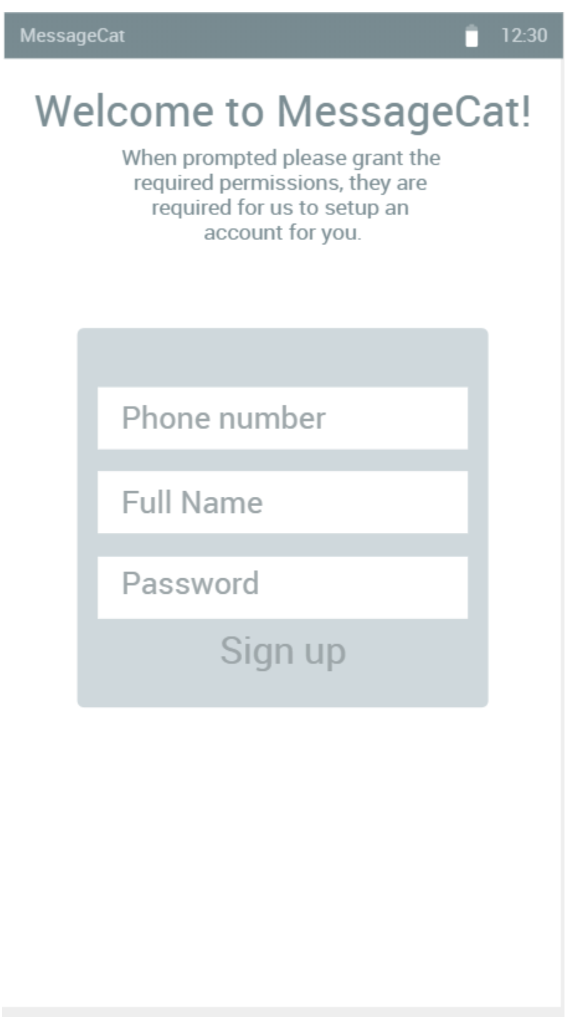
When the user presses the submit button, the application will send a request to the server to create a new user with the entered information, but before this it should perform some checks to ensure that the data is valid:

- Request the entered username from the server to ensure there are no other users with the same username (although this check may be implemented server side as it is just easier to do it there rather than have unnecessary requests into the server).
- Check that the password is entered correctly by having a password re-entry box and checking if the entries match.

These checks should ensure that the user's entry is valid and acceptable by the application and the server. Furthermore, to improve the security of the user's authentication information, their password will be hashed using the *SHA-256* hashing algorithm before being sent off to the server. This will be implemented client side to ensure that the user's actual password never leaves the device they entered it on.

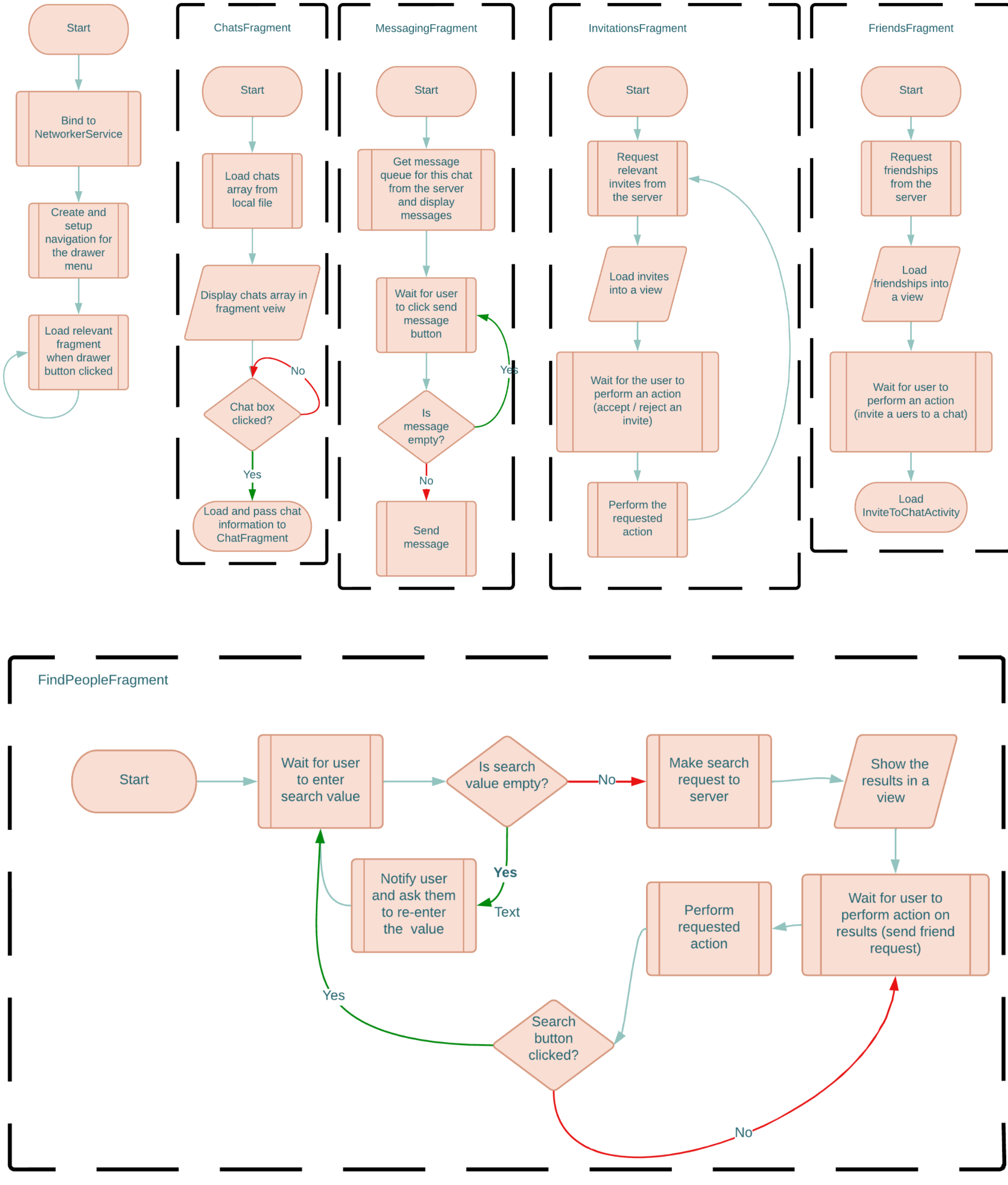


And a wireframe design of this page.



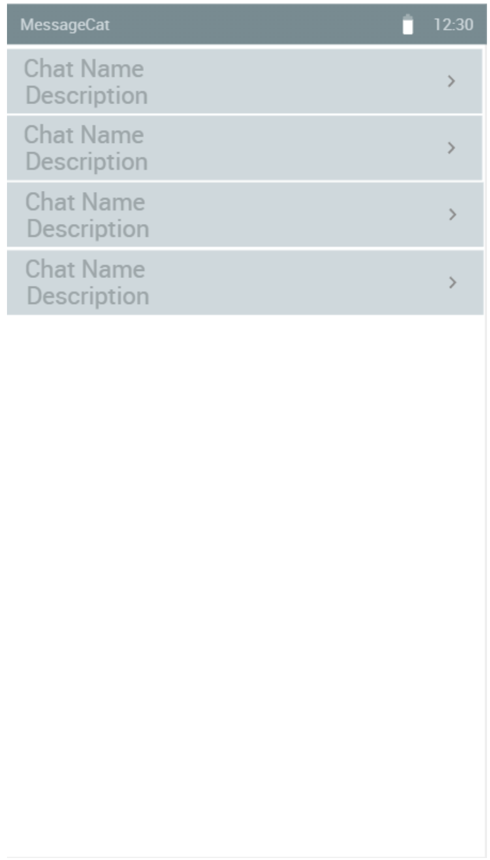
MainActivity

This is the main page of the application. Once all checks have been done and the user is authenticated, this page will be loaded. All the primary functions of the application will be accessible from this page via a drawer menu which can be pulled out from the side of the window.

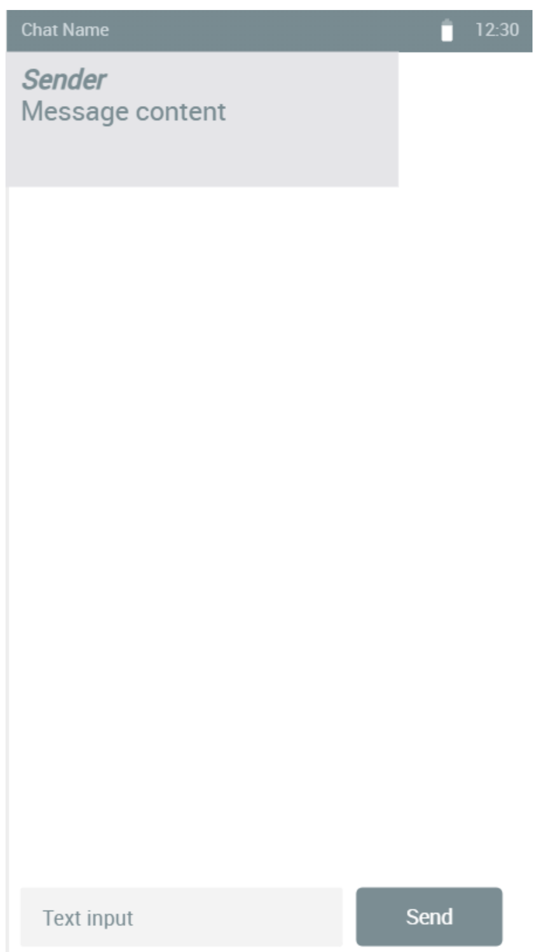


This page will include a main window, which has a pull out menu on the side of the window, and a main display port, which displays a *fragment*, which is essentially a sub-page.

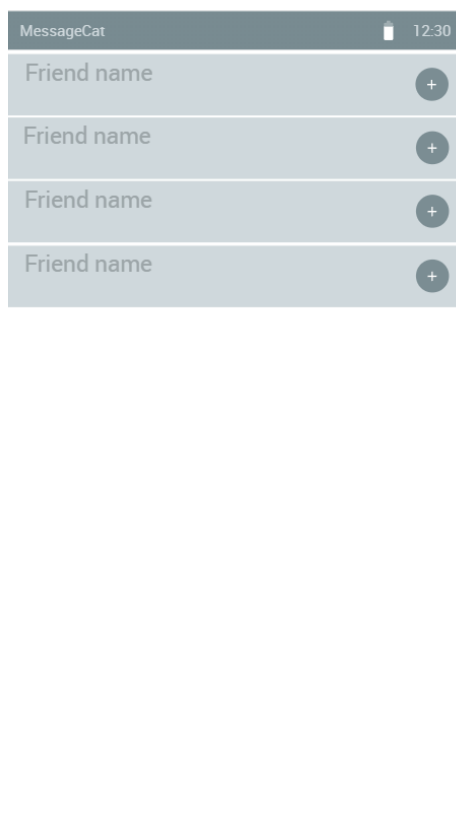
Here is the fragment used to display the chats the user is a part of (*ChatsFragment*):



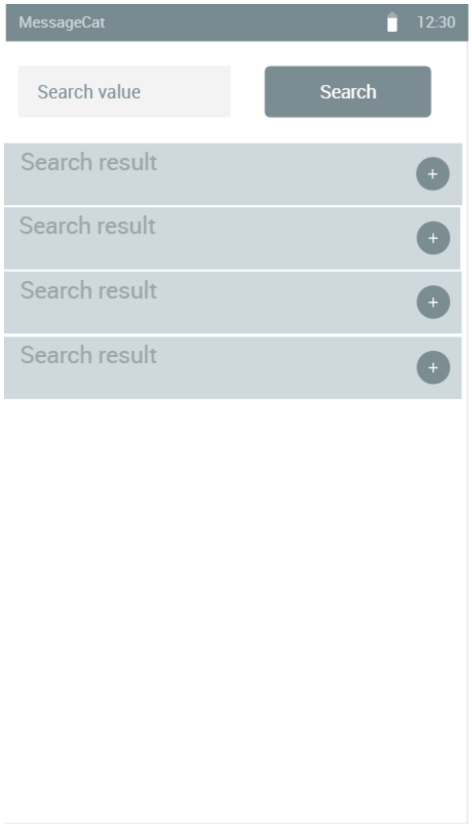
The fragment used to display messages in a chat, and that allows users to send messages to that chat (*MessagingFragment*):



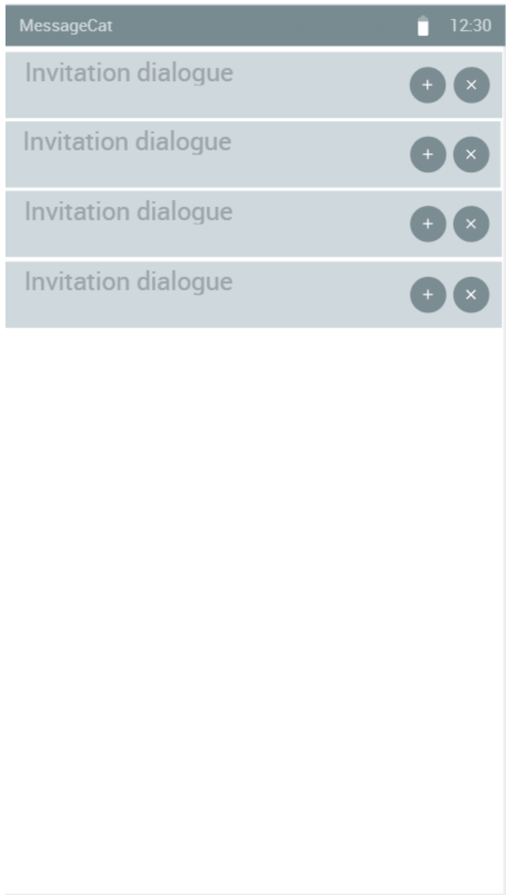
A fragment to display the user’s friends list (*FriendsFragment*):



A fragment that allows users to search for other users by their display name (*FindUserFragment*):



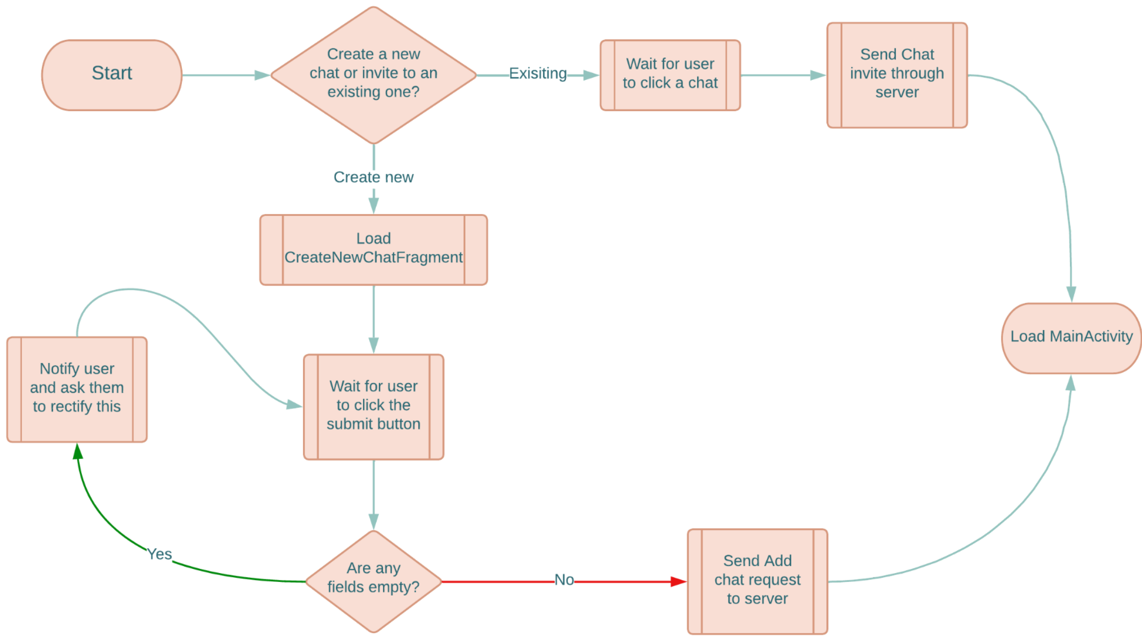
And finally, a fragment which displays invitations sent to the user, allowing them to accept or decline them (*InvitationsFragment*):



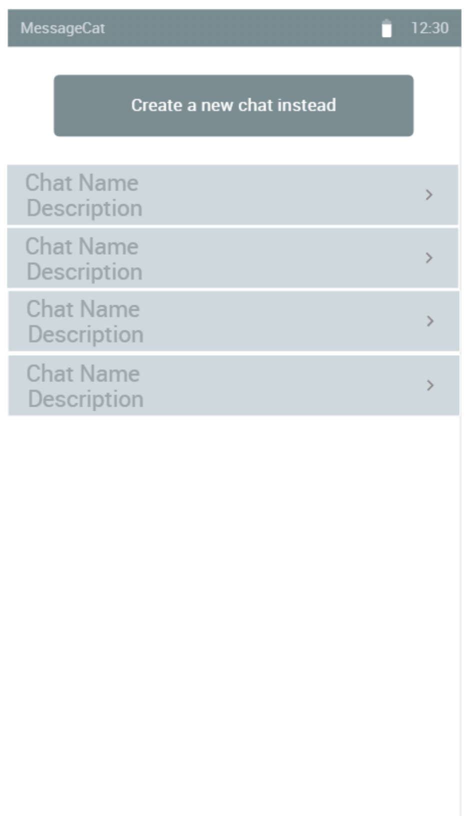
The invitation dialogue will be specific to a type of invitation, for example a friend request might say, "{displayName} wants to be friends!", or a chat invitation might say "{displayName} invited you to chat!".

InviteToChatActivity

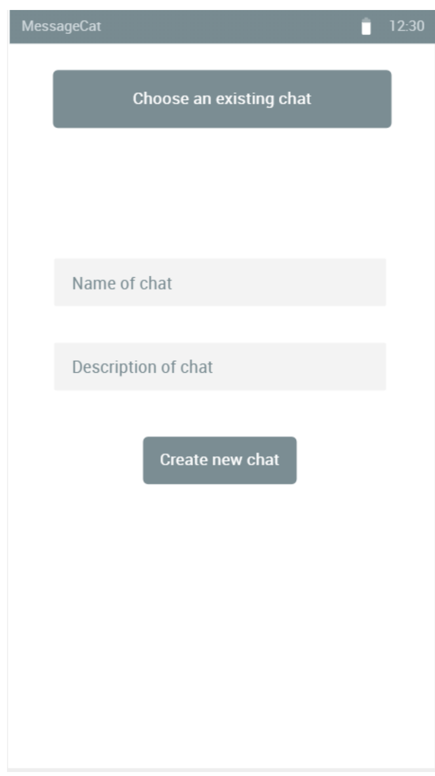
This page will be responsible for allowing the user to invite others to a chat. They will be given the option to choose between creating a new chat and inviting their friend to that new chat, or just inviting their friend to an existing chat.



And a wireframe design of this page for inviting a user to an existing chat:



And for a new chat:



Notifications
Notifications are an important part of a lot of applications, especially in this one since users will not be telepathically able to know when they have received a new message, or a friend request, or maybe a chat request, so the application should have a method of alerting the user when these events occur so that they are aware of it and can respond how they see fit.

Conveniently, the Android SDK provides a way for us to create notifications from the app using notification channels. These channels have a set priority and certain rules that notifications entered into them must follow, and they are a way of easily grouping together notifications from apps which have different purposes.

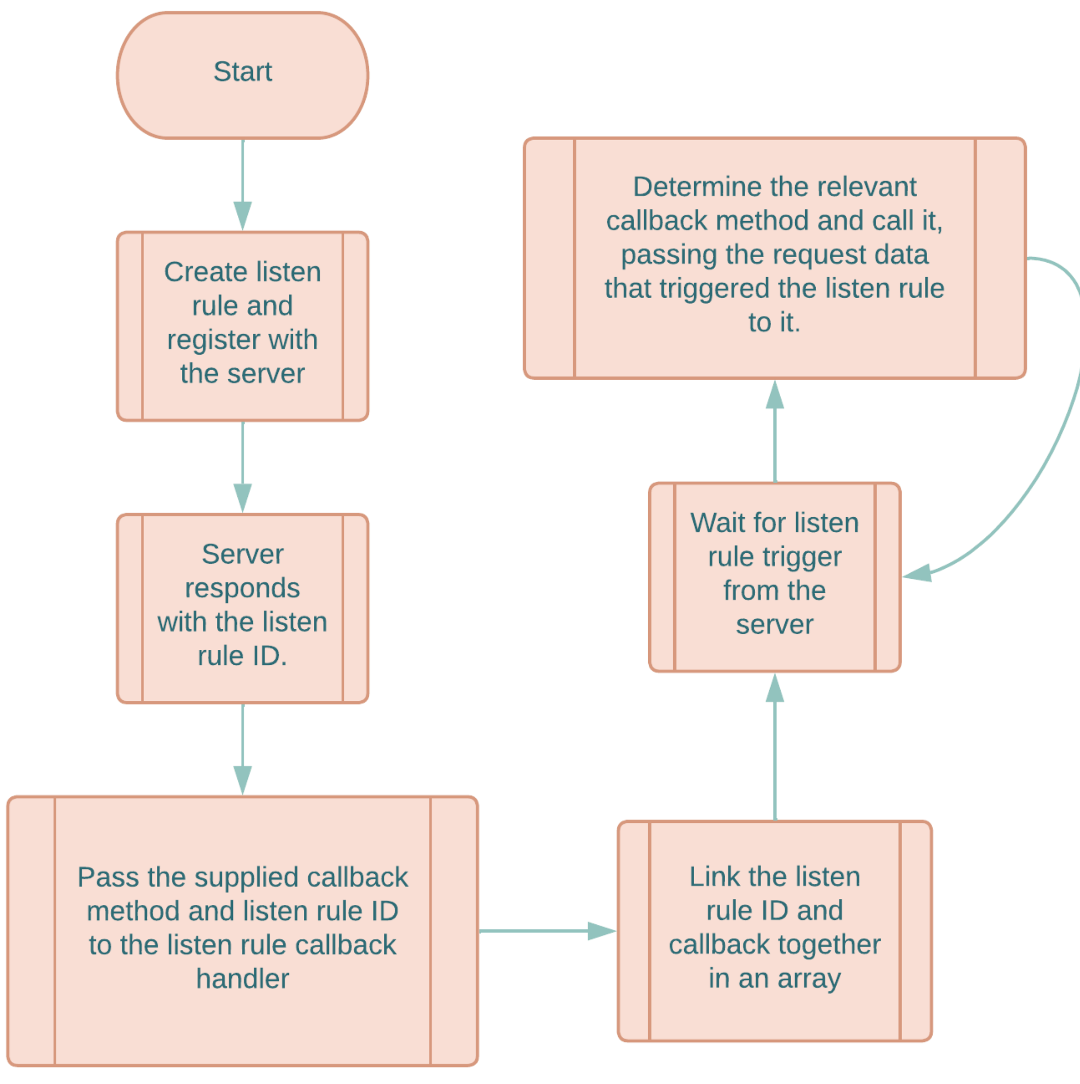
In this application there will be two main types of notification, the first type is the service notifications. This is a notification which is always present in the notification bar, and alerts the user that the networker service is running, it will say something like, *"The MessageCat Service is running."*. This notification is required to create a persistent service, like we are trying to create with the notification service.

The second type of notification is alerts for social events within the application, this will cover events such as:

- New messages in chats that the user is a member of.
- Friend requests sent to the user.
- Chat requests sent to the user.

So, we know how to implement notifications from an Android point of view, but how do we actually know when these events occur, and how do we get the data required to display information about them to the user? The answer to that lies in the listen rule system which will be implemented in the server, I won't go into detail about this system here as it is covered in the server design section above,

but the point is, we can register listen rules with the server and assign callbacks to them within the application which take the request data as a parameter, these callback methods would then create the notification with the information about the event. This would require another process called the *ListenRuleCallbackHandler* as part of the networker service, since this part of the application would be very similar in functionality to the *ConnectionHandler*, I won't cover it in excessive detail, but the main difference is that rather than handling incoming requests, it only receives listen rule trigger messages from the server for listen rules that the client has created, it should then link this to a callback given to it when the listen rule was created, and call said callback with the request data, this suggests the following flow chart illustrating the listen rule process from the client side.



Local storage

To improve the efficiency of the application, we can, and should, store some data in the client device's local storage. Some data of course *must* be stored locally for security reasons. Following is a list of data which will be stored locally.

- User data, used for authentication.

- Chats the user is a member of
- Private keys for chats the user is a member of

This data will be stored in three files, each of which containing a Java object which contains the required data:

- UserData.bin
  - Contains an instance of the *User* class.
- Chats.bin
  - Contains an array of chats, *Chat[]*.
- KeyStore.bin
  - Contains a hashmap linking the *public* key ID of a chat to its *private* key.
  - *HashMap<int, KeyPair>*.

Main process to connection handler process communication
The main process must connect to the connection handler in order to make network requests to the server. Since the connection handler is in a separate process, we must have a method of communicating between processes. This is the aim of the networker service.

The point of the networker service is that it *runs persistently*, meaning that it should not be interrupted by Android or another process (with the exception of the main application process). It is important to state that the service and the UI thread execute in the same process, which means that I cannot simply perform network requests from the networker service, since it will still block the UI thread, which is something Android does not allow. So the networker service manages the state and function of the connection handler, which manages the connection between the client and the server. The UI thread *binds* to the networker service, obtaining an instance of it, and is then able to send requests through it which are forwarded to the connection handler, which forwards them on to the server.

This process avoids having networking operations on the UI thread process, but introduces a new problem, if the UI thread is not blocked while this network request is waiting to be fulfilled, how does it know when it has completed, and how should it pass the response back so that the UI thread can handle it appropriately? In order to solve this problem I took inspiration from *Node.js*' asynchronous functions (sort of). A long time ago I developed a web application which made a number of web requests fairly frequently, and I used *Node.js*' asynchronous processing model to do this, and I created *callback functions* which were executed when these requests were completed. These callback functions took the response from the request as a parameter, and handled them accordingly, and I managed them using *function pointers*, although a more accurate term outside of C/C++ might be *function references*. By passing a reference to a function, you can execute that function anonymously, and pass it as an argument into other functions, which allows me to pass these callback functions into the asynchronous web request functions.

Now, Java does not (as far as I am aware) have an asynchronous programming model, but the same concept can be applied to synchronous programming in separate processes. I could create *interfaces* which declare callback methods which can be implemented when a request is made, and then pass the callback to the connection handler to be executed when a response is received from the server. I should consider the fact that this callback will be executed in a different process and a different context than the the UI thread, Java will handle the context by using *instance capturing*, but UI operations must be performed on the main thread so we should use Android SDK methods for this,

there is one in particular called *Activity.runOnUIThread(Runnable action)*, which allows us to execute a lambda function on the UI thread.

Algorithms

The two formal algorithms I will be using in this project are implementations of a Queue data structure, and RSA asymmetric encryption. I will not use a library for these, although they are available, since I can then create a more customised implementation which works better with this application.

Queue Data Structure

A key part of this application is the use of queue data structures, these allow for efficient processing of multiple requests and are integral in a number of the application systems. A queue is a first in first out data structure, following is a pseudocode implementation of a queue.

```
CLASS Queue
    VAR data        // Data array
    VAR maxSize     // The maximum size of the queue
    VAR endPointer  // Pointer for the end of the queue

    FUNCTION Push(obj)
        IF size of data > maxSize THEN
            RETURN FAILED
        END IF

        data[endPointer] = obj
        endPointer++
    END FUNCTION

    FUNCTION Pop()
        IF size of data == 0 THEN
            RETURN FAILED
        END IF

        VAR val = data[0]
        data[0] = NULL
        RETURN val
    END FUNCTION
END CLASS
```

This will be implemented as part of the database.

RSA Encryption Algorithm

End-to-end encryption plays a big role in securing user data, so it is important to ensure that this system is well built and works properly, so that the user's data is properly secured.

This application will use an Asymmetric encryption algorithm called RSA. RSA is a very common encryption algorithm used in many modern applications. It makes use of very large prime numbers and modular arithmetic to encrypt and decrypt messages using a public and private key. The following is the process for generating an RSA key pair for encryption and decryption.

First, choose two random coprime integers p and q.

Let $n = pq$.

We must now choose a value for e, also known as the public key exponent, the reason for this name will become clear later. The standard value for e is 65537, so we will use that.

Let $e = 65537$.

The public key is defined as $K_{public} = \{n, e\}$, and the private key is defined as $K_{private} = \{n, d\}$, so we now need to find d. This is where we need to use modular arithmetic,
$ed = 1 \bmod (p - 1)(q - 1)$, therefore d can be found by finding the inverse modulus of e and $(p - 1)(q - 1)$.

Now we have our key pair, we can perform encryption and decryption:
$K_{encrypt}(x) = x^e \bmod n$, $K_{decrypt}(x) = x^d \bmod n$.

We will use this method to generate key pairs which can encrypt and decrypt entire Java objects. The encrypted objects can then be sent over a socket and decrypted into the original object at the other end. We could do this by splitting an object into its bytes in memory, then combining those bytes into large integers, and encrypting each of those large integers, to form an *EncryptedObject*, which is then what we send over the socket to be decrypted by the recipient.

In order to maintain the security of the user's data, all data that is transferred between the application and the server will be encrypted using public and private keys specific to the user. An RSA key pair will be created for a user upon creation of their account, the public key will be stored on the server, available for anyone to access, and the private key will be stored on the user's local device, so that it cannot be easily accessed by an external user. Messages sent to a user will be encrypted with the recipient's public key, and decrypted with their private key upon receipt of the message. This means that when a user sends a new chat invite, they must give the private key to the server to be stored temporarily while the invitation is pending. Once an action has been taken or the invite has expired it should be deleted from the server along with the private key. Furthermore, communications between the server and the user will be encrypted using a similar method, although slightly modified. The Server could send data to the user encrypted with their public key, but the user would not be able to respond with the same key pair, instead each communication will be encrypted using a key pair generated by the server, at the start of the communication. This will require the development of a sort of handshake between the user and the server, before the encrypted data is transmitted.

Messages in chats will be encrypted using a keypair, each chat will have a set keypair that will be created upon the creation of the chat. The public key will be stored on the server, and the private key will be stored on devices which are members of the chat. This will allow for chats with multiple members to be encrypted using this method, without storing duplicate messages in the database, encrypted for every user in the chat.

With recent advancements in Quantum Computing, we are realising that encryption systems such as RSA are inherently vulnerable, as Shor's algorithm allows Quantum Computers to break such systems almost instantaneously. There are a number of Post-Quantum Computing (PQC) encryption

algorithms, but I do not plan to implement them here as they can be very complex. However, future developments to this application may include an implementation of the CRYSTALS-Kyber PQC encryption algorithm, but for now I will stick with RSA.

I will implement this as a package separate from the rest of the application.

This implementation works with integers, specifically positive integers. If we aim to encrypt a Java object, we should first break it down into integers using a byte stream, then combining the bytes into a number of 2048 bit integers, then encrypting each of those 2048 bit integers using this method. There is a problem in that some of these integers might be negative, so we should create some kind of wrapper object which will allow the encryption system to handle these integers with a positive sign, but ensures that it decrypts to a negative integer as required.

SHA-256 Hashing Algorithm

In order to increase the security of storing passwords in the SQL database, I will use a hashing algorithm called *SHA-256* (another, potentially simpler option is *MD5*, although this algorithm has been broken, and *SHA-256* is significantly more secure than *MD5*).

The main advantage of using a hashing algorithm is that the plain text passwords are not stored directly in the database, which improves the security of the application significantly, as hashing algorithms cannot be reversed, so even if someone was able to gain access to the passwords in the database, they would still be unable to access that account, since they cannot obtain the original plain text password.

Java provides a library which allows us to use some standard hashing algorithms, so I will use this rather than attempting to implement it myself since these algorithms can be rather complex.

Overall code structure

The overall code structure can be derived from what each part of the application needs to function properly, in terms of library requirements.

Network requests will be created as JSON objects, there is a Java library which provides classes which allow us to do this effectively, so this library will be included in both the server and the client application, since both will require the use of such *JSONObjects*.

Moving the code I will implement, I will separate different parts of the application into the following packages.

com.nathcat.RSA		
EncryptedObject		
Name	***Type***	***Justification***

flipSign	Boolean attribute	Tells the program whether it not the integer contained in this object should have its sign flipped when decrypted.
object	BigInteger attribute	The encrypted integer.
GetInteger()	Method - returns BigInteger	Returns the integer contained by this object, taking into account flipSign.
GetNaturalNumber()	Method - returns BigInteger	Returns the integer contained by this object in positive form.
GetObject()	Method - returns Object	Get the object described by this integer by deserializing the bytes of the integer.
SerializeObject(Object obj)	Method - returns byte[]	Serialise an object into an array of bytes.
DeserializeObject(byte[] bytes)	Method - returns Object	Deserialize a byte array into an object.
ObjectToNumArray(Object obj)	Method - returns BigInteger[]	Turns an object into an array of 2048 bit integers.
NumArrayToObject(BigInteger [] arr)	Method - returns Object	Turns an array of 2048 bit integers into an object.
KeyPair		
pub	PublicKey attribute	The public key of this key pair.
pri	PrivateKey attribute	The private key of this key pair.
Encrypt(BigInteger message)	Method - returns EncryptedObject	Encrypt a single integer.
Decrypt(BigInteger message)	Method - returns BigInteger	Decrypt a single integer.
EncryptBigObject(Object message)	Method - returns EncryptedObject[]	Encrypt a large object (greater than 2048 bits) into an array of encrypted integers).
DecryptBigObject(EncryptedObject[] message)	Method - returns Object	Decrypt an array of EncryptedObjects into the original object.
PrivateKey		
n	BigInteger attribute	The n parameter of this key.

d	BigInteger attribute	The d parameter of this key.
PublicKey		
n	BigInteger attribute	The n parameter of this key.
e	BigInteger attribute	The e parameter of this key.
RSA		
GenerateRSAKeyPair()	Method - returns KeyPair	Generate an RSA key pair object.
com.nathcat.messagecat_database		
Result (enum)		
SUCCESS		
FAILED		
MySQLHandler		
conn	Connection attribute	Object which represents the connection to the SQL database.
config	JSONObject attribute	JSONObject which contains the information given in the MySQL config file.
StartConnection()	Method	Attempts to create a connection to the MySQL server.
GetMySQLConfig()	Method - returns JSONObject	Get the data from the MySQL config file.
Select	Method - returns ResultSet	Perform a select query using the active connection (or any query that returns a result set).
Update	Method	Perform an update query using the active connection (or any query that does not return a result set).
GetUserByID(int userID)	Method - returns User	Get a user by their ID.
GetUserByUsername(String username)	Method - returns User	Get a user by their username.
GetUserByDisplayName(Strin	Method - returns User[]	Get a list of users whose

g displayName		display name matches the pattern “{displayName}%”.
AddUser(User user)	Method	Add a user to the database.
GetFriendshipByID(int friendshipID)	Method - returns Friendship	Get a friendship record by its ID.
GetFriendshipByUserID(int userID)	Method - returns Friendship[]	Get a list of friendships with a given userID
GetFriendshipByUserIDAndFriendID(int userID, int friendID)	Method - returns Friendship	Get a friendship record by its userID and friendID.
AddFriendship(Friendship friendship)	Method	Add a new friendship record to the database.
GetFriendRequestsByRecipientID(int recipientID)	Method - returns FriendRequest[]	Get a list of friend requests by their recipientID.
GetFriendRequestsBySenderID(int senderID)	Method - returns FriendRequest[]	Get a list of friend requests by their senderID.
DeleteFriendRequest(int requestID)	Method	Delete a friend request from the database.
AddFriendRequest(FriendRequest fr)	Method	Add a friend request to the database.
GetChatByID(int chatID)	Method - returns Chat	Get a chat by its ID.
GetChatByPublicKeyID(int keyID)	Method - returns Chat	Get a chat by its public key ID.
AddChat(Chat chat)	Method	Add a new chat to the database.
GetChatInviteByID(int chatInviteID)	Method - returns ChatInvite	Get a chat invite by its ID.
GetChatInvitesByRecipientID(int recipientID)	Method - returns ChatInvite[]	Get a list of chat invites by their recipientID.
GetChatInvitesBySenderID(int senderID)	Method - returns ChatInvite[]	Get a list of chat invites by their senderID.
DeleteChatInvite(int chatInviteID)	Method	Delete a chat invite from the database.
AddChatInvite(ChatInvite ci)	Method	Add a new chat invite to the database.
MessageQueue		
ChatID	Integer attribute	The ID of the chat this message

		queue is linked to.
data	Queue attribute	The queue structure which stores the messages in this chat.
Push(Message msg)	Method	Pushes a new message to the queue.
Pop()	Method - returns Message	Pops a message from the queue and returns the value that was popped.
Get(int i)	Method - returns Message	Gets an item from the queue at index i.
MessageStore		
data	HashMap<Integer, MessageQueue> attribute	Hash table which stores all the message queues, where the key is the ID of the chat they are linked to.
ReadFromFile()	Method - returns HashMap<Integer, MessageQueue>	Attempts to read the hash table from the data file.
WriteToFile()	Method	Attempts to write the hash table to the data file.
GetMessageQueue(int id)	Method - returns MessageQueue	Gets the MessageQueue linked to the key given as the argument of this method.
AddMessageQueue(MessageQueue queue)	Method - returns Result	Adds a new MessageQueue to the hash table.
RemoveMessageQueue(int id)	Method - returns Result	Removes the MessageQueue with the key given as an argument of this method.
KeyStore		
data	HashMap<Integer, KeyPair> attribute	Hash table which stores all of the key pairs, linked to an ID.
dataFile	File attribute	The file which contains the pre-existing data for this hash table.
ReadFromFile()	Method - returns HashMap<Integer, KeyPair>	Attempts to read the hash table from dataFile.
WriteToFile()	Method	Attempts to write the contents

		of the hash table to dataFile.
GetKeyPair(int keyID)	Method - returns KeyPair	Gets the KeyPair with ID keyID from the hash table.
AddKeyPair(KeyPair pair)	Method - returns Result	Adds a new KeyPair to the hash table.
RemoveKeyPair(int keyID)	Method - returns Result	Removes a KeyPair from the hash table at ID keyID.
ExpirationManager extends Thread		
db	Database attribute	Stores a reference to the active Database object.
maxTimeElapsed	Long attribute	The maximum amount of time that can elapse for any given request.
run()	Method	Overrides *Thread.run*, executes in a separate process from the rest of the application.
Database		
mySQLHandler	MySQLHandler attribute	Reference to the MySQLHandler instance.
keyStore	KeyStore attribute	Reference to the KeyStore instance.
messageStore	MessageStore attribute	Reference to the MessageStore instance.
expirationManager	ExpirationManager attribute	Reference to the ExpirationManager instance.
SaveKeyAndMessageStore()	Method	Manually saves the KeyStore and MessageStore instances to their respective data files.
Wrapper implementations of methods that are defined in MySQLHandler, KeyStore, and MessageStore	Methods - various return types	Methods which call other methods, so this class kind of acts as a sort of switchboard to the other database systems.
com.nathcat.messagecat_database_entities		
User		
userID	Integer attribute	ID of the user.

username	String attribute	The user's username.
password	String attribute	The user's password.
displayName	String attribute	The user's display name.
dateCreated	String attribute	The date this user was created.
profilePicturePath	String attribute	The path of this user's profile picture.
Chat		
ChatID	Integer attribute	The ID of this chat.
Name	String attribute	The name of this chat.
Description	String attribute	The description of this chat.
PublicKeyID	Integer attribute	The ID of the public key used to encrypt this chat.
ChatInvite		
ChatInviteID	Integer attribute	The ID of this chat invite.
ChatID	Integer attribute	The ID this chat invite is linked to.
SenderID	Integer attribute	The ID of the user that sent this invite.
RecipientID	Integer attribute	The ID of the user that should receive this invite.
TimeSent	Long attribute	The time this invite was sent.
PrivateKeyID	Integer attribute	The ID of the private key used to decrypt messages in this chat.
FriendRequest		
FriendRequestID	Integer attribute	The ID of this friend request.
SenderID	Integer attribute	The ID of the user that sent this friend request.
RecipientID	Integer attribute	The ID of the user that is to receive this request.
TimeSent	Long attribute	The time this request was sent.

Friendship		
FriendshipID	Integer attribute	The ID of this friendship.
UserID	Integer attribute	The ID of the user.
FriendID	Integer attribute	The ID of the friend.
DateEstablished	String attribute	The date this friendship was created.
Message		
SenderID	Integer attribute	The ID of the user that sent this message.
ChatID	Integer attribute	The ID of the chat this message was sent to.
TimeSent	Long attribute	The time this message was sent.
Content	Object attribute	The encrypted content of this message.
com.nathcat.messagecat_server		
Queue		
startNode	Node attribute	The start node of the internal linked list.
maxLength	Integer attribute	The maximum length of this queue.
length	Integer attribute	The current length of this queue.
Push(Object data)	Method	Push a new object to the queue.
Pop()	Method - returns Object	Pop an object off the queue.
Get(int i)	Method - returns Object	Get the object at index i from the queue.
Queue.Node		
data	Object attribute	The data stored at this node.
nextNode	Node attribute	The next node in the list.

QueueManager extends Thread		
server	Server attribute	Stores a reference to the currently active server.
queue	Queue attribute	The queue this manager should manage.
pool	Handler[] attribute	The Handler pool assigned to this manager.
run()	Method	Override from *Thread.run*, executes in a different process.
RequestType (enum)		
Authenticate		
GetUser		
GetFriendship		
GetFriendRequests		
GetChat		
GetChatInvite		
GetPublicKey		
GetMessageQueue		
AddUser		
AddChat		
AddListenRule		
RemoveListenRule		
AcceptFriendRequest		
DeclineFriendRequest		
AcceptChatInvite		
DeclineChatInvite		
SendMessage		
SendFriendRequest		
SendChatInvite		

ListenRule		
id	Integer attribute	The ID of this listen rule.
connectionHandlerID	Integer attribute	The ID of the connection handler that created this listen rule.
handler	ConnectionHandler attribute	A reference to the connection handler that created this listen rule.
listenForType	RequestType attribute	The request type this rule listens for.
fieldToMatch	Field attribute	The request field that should be checked.
objectToMatch	Object attribute	The object that should be used to compare against the field. If the field matches this object, the rule is triggered.
objectsToMatch	Object[] attribute	Multiple objects which should be compared against the field. If any of these objects match the field, this rule is triggered.
getID()	Method - returns Integer	Get the ID of this listen rule.
setID(int id)	Method	Set the ID of this listen rule. Only allows the ID to be set once.
CheckRequest(RequestType type, Object data)	Method - returns boolean	Checks if certain data triggers this listen rule.
Handler		
socket	Socket attribute	The main TCP/IP socket for communications with the client.
oos	ObjectOutputStream attribute	The main output stream.
ois	ObjectInputStream attribute	The main input stream.
threadNum	Integer attribute	The number assigned to this thread when it was created, used for debugging.
className	String attribute	The name of this class, again, used for debugging.

keyPair	KeyPair attribute	The KeyPair used to *send* encrypted messages to the client.
clientKeyPair	KeyPair attribute	The KeyPair used to *receive* encrypted messages from the client.
busy	Boolean attribute	Tells the rest of the program if this handler is busy or not.
server	Server attribute	Stores a reference to the currently active server.
queueObject	Object	The object passed to this handler by the QueueManager.
authenticated	Boolean attribute	Tells the rest of the program if the client connected to this handler has authenticated their connection or not.
lrSocket	Socket attribute	The listen rule communication socket.
lrOos	ObjectOutputStream attribute	The listen rule communication stream.
InitialiseIO()	Method	Initialise input / output operations on the main socket.
StopHandler()	Method	Stop the handler process.
DebugLog(String message)	Method	Print a message to the console, giving a unique identifier for this process.
Send(Object obj)	Method	Send an object to the client.
LrSend(Object obj)	Method	Send an object to the client over the listen rule socket.
Receive()	Method - returns Object	Receive an object from the client.
Close()	Method	Close the connection to the client.
ConnectionHandler extends Handler		
request	JSONObject request	The request which is currently being processed.
run()	Method	Overrides from *Thread.run*, this method executes in a

		different process, hence this conducts the main operation of the handler.
DoHandshake()	Method - returns boolean	Attempts to complete the handshake process with the client, key pairs are exchanged and the listen rule socket is created. Returns a boolean describing the success state of this process.
Mainloop()	Method	Main process for this handler, waits for a request to be received and passes this to the handle request method and returns the result to the client.
HandleRequest()	Method - returns Object	Determines the correct method to handle a request and returns the result of that method.
Authenticate()	Method - returns Object	Authenticate the connection with the given user data. Return the full data of the user if successful, or the string “failed”.
GetUser()	Method - returns Object.	Get a user from the database with the given data.
GetFriendship()	Method - returns Object	Get a friendship with the given data.
GetFriendRequests()	Method - returns Object	Get friend requests with the given data.
GetChat()	Method - returns Object	Get a chat with the given data.
GetChatInvite()	Method - returns Object	Get a chat invite with the given data.
GetPublicKey()	Method - returns Object	Get a public key with the given data.
GetMessageQueue()	Method - returns Object	Get a message queue with the given data.
AddUser()	Method - returns Object	Add a new user to the database and return the full data of the new user.
AddChat()	Method - returns Object	Add a new chat to the database and return the full data of the new chat.

AddListenRule()	Method - returns Object	Add a new listen rule and return its ID.
RemoveListenRule()	Method - returns Object	Remove a listen rule given its ID.
AcceptFriendRequest()	Method - returns Object	Accept a friend request. This will remove the friend request from the database and add two new friendship records.
DeclineFriendRequest()	Method - returns Object	Deletes a friend request from the database.
AcceptChatInvite()	Method - returns Object	Deletes a chat invite from the database and returns the private key for that chat.
DeclineChatInvite()	Method - returns Object	Deletes a chat invite from the database.
SendMessage()	Method - returns Object	Add a message to the relevant message queue.
SendFriendRequest()	Method - returns Object	Add a friend request to the database.
SendChatInvite()	Method - returns Object	Add a chat invite to the database and the chat's private key.
Server		
port	Integer attribute	The port the server will receive connections on.
maxThreadCount	Integer attribute	The maximum number of handlers allowed to run.
connectionHandlerPool	Handler[] attribute	The pool of connection handlers.
connectionHandlerQueueManager	QueueManager attribute	The QueueManager for the connection handlers.
db	Database attribute	The active instance of the database.
listenRules	ArrayList<ListenRule> attribute	A list of active listen rules.
main(String[] args)	Method	The main method. Sets up the server and accepts connections.
GetConfigFile()	Method - returns JSONObject	Gets the contents of the server

		config file.
DebugLog(String message)	Method	Prints a message to the console with identifier “Server”.
com.nathcat.messagecat_client		
AutoStartService extends BroadcastReceiver		
onReceive(Context context, Intent intent)	Method	Executes when the application is started, starts the networker service.
LoadingActivity		
networkerServiceConnection	ServiceConnection attribute	Used to bind to the networker service.
networkerService	NetworkerService attribute	Stores a reference to the networker service instance.
bound	Boolean attribute	Tells the rest of the program whether or not this activity is bound to the service.
onCreate(Bundle savedInstanceState)	Method	Called when the activity is created.
isServiceRunning()	Method - returns boolean	Determines if the networker service is currently running or not.
NewUserActivity		
networkerServiceConnection	ServiceConnection attribute	Used to bind to the networker service.
networkerService	NetworkerService attribute	Stores a reference to the networker service instance.
bound	Boolean attribute	Tells the rest of the program whether or not this activity is bound to the service.
displayNameEntry	EditText attribute	Reference to the entry box for the new user’s display name.
phoneNumberEntry	EditText attribute	Reference to the entry box for the new user’s phone number, which will be used as their username.

loadingWheel	ProgressBar attribute	Reference to the loading wheel widget which will be displayed while the page sends the request.
passwordEntry	EditText attribute	Reference to the entry box for the new user's password.
passwordRetypeEntry	EditText attribute	Reference to the entry box for the new user's password retype.
phoneNumber	String attribute	Stores the user's phone number as found by the application.
digest	MessageDigest attribute	Used for password hashing, stores a reference to the SHA-256 algorithm.
onCreate(Bundle savedInstanceState)	Method	Called when the activity is first created.
OnSubmitButtonClicked(View v)	Method	Called when the submit button is clicked, checks that the data entered is valid and then sends a request to the server to create a new user.
NetworkerService extends Service		
notificationChannel	NotificationChannel attribute	Stores a reference to the notification channel object used to give notifications about events within the application.
serviceStatusChannel	NotificationChannel attribute	Stores a reference to the notification channel for the service status notification.
connectionHandler	ConnectionHandler attribute	Stores a reference to the active connection handler.
connectionHandlerLooper	Looper attribute	The handler looper to use with the connection handler process.
authenticated	Boolean attribute	Tells the rest of the program if the connection is authenticated or not.
waitingForResponse	Boolean attribute	Tells the rest of the program if the service is currently waiting for a response from the server.
binder	NetworkerServiceBinder	Used to allow the client

	attribute	application to bind to the service.
bound	Boolean attribute	Identifies whether or not the service is currently bound to the main application.
user	User attribute	The user that is currently authenticated on this connection.
activeChatID	Integer attribute	The ID of the chat that is currently open in the application, used to filter out notifications from messages sent in this chat.
onBind(Intent intent)	Method - returns IBinder	Called when a process binds to this service.
onUnbind(Intent intent)	Method	Called when a process unbinds from this service.
onCreate()	Method	Called when the service is first created.
onStartCommand(Intent intent, int flags, int startID)	Method - returns integer	Called when the service is commanded to start. The return value determines the behaviour of the service when it is closed. We will return the value of the constant *START_STICKY*, which means that the service should restart if it is closed.
SendRequest(Request request)	Method	Passes a request object to the connection handler to be sent to the server.
onDestroy()	Method	Called when the service is destroyed, should stop the connection handler.
startConnectionHandler()	Method	Start the connection handler and initialise the connection to the server and attempt to authenticate.
ConnectionHandler extends Handler		
s	Socket attribute	The TCP/IP socket used to communicate with the server.
oos	ObjectOutputStream attribute	The output stream for the main

		socket.
ois	ObjectInputStream attribute	The input stream of the main socket.
keyPair	KeyPair attribute	The key pair generated by this client to communicate with the server.
serverKeyPair	KeyPair attribute	The key pair generated by the server.
context	Context attribute	The application context.
connectionHandlerID	Integer attribute	The ID of the connection handler that has taken this connection.
callbackHandler	ListenRuleCallbackHandler attribute	The listen rule callback handler thread which handles all callbacks from listen rules added registered by this client.
listenRules	ArrayList<ListenRuleRecord> attribute	Stores listen rules that have been created by this client.
Send(Object obj)	Method	Send an object to the server.
Receive()	Method - returns Object	Receive an object from the server.
handleMessage(Message msg)	Method	Handle a message sent to this handler by the Android Handler IPC protocol. A value message of 0 tells the handler to initialise a new connection, 1 tells the handler to send a request to the server, and 2 tells the handler to close the current connection.
ListenRuleCallbackHandler extends Thread		
connectionHandler	ConnectionHandler attribute	The ConnectionHandler instance which created this process.
s	Socket attribute	The TCP/IP socket used to communicate with the server.
oos	ObjectOutputStream attribute	The object output stream
ois	ObjectInputStream attribute	The object input stream
keyPair	KeyPair attribute	The client’s key pair

serverKeyPair	KeyPair attribute	The server’s key pair
port	Int attribute	The port assigned by the server.
Send(Object obj)	Method	Sends an object through the connected socket.
Receive()	Method - returns Object	Receives an object through the connected socket.
run()	Method	Overrides from parent class *Thread*, executes in a different thread. Handles the main process of the listen rule callback handler.
MainActivity		
connection	ServiceConnection attribute	Used to bind to the networker service.
networkerService	NetworkerService attribute	Stores a reference to the networker service instance when bound.
searchResults	User[] attribute	Stores the results of a search on the find users page.
friends	User[] attribute	Array of friends, displayed on the friends page.
invitationsFragment	InvitationsFragment attribute	Stores a reference to the active invitations fragment.
invitationFragments	InvitationFragment[] attribute	An array of invitations represented as fragments within the invitations fragment page.
users	HashMap<Integer, User> attribute	Effectively a cache of users so the app doesn’t have to request a user by their ID for every message when displaying messages on the messaging fragment page.
messagingFragment	MessagingFragment attribute	Stores a reference to the currently active messaging fragment.
chatFragments	ArrayList<ChatFragment>	A list of chats displayed as fragments on the chats page.

onCreate(Bundle savedInstanceState)	Method	Called when this activity is created.
onSearchButtonClicked(View v)	Method	Called when the user searches for a display name on the find users page. Sends a request to the server and then displays the result.
onAddFriendButtonClicked(View v)	Method	Called when the user sends another user a friend request from the find users page.
onInviteToChatClicked(View v)	Method	Called when the user wants to send a friend an invite to a chat.
onChatClicked(View v)	Method	Called when a user clicks on a chat on the chats page.
onAcceptInviteClicked(View v)	Method	Called when a user accepts an invite.
onDeclineInviteClicked(View v)	Method	Called when a user declines an invite.
SendMessage(View v)	Method	Called when a user sends a message on the messaging page.
Other…		
All other classes are either small storage classes that have no functional value other than encapsulation, or are very simple / similar to other classes so I have not included them here for the sake of saving space on this document. Such classes are listed below. ● ListenRuleRecord ○ Acts as a storage class for information data about registered listen rules. ● Request ○ Used as a storage class to encapsulate data required for a network request. ● ListenRuleRequest ○ Extends Request ○ Same purpose as the Request class but specifically for network requests involving listen rules. ● All fragment classes.		

Testing

RSA

Testing the RSA Asymmetric encryption system is imperative, since it will be used when transferring any data over an active connection. It will be required to encrypt Java objects, so we should attempt to supply each of the database entity classes to it with data supplied to them, and attempt to encrypt an RSA key pair, while this may seem odd it will be required by the application and is likely the largest data object that will have to be passed through this system, so it is very important that we test that the system can handle this kind of encryption.

Data	***Expected outcome from sequential encrypt then decrypt***
User class	An identical class
Friendship class	An identical class
Friend request class	An identical class
Chat invite class	An identical class
Chat class	An identical class
Message class	An identical class
KeyPair class	An identical class

Database

The database will be tested by supplying data to the database through implemented methods, and then trying to retrieve that data, also through implemented methods. There will be a lot of these methods, since there are a significant number of database functions.

The *MySQLHandler* class will be tested using the following data. For methods that add data to the database we will verify that the data has been correctly added by performing a select statement directly on the SQL database.

AddUser()

Data	**Expected outcome from select statement**
Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png

Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated: 25/07/2022 ProfilePicturePath: default.png	Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated: 25/07/2022 ProfilePicturePath: default.png

AddFriendship()

Data	**Expected outcome from select statement**
UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 2 FriendID: 1 DateEstablished: 25/07/2022	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022

AddFriendRequest()

Data	**Expected outcome from select statement**
FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659277170486	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent:
FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent: 1659277170486	FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent:

AddChat()

Data	**Expected outcome from select statement**
ChatID: 1 Name: “test1” Description: “test1-description” PublicKeyID: 1	ChatID: 1 Name: “test1” Description: “test1-description” PublicKeyID: 1

ChatID: 2 Name: “test2” Description: “test2-description” PublicKeyID: 2	ChatID: 2 Name: “test2” Description: “test2-description” PublicKeyID: 2

AddChatInvite()

Data	Expected outcome from select statement
ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1
ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2	ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2

GetUser()

Data	Expected outcome
UserID: 1	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png
Username: MyPhoneNumber	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png
DisplayName: Nathcat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png

UserID: 3	null
Username: aiwdwid	null
DisplayName: Nat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png

GetFriendship()

Data	**Expected outcome**
FriendshipID: 1	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 1	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 1 FriendID: 2	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
FriendshipID: 3	null
UserID: 3	“”
UserID: 2 FriendID: 1	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022

GetFriendRequests()

Data	**Expected outcome**
SenderID: 1	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305

RecipientID: 1	FriendRequestID: 2 SenderID: 2 RecipientID: 1 TimeSent: 1659278362305
SenderID: 3	“”
RecipientID: 3	“”

GetChat()

Data	Expected outcome
ChatID: 1	ChatID: 1 Name: test1 Description: test1-description PublicKeyID: 1
ChatID: 2	ChatID: 2 Name: test2 Description: test2-description PublicKeyID: 2
ChatID: 3	null

GetChatInvite()

Data	Expected outcome
ChatInviteID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1
ChatInviteID: 3	null
RecipientID: 2	ChatInviteID: 1 ChatID: 1

	SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2
RecipientID: 3	null
SenderID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2
SenderID: 3	null

Server

The server will be tested using a test program which supplies fabricated data, to ensure that this data is processed as expected. Most of the server's functions simply employ database functionality, which will have been tested before we reach this stage, so we simply need to test that the server takes the requests and directs them to the correct processing method.

Request Type	***Test outline***
Authenticate	We will supply different sets of user data, one which is correct and should be authenticated, and others which are incorrect (either do not exist in the database or include invalid data for a certain field), and observe the action the program takes based on this input.
GetUser	We will again supply different sets of user data, with individual fields filled out as per the selector options for this request type, those selector options being: ● ID ● Username ● DisplayName

	Also, in order to protect the privacy of users, the password field will be made blank before the results are passed back to the client application, so we should make sure that this is done correctly.
GetFriendship	Supply different sets of friendship data, according to the selector options for this request: ● ID ● UserID ● UserID and FriendID
GetFriendRequests	Supply different sets of friend request data, according to the selectors for this request type: ● ID ● SenderID ● RecipientID
GetChat	Supply different sets of chat data, according to the selectors for this request type: ● ID ● PublicKeyID
GetChatInvites	Supply different sets of chat invite data, according to the selectors for this request type: ● ID ● RecipientID
GetPublicKey	Supply different public key IDs to the server and check what it returns, if the ID is linked to a public key which exists in the key store it should return that key, but if it does not exist it should return *null*.
GetMessageQueue	Supply different chat IDs to the server and check what it returns, if the ID is linked to a queue which exists in the message store it should return that queue, but if it does not exist it should return *null*.
AddUser	Create some user data to add to the server and then check the SQL database to see if the data has been entered correctly.
AddChat	Create some chat data to add to the server and then check the SQL database to see if the data has been entered correctly.
AddListenRule	Register a listen rule with the server and then perform the action which triggers this listen rule

	to check that it is triggered and handled correctly, and also perform actions that should not trigger it to see how that is handled by the server, these should be requests of the same type that do not have a field match, and requests of different types.
RemoveListenRule	Remove the registered listen rule and perform the action which should trigger it and watch how the server handles this.
AcceptFriendRequest	After adding a friend request to the server, attempt to accept it, and check the SQL database afterwards to see if the request acceptance is handled correctly. The server should create 2 new friendship records.
DeclineFriendRequest	The server should simply delete the friend request and make no other changes to the database.
AcceptChatInvite	After adding a chat invite to the server, attempt to accept it, and check that the server responds with the private key sent with the chat invite.
DeclineChatInvite	The server should delete the invite record in the database and also delete the private key from the key store.
SendMessage	Send a message to the server and check the relevant message queue for the message. Also try a variety of characters to see how it handles this variety.
SendFriendRequest	Send a friend request and check that it is added to the database correctly.
SendChatInvite	Send a chat invite and check that it is added to the SQL database correctly, and that the private key is added to the key store under the hash code of the key.

Client application

Each part / page of the client application will have its functionality tested once it has been implemented, although there are some pages which rely on other pages being functional before they can have their full functionality implemented, for example the *ChatsFragment* page, which is the page the *MainActivity* starts on cannot implement the change to the *ChatFragment* page before we have implemented *ChatFragment*, which will likely be the last thing we implement. In this specific case we can write a function which writes a message to the console when it would open a chat, perhaps outputting the information of the chat that is to be opened.

Page	***Test***	***Expected outcome***
LoadingActivity	Do nothing, allow the application to set itself up. Perform when there is a user account on the device and when there is not a user account on the device.	When there is a user account, this activity should start the networker service and load the *MainActivity*. When there is not a user account, the activity should start the networker service and load the *NewUserActivity*.
NewUserActivity	Enter data into the fields and click submit. This data should contain a variety of characters, some which are included in only ASCII text and some which are only included in Unicode text.	The application should submit the new user request, create a new user on the server, then reload the *LoadingActivity*, which attempts to authenticate the new user and then load the *MainActivity*. The application should not be phased by the variance of characters.
MainActivity	Attempt to open the drawer menu and navigate to each of the sub pages through this menu by clicking on the relevant buttons.	The application should change the page to the page which was clicked, for example if the find users button was clicked, it should open the *FindUsersFragment*.
ChatsFragment	Click on the chats in the window.	The application should open the *ChatFragment* for this chat. Or, it should output a message to the console containing information about the chat that was clicked, if the *ChatFragment* has not been implemented by this point.
FriendsFragment	Click on the invite to chat button on a friend displayed in the window.	The application should load the *InviteToChatActivity*.
InvitationsFragment	Click on both the accept invite button and decline invite button for different invites.	The application should perform the relevant action, if the accept button is clicked it should request that the server accept the invite, and if the decline button is clicked it should request that the server declines this invite.
FindUserFragment	Attempt to search for a user which exists in the database, using variations in the length	The application should not show any results for users which do not exist, and should

	of the name. Also use a variety of characters. Also search for users which do not exist.	show results for users which do exist, regardless of how complete their name is.
InviteToChatActivity	Test both inviting a user to an existing chat and creating a new chat which the user is then invited to. Use a variety of characters in the fields for information about the new chat.	The application should send a chat invite to the user that was clicked, it should also create a new chat if this is required.
MessagingFragment	Try sending messages to chats using a variety of characters. Also try receiving messages.	The application should not be phased by the variety of characters and the entered characters should appear as they were entered on the other client devices. The page should not be refreshed unless a new message is received. The correct user display name should also be shown on each message.
Notifications	Try performing the actions which should cause a notification to be shown on other devices	The application should show the required notifications for the given context and action, which will be detailed more specifically in the actual tests later on.

End-user testing

This stage of testing will test the entire application's function as a single unit, rather than testing the individual components, black box testing, as it is more formally named. In order to perform such testing I will gather a number of people (including the original stakeholder) and install the application on their devices and ask that they use it for a period of time, and provide feedback on it through a questionnaire which specifically targets the success criteria and requirements of the application, which will allow us to evaluate the extent to which they were met in the final product.

Following is the questionnaire which I will give to testers.

1. What was your experience of the actual messaging system? I.e. did you find it easy to use, was it simple to look at, was there too much information displayed, not enough information displayed?
2. What was your experience of the contact management system? (the ability to see your friend's list, send friend and chat invites, and manage invitations you have received from others).
3. How did you find the performance of the application, was it fast, slow, did it crash a lot?
4. How easy was the application to use, were there any areas where you felt that usability could be improved?

5. How secure do you feel your data is when using this application? Do you think the application should be more transparent about how it stores your data and why it requires that data?
6. Do you feel that there were any missing features that you expected to be in an application like this?
7. Any other comments?

This questionnaire covers the success criteria and requirements laid out earlier in the analysis stage, and also requests any other information or suggestions on how the application could be improved. There is also a question specifically targeting the usability of the application, asking testers how the application's usability could be improved or what is already good about it.

This will allow us to assess how well the product at the end of the development cycle meets the success criteria and requirements, and how we may improve the application in future development / maintenance.

Privacy policy

Given that this application collects sensitive user data, I should create a privacy policy to ensure transparency about the way the data is stored. This privacy policy could be hosted on the same device as the server, although operating on a different web server program. Creating another website in this way could also offer other benefits since we could create a further website to promote the application and potentially provide information on how to use the application, although this may be a future development goal.

In any case, here is an example of a privacy policy statement I could use.

MessageCat, and its developer "Nathcat", collects the following sensitive information about its data subjects / users:

- *Phone number*
    - *Used as a unique identifier to facilitate authentication of users when connecting to the application.*
    - *May also be used in other future MessageCat application variations to simplify the process of authentication in these applications, but these do not exist and hence this point is not applicable at this time.*
    - *This information is stored on the MessageCat server and is not made available to access through the application, and is not sold to third parties. Client connections must also be authenticated with their own information in order to request data about other users.*
- *Password*
    - *This is also used to authenticate users. It acts as a phrase only the user that created it knows to protect against other people accessing their account.*
    - *This information is stored on the MessageCat server and is not shared with any third parties, and is stored using a SHA-256 hash of the password they originally entered.*

Other information which is gathered by the application but may not be specifically regarded as sensitive is the "display name", the name displayed to other users of the application. This information is available to view through the application.

Please contact the email address below with any queries should you have any, and also to request that your data be deleted. Under the data protection act you have the right to request that we delete your data and we will comply with this request should that be your intention.

Contact information will also be displayed at the end of the document to allow users to send queries to the development team to get a better understanding of how their data is being handled. Users will also be able to request that their data be deleted through this system. In the future I should provide a way for users to do this through the application but for now this will suffice.

Development timeline

Given the proposed functionality of different parts of this application there is a clear order in which we should implement this project. We should implement the application piece by piece, ensuring that the system we are implementing does not require other features to function properly. Given this rule set, we could have the following development timeline.

1. Database entities
    a. *User*
    b. *Friendship*
    c. *FriendRequest*
    d. *ChatInvite*
    e. *Chat*
    f. *Message*
2. RSA CryptoSystem
    a. *PublicKey & Private Key*
    b. *EncryptedObject*
    c. *KeyPair*
    d. *RSA* (includes key pair generation methods)
3. Database
    a. *MySQLHandler*
    b. *KeyStore*
    c. *Queue*
    d. *MessageQueue*
    e. *MessageStore*
    f. *Database*
    g. *ExpirationManager*
4. Server
    a. *ListenRule*
    b. *Handler*
    c. *ConnectionHandler* (server)
    d. *QueueManager*
    e. *Server*
    f. *Server main method*
5. Client application
    a. *ConnectionHandler* (client)
    b. *NetworkerService*
    c. *AutoStartService*

d. *LoadingActivity*
e. *NewUserActivity*
f. *MainActivity* (developed progressively with sub-pages)
g. *ChatsFragment*
h. *FriendsFragment*
i. *FindUserFragment*
j. *InvitationsFragment*
k. *InviteToChatActivity*
l. *MessagingFragment*

Implementation

Database entities

These are classes which represent records from the database, so that we can access this data through native Java means, rather than an object from a library. This also provides a layer of abstraction to the final code which will make it easier to use and more maintainable.

This section does not require testing since it does not offer any functionality per say, only implements a few data classes.

This part of the application will be contained within a package called *com.nathcat.messagecat_database_entities.*

User

```
package com.nathcat.messagecat_database_entities;

import java.io.Serializable;

/**
* Represents a User from the database
*/
public class User implements Serializable {
   public final int UserID;
   public final String Username;
   public final String Password;
   public final String DisplayName;
   public final String DateCreated;
   public final String ProfilePicturePath;

   public User(int userID, String username, String password, String displayName, String dateCreated, String
profilePicturePath) {
       UserID = userID;
       Username = username;
       Password = password;
       DisplayName = displayName;
       DateCreated = dateCreated;
       ProfilePicturePath = profilePicturePath;
   }
```

```
    @Override
    public String toString() {
        return "User{" +
                "UserID=" + UserID +
                ", Username='" + Username + '\'' +
                ", Password='" + Password + '\'' +
                ", DisplayName='" + DisplayName + '\'' +
                ", DateCreated='" + DateCreated + '\'' +
                ", ProfilePicturePath='" + ProfilePicturePath + '\'' +
                '}';
    }
}
```

Friendship

```
package com.nathcat.messagecat_database_entities;

import java.io.Serializable;

/**
* Represents a friendship from the database.
*/
public class Friendship implements Serializable {
    public final int FriendshipID;
    public final int UserID;
    public final int FriendID;
    public final String DateEstablished;

    public Friendship(int friendshipID, int userID, int friendID, String dateEstablished) {
        FriendshipID = friendshipID;
        UserID = userID;
        FriendID = friendID;
        DateEstablished = dateEstablished;
    }

    @Override
    public String toString() {
        return "Friendship{" +
                "FriendshipID=" + FriendshipID +
                ", UserID=" + UserID +
                ", FriendID=" + FriendID +
                ", DateEstablished='" + DateEstablished + '\'' +
                '}';
    }
}
```

FriendRequest

```
package com.nathcat.messagecat_database_entities;

import java.io.Serializable;

/**
* Represents a friend request from the database.
```

```java
*/
public class FriendRequest implements Serializable {
   public final int FriendRequestID;
   public final int SenderID;
   public final int RecipientID;
   public final long TimeSent;

   public FriendRequest(int friendRequestID, int senderID, int recipientID, long timeSent) {
       FriendRequestID = friendRequestID;
       SenderID = senderID;
       RecipientID = recipientID;
       TimeSent = timeSent;
   }

   @Override
   public String toString() {
       return "FriendRequest{" +
               "FriendRequestID=" + FriendRequestID +
               ", SenderID=" + SenderID +
               ", RecipientID=" + RecipientID +
               ", TimeSent=" + TimeSent +
               '}';
   }
}
```

ChatInvite

```java
package com.nathcat.messagecat_database_entities;

import java.io.Serializable;

/**
* Represents a chat invite from the database.
*/
public class ChatInvite implements Serializable {
   public final int ChatInviteID;
   public final int ChatID;
   public final int SenderID;
   public final int RecipientID;
   public final long TimeSent;
   public final int PrivateKeyID;

   public ChatInvite(int chatInviteID, int chatID, int senderID, int recipientID, long timeSent, int privateKeyID) {
       ChatInviteID = chatInviteID;
       ChatID = chatID;
       SenderID = senderID;
       RecipientID = recipientID;
       TimeSent = timeSent;
       PrivateKeyID = privateKeyID;
   }

   @Override
   public String toString() {
       return "ChatInvite{" +
               "ChatInviteID=" + ChatInviteID +
```

```
               ", ChatID=" + ChatID +
               ", SenderID=" + SenderID +
               ", RecipientID=" + RecipientID +
               ", TimeSent=" + TimeSent +
               ", PrivateKeyID=" + PrivateKeyID +
               '}';
   }
}
```

Chat

```
package com.nathcat.messagecat_database_entities;

import java.io.Serializable;

/**
* Represents a chat from the database.
*/
public class Chat implements Serializable {
   public final int ChatID;
   public final String Name;
   public final String Description;
   public final int PublicKeyID;

   public Chat(int chatID, String name, String description, int publicKeyID) {
       ChatID = chatID;
       Name = name;
       Description = description;
       PublicKeyID = publicKeyID;
   }

   @Override
   public String toString() {
       return "Chat{" +
               "ChatID=" + ChatID +
               ", Name='" + Name + '\'' +
               ", Description='" + Description + '\'' +
               ", PublicKeyID=" + PublicKeyID +
               '}';
   }
}
```

Message

```
package com.nathcat.messagecat_database_entities;

import org.json.simple.JSONObject;

import java.io.Serializable;

/**
* Represents a message from the database.
*/
public class Message implements Serializable {
```

```
    public final int SenderID;
    public final int ChatID;
    public final long TimeSent;
    public final Object Content;

    public Message(int senderID, int chatID, long timeSent, Object content) {
        SenderID = senderID;
        ChatID = chatID;
        TimeSent = timeSent;
        Content = content;
    }

    @Override
    public String toString() {
        return "Message{" +
                "SenderID=" + SenderID +
                ", ChatID=" + ChatID +
                ", TimeSent=" + TimeSent +
                ", Content='" + Content + '\'' +
                '}';
    }

    public JSONObject GetJSONObject() {
        JSONObject json = new JSONObject();
        json.put("SenderID", this.SenderID);
        json.put("ChatID", this.ChatID);
        json.put("TimeSent", this.TimeSent);
        json.put("Content", this.Content);

        return json;
    }
}
```

RSA Asymmetric Encryption system

This part of the application will be contained within a package called *com.nathcat.RSA*, an implementation of RSA asymmetric encryption as detailed in my design.

PublicKey

```
package com.nathcat.RSA;

import java.io.Serializable;
import java.math.BigInteger;

/**
* Contains data for an RSA encryption public key
*
* @author Nathan "Nathcat" Baines
*/

public class PublicKey implements Serializable {
    public final BigInteger n;
    public final BigInteger e;
```

```java
    public PublicKey(BigInteger n, BigInteger e) {
        this.n = n;
        this.e = e;
    }
}
```

PrivateKey

```java
package com.nathcat.RSA;

import java.io.Serializable;
import java.math.BigInteger;

/**
* Contains data for an RSA encryption private key
*
* @author Nathan "Nathcat" Baines
*/

public class PrivateKey implements Serializable {
    public final BigInteger n;
    public final BigInteger d;

    public PrivateKey(BigInteger n, BigInteger d) {
        this.n = n;
        this.d = d;
    }
}
```

EncryptedObject

```java
package com.nathcat.RSA;

import java.io.*;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;

public class EncryptedObject implements Serializable {
    public final boolean flipSign;   // Determine whether the sign should be flipped
    public BigInteger object;        // The object as a big integer

    public EncryptedObject(boolean flipSign, BigInteger object) {
        this.flipSign = flipSign;
        this.object = object;
    }

    public EncryptedObject(Object object) {
        BigInteger o = new BigInteger(SerializeObject(object));

        if (o.compareTo(new BigInteger("0")) < 0) {
            this.flipSign = o.compareTo(new BigInteger("0")) < 0;
```

```
            this.object = o.abs();
        }
        else {
            this.flipSign = false;
            this.object = o;
        }
    }

    public BigInteger GetInteger() {
        if (this.flipSign) {
            return this.object.multiply(new BigInteger("-1"));
        }
        else {
            return this.object;
        }
    }

    public BigInteger GetNaturalNumber() {
        return this.object;
    }

    public Object GetObject() {
        BigInteger o = this.GetInteger();
        return DeserializeObject(o.toByteArray());
    }

    public static byte[] SerializeObject(Object obj) {
        try {
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            oos.writeObject(obj);
            oos.flush();
            byte[] objBytes = baos.toByteArray();
            oos.close();
            baos.close();

            return objBytes;

        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    public static Object DeserializeObject(byte[] bytes) {
        try {
            ByteArrayInputStream baos = new ByteArrayInputStream(bytes);
            ObjectInputStream ois = new ObjectInputStream(baos);
            Object obj = ois.readObject();
            ois.close();
            baos.close();

            return obj;

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return null;
```

```
        }
    }

    public static BigInteger[] ObjectToNumArray(Object obj) {
        byte[] byteArray = EncryptedObject.SerializeObject(obj);
        assert byteArray != null;

        byte[] currentNum = new byte[256];
        ArrayList<BigInteger> numArrayList = new ArrayList<>();

        for (int i = 0; i < byteArray.length; i++) {
            currentNum[i % 256] = byteArray[i];

            if ((i + 1) % 256 == 0) {
                numArrayList.add(new BigInteger(currentNum));
                currentNum = new byte[256];
            }
        }

        if (byteArray.length % 256 != 0) {
            numArrayList.add(new BigInteger(currentNum));
        }

        Object[] arr = numArrayList.toArray();
        BigInteger[] result = new BigInteger[arr.length];
        for (int i = 0; i < arr.length; i++) {
            result[i] = (BigInteger) arr[i];
        }

        return result;
    }

    public static Object NumArrayToObject(BigInteger[] arr) {
        byte[] byteArray = new byte[arr.length * 256];
        int byteCounter = 0;
        for (BigInteger num : arr) {
            byte[] currentNum = num.toByteArray();
            for (int i = 0; i < currentNum.length; i++) {
                byteArray[byteCounter] = currentNum[i];
                byteCounter++;
            }
        }

        return EncryptedObject.DeserializeObject(byteArray);
    }
}
```

KeyPair

```
package com.nathcat.RSA;

import java.io.Serializable;
import java.math.BigInteger;
import java.util.Arrays;

/**
```

```
* Contains a pair of RSA encryption keys
*
* @author Nathan "Nathcat" Baines
*/

public class KeyPair implements Serializable {
    public PublicKey pub = null;
    public PrivateKey pri = null;

    public KeyPair(PublicKey pub, PrivateKey pri) {
        this.pub = pub;
        this.pri = pri;
    }

    /**
     * Give a string representation of this KeyPair
     * @return string representation of this KeyPair
     */
    public String toString() {
        String result = "";
        if (this.pub != null) {
            result += "Public key----------\nn = " + this.pub.n.toString() + "\ne = " + this.pub.e.toString();
        }

        if (this.pub != null && this.pri != null) {
            result += "\n\n";
        }

        if (this.pri != null) {
            result += "Private Key-----\nn = " + this.pri.n.toString() + "\nd = " + this.pri.d.toString();
        }

        return result;
    }

    /**
     * Encrypt a BigInteger array
     * @param message The BigInteger array to encrypt
     * @return Encrypted BigInteger array
     * @deprecated Use object encryption methods instead
     */
    public BigInteger[] encrypt(BigInteger[] message) throws PublicKeyException {
        if (this.pub == null) {
            throw new PublicKeyException();
        }

        BigInteger[] result = new BigInteger[message.length];

        for (int i = 0; i < message.length; i++) {
            result[i] = (message[i].modPow(this.pub.e, this.pub.n));
        }

        return result;
    }

    /**
     * Decrypt a BigInteger array
```

```java
     * @param message The BigInteger array to decrypt
     * @return The decrypted BigInteger array
     * @deprecated Use object encryption methods instead
     */
    public BigInteger[] decrypt(BigInteger[] message) throws PrivateKeyException {
        if (this.pri == null) {
            throw new PrivateKeyException();
        }

        BigInteger[] result = new BigInteger[message.length];

        for (int i = 0; i < message.length; i++) {
            result[i] = (message[i].modPow(this.pri.d, this.pri.n));
        }

        return result;
    }

    /**
     * Encrypt an object
     * @param message The object to encrypt
     * @return The encrypted object
     * @throws PublicKeyException Thrown if this pair has no public key
     */
    public EncryptedObject encrypt(Object message) throws PublicKeyException {
        if (this.pub == null) {
            throw new PublicKeyException();
        }

        BigInteger cipherNum = new EncryptedObject(message)
                .GetNaturalNumber()
                .modPow(this.pub.e, this.pub.n);

        return new EncryptedObject(new EncryptedObject(message).flipSign, cipherNum);
    }

    /**
     * Decrypt an object
     * @param message The object to decrypt
     * @return The decrypted object
     * @throws PrivateKeyException Thrown if this pair has no private key
     */
    public Object decrypt(EncryptedObject message) throws PrivateKeyException {
        if (this.pri == null) {
            throw new PrivateKeyException();
        }

        BigInteger plain = message.GetNaturalNumber().modPow(this.pri.d, this.pri.n);
        message.object = plain;

        return message.GetObject();
    }

    /**
     * Encrypt a large object.
     * @param message The object to encrypt
     * @return Array of encrypted objects
```

```java
     * @throws PublicKeyException Thrown if the public key on this object is null
     */
    public EncryptedObject[] encryptBigObject(Object message) throws PublicKeyException {
        if (this.pub == null) {
            throw new PublicKeyException();
        }

        ObjectContainer container = new ObjectContainer(message);
        BigInteger[] numArray = EncryptedObject.ObjectToNumArray(container);

        EncryptedObject[] result = new EncryptedObject[numArray.length];
        for (int i = 0; i < numArray.length; i++) {
            result[i] = this.encrypt(numArray[i]);
        }

        return result;
    }

    /**
     * Decrypt a large object
     * @param message The array of encrypted objects you wish to decrypt
     * @return The resulting object
     * @throws PrivateKeyException Thrown if the private key on this object is null
     */
    public Object decryptBigObject(EncryptedObject[] message) throws PrivateKeyException {
        if (this.pri == null) {
            throw new PrivateKeyException();
        }

        BigInteger[] numArray = new BigInteger[message.length];
        for (int i = 0; i < message.length; i++) {
            numArray[i] = (BigInteger) this.decrypt(message[i]);
        }

        return ((ObjectContainer) EncryptedObject.NumArrayToObject(numArray)).obj;
    }
}
```

RSA

```java
package com.nathcat.RSA;

import java.math.BigInteger;
import java.security.NoSuchAlgorithmException;
import java.security.SecureRandom;

/**
* Methods for generating and using RSA asymmetric encryption keys.
*
* @author Nathan "Nathcat" Baines
*/

public class RSA {
    /**
     * Generate an RSA public key/private key pair.
     * @return An RSA key pair
```

```
     * @throws NoSuchAlgorithmException Thrown by crypto-secure generation of random integers
     */
    public static KeyPair GenerateRSAKeyPair() throws NoSuchAlgorithmException {
        // Generate two random 2048-bit prime numbers
        BigInteger p = new BigInteger(2048, 1, new SecureRandom());
        BigInteger q = new BigInteger(2048, 1, new SecureRandom());

        // n = p * q
        BigInteger n = p.multiply(q);
        // Standard value for e (public key exponent) is 65537
        BigInteger e = new BigInteger("65537");
        // ed = 1 (mod (p - 1) * (q - 1)), d is the private key exponent
        BigInteger d = e.modInverse(p.subtract(BigInteger.ONE).multiply(q.subtract(BigInteger.ONE)));

        return new KeyPair(
                new PublicKey(n, e),
                new PrivateKey(n, d)
        );
    }
}
```

Testing

Data	***Expected outcome from sequential encrypt then decrypt***	***Actual outcome***
User class	An identical class	*StreamCorruptedException*
Friendship class	An identical class	*StreamCorruptedException*
Friend request class	An identical class	*StreamCorruptedException*
Chat invite class	An identical class	*StreamCorruptedException*
Chat class	An identical class	*StreamCorruptedException*
Message class	An identical class	*StreamCorruptedException*
KeyPair class	An identical class	*StreamCorruptedException*

All of the tests I conducted resulted in a *StreamCorruptedException*, which is thrown whilst the program attempts to decrypt an object. All of the bytes of the object are decrypted as normal, following the mathematics, and the resulting array of bytes is *serialised* back into a regular Java object using a byte stream wrapped in an object stream. This problem indicates that the object stream failed to interpret the byte stream as an object, which means that the bytes in the array do not form a Java object.

I first decided to test whether or not I could deserialise, and then serialise an object normally, without putting it through the encryption process. This was successful and I obtained an identical object with no errors, so clearly it is possible to handle objects in this way, which suggests that the decrypted byte stream is not the same as the object's original byte stream.

To verify this I outputted the byte array produced by serialisation, and then the byte array produced after decryption, into the console to examine any differences. Here is a sample output which shows the problem.

```
... 34, 62, 124, -78, 23, 0, 0, 0, 12, -43, 102 ...
... 34, 62, 124, -78, 23, 0, -43, 102, 123, 86 ...
```

As you can see, 2 bytes with a magnitude of 0 have been lost from the array, this is clearly the cause of the stream corruption. The question now is why is this happening.

This took a while to work out, but I eventually noticed a pattern. Not all 0 bytes were being removed, only the ones that occurred at each 64 byte boundary, by which I mean that if, 64 bytes into the stream, there was a 0, that byte and any 0 bytes immediately following it were lost. In order to perform the mathematical operations, the program creates an array of *BigInteger* objects, which are part of the Java standard library, each of these objects is assigned 64 bytes, except for the last one which may be assigned less if the number of bytes in the stream is not a multiple of 64. It cannot be a coincidence that these zeros are being lost at an interval equivalent to the boundary of each *BigInteger*. This leads me to believe that when the program passes a set of 64 bytes to the *BigInteger* class constructor, it removes all the bytes that form leading zeros, which makes sense, but as a result these zeros are lost in the final decrypted stream, leading to a corrupted stream.

So how do we fix this?

We could create another class called *ByteChunk*, which acts as a wrapper for the *BigInteger*, and manages both the sign and lost bytes in the stream. This way the mathematics will not be broken by negative integers, and no bytes will be lost from the final array.

I will also take this opportunity to improve the *EncryptedObject* class, which is not the most efficient or maintainable in its current state. Now, rather than encrypting an object to an array of *EncryptedObject*s, you will be able to encrypt to / decrypt from a single *EncryptedObject*, since it will act as a wrapper for an array of *ByteChunk* objects, and provide utility methods to serialise and deserialise objects into arrays of *ByteChunk*s.

ByteChunk

```java
package com.nathcat.RSA;

import java.math.BigInteger;
import java.util.Arrays;

/**
* Represents a chunk of 64 bytes
*/
public class ByteChunk {
   private final byte[] lostBytes;  // The lost bytes in the conversion from byte array to integer
   public BigInteger integer;       // The big integer created from the original bytes array
   private final boolean flipSign;  // Will the integer sign need to be changed?
```

```java
    public ByteChunk(byte[] bytes) {
        this.integer = new BigInteger(bytes);
        this.lostBytes = new byte[bytes.length];

        if (bytes.length - this.integer.toByteArray().length >= 0)
            System.arraycopy(bytes, 0, lostBytes, 0, bytes.length - this.integer.toByteArray().length);

        if (this.integer.compareTo(BigInteger.ZERO) < 0) {
            this.flipSign = true;
            this.integer = this.integer.abs();
        } else {
            this.flipSign = false;
        }
    }

    /**
     * Create a from this chunk, taking into account the number of leading zeros in the original byte array
     *
     * @return The byte array
     */
    public byte[] GetByteArray() {
        // Flip the sign of the integer if required
        if (this.flipSign) this.integer = this.integer.multiply(BigInteger.valueOf(-1));

        return EncryptedObject.CombineByteArrays(this.lostBytes, this.integer.toByteArray());
    }

    public String toString() {
        return Arrays.toString(this.GetByteArray());
    }
}
```

EncryptedObject

```java
package com.nathcat.RSA;

import java.io.*;
import java.math.BigInteger;
import java.util.ArrayList;
import java.util.Arrays;

public class EncryptedObject implements Serializable {
    public final ByteChunk[] byteChunks; // The byte chunks of this encrypted object

    public EncryptedObject(ByteChunk[] byteChunks) {
        this.byteChunks = byteChunks;
    }

    /**
     * Turn an object into an array of bytes
     * @param obj The object to serialize
     * @return A corresponding array of bytes, or null if a byte array could not be created
     */
    public static byte[] SerializeObject(Object obj) {
        // My process here is to pass the object through two input streams which handle different types of data.
        // At the lowest level of course they both handle binary data, hence they are compatible and can pass
        // data to each other.
        try {
```

```
            // Create a byte array output stream, this will allow us to get a byte array output from whatever we
input into the stream.
            ByteArrayOutputStream baos = new ByteArrayOutputStream();
            // Now we create a object output stream as a wrapper over the byte array stream.
            // This means that when we pass an object into the object output stream, it will be passed straight into
the byte array stream.
            ObjectOutputStream oos = new ObjectOutputStream(baos);
            // Writing the object to the object stream and flushing the changes
            oos.writeObject(obj);
            oos.flush();
            // We can now extract the byte array from the byte stream
            byte[] objBytes = baos.toByteArray();
            // ... and close the two streams
            oos.close();
            baos.close();

            return objBytes;

        } catch (IOException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * Turn an array of bytes into an object.
     * @param bytes The array of bytes to deserialize.
     * @return The resulting object, or null if no object could be created.
     */
    public static Object DeserializeObject(byte[] bytes) {
        // The process here is effectively the same as in the serialize object method, except in reverse.
        try {
            // Create the byte array INPUT stream from the byte array, this provides the stream with somewhere to get
data from.
            ByteArrayInputStream baos = new ByteArrayInputStream(bytes);
            // Create an object input stream as a wrapper of the byte array stream, this allows us to extract objects
from the byte array stream.
            ObjectInputStream ois = new ObjectInputStream(baos);
            // Attempt to read an object from the stream.
            Object obj = ois.readObject();
            // Close the streams
            ois.close();
            baos.close();

            return obj;

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return null;
        }
    }


    /**
     * Split an object into chunks of bytes, each chunk contains 64 bytes.
     * @param obj The object to split.
     * @return An array of ByteChunk objects.
     */
    public static ByteChunk[] SplitObjectToByteChunks(Object obj) {
        // Serialize the object into an array of bytes
        byte[] bytes = EncryptedObject.SerializeObject(obj);
```

```
        // Create two lists, one for sets of 64 bytes, and one for the current set of 64 bytes
        ArrayList<byte[]> byte64Chunks = new ArrayList<>();
        ArrayList<Byte> currentBytes = new ArrayList<>();

        for (byte aByte : bytes) {
            // Add a byte to the current bytes list
            currentBytes.add(aByte);

            // If the current bytes list has 64 bytes in it, create a byte array from this and add it to the set of
64 bytes,
            // then clear the current bytes list.
            if (currentBytes.size() == 64) {
                byte[] chunk = new byte[64];
                for (int b = 0; b < 64; b++) {
                    chunk[b] = currentBytes.get(b);
                }

                byte64Chunks.add(chunk);
                currentBytes.clear();
            }
        }

        // If the number of bytes was not a multiple of 64, there will be some bytes left over in the current bytes
        // array after this loop finishes, so create a byte array from the remaining bytes and add then to the set of
64 bytes.
        if ((bytes.length % 64) != 0) {
            byte[] chunk = new byte[currentBytes.size()];
            for (int b = 0; b < currentBytes.size(); b++) {
                chunk[b] = currentBytes.get(b);
            }

            byte64Chunks.add(chunk);
        }

        // Turn the set of 64 bytes into an array of ByteChunk objects
        ByteChunk[] result = new ByteChunk[byte64Chunks.size()];
        for (int i = 0; i < byte64Chunks.size(); i++) {
            result[i] = new ByteChunk(byte64Chunks.get(i));
        }

        return result;
    }

    /**
     * Turn an array of byte chunks in an object.
     * @param chunks The byte chunks to compile.
     * @return The resulting object, or null if one could not be created from the data.
     */
    public static Object CompileByteChunks(ByteChunk[] chunks) {
        // Get the byte arrays from each of the byte chunks
        ArrayList<byte[]> chunkArrays = new ArrayList<>();
        int byteCount = 0;
        for (ByteChunk chunk : chunks) {
            byte[] arr = chunk.GetByteArray();
            chunkArrays.add(arr);
            byteCount += arr.length;
        }

        // Create just a straight array of bytes by copying the chunk arrays into a single, new, empty array
        byte[] bytes = new byte[byteCount];
```

```java
        int bytesPointer = 0;
        for (byte[] chunkArray : chunkArrays) {
            System.arraycopy(chunkArray, 0, bytes, bytesPointer, chunkArray.length);
            bytesPointer += chunkArray.length;
        }

        // Deserialize of bytes array and return the result
        return EncryptedObject.DeserializeObject(bytes);
    }

    /**
     * Combines byte arrays a and b. Array b is overlaid onto a from the end of a.
     * For example, let a = [1, 2, 3, 4, 5], b = [6, 7, 8]
     * The result will be [1, 2, 6, 7, 8]
     * @param a Array a
     * @param b Array b
     * @return The combined array
     */
    public static byte[] CombineByteArrays(byte[] a, byte[] b) {
        byte[] result = new byte[a.length];

        int lengthDifference = a.length - b.length;
        if (lengthDifference < 0) {
            throw new IllegalArgumentException("Array a should be larger than or the same size as array b!");
        }

        int cursor = 0;
        while (cursor < lengthDifference) {
            result[cursor] = a[cursor];
            cursor++;
        }

        while ((cursor - lengthDifference) < b.length) {
            result[cursor] = b[cursor - lengthDifference];
            cursor++;
        }

        return result;
    }
}
```

KeyPair

```java
package com.nathcat.RSA;

import java.io.Serializable;
import java.math.BigInteger;
import java.util.Arrays;

/**
* Contains a pair of RSA encryption keys
*
* @author Nathan "Nathcat" Baines
*/
```

```java
public class KeyPair implements Serializable {
    public PublicKey pub = null;
    public PrivateKey pri = null;

    public KeyPair(PublicKey pub, PrivateKey pri) {
        this.pub = pub;
        this.pri = pri;
    }

    /**
     * Give a string representation of this KeyPair
     * @return string representation of this KeyPair
     */
    public String toString() {
        String result = "";
        if (this.pub != null) {
            result += "Public key----------\nn = " + this.pub.n.toString() + "\ne = " + this.pub.e.toString();
        }

        if (this.pub != null && this.pri != null) {
            result += "\n\n";
        }

        if (this.pri != null) {
            result += "Private Key-----\nn = " + this.pri.n.toString() + "\nd = " + this.pri.d.toString();
        }

        return result;
    }

    /**
     * Encrypt an object
     * @param obj The object to encrypt
     * @return The encypted object
     * @throws PublicKeyException Thrown if the public key is null
     */
    public EncryptedObject encrypt(Object obj) throws PublicKeyException {
        if (this.pub == null) {
            throw new PublicKeyException();
        }

        ByteChunk[] chunks = EncryptedObject.SplitObjectToByteChunks(obj);
        for (int i = 0; i < chunks.length; i++) {
            chunks[i].integer = chunks[i].integer.modPow(this.pub.e, this.pub.n);
        }

        return new EncryptedObject(chunks);
    }

    /**
     * Decrypt an object
     * @param obj The object to decrypt
     * @return The original object
     * @throws PrivateKeyException Thrown if the private key is null
     */
    public Object decrypt(EncryptedObject obj) throws PrivateKeyException {
        if (this.pri == null) {
```

```
            throw new PrivateKeyException();
        }

        ByteChunk[] chunks = obj.byteChunks;
        for (int i = 0; i < chunks.length; i++) {
            chunks[i].integer = chunks[i].integer.modPow(this.pri.d, this.pri.n);
        }

        return EncryptedObject.CompileByteChunks(chunks);
    }
}
```

Secondary testing

Data	*Expected outcome from sequential encrypt then decrypt*	*Actual outcome*
User class	An identical class	An identical class
Friendship class	An identical class	An identical class
Friend request class	An identical class	An identical class
Chat invite class	An identical class	An identical class
Chat class	An identical class	An identical class
Message class	An identical class	An identical class
KeyPair class	An identical class	An identical class

Testing after implementing the fix displayed the expected behaviour, so this issue has been resolved and we can move onto the next stage of development.

Database

This part of the application will be contained within a package called *com.nathcat.messagecat_database.* This contains a number of classes which provide layers of abstraction to the process of accessing the database behind the application. This makes the program more maintainable and simpler to develop.

MySQLHandler

```
package com.nathcat.messagecat_database;

import com.mysql.cj.jdbc.exceptions.CommunicationsException;
import com.nathcat.messagecat_database_entities.*;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileNotFoundException;
```

```java
import java.sql.*;
import java.util.Scanner;

/**
* This class handles calls to the MySQL database made through the Database class.
*
* @author Nathan "Nathcat" Baines
*/
public class MySQLHandler {
   private Connection conn;          // The connection to the MySQL database.
   private final JSONObject config;  // Config JSONObject

   /**
    * Default constructor, creates a connection to the database.
    */
   public MySQLHandler() throws FileNotFoundException, ParseException, SQLException {
       // Instead of catching those exceptions, we should leave them to be caught elsewhere.
       // If any of them are thrown, this object shouldn't be allowed to be created.

       // Get the MySQL config file
       this.config = this.GetMySQLConfig();

       StartConnection();
   }

   private void StartConnection() throws SQLException {
       // Create a connection to the MySQL database.
       conn = DriverManager.getConnection((String) config.get("connection_url"), (String) config.get("username"),
(String) config.get("password"));
   }

   /**
    * Get the MySQL config JSON file.
    * @return A JSON Object containing MySQL config data.
    * @throws FileNotFoundException Thrown if the config file cannot be found.
    * @throws ParseException Thrown if the data in the config file contains a syntax error.
    */
   private JSONObject GetMySQLConfig() throws FileNotFoundException, ParseException {
       Scanner reader = new Scanner(new File("Assets/MySQL_Config.json"));
       StringBuilder sb = new StringBuilder();

       while (reader.hasNextLine()) {
           sb.append(reader.nextLine());
       }

       return (JSONObject) new JSONParser().parse(sb.toString());
   }

   /**
    * Perform a Select query on the database.
    * @param query The query to be executed
    * @return The ResultSet returned from the query
    * @throws SQLException Thrown by SQL errors.
    */
   protected ResultSet Select(String query) throws SQLException {
       // Create and execute the statement
       Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
```

```
        stmt.execute(query);
        // Get the result set and close the statement
        ResultSet rs = stmt.getResultSet();

        // Return the result set
        return rs;
    }

    /**
     * Perform an update query on the database (or any query that does not have a result set)
     * @param query The query to be executed
     * @throws SQLException Thrown by SQL errors
     */
    protected void Update(String query) throws SQLException {
        // Create and execute the statement
        Statement stmt = conn.createStatement();
        stmt.execute(query);

        // Close the statement
        stmt.close();
    }

    /**
     * Get a User by their ID.
     * @param UserID The UserID to search for.
     * @return The User that was found, or null if none were found.
     * @throws SQLException Thrown by SQL errors.
     */
    public User GetUserByID(int UserID) throws SQLException {
        // Get the result set from the query
        ResultSet rs = this.Select("SELECT * FROM `Users` WHERE `UserID` like " + UserID);

        // Check if there are no results
        rs.last();
        if (rs.getRow() == 0) {
            return null;
        }

        // There must be 1 result, since we are searching for a primary key.
        // Get the data from this 1 result, close the result set, and return the user.
        rs.first();
        User result = new User(
                rs.getInt("UserID"),
                rs.getString("Username"),
                rs.getString("Password"),
                rs.getString("DisplayName"),
                rs.getString("DateCreated"),
                rs.getString("ProfilePicturePath")
        );

        rs.close();
        return result;
    }

    /**
     * Get a user by their username.
     * @param Username The username to search for
```

```
 * @return The User that is found, or null, if none are found.
 * @throws SQLException Thrown by SQL errors.
 */
public User GetUserByUsername(String Username) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `Users` WHERE `Username` like '" + Username + "'");

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return null;
    }

    // There must be 1 result, since we are searching for a unique value.
    // Get the data from this 1 result, close the result set, and return the user.
    rs.first();
    User result = new User(
            rs.getInt("UserID"),
            rs.getString("Username"),
            rs.getString("Password"),
            rs.getString("DisplayName"),
            rs.getString("DateCreated"),
            rs.getString("ProfilePicturePath")
    );

    rs.close();
    return result;
}

/**
 * Get a list of users by their display name.
 * @param DisplayName The display name to search for
 * @return A list of users whose display names start with DisplayName
 * @throws SQLException Thrown by SQL errors.
 */
public User[] GetUserByDisplayName(String DisplayName) throws SQLException {
    // Get the result set from the query, include a wildcard character in the query so that we get the users whose
    // display names start with DisplayName.
    ResultSet rs = this.Select("SELECT * FROM `Users` WHERE `DisplayName` like '" + DisplayName + "%'");

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return new User[0];
    }

    // There must be at least one result, so create an array of Users.
    User[] results = new User[rs.getRow()];

    rs.beforeFirst();

    while (rs.next()) {
        results[rs.getRow() - 1] = new User(
                rs.getInt("UserID"),
                rs.getString("Username"),
                rs.getString("Password"),
                rs.getString("DisplayName"),
```

```
                rs.getString("DateCreated"),
                rs.getString("ProfilePicturePath")
        );
    }

    rs.close();
    return results;
}

/**
 * Get a friendship record from the database
 * @param FriendshipID The ID of the record
 * @return The record found, or null if none are found
 * @throws SQLException Thrown by SQL errors
 */
public Friendship GetFriendshipByID(int FriendshipID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `Friendships` WHERE `FriendshipID` like " + FriendshipID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return null;
    }

    // There must be 1 result, since we are searching for a unique value.
    // Get the data from this 1 result, close the result set, and return the friendship.
    rs.first();
    Friendship result = new Friendship(
            rs.getInt("FriendshipID"),
            rs.getInt("UserID"),
            rs.getInt("FriendID"),
            rs.getString("DateEstablished")
    );

    rs.close();
    return result;
}

/**
 * Get a friendship record by the UserID
 * @param UserID The UserID to search for
 * @return The records found, or an empty array if none are found
 * @throws SQLException Thrown by SQL errors
 */
public Friendship[] GetFriendshipByUserID(int UserID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `Friendships` WHERE `UserID` like " + UserID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return new Friendship[0];
    }

    // There must be at least one result, so create an array of friendships.
    Friendship[] results = new Friendship[rs.getRow()];
```

```java
        rs.beforeFirst();

        while (rs.next()) {
            results[rs.getRow() - 1] = new Friendship(
                    rs.getInt("FriendshipID"),
                    rs.getInt("UserID"),
                    rs.getInt("FriendID"),
                    rs.getString("DateEstablished")
            );
        }

        rs.close();
        return results;
    }

    /**
     * Get a friendship record by its UserID and FriendID
     * @param UserID The UserID
     * @param FriendID The FriendID
     * @return The Friendship record that is found, or null if none are found
     * @throws SQLException Thrown by SQL errors
     */
    public Friendship GetFriendshipByUserIDAndFriendID(int UserID, int FriendID) throws SQLException {
        // Get the result set from the query
        ResultSet rs = this.Select("SELECT * FROM `Friendships` WHERE `UserID` like " + UserID + " AND `FriendID` like
" + FriendID);

        // Check if there are no results
        rs.last();
        if (rs.getRow() == 0) {
            return null;
        }

        // There must be 1 result, since we are searching for a unique value.
        // Get the data from this 1 result, close the result set, and return the friendship.
        rs.first();
        Friendship result = new Friendship(
                rs.getInt("FriendshipID"),
                rs.getInt("UserID"),
                rs.getInt("FriendID"),
                rs.getString("DateEstablished")
        );

        rs.close();
        return result;
    }

    /**
     * Get a User's friend requests (requests where they are the recipient)
     * @param RecipientID The UserID of the recipient
     * @return An array of friend requests
     * @throws SQLException Thrown by SQL errors
     */
    public FriendRequest[] GetFriendRequestsByRecipientID(int RecipientID) throws SQLException {
        // Get the result set from the query
        ResultSet rs = this.Select("SELECT * FROM `FriendRequests` WHERE `RecipientID` like " + RecipientID);
```

```java
        // Check if there are no results
        rs.last();
        if (rs.getRow() == 0) {
            return new FriendRequest[0];
        }

        // There must be at least one result, so create an array of requests.
        FriendRequest[] results = new FriendRequest[rs.getRow()];

        rs.beforeFirst();

        while (rs.next()) {
            results[rs.getRow() - 1] = new FriendRequest(
                    rs.getInt("FriendRequestID"),
                    rs.getInt("SenderID"),
                    rs.getInt("RecipientID"),
                    rs.getLong("TimeSent")
            );
        }

        rs.close();
        return results;
    }

    /**
     * Get a User's friend requests (requests they have sent)
     * @param SenderID The UserID of the sender
     * @return An array of friend requests
     * @throws SQLException Thrown by SQL errors
     */
    public FriendRequest[] GetFriendRequestsBySenderID(int SenderID) throws SQLException {
        // Get the result set from the query
        ResultSet rs = this.Select("SELECT * FROM `FriendRequests` WHERE `SenderID` like " + SenderID);

        // Check if there are no results
        rs.last();
        if (rs.getRow() == 0) {
            return new FriendRequest[0];
        }

        // There must be at least one result, so create an array of requests.
        FriendRequest[] results = new FriendRequest[rs.getRow()];

        rs.beforeFirst();

        while (rs.next()) {
            results[rs.getRow() - 1] = new FriendRequest(
                    rs.getInt("FriendRequestID"),
                    rs.getInt("SenderID"),
                    rs.getInt("RecipientID"),
                    rs.getLong("TimeSent")
            );
        }

        rs.close();
        return results;
```

```
}

/**
 * Delete a friend request
 * @param FriendRequestID The ID of the friend request to delete
 * @throws SQLException Thrown by SQL errors
 */
public void DeleteFriendRequest(int FriendRequestID) throws SQLException {
    this.Update("DELETE FROM `FriendRequests` WHERE `FriendRequestID` like " + FriendRequestID);
}

/**
 * Get a chat record from the database
 * @param ChatID The ID of the record
 * @return The record found, or null if none are found
 * @throws SQLException Thrown by SQL errors
 */
public Chat GetChatByID(int ChatID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `Chats` WHERE `ChatID` like " + ChatID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return null;
    }

    // There must be 1 result, since we are searching for a unique value.
    // Get the data from this 1 result, close the result set, and return the chat.
    rs.first();
    Chat result = new Chat(
            rs.getInt("ChatID"),
            rs.getString("Name"),
            rs.getString("Description"),
            rs.getInt("PublicKeyID")
    );

    rs.close();
    return result;
}

/**
 * Get a chat record from the database by the public key id associated with it
 * @param PublicKeyID The public key id to search for
 * @return The Chat that is found
 * @throws SQLException Thrown by SQL errors
 */
public Chat GetChatByPublicKeyID(int PublicKeyID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `Chats` WHERE `PublicKeyID` like " + PublicKeyID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return null;
    }
```

```java
    // There must be 1 result, since we are searching for a unique value.
    // Get the data from this 1 result, close the result set, and return the chat.
    rs.first();
    Chat result = new Chat(
            rs.getInt("ChatID"),
            rs.getString("Name"),
            rs.getString("Description"),
            rs.getInt("PublicKeyID")
    );

    rs.close();
    return result;
}

/**
 * Get a chat invite record from the database
 * @param ChatInviteID The ID of the record
 * @return The record found, or null if none are found
 * @throws SQLException Thrown by SQL errors
 */
public ChatInvite GetChatInviteByID(int ChatInviteID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `ChatInvitations` WHERE `ChatInviteID` like " + ChatInviteID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return null;
    }

    // There must be 1 result, since we are searching for a unique value.
    // Get the data from this 1 result, close the result set, and return the invite
    rs.first();
    ChatInvite result = new ChatInvite(
            rs.getInt("ChatInviteID"),
            rs.getInt("ChatID"),
            rs.getInt("SenderID"),
            rs.getInt("RecipientID"),
            rs.getLong("TimeSent"),
            rs.getInt("PrivateKeyID")
    );

    rs.close();
    return result;
}

/**
 * Get a user's invitations to chats
 * @param RecipientID The UserID of the recipient
 * @return An array of chat invites
 * @throws SQLException Thrown by SQL errors
 */
public ChatInvite[] GetChatInvitesByRecipientID(int RecipientID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `ChatInvitations` WHERE `RecipientID` like " + RecipientID);

    // Check if there are no results
```

```
    rs.last();
    if (rs.getRow() == 0) {
        return new ChatInvite[0];
    }

    // There must be at least one result, so create an array of invites.
    ChatInvite[] results = new ChatInvite[rs.getRow()];

    rs.beforeFirst();

    while (rs.next()) {
        results[rs.getRow() - 1] = new ChatInvite(
                rs.getInt("ChatInviteID"),
                rs.getInt("ChatID"),
                rs.getInt("SenderID"),
                rs.getInt("RecipientID"),
                rs.getLong("TimeSent"),
                rs.getInt("PrivateKeyID")
        );
    }

    rs.close();
    return results;
}

/**
 * Get a user's invitations to chats
 * @param SenderID The UserID of the sender
 * @return An array of chat invites
 * @throws SQLException Thrown by SQL errors
 */
public ChatInvite[] GetChatInvitesBySenderID(int SenderID) throws SQLException {
    // Get the result set from the query
    ResultSet rs = this.Select("SELECT * FROM `ChatInvitations` WHERE `SenderID` like " + SenderID);

    // Check if there are no results
    rs.last();
    if (rs.getRow() == 0) {
        return new ChatInvite[0];
    }

    // There must be at least one result, so create an array of invites.
    ChatInvite[] results = new ChatInvite[rs.getRow()];

    rs.beforeFirst();

    while (rs.next()) {
        results[rs.getRow() - 1] = new ChatInvite(
                rs.getInt("ChatInviteID"),
                rs.getInt("ChatID"),
                rs.getInt("SenderID"),
                rs.getInt("RecipientID"),
                rs.getLong("TimeSent"),
                rs.getInt("PrivateKeyID")
        );
    }
```

```
        rs.close();
        return results;
    }

    /**
     * Delete a chat invite from the database
     * @param ChatInviteID The ID of the chat invite
     * @throws SQLException Thrown by SQL errors
     */
    public void DeleteChatInvite(int ChatInviteID) throws SQLException {
        this.Update("DELETE FROM `ChatInvitations` WHERE `ChatInviteID` like " + ChatInviteID);
    }

    /**
     * Adds a user to the database
     * @param user The user to add
     * @throws SQLException Thrown by SQL errors
     */
    public void AddUser(User user) throws SQLException {
        this.Update("insert into `Users` (`Username`, `Password`, `DisplayName`, `DateCreated`, `ProfilePicturePath`)
values (" +
                "\"" + user.Username + "\", " +
                "\"" + user.Password + "\", " +
                "\"" + user.DisplayName + "\", " +
                "\"" + user.DateCreated + "\", " +
                "\"" + user.ProfilePicturePath + "\");"
        );
    }

    /**
     * Adds a friendship to the database
     * @param friendship The friendship to add
     * @throws SQLException Thrown by SQL errors
     */
    public void AddFriendship(Friendship friendship) throws SQLException {
        this.Update("insert into `Friendships` (`UserID`, `FriendID`, `DateEstablished`) values (" +
                friendship.UserID + ", " +
                friendship.FriendID + ", " +
                "\"" + friendship.DateEstablished + "\");"
        );
    }

    /**
     * Adds a friend request to the database
     * @param friendRequest The friend request to add
     * @throws SQLException Thrown by SQL errors
     */
    public void AddFriendRequest(FriendRequest friendRequest) throws SQLException {
        this.Update("insert into `FriendRequests` (`SenderID`, `RecipientID`, `TimeSent`) values (" +
                friendRequest.SenderID + ", " +
                friendRequest.RecipientID + ", " +
                friendRequest.TimeSent + ");"
        );
    }

    /**
     * Adds a chat to the database
```

```
     * @param chat The chat to add
     * @throws SQLException Thrown by SQL errors
     */
    public void AddChat(Chat chat) throws SQLException {
        this.Update("insert into `Chats` (`Name`, `Description`, `PublicKeyID`) values (" +
                "\"" + chat.Name + "\", " +
                "\"" + chat.Description + "\", " +
                chat.PublicKeyID + ");"
        );
    }

    /**
     * Adds a chat invite to the database
     * @param chatInvite The chat invite to add
     * @throws SQLException Thrown by SQL errors
     */
    public void AddChatInvite(ChatInvite chatInvite) throws SQLException {
        this.Update("insert into `ChatInvitations` (`ChatID`, `SenderID`, `RecipientID`, `TimeSent`, `PrivateKeyID`)
values (" +
                chatInvite.ChatID + ", " +
                chatInvite.SenderID + ", " +
                chatInvite.RecipientID + ", " +
                chatInvite.TimeSent + ", " +
                chatInvite.PrivateKeyID + ");"
        );
    }
}
```

KeyStore

```
package com.nathcat.messagecat_database;

import com.nathcat.RSA.*;

import java.io.*;
import java.util.HashMap;

/**
* Stores Public and Private encryption keys
*
* @author Nathan "Nathcat" Baines
*/
public class KeyStore {
    private HashMap<Integer, KeyPair> data;
    private File dataFile;

    /**
     * Default constructor
     */
    public KeyStore() throws IOException {
        dataFile = new File("Assets/Data/KeyStore.bin");

        try {
            // Try to read the data file
            data = this.ReadFromFile();
```

```java
    } catch (FileNotFoundException e) {  // Thrown if the file does not exist
        // Create a new empty hash map and create a new file for it
        data = new HashMap<Integer, KeyPair>();
        this.WriteToFile();

    } catch (IOException | ClassNotFoundException e) {  // Potentially thrown by I/O operations
        e.printStackTrace();
    }

    assert this.data != null;
}

/**
 * Default constructor
 */
public KeyStore(File file) throws IOException {
    this.dataFile = file;

    try {
        // Try to read the data file
        data = this.ReadFromFile();

    } catch (FileNotFoundException e) {  // Thrown if the file does not exist
        // Create a new empty hash map and create a new file for it
        data = new HashMap<Integer, KeyPair>();
        this.WriteToFile();

    } catch (IOException | ClassNotFoundException e) {  // Potentially thrown by I/O operations
        e.printStackTrace();
    }

    assert this.data != null;
}

/**
 * Read data from data file
 * @return The HashMap found in the data file
 * @throws IOException Can be thrown by I/O operations
 * @throws ClassNotFoundException Thrown if the Serialized class cannot be found
 */
public HashMap<Integer, KeyPair> ReadFromFile() throws IOException, ClassNotFoundException {
    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(this.dataFile));
    return (HashMap<Integer, KeyPair>) ois.readObject();
}

/**
 * Write data to data file
 * @throws IOException Can be thrown by I/O operations
 */
public void WriteToFile() throws IOException {
    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(this.dataFile));
    oos.writeObject(data);
    oos.flush();
    oos.close();
}
```

```java
    /**
     * Get a key pair object given its identifier
     * @param keyID The identifier to search for
     * @return The KeyPair object found, note that this KeyPair con contain one or both key, depending on the use case
     */
    public KeyPair GetKeyPair(int keyID) {
        return this.data.get(keyID);
    }

    /**
     * Add a new key pair, using the hash code of the KeyPair object as the id
     * @param pair The KeyPair to add
     * @return The result code
     */
    public Result AddKeyPair(KeyPair pair) {
        // Create a copy of the original data
        Object oldData = this.data.clone();

        // Make the changes to the hash map
        this.data.put(pair.hashCode(), pair);

        // Try to write the changes to the data file, or revert to the original state
        try {
            this.WriteToFile();
            return Result.SUCCESS;

        } catch (IOException e) {
            this.data = (HashMap<Integer, KeyPair>) oldData;
            return Result.FAILED;
        }
    }

    /**
     * Remove a key pair
     * @param keyID The ID of the KeyPair object
     * @return The result code
     */
    public Result RemoveKeyPair(int keyID) {
        // Create a copy of the original data
        Object oldData = this.data.clone();

        // Make the changes to the hash map
        this.data.remove(keyID);

        // Try to write the changes to the data file, or revert to the original state
        try {
            this.WriteToFile();
            return Result.SUCCESS;

        } catch (IOException e) {
            this.data = (HashMap<Integer, KeyPair>) oldData;
            return Result.FAILED;
        }
    }
}
```

Queue

```java
package com.nathcat.messagecat_server;

import java.io.Serializable;
import java.util.Arrays;

/**
* Queue data structure implemented using an internal linked list
*
* @author Nathan "Nathcat" Baines
*/
public class Queue implements Cloneable, Serializable {
   private Node startNode = null;  // The start node of the linked list
   private int maxLength = -1;     // The maximum length of the queue
   private int length = 0;         // The current length of the queue

   /**
    * Represents a node of the linked list
    */
   private static class Node implements Cloneable, Serializable {
       public final Object data;  // The data contained by the node
       public Node nextNode;      // The next node in the linked list

       public Node(Object data, Node nextNode) {
           this.data = data;
           this.nextNode = nextNode;
       }

       /**
        * Create an identical copy of this object
        * @return The clone of this object
        */
       @Override
       public Object clone() {
           try {
               return super.clone();

           } catch (CloneNotSupportedException e) {
               e.printStackTrace();
           }

           return null;
       }
   }

   public Queue() {}

   public Queue(int maxLength) {
       this.maxLength = maxLength;
   }

   /**
    * Push a new object to the end of the queue
    * @param data The object to push
```

```
 */
public void Push(Object data) {
    if (length >= maxLength && maxLength != -1) {
        this.Pop();
    }

    if (this.startNode == null) {
        this.startNode = new Node(data, null);
        length++;
        return;
    }

    Node currentNode = this.startNode;
    while (currentNode.nextNode != null) {
        currentNode = currentNode.nextNode;
    }

    currentNode.nextNode = new Node(data, null);
    this.length++;
}

/**
 * Remove the object from the front of the queue
 * @return The object that was at the front of the queue
 */
public Object Pop() {
    if (this.startNode == null) {
        return null;
    }

    Object data = this.startNode.data;
    this.startNode = this.startNode.nextNode;
    this.length--;
    return data;
}

/**
 * Get an object from a given index
 * @param index The index of the object to get
 * @return The object at the given index
 */
public Object Get(int index) {
    Node currentNode = this.startNode;
    if (currentNode == null) {
        return null;
    }

    for (int i = 0; i < index; i++) {
        currentNode = currentNode.nextNode;
        if (currentNode == null) {
            return null;
        }
    }

    return currentNode.data;
}
```

```java
    @Override
    public String toString() {
        if (this.startNode == null) {
            return "";
        }

        StringBuilder sb = new StringBuilder();
        Node currentNode = this.startNode;
        sb.append(this.startNode.data);
        while (currentNode.nextNode != null) {
            currentNode = currentNode.nextNode;
            sb.append(currentNode.data).append(" ");
        }

        return sb.toString();
    }

    /**
     * Create an identical copy of this object
     * @return The clone of this object
     */
    @Override
    public Object clone() {
        try {
            return super.clone();

        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }

        return null;
    }
}
```

MessageQueue

```java
package com.nathcat.messagecat_database;

import com.nathcat.messagecat_database_entities.Message;
import com.nathcat.messagecat_server.Queue;

import java.io.Serializable;

/**
* Queue data structure for storing messages.
*
* @author Nathan "Nathcat" Baines
*/
public class MessageQueue implements Serializable {
    public final int ChatID;        // The ID of the chat this queue is linked to
    private Queue data;             // The Queue which will be used to store data

    /**
     * Default constructor
     * @param ChatID The ID of the chat this queue is to be linked to
```

```
     */
    public MessageQueue(int ChatID) {
        this.ChatID = ChatID;
        data = new Queue(10);
    }

    /**
     * Push a new message to the back of the queue
     * @param message The new message
     */
    public void Push(Message message) {
        this.data.Push(message);
    }

    /**
     * Remove the item at the front of the queue
     */
    public void Pop() {
        this.data.Pop();
    }

    /**
     * Get the message at index i from the data array
     * @param i The index to get
     * @return The message object at that index
     */
    public Message Get(int i) {
        return (Message) this.data.Get(i);
    }

    /**
     * Return an array of JSON strings for all messages in the queue
     * @return An array of JSON strings for all the messages in the queue
     */
    public String[] GetJSONString() {
        String[] result = new String[50];
        for (int i = 0; i < 50; i++) {
            if (this.data.Get(i) == null) {
                continue;
            }

            result[i] = ((Message) this.data.Get(i)).GetJSONObject().toJSONString();
        }

        return result;
    }

    public Object Clone() throws CloneNotSupportedException {
        return this.clone();
    }
}
```

MessageStore

```
package com.nathcat.messagecat_database;
```

```java
import java.io.*;
import java.util.HashMap;

/**
* This class will handle messages.
*
* @author Nathan "Nathcat" Baines
*/
public class MessageStore {
   private HashMap<Integer, MessageQueue> data = null;  // Keys are the chat ids, and the values are the message
queues

   /**
    * Default constructor
    */
   public MessageStore() throws IOException {
       try {
           // Try to read the data file
           data = this.ReadFromFile();

       } catch (FileNotFoundException e) {  // Thrown if the file does not exist
           // Create a new empty hash map and create a new file for it
           data = new HashMap<Integer, MessageQueue>();
           this.WriteToFile();

       } catch (IOException | ClassNotFoundException e) {  // Potentially thrown by I/O operations
           e.printStackTrace();
       }

       assert this.data != null;
   }

   /**
    * Read data from data file
    * @return The HashMap found in the data file
    * @throws IOException Can be thrown by I/O operations
    * @throws ClassNotFoundException Thrown if the Serialized class cannot be found
    */
   public HashMap<Integer, MessageQueue> ReadFromFile() throws IOException, ClassNotFoundException {
       ObjectInputStream ois = new ObjectInputStream(new FileInputStream("Assets/Data/MessageStore.bin"));
       return (HashMap<Integer, MessageQueue>) ois.readObject();
   }

   /**
    * Write data to data file
    * @throws IOException Can be thrown by I/O operations
    */
   public void WriteToFile() throws IOException {
       ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream("Assets/Data/MessageStore.bin"));
       oos.writeObject(data);
       oos.flush();
       oos.close();
   }

   /**
    * Get a message queue
```

```java
     * @param ChatID The ChatID of the chat for which you are retrieve the message queue
     * @return The MessageQueue
     */
    public MessageQueue GetMessageQueue(int ChatID) {
        return this.data.get(ChatID);
    }

    /**
     * Add a message queue
     * @param queue The message queue to add
     * @return Result code
     */
    public Result AddMessageQueue(MessageQueue queue) {
        // Create a copy of the original data
        Object dataCopy = this.data.clone();
        // Add the new queue
        this.data.put(queue.ChatID, queue);

        // Try to write the new data to the data file
        try {
            this.WriteToFile();
            return Result.SUCCESS;

        } catch (IOException e) {
            // Revert the data to its original form
            e.printStackTrace();
            this.data = (HashMap<Integer, MessageQueue>) dataCopy;
            return Result.FAILED;
        }
    }

    /**
     * Remove a message queue
     * @param ChatID The ID of the chat whose message queue you are trying to delete
     * @return Result code
     */
    public Result RemoveMessageQueue(int ChatID) {
        // Create a copy of the original data
        Object dataCopy = this.data.clone();
        // Add the new queue
        this.data.remove(ChatID);

        // Try to write the new data to the data file
        try {
            this.WriteToFile();
            return Result.SUCCESS;

        } catch (IOException e) {
            // Revert the data to its original form
            e.printStackTrace();
            this.data = (HashMap<Integer, MessageQueue>) dataCopy;
            return Result.FAILED;
        }
    }
}
```

Database

```java
package com.nathcat.messagecat_database;

import com.nathcat.RSA.KeyPair;
import com.nathcat.messagecat_database_entities.*;
import org.json.simple.parser.ParseException;

import java.io.IOException;
import java.sql.SQLException;

/**
* Wrapper which combines the different database systems into one unit.
*
* @author Nathan "Nathcat" Baines
*/
public class Database {
   protected MySQLHandler mySQLHandler = null;   // MySQLHandler instance
   protected MessageStore messageStore = null;   // MessageStore instance
   protected KeyStore keyStore = null;           // KeyStore instance
   private final ExpirationManager expirationManager;  // The expiration manager

   /**
    * Default constructor
    */
   public Database() {
       // Try to create instances of the three database systems
       try {
           this.mySQLHandler = new MySQLHandler();
           this.keyStore = new KeyStore();
           this.messageStore = new MessageStore();

       } catch (ParseException | SQLException | IOException e) {
           e.printStackTrace();
       }

       // Ensure that all the systems have been initialised correctly
       assert this.mySQLHandler != null && this.messageStore != null && this.keyStore != null;

       // Start the expiration manager
       expirationManager = new ExpirationManager(this);
       expirationManager.setDaemon(true);
       expirationManager.start();
   }

   /**
    * @see MessageStore#WriteToFile()
    * @see KeyStore#WriteToFile()
    */
   public void SaveKeyAndMessageStore() {
       try {
           this.messageStore.WriteToFile();
           this.keyStore.WriteToFile();

       } catch (IOException e) {
```

```
            e.printStackTrace();
        }
    }

    /**
     * @see com.nathcat.messagecat_database.MySQLHandler#GetUserByID(int)
     */
    public User GetUserByID(int UserID) {
        try {
            return this.mySQLHandler.GetUserByID(UserID);

        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * @see com.nathcat.messagecat_database.MySQLHandler#GetUserByUsername(String)
     */
    public User GetUserByUsername(String Username) {
        try {
            return this.mySQLHandler.GetUserByUsername(Username);

        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * @see com.nathcat.messagecat_database.MySQLHandler#GetUserByDisplayName(String)
     */
    public User[] GetUserByDisplayName(String DisplayName) {
        try {
            return this.mySQLHandler.GetUserByDisplayName(DisplayName);

        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
     * @see com.nathcat.messagecat_database.MySQLHandler#GetFriendshipByID(int)
     */
    public Friendship GetFriendshipByID(int FriendshipID) {
        try {
            return this.mySQLHandler.GetFriendshipByID(FriendshipID);

        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }

    /**
```

```java
 * @see com.nathcat.messagecat_database.MySQLHandler#GetFriendshipByUserID(int)
 */
public Friendship[] GetFriendshipByUserID(int UserID) {
    try {
        return this.mySQLHandler.GetFriendshipByUserID(UserID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetFriendshipByUserIDAndFriendID(int, int)
 */
public Friendship GetFriendshipByUserIDAndFriendID(int UserID, int FriendID) {
    try {
        return this.mySQLHandler.GetFriendshipByUserIDAndFriendID(UserID, FriendID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetFriendRequestsBySenderID(int)
 */
public FriendRequest[] GetFriendRequestsBySenderID(int SenderID) {
    try {
        return this.mySQLHandler.GetFriendRequestsBySenderID(SenderID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetFriendRequestsByRecipientID(int)
 */
public FriendRequest[] GetFriendRequestsByRecipientID(int RecipientID) {
    try {
        return this.mySQLHandler.GetFriendRequestsByRecipientID(RecipientID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#DeleteFriendRequest(int)
 */
public Result DeleteFriendRequest(int FriendRequestID) {
    try {
        this.mySQLHandler.DeleteFriendRequest(FriendRequestID);
```

```
        return Result.SUCCESS;

    } catch (SQLException e) {
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetChatByID(int)
 */
public Chat GetChatByID(int ChatID) {
    try {
        return this.mySQLHandler.GetChatByID(ChatID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetChatByPublicKeyID(int)
 */
public Chat GetChatByPublicKeyID(int PublicKeyID) {
    try {
        return this.mySQLHandler.GetChatByPublicKeyID(PublicKeyID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetChatInviteByID(int)
 */
public ChatInvite GetChatInviteByID(int ChatInviteID) {
    try {
        return this.mySQLHandler.GetChatInviteByID(ChatInviteID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetChatInvitesBySenderID(int)
 */
public ChatInvite[] GetChatInvitesBySenderID(int SenderID) {
    try {
        return this.mySQLHandler.GetChatInvitesBySenderID(SenderID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
```

```java
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#GetChatInvitesByRecipientID(int)
 */
public ChatInvite[] GetChatInvitesByRecipientID(int RecipientID) {
    try {
        return this.mySQLHandler.GetChatInvitesByRecipientID(RecipientID);

    } catch (SQLException e) {
        e.printStackTrace();
        return null;
    }
}

public Result DeleteChatInvite(int ChatInviteID) {
    try {
        this.mySQLHandler.DeleteChatInvite(ChatInviteID);
        return Result.SUCCESS;

    } catch (SQLException e) {
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#AddUser(User)
 */
public Result AddUser(User user) {
    try {
        this.mySQLHandler.AddUser(user);
        return Result.SUCCESS;

    } catch (SQLException e) {
        e.printStackTrace();
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#AddFriendship(Friendship)
 */
public Result AddFriendship(Friendship friendship) {
    try {
        this.mySQLHandler.AddFriendship(friendship);
        return Result.SUCCESS;

    } catch (SQLException e) {
        e.printStackTrace();
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#AddFriendRequest(FriendRequest)
 */
public Result AddFriendRequest(FriendRequest friendRequest) {
```

```java
    try {
        this.mySQLHandler.AddFriendRequest(friendRequest);
        return Result.SUCCESS;

    } catch (SQLException e) {
        e.printStackTrace();
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#AddChat(Chat)
 */
public Result AddChat(Chat chat) {
    try {
        this.mySQLHandler.AddChat(chat);
        return Result.SUCCESS;

    } catch (SQLException e) {
        e.printStackTrace();
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MySQLHandler#AddChatInvite(ChatInvite)
 */
public Result AddChatInvite(ChatInvite chatInvite) {
    try {
        this.mySQLHandler.AddChatInvite(chatInvite);
        return Result.SUCCESS;

    } catch (SQLException e) {
        e.printStackTrace();
        return Result.FAILED;
    }
}

/**
 * @see com.nathcat.messagecat_database.MessageStore#GetMessageQueue(int)
 */
public MessageQueue GetMessageQueue(int ChatID) {
    return this.messageStore.GetMessageQueue(ChatID);
}

/**
 * @see com.nathcat.messagecat_database.MessageStore#AddMessageQueue(MessageQueue)
 */
public Result AddMessageQueue(MessageQueue messageQueue) {
    return this.messageStore.AddMessageQueue(messageQueue);
}

/**
 * Add a key pair to the key store
 * @param pair The key pair to add
 * @return The result code
 * @see com.nathcat.messagecat_database.KeyStore#AddKeyPair(KeyPair)
```

```
     */
    public Result AddKeyPair(KeyPair pair) {
        return this.keyStore.AddKeyPair(pair);
    }

    /**
     * Get a key pair from the key store
     * @param keyID The ID of the key pair
     * @return The key pair found at this ID
     */
    public KeyPair GetKeyPair(int keyID) {
        return this.keyStore.GetKeyPair(keyID);
    }

    /**
     * Remove a key pair from the key store
     * @param id The id of the key pair to remove
     * @return The result code
     * @see com.nathcat.messagecat_database.KeyStore#RemoveKeyPair(int)
     */
    public Result RemoveKeyPair(int id) {
        return this.keyStore.RemoveKeyPair(id);
    }
}
```

ExpirationManager

```
package com.nathcat.messagecat_database;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Date;

/**
* Manages the expiry of things such as friend requests, or chat invitations.
*/
public class ExpirationManager extends Thread {
    private final Database db;
    private final long maxTimeElapsed = 2592000000L;

    public ExpirationManager(Database db) {
        this.db = db;
    }

    @Override
    public void run() {
        // Get all friend requests from the database
        ResultSet rs;

        // Check friend requests
        try {
            rs = this.db.mySQLHandler.Select("select * from `FriendRequests`");

            while (rs.next()) {
                long timeSent = rs.getLong("TimeSent");
```

```
                long currentTime = new Date().getTime();
                if (currentTime >= (timeSent + this.maxTimeElapsed)) {
                    this.db.mySQLHandler.Update("delete from `FriendRequests` where `FriendRequestID` like " +
rs.getInt("FriendRequestID"));
                }
            }

        } catch (SQLException e) {
            this.DebugLog(e.getMessage() + " when getting friend requests.");
        }

        // Check chat invitations
        try {
            rs = this.db.mySQLHandler.Select("select * from `ChatInvitations`");

            while (rs.next()) {
                long timeSent = rs.getLong("TimeSent");
                long currentTime = new Date().getTime();
                if (currentTime >= (timeSent + this.maxTimeElapsed)) {
                    this.db.keyStore.RemoveKeyPair(rs.getInt("PrivateKeyID"));
                    this.db.mySQLHandler.Update("delete from `ChatInvitations` where `ChatInviteID` like " +
rs.getInt("ChatInviteID"));
                }
            }

        } catch (SQLException e) {
            this.DebugLog(e.getMessage() + " when getting chat invites");
        }
    }

    private void DebugLog(String message) {
        System.out.println("Database (ExpirationManager): " + message);
    }
}
```

Testing

AddUser()

Data	Expected outcome from select statement	Actual outcome from select statement
Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Access denied
Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated:	Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated:	Access denied

25/07/2022 ProfilePicturePath: default.png	25/07/2022 ProfilePicturePath: default.png	

I received an "Access denied" error when the program attempted to connect to the *MySQL* server, I discovered that this was because the IP address JDBC tried to connect to was "127.0.0.1" (localhost), which did not fit the pattern set in the user access limitations on the *MySQL* server, so I changed the IP address to "192.168.1.26", the IP address of my device.

This fixed the "Access denied" error but uncovered another one, saying that "no database was selected", to rectify this I further modified the connection URL in the config file to "jdbc:mysql://192.168.1.26:3306/messagecat", as the path of the url tells JDBC which database to use upon connection to the database, according to the documentation. This fixed all runtime errors.

Data	Expected outcome from select statement	Actual outcome from select statement
Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png
Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 2 Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 2 Username: OtherPhoneNumber Password: HelloWorld123456 DisplayName: Herman DateCreated: 25/07/2022 ProfilePicturePath: default.png

AddFriendship()

Data	Expected outcome from select statement	Actual outcome from select statement
UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 2 FriendID: 1 DateEstablished: 25/07/2022	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022

AddFriendRequest()

Data	Expected outcome from select statement	Actual outcome from select statement
FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659277170486	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent:	Data truncation: Out of range value for column 'TimeSent' at row 1
FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent: 1659277170486	FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent:	Data truncation: Out of range value for column 'TimeSent' at row 1

The value for TimeSent was chosen by the test program, it is the number of milliseconds that have passed since January 1st 1970, 00:00:00 GMT (according to the Java documentation). This value is too large to be stored in a simple integer, so it must be stored in a long integer type, which contains 64 bits, rather than 32 bits. This means that the *MySQL* server will need to be reconfigured slightly to allow it to store integers that large. To do this I will change the timestamp columns which require an INT type, to require a BIGINT type, which should be able to store the required amount of data.

```
alter table `FriendRequests` modify column `TimeSent` BIGINT;
alter table `ChatInvitations` modify column `TimeSent` BIGINT;
```

This short SQL script should perform this action. Let's try again.

Data	Expected outcome from select statement	Actual outcome from select statement
FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305
FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent: 1659278362305	FriendRequestID: 1 SenderID: 2 RecipientID: 1 TimeSent: 1659278362305	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305

AddChat()

Data	Expected outcome from select statement	Actual outcome from select statement
ChatID: 1 Name: "test1" Description: "test1-description" PublicKeyID: 1	ChatID: 1 Name: "test1" Description: "test1-description" PublicKeyID: 1	ChatID: 1 Name: "test1" Description: "test1-description" PublicKeyID: 1

ChatID: 2 Name: “test2” Description: “test2-description” PublicKeyID: 2	ChatID: 2 Name: “test2” Description: “test2-description” PublicKeyID: 2	ChatID: 2 Name: “test2” Description: “test2-description” PublicKeyID: 2

AddChatInvite()

Data	Expected outcome from select statement	Actual outcome from select statement
ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1
ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2	ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2	ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2

GetUser()

Data	Expected outcome	Actual outcome
UserID: 1	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Operation not allowed after ResultSet closed
Username: MyPhoneNumber	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Operation not allowed after ResultSet closed
DisplayName: Nathcat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Operation not allowed after ResultSet closed

UserID: 3	null	Operation not allowed after ResultSet closed
Username: aiwdwid	null	Operation not allowed after ResultSet closed
DisplayName: Nat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	Operation not allowed after ResultSet closed

This error is fairly self-explanatory, we are trying to perform an operation on a *ResultSet* object after it has been closed. The question is, where is it being closed? As it turns out, when we close the *Statement* object in the *Select* method, we also close the *ResultSet,* so removing this line fixed this error, but also brought up another one: *Operation not allowed for a result set of type ResultSet.TYPE_FORWARD_ONLY.*

After some browsing through the documentation I found a fix for this, when we create the statement in the *Select* method, we use the default constructor, which gives a default setting which means that we can only move forward through the *ResultSet,* but we need to move in both directions. To allow this, we change this line:

```
Statement stmt = conn.createStatement();
```

To this:

```
Statement stmt = conn.createStatement(
        ResultSet.TYPE_SCROLL_SENSITIVE,
        ResultSet.CONCUR_READ_ONLY
);
```

This fixed the error.

Data	**Expected outcome**	**Actual outcome**
UserID: 1	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png
Username: MyPhoneNumber	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png

DisplayName: Nathcat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png
UserID: 3	null	null
Username: aiwdwid	null	null
DisplayName: Nat	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png	UserID: 1 Username: MyPhoneNumber Password: HelloWorld1234 DisplayName: Nathcat DateCreated: 25/07/2022 ProfilePicturePath: default.png

GetFriendship()

Data	Expected outcome	Actual outcome
FriendshipID: 1	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 1	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
UserID: 1 FriendID: 2	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022	FriendshipID: 1 UserID: 1 FriendID: 2 DateEstablished: 25/07/2022
FriendshipID: 3	null	null
UserID: 3	“”	“”
UserID: 2 FriendID: 1	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022	FriendshipID: 2 UserID: 2 FriendID: 1 DateEstablished: 25/07/2022

GetFriendRequests()

Data	Expected outcome	Actual outcome

SenderID: 1	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305	FriendRequestID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659278362305
RecipientID: 1	FriendRequestID: 2 SenderID: 2 RecipientID: 1 TimeSent: 1659278362305	FriendRequestID: 2 SenderID: 2 RecipientID: 1 TimeSent: 1659278362305
SenderID: 3	“”	“”
RecipientID: 3	“”	“”

GetChat()

Data	Expected outcome	Actual outcome
ChatID: 1	ChatID: 1 Name: test1 Description: test1-description PublicKeyID: 1	ChatID: 1 Name: test1 Description: test1-description PublicKeyID: 1
ChatID: 2	ChatID: 2 Name: test2 Description: test2-description PublicKeyID: 2	ChatID: 2 Name: test2 Description: test2-description PublicKeyID: 2
ChatID: 3	null	null

GetChatInvite()

Data	Expected outcome	Actual outcome
ChatInviteID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1	ChatID: 1 Name: test1 Description: test1-description PublicKeyID: 1
ChatInviteID: 3	null	null
RecipientID: 2	ChatInviteID: 1 ChatID: 1 SenderID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1

	RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2	RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2
RecipientID: 3	null	null
SenderID: 1	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2	ChatInviteID: 1 ChatID: 1 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 1, ChatInviteID: 2 ChatID: 2 SenderID: 1 RecipientID: 2 TimeSent: 1659357276989 PrivateKeyID: 2
SenderID: 3	null	null

Having completed the testing process for this part of the application we can see that it functions as expected after some changes, so this part of the development process has been successful and we can safely move on.

Connection timeout issue

During testing I decided to have the database run for a long time, and noticed that the MySQL connection timed out after a while. A way to fix this would be to catch any connection errors and restart the connection to the server to ensure that it is always active. Following are the changes I made in order to fix this issue.

```
18  18      public class MySQLHandler {
19        -     private final Connection conn;  // The connection to the MySQL database.
    19    +     private Connection conn;          // The connection to the MySQL database.
    20    +     private final JSONObject config;  // Config JSONObject
20  21
21  22          /**
22  23           * Default constructor, creates a connection to the database.
@@ -26,8 +27,12 @@ public MySQLHandler() throws FileNotFoundException, ParseException, SQLException
26  27              // If any of them are thrown, this object shouldn't be allowed to be created.
27  28
28  29              // Get the MySQL config file
29        -         JSONObject config = this.GetMySQLConfig();
    30    +         this.config = this.GetMySQLConfig();
30  31
    32    +         StartConnection();
    33    +     }
    34    +
    35    +     private void StartConnection() throws SQLException {
31  36              // Create a connection to the MySQL database.
32  37              conn = DriverManager.getConnection((String) config.get("connection_url"), (String) config.get("username"), (String)
        config.get("password"));
33  38          }
@@ -56,14 +61,29 @@ private JSONObject GetMySQLConfig() throws FileNotFoundException, ParseException
56  61           * @throws SQLException Thrown by SQL errors.
57  62           */
58  63          protected ResultSet Select(String query) throws SQLException {
59        -         // Create and execute the statement
60        -         Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
61        -         stmt.execute(query);
62        -         // Get the result set and close the statement
63        -         ResultSet rs = stmt.getResultSet();
64        -
65        -         // Return the result set
66        -         return rs;
    64    +         try {
    65    +             // Create and execute the statement
    66    +             Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
    67    +             stmt.execute(query);
```

```
    71  +            // Return the result set
    72  +            return rs;
    73  +
    74  +        } catch (SQLNonTransientConnectionException e) {
    75  +            // Restart the connection and try again
    76  +            StartConnection();
    77  +
    78  +            // Create and execute the statement
    79  +            Statement stmt = conn.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_READ_ONLY);
    80  +            stmt.execute(query);
    81  +            // Get the result set and close the statement
    82  +            ResultSet rs = stmt.getResultSet();
    83  +
    84  +            // Return the result set
    85  +            return rs;
    86  +        }
67  87          }
68  88
69  89          /**
@@ -72,12 +92,25 @@ protected ResultSet Select(String query) throws SQLException {
72  92           * @throws SQLException Thrown by SQL errors
73  93           */
74  94          protected void Update(String query) throws SQLException {
75      -            // Create and execute the statement
76      -            Statement stmt = conn.createStatement();
77      -            stmt.execute(query);
    95  +        try {
    96  +            // Create and execute the statement
    97  +            Statement stmt = conn.createStatement();
    98  +            stmt.execute(query);
    99  +
   100  +            // Close the statement
   101  +            stmt.close();
78 102
79      -            // Close the statement
80      -            stmt.close();
   103  +        } catch (SQLNonTransientConnectionException e) {
   104  +            // Restart the connection and try again
   105  +            StartConnection();
   106  +
   107  +            // Create and execute the statement
   108  +            Statement stmt = conn.createStatement();
   109  +            stmt.execute(query);
   110  +
   111  +            // Close the statement
   112  +            stmt.close();
   113  +        }
```

Server

This part of the application will be contained within a package called *com.nathcat.messagecat_server*. This is the central part of the system which accepts incoming connections from client applications and handles requests to the database.

ListenRule

```java
package com.nathcat.messagecat_server;

import java.io.Serializable;
import java.lang.reflect.Field;
```

```java
/**
* A listening rule to be supplied to a client when they want to listen for actions performed
* on the server by other clients.
*
* @author Nathan "Nathcat" Baines
*/

public class ListenRule implements Serializable {
   public class IDAlreadySetException extends Exception { }

   private int id = -1;                // Unique identifier for this listening rule
   public int connectionHandlerId = -1;
   public ConnectionHandler handler;   // Handler which is handling the client this listen rule was created by
   private RequestType listenForType;  // The request type which this listen rule is listening for

   // The listen rule will only be triggered if the data in fieldToMatch matches the data in objectToMatch
   private String fieldNameToMatch;
   private Object objectToMatch;
   private Object[] objectsToMatch;

   public ListenRule(ConnectionHandler handler, RequestType listenForType, String fieldNameToMatch, Object
objectToMatch) {
       this.handler = handler;
       this.listenForType = listenForType;
       this.fieldNameToMatch = fieldNameToMatch;
       this.objectToMatch = objectToMatch;
   }

   public ListenRule(int connectionHandlerId, RequestType listenForType, String fieldNameToMatch, Object
objectToMatch) {
       this.listenForType = listenForType;
       this.fieldNameToMatch = fieldNameToMatch;
       this.objectToMatch = objectToMatch;
       this.connectionHandlerId = connectionHandlerId;
   }

   public ListenRule(RequestType listenForType, String fieldNameToMatch, Object objectToMatch) {
       this.listenForType = listenForType;
       this.fieldNameToMatch = fieldNameToMatch;
       this.objectToMatch = objectToMatch;
       this.objectsToMatch = null;
   }
   public ListenRule(RequestType listenForType, String fieldNameToMatch, Object[] objectsToMatch) {
       this.listenForType = listenForType;
       this.fieldNameToMatch = fieldNameToMatch;
       this.objectToMatch = null;
       this.objectsToMatch = objectsToMatch;
   }

   public ListenRule(RequestType listenForType) {
       this.listenForType = listenForType;
   }

   public ListenRule(int connectionHandlerId, RequestType listenForType) {
       this.listenForType = listenForType;
       this.connectionHandlerId = connectionHandlerId;
   }
```

```
    public int getId() {
        return this.id;
    }

    public void setId(int id) throws IDAlreadySetException {
        if (this.id != -1) {
            throw new IDAlreadySetException();
        }

        this.id = id;
    }

    /**
     * Checks if a request matches the listen rule's criteria, and send the request to the client if it does
     * @param type The type of request
     * @param data The data object to compare to
     * @return True if the listen rule criteria is met, False if not.
     */
    public boolean CheckRequest(RequestType type, Object data) throws NoSuchFieldException, IllegalAccessException {
        if (fieldNameToMatch == null) {
            return type == listenForType;
        }

        if (objectToMatch != null) {
            return type == listenForType && data.getClass()  // Compare request type to the type we are listening for
and get the class object of data
                    .getField(fieldNameToMatch)              // Get the field we are comparing
                    .get(data).equals(objectToMatch);        // Get the data from the field in the instance of data
and compare to objectToMatch
        }
        else {
            boolean contains = false;
            for (Object toMatch : objectsToMatch) {
                if (data.getClass().getField(fieldNameToMatch).get(data).equals(toMatch)) {
                    contains = true;
                    break;
                }
            }

            return type == listenForType && contains;
        }
    }
}
```

Handler

```
package com.nathcat.messagecat_server;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.lang.reflect.Field;
import java.math.BigInteger;
import java.net.*;
```

```java
import com.nathcat.RSA.*;

/**
* Parent class for the three handler classes
*
* @author Nathan "Nathcat" Baines
*/
public class Handler extends Thread {
    public Socket socket;                   // The TCP/IP connection socket
    public ObjectOutputStream oos;          // Stream to output objects to through the socket
    public ObjectInputStream ois;           // Stream to receive objects from the socket
    public final int threadNum;             // The thread number, used in debug messages
    private final String className;         // The name of the class to be used in debug messages
    public KeyPair keyPair;                 // The encryption key pair to be used in communications
    public KeyPair clientKeyPair;           // The client's encryption key pair (will only contain the public key)
    public boolean busy = false;            // Indicates whether the handler is busy
    public Server server;                   // Parent server object
    public Object queueObject;              // The object supplied to the handler from the QueueManager
    public boolean authenticated;           // Whether the connection is authenticated or not
    public Socket lrSocket;                 // A socket specifically for communicating listen rule triggers
    public ObjectOutputStream lrOos;        // Stream to output objects to the lrSocket

    /**
     * Constructor method, assigns private and constant fields
     * @param socket The TCP/IP connection socket to be used by this handler
     * @param threadNum The Thread number of this handler, used in debug messages
     * @param className The name of the class, used in debug messages
     */
    public Handler(Socket socket, int threadNum, String className) {
        this.socket = socket;
        this.threadNum = threadNum;
        this.className = className;

        // Make this thread a daemon of the main process
        // This means that this thread will quit when the main program quits.
        this.setDaemon(true);
    }

    /**
     * Starts I/O streams and creates RSA key pair
     * @throws IOException Thrown by I/O operations
     */
    public void InitializeIO() throws IOException {
        this.oos = new ObjectOutputStream(this.socket.getOutputStream());
        this.ois = new ObjectInputStream(this.socket.getInputStream());
    }

    /**
     * Stop the handler thread using the Thread.wait() method
     */
    public synchronized void StopHandler() {
        try {
            this.wait();

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
```

```java
}

/**
 * Create a debug log message using the identification data supplied to this handler
 * @param message The message to output
 */
public void DebugLog(String message) {
    System.out.println(className + " (" + threadNum + "): " + message);
}

/**
 * Send an object over the socket
 * @param obj The object to send
 * @throws IOException Thrown if there is an I/O issue
 */
public void Send(Object obj) throws IOException {
    this.oos.writeObject(obj);
    this.oos.flush();
}

/**
 * Send an object via the listen rule socket
 * @param obj The object to send
 */
public void LrSend(Object obj) throws IOException {
    lrOos.writeObject(obj);
    lrOos.flush();
}

/**
 * Receive an object from the socket
 * @return The object that is received
 * @throws IOException Thrown if there is an I/O issue
 * @throws ClassNotFoundException Thrown if a requested class cannot be found
 */
public Object Receive() throws IOException, ClassNotFoundException {
    return this.ois.readObject();
}

/**
 * Try to close the socket
 */
public void Close() {
    try {
        this.socket.close();
        this.lrSocket.close();

    } catch (Exception e) {
        this.DebugLog("Failed to close socket (" + e.getMessage() + ")");
    }

    boolean emptyPass = false;
    while (!emptyPass) {
        emptyPass = true;

        for (int i = 0 ; i < this.server.listenRules.size(); i++) {
            if (this.server.listenRules.get(i).handler.equals(this)) {
```

```
                this.server.listenRules.remove(i);
                emptyPass = false;
                break;
            }
        }
    }

    authenticated = false;
    busy = false;
  }
}
```

ConnectionHandler

```
package com.nathcat.messagecat_server;

import com.nathcat.RSA.*;
import com.nathcat.messagecat_database.MessageQueue;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.*;
import org.json.simple.JSONObject;

import java.io.IOException;
import java.io.ObjectOutputStream;
import java.net.*;
import java.security.NoSuchAlgorithmException;
import java.util.Arrays;
import java.util.Objects;

/**
* Maintains a connection with the client device
*
* @author Nathan "Nathcat" Baines
*/
public class ConnectionHandler extends Handler {
   private JSONObject request;

   /**
    * Constructor method, assigns private and constant fields
    *
    * @param socket    The TCP/IP connection socket to be used by this handler
    * @param threadNum The Thread number of this handler, used in debug messages
    */
   public ConnectionHandler(Socket socket, int threadNum) throws NoSuchAlgorithmException, IOException {
       super(socket, threadNum, "ConnectionHandler");
   }

   /**
    * This method will be executed in a different thread
    */
   @Override
   public void run() {
       while (true) {
           this.busy = false;
           this.StopHandler();
```

```
        if (this.queueObject == null) {
            continue;
        }

        this.busy = true;
        this.authenticated = false;
        this.DebugLog("Assigned to task");

        this.socket = (Socket) ((CloneableObject) this.queueObject).object;

        try {
            this.InitializeIO();

        } catch (IOException e) {
            this.DebugLog("Failed to initialise I/O (" + e.getMessage() + ").");
            this.Close();
            continue;
        }

        // Perform handshake
        if (this.DoHandshake()) {
             // Open listen rule socket
             try {
                 int port = (int) this.keyPair.decrypt((EncryptedObject) this.Receive());
                 this.lrSocket = new Socket(this.socket.getInetAddress().getHostAddress(), port);
                 this.lrOos = new ObjectOutputStream(lrSocket.getOutputStream());

             } catch (IOException | PrivateKeyException | ClassNotFoundException e) {
                 e.printStackTrace();
                 this.Close();
                 return;
             }

            // Start connection main loop
            this.MainLoop();
        }
        else {
            this.DebugLog("Handshake failed!");
            this.Close();
        }
    }
}

/**
 * Perform the handshake between the server and the client
 * @return Whether the handshake was successful or not
 */
private boolean DoHandshake() {
    // Try to generate an RSA key pair
    try {
        this.keyPair = RSA.GenerateRSAKeyPair();
    } catch (NoSuchAlgorithmException e) {
        this.DebugLog("Failed to generate RSA key pair! (" + e.getMessage() + ")");
        this.queueObject = null;
        return false;
    }
```

```
    // Perform the handshake process
    boolean handshakeSuccessful = true;

    // Try to send the server's public key to the client
    try {
        this.Send(new KeyPair(this.keyPair.pub, null));

    } catch (IOException e) {
        this.DebugLog(e.getMessage());
        handshakeSuccessful = false;
    }

    // Try to receive the client's key pair
    try {
        this.clientKeyPair = (KeyPair) this.Receive();

    } catch (IOException | ClassNotFoundException e) {
        this.DebugLog(e.getMessage());
        handshakeSuccessful = false;
    }

    // Send the connection handler identifier to the client
    try {
        this.Send(this.clientKeyPair.encrypt(threadNum));
    } catch (IOException | PublicKeyException e) {
        this.DebugLog(e.getMessage());
        handshakeSuccessful = false;
    }

    return handshakeSuccessful;
}

/**
 * The main handler loop for the connection
 */
private void MainLoop() {
    while (true) {
        try {
            JSONObject request = (JSONObject) this.keyPair.decrypt((EncryptedObject) this.Receive());
            Object response = this.HandleRequest(request);
            this.Send(this.clientKeyPair.encrypt(response));

        } catch (Exception e) {
            this.DebugLog("Exception in main protocol: " + e.getMessage());
            this.Close();
            return;
        }
    }
}

/**
 * Handle a JSON request object
 * @param request The JSON request object
 * @return The response object
 */
private Object HandleRequest(JSONObject request) {
```

```
        this.request = request;

        if (request == null) {
            this.Close();
            return null;
        }

        switch ((RequestType) request.get("type")) {
            case Authenticate -> {
                return this.Authenticate();
            }

            case GetUser -> {
                return this.GetUser();
            }

            case GetFriendship -> {
                return this.GetFriendship();
            }

            case GetFriendRequests -> {
                return this.GetFriendRequests();
            }

            case GetChat -> {
                return this.GetChat();
            }

            case GetChatInvite -> {
                return this.GetChatInvite();
            }

            case GetPublicKey -> {
                return this.GetPublicKey();
            }

            case GetMessageQueue -> {
                return this.GetMessageQueue();
            }

            case AddUser -> {
                return this.AddUser();
            }

            case AddChat -> {
                return this.AddChat();
            }

            case AddListenRule -> {
                return this.AddListenRule();
            }

            case RemoveListenRule -> {
                return this.RemoveListenRule();
            }

            case AcceptFriendRequest -> {
```

```
                return this.AcceptFriendRequest();
            }

            case DeclineFriendRequest -> {
                return this.DeclineFriendRequest();
            }

            case AcceptChatInvite -> {
                return this.AcceptChatInvite();
            }

            case DeclineChatInvite -> {
                return this.DeclineChatInvite();
            }

            case SendMessage -> {
                return this.SendMessage();
            }

            case SendFriendRequest -> {
                return this.SendFriendRequest();
            }

            case SendChatInvite -> {
                return this.SendChatInvite();
            }
        }

        return null;
    }

    private Object Authenticate() {
        // Get the authentication data from the request
        User authData = (User) this.request.get("data");

        // Get the corresponding user from the database (by username)
        User user = this.server.db.GetUserByUsername(authData.Username);

        // Check if the user is null (i.e. the username is incorrect)
        if (user == null) {
            this.authenticated = false;
            return "failed";
        }
        else {  // Check the auth data is valid
            if (user.Password.contentEquals(authData.Password)) {
                this.authenticated = true;
                return user;
            }
            else {
                this.authenticated = false;
                return "failed";
            }
        }
    }

    private Object GetUser() {
        if (!this.authenticated) {
```

```
            return null;
        }

        // Get the user from the request and decrypt
        User requestedUser = (User) this.request.get("data");

        // Get the selector
        String selector = (String) this.request.get("selector");

        // Search the database and return the result
        Object result = null;

        if (selector.contentEquals("id")) {
            result = this.server.db.GetUserByID(requestedUser.UserID);
        }
        else if (selector.contentEquals("username")) {
            result = this.server.db.GetUserByUsername(requestedUser.Username);
        }
        else if (selector.contentEquals("displayName")) {
            result = this.server.db.GetUserByDisplayName(requestedUser.DisplayName);
        }
        else {
            this.DebugLog("Invalid selector!");
            this.Close();
            return null;
        }

        // Remove the password from the result/s
        if (result.getClass() == User.class) {
            result = new User(((User) result).UserID, ((User) result).Username, null, ((User) result).DisplayName,
((User) result).DateCreated, ((User) result).ProfilePicturePath);
        }
        else {
            User[] uArray = (User[]) result;
            for (int i = 0; i < uArray.length; i++) {
                uArray[i] = new User(uArray[i].UserID, uArray[i].Username, null, uArray[i].DisplayName,
uArray[i].DateCreated, uArray[i].ProfilePicturePath);
            }

            result = uArray;
        }

        return result;
    }

    private Object GetFriendship() {
        if (!this.authenticated) {
            return null;
        }

        // Get the friendship from the request and decrypt
        Friendship requestedFriendship = (Friendship) this.request.get("data");

        // Get the selector
        String selector = (String) this.request.get("selector");

        // Search the database and return the result
```

```
        Object result = null;
        if (selector.contentEquals("id")) {
            result = this.server.db.GetFriendshipByID(requestedFriendship.FriendshipID);
        }
        else if (selector.contentEquals("userID")) {
            result = this.server.db.GetFriendshipByUserID(requestedFriendship.UserID);
        }
        else if (selector.contentEquals("userID&FriendID")) {
            result = this.server.db.GetFriendshipByUserIDAndFriendID(requestedFriendship.UserID,
requestedFriendship.FriendID);
        }
        else {
            this.DebugLog("Invalid selector!");
            this.Close();
            return null;
        }

        return result;
    }

    private Object GetFriendRequests() {
        if (!this.authenticated) {
            return null;
        }

        // Get the friend request from the request and decrypt it
        FriendRequest requestedFriendRequest = (FriendRequest) this.request.get("data");

        // Get the selector
        String selector = (String) this.request.get("selector");

        // Search the database and return the result
        Object result = null;

        if (selector.contentEquals("senderID")) {
            result = this.server.db.GetFriendRequestsBySenderID(requestedFriendRequest.SenderID);
        }
        else if (selector.contentEquals("recipientID")) {
            result = this.server.db.GetFriendRequestsByRecipientID(requestedFriendRequest.RecipientID);
        }
        else {
            this.DebugLog("Invalid selector!");
            this.Close();
            return null;
        }

        return result;
    }

    private Object GetChat() {
        if (!this.authenticated) {
            return null;
        }

        // Get the chat from the request and decrypt
        Chat requestedChat = (Chat) this.request.get("data");
```

```
    // Search the database and return the result
    return this.server.db.GetChatByID(requestedChat.ChatID);
}

private Object GetChatInvite() {
    if (!this.authenticated) {
        return null;
    }

    // Get the chat invite from the request and decrypt it
    ChatInvite requestedChatInvite = (ChatInvite) this.request.get("data");

    // Get the selector
    String selector = (String) this.request.get("selector");

    // Search the database and return the result
    Object result = null;

    if (selector.contentEquals("id")) {
        result = this.server.db.GetChatInviteByID(requestedChatInvite.ChatInviteID);
    }
    else if (selector.contentEquals("senderID")) {
        result = this.server.db.GetChatInvitesBySenderID(requestedChatInvite.SenderID);
    }
    else if (selector.contentEquals("recipientID")) {
        result = this.server.db.GetChatInvitesByRecipientID(requestedChatInvite.RecipientID);
    }
    else {
        this.DebugLog("Invalid selector!");
        this.Close();
        return null;
    }

    return result;
}

private Object GetPublicKey() {
    if (!this.authenticated) {
        return null;
    }

    // Get the public key id from the request
    int keyID = (int) this.request.get("data");

    // Get the key pair from the database
    return this.server.db.GetKeyPair(keyID);
}

private Object GetMessageQueue() {
    if (!this.authenticated) {
        return null;
    }

    // Get the chat id from the request
    int chatID = (int) this.request.get("data");

    // Get the message queue
```

```
        return this.server.db.GetMessageQueue(chatID);
    }

    private Object AddUser() {
        // Get the user from the request and decrypt
        User newUser = (User) this.request.get("data");

        if (!Arrays.equals(this.server.db.GetUserByDisplayName(newUser.DisplayName), new User[0]) ||
this.server.db.GetUserByUsername(newUser.Username) != null) {
            return null;
        }

        // Add the new user through the database
        this.server.db.AddUser(newUser);

        // Get the new user from the database and send back to the client
        return this.server.db.GetUserByUsername(newUser.Username);
    }

    private Object AddChat() {
        if (!this.authenticated) {
            return null;
        }

        // Create a new public key
        KeyPair chatKeyPair = (KeyPair) this.request.get("keyPair");

        // Get the chat from the request and decrypt
        Chat newChat = (Chat) this.request.get("data");
        newChat = new Chat(newChat.ChatID, newChat.Name, newChat.Description, chatKeyPair.hashCode());

        // Add the chat, public key, and message queue to the database
        this.server.db.AddChat(newChat);
        this.server.db.AddKeyPair(chatKeyPair);
        // Get the chat from the database, so we know it's ID
        newChat = this.server.db.GetChatByPublicKeyID(newChat.PublicKeyID);
        // Create a new message queue
        MessageQueue messageQueue = new MessageQueue(newChat.ChatID);
        this.server.db.AddMessageQueue(messageQueue);

        // Send the new chat back to the client
        return this.server.db.GetChatByPublicKeyID(chatKeyPair.hashCode());
    }

    private Object AddListenRule() {
        if (!this.authenticated) {
            return null;
        }

        // Get the listen rule object from the request
        ListenRule listenRule = (ListenRule) this.request.get("data");

        // Set the listen rule handler
        if (listenRule.connectionHandlerId == -1) {
            listenRule.handler = this;
        }
        else {
```

```
            listenRule.handler = (ConnectionHandler)
this.server.connectionHandlerPool[listenRule.connectionHandlerId];
        }

        try {
            // Assign an id to the listen rule and add the rule to the list
            listenRule.setId(this.server.listenRules.size());
            this.server.listenRules.add(listenRule);
            // Return the id of the listen rule
            return listenRule.getId();

        } catch (ListenRule.IDAlreadySetException e) {
            e.printStackTrace();
            return "failed";
        }
    }

    private Object RemoveListenRule() {
        if (!this.authenticated) {
            return null;
        }

        int id = (int) this.request.get("data");
        boolean found = false;
        for (int i = 0; i < this.server.listenRules.size(); i++) {
            if (this.server.listenRules.get(i).getId() == id) {
                this.server.listenRules.remove(i);
                found = true;
                break;
            }
        }

        return found ? "done" : "failed";
    }

    private Object AcceptFriendRequest() {
        if (!this.authenticated) {
            return null;
        }

        // Get the friend request from the request, since the data contained are all integers we do not need to
decrypt
        FriendRequest fr = (FriendRequest) this.request.get("data");

        // Check if this request triggers any listen rules
        for (ListenRule rule : this.server.listenRules) {
            try {
                if (rule.CheckRequest(RequestType.AcceptFriendRequest, fr)) {
                    this.request.put("triggerID", rule.getId());
                    rule.handler.LrSend(rule.handler.clientKeyPair.encrypt(this.request));
                }
            } catch (IllegalAccessException | NoSuchFieldException | PublicKeyException | IOException ignored) {}
        }

        // Check if the two users involved are already friends
        if (this.server.db.GetFriendshipByUserIDAndFriendID(fr.SenderID, fr.RecipientID) != null) {
            return "done";
```

```
        }

        // Create the friend objects
        Friendship friendA = new Friendship(-1, fr.SenderID, fr.RecipientID, "");
        Friendship friendB = new Friendship(-1, fr.RecipientID, fr.SenderID, "");

        // Add the friendships to the database
        this.server.db.AddFriendship(friendA);
        this.server.db.AddFriendship(friendB);

        // Delete the friend requests from the database
        if (this.server.db.DeleteFriendRequest(fr.FriendRequestID) == Result.FAILED) {
            return "failed";
        }

        // Reply to the client
        return "done";
    }

    private Object DeclineFriendRequest() {
        if (!this.authenticated) {
            return null;
        }

        // Get the friend request from the request, since the data contained are all integers we do not need to
decrypt
        FriendRequest fr = (FriendRequest) this.request.get("data");

        // Delete the friend requests from the database
        if (this.server.db.DeleteFriendRequest(fr.FriendRequestID) == Result.FAILED) {
            return "failed";
        }

        // Reply to the client
        return "done";
    }

    private Object AcceptChatInvite() {
        if (!this.authenticated) {
            return null;
        }

        // Get the chat invite from the request and the private key from the database
        ChatInvite ci = (ChatInvite) this.request.get("data");
        KeyPair privateKey = this.server.db.GetKeyPair(ci.PrivateKeyID);

        // Check if this request triggers any listen rules
        for (ListenRule rule : this.server.listenRules) {
            try {
                if (rule.CheckRequest(RequestType.AcceptChatInvite, ci)) {
                    this.request.put("triggerID", rule.getId());
                    rule.handler.LrSend(rule.handler.clientKeyPair.encrypt(this.request));
                }
            } catch (IllegalAccessException | NoSuchFieldException | PublicKeyException | IOException ignored) {}
        }

        // Delete the chat invite from the database
```

```
        if (this.server.db.DeleteChatInvite(ci.ChatInviteID) == Result.FAILED ||
this.server.db.RemoveKeyPair(ci.PrivateKeyID) == Result.FAILED) {
            return "failed";
        }

        return privateKey;
    }

    private Object DeclineChatInvite() {
        if (!this.authenticated) {
            return null;
        }

        // Get the chat invite from the request and the private key from the database
        ChatInvite ci = (ChatInvite) this.request.get("data");

        // Delete the chat invite from the database
        if (this.server.db.DeleteChatInvite(ci.ChatInviteID) == Result.FAILED ||
this.server.db.RemoveKeyPair(ci.PrivateKeyID) == Result.FAILED) {
            return "failed";
        }

        return "done";
    }

    private Object SendMessage() {
        if (!this.authenticated) {
            return null;
        }

        // Get the message from the database
        Message message = (Message) this.request.get("data");

        // Check if this request triggers any listen rules
        for (ListenRule rule : this.server.listenRules) {
            try {
                if (rule.CheckRequest(RequestType.SendMessage, message)) {
                    this.request.put("triggerID", rule.getId());
                    rule.handler.LrSend(rule.handler.clientKeyPair.encrypt(this.request));
                }
            } catch (IllegalAccessException | NoSuchFieldException | PublicKeyException | IOException ignored) {}
        }

        // Add the message to the database
        this.server.db.GetMessageQueue(message.ChatID).Push(message);
        this.server.db.SaveKeyAndMessageStore();

        // Reply to the client
        return "done";
    }

    private Object SendFriendRequest() {
        if (!this.authenticated) {
            return null;
        }

        // Get the friend request from the request
```

```
        FriendRequest fr = (FriendRequest) this.request.get("data");

        // Check if this request triggers any listen rules
        for (ListenRule rule : this.server.listenRules) {
            try {
                if (rule.CheckRequest(RequestType.SendFriendRequest, fr)) {
                    this.request.put("triggerID", rule.getId());
                    rule.handler.LrSend(rule.handler.clientKeyPair.encrypt(this.request));
                }
            } catch (IllegalAccessException | NoSuchFieldException | PublicKeyException | IOException ignored) {}
        }

        // Add the request to the database
        if (this.server.db.AddFriendRequest(fr) == Result.FAILED) {
            return "failed";
        }

        return "done";
    }

    private Object SendChatInvite() {
        if (!this.authenticated) {
            return null;
        }

        // Get the chat invite and public key from the request
        ChatInvite chatInvite = (ChatInvite) this.request.get("data");
        KeyPair privateKey = (KeyPair) this.request.get("keyPair");
        chatInvite = new ChatInvite(chatInvite.ChatInviteID, chatInvite.ChatID, chatInvite.SenderID,
chatInvite.RecipientID, chatInvite.TimeSent, privateKey.hashCode());

        // Check if this request triggers any listen rules
        for (ListenRule rule : this.server.listenRules) {
            try {
                if (rule.CheckRequest(RequestType.SendChatInvite, chatInvite)) {
                    this.request.put("triggerID", rule.getId());
                    rule.handler.LrSend(rule.handler.clientKeyPair.encrypt(this.request));
                }
            } catch (IllegalAccessException | NoSuchFieldException | PublicKeyException | IOException ignored) {}
        }

        // Add the chat invite and private key to the database
        if (this.server.db.AddKeyPair(privateKey) == Result.FAILED || this.server.db.AddChatInvite(chatInvite) ==
Result.FAILED) {
            return "failed";
        }

        // Reply to the client
        return "done";
    }
}
```

QueueManager

```
package com.nathcat.messagecat_server;
```

```java
/**
* Handles a connection queue
*
* @author Nathan "Nathcat" Baines
*/
public class QueueManager extends Thread{
   private final Server server;   // The Server object
   public Queue queue;            // The queue assigned to this manager
   private final Handler[] pool;  // The handler pool assigned to this manager

   /**
    * Constructor method
    * @param server The Server object
    * @param queue The queue assigned to this manager
    * @param pool The pool assigned to this manager
    */
   public QueueManager(Server server, Queue queue, Handler[] pool) {
       this.server = server;
       this.queue = queue;
       this.pool = pool;
       // Make this thread a daemon to the program
       // This means that this thread will quit when the program quits
       this.setDaemon(true);
   }

   /**
    * This method will be executed in a separate thread once Thread.start() method is called on this object
    */
   @Override
   public void run() {
       while (true) {
           // Get the object at the front of the queue
           Object frontObj = this.queue.Pop();

           // Check if the front object is null or not
           if (frontObj == null) {
               try {
                   Thread.sleep(1000);

               } catch (InterruptedException e) {
                   e.printStackTrace();
               }

               continue;
           }

           // Check if any of the handlers in the queue are not busy
           boolean threadAvailable = false;
           for (int i = 0; i < this.pool.length; i++) {
               // If the current handler is not busy, assign it to the front object
               if (!this.pool[i].busy) {
                   threadAvailable = true;
                   this.pool[i].queueObject = frontObj;

                   synchronized (this.pool[i]) {
                       this.pool[i].notify();
```

```
                }

                break;
            }
        }

        // If no handler is available, push the object back into the queue
        if (!threadAvailable) {
            this.queue.Push(frontObj);
        }

        try {
            Thread.sleep(1000);

        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
  }
}
```

Server

```
package com.nathcat.messagecat_server;

import com.nathcat.messagecat_database.Database;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.Scanner;

/**
* This is Server object, it will initialise the Server and prepare the program to receive connections.
*
* @author Nathan "Nathcat" Baines
*/
public class Server {
   private final int port;
   private final int maxThreadCount;
   public final Handler[] connectionHandlerPool;
   public final QueueManager connectionHandlerQueueManager;
   public final Database db;
   public final ArrayList<ListenRule> listenRules = new ArrayList<>();

   /**
    * Constructor method
    */
```

```java
public Server() {
    this.DebugLog("Getting config file");

    // Get the config file and set constant fields
    JSONObject config = this.GetConfigFile();

    // Define the constant fields
    this.port = Integer.parseInt((String) config.get("port"));
    this.maxThreadCount = Integer.parseInt((String) config.get("maxThreadCount"));

    this.DebugLog("Starting database");
    this.db = new Database();

    this.DebugLog("Creating thread pools");

    // Create the thread pools
    connectionHandlerPool = new Handler[this.maxThreadCount];

    this.DebugLog("Creating handlers (" + this.maxThreadCount + " handlers to create)");
    // Populate the thread pools with handlers
    for (int i = 0; i < this.maxThreadCount; i++) {
        try {
            connectionHandlerPool[i] = new ConnectionHandler(null, i);
            connectionHandlerPool[i].server = this;
            connectionHandlerPool[i].start();

        } catch (NoSuchAlgorithmException | IOException e) {
            this.DebugLog("Failed to create handler pools! (" + e.getMessage() + ")");
            System.exit(1);
        }
    }

    this.DebugLog("Starting queue managers");
    // Start the queue managers
    connectionHandlerQueueManager = new QueueManager(this, new Queue(), this.connectionHandlerPool);
    connectionHandlerQueueManager.start();

    this.DebugLog("Initial setup complete");
}

/**
 * Get the server config file located at Assets/Server_Config.json
 * @return A JSON object parsed from the file's contents
 */
private JSONObject GetConfigFile() {
    Scanner file = null;
    try {
        file = new Scanner(new File("Assets/Server_Config.json"));

    } catch (FileNotFoundException e) {
        this.DebugLog("Couldn't find config file at \"Assets/Server_Config.json\".");
        System.exit(1);
    }

    StringBuilder sb = new StringBuilder();
    while (file.hasNextLine()) {
        sb.append(file.nextLine());
```

```java
        }

        try {
            return (JSONObject) new JSONParser().parse(sb.toString());
        } catch (ParseException e) {
            this.DebugLog("JSON Syntax error present in config file (" + e.getMessage() + ").");
            System.exit(1);
        }

        return null;
    }

    /**
     * Output a debug message to the console
     * @param message The message to output
     */
    public void DebugLog(String message) {
        System.out.println("Server: " + message);
    }
}

/**
* This will be used with a shutdown hook so that the server is shutdown correctly when the program is terminated
*/
class ShutdownProcess extends Thread {
    private final ServerSocket ss;
    private final Server s;

    public ShutdownProcess(ServerSocket ss, Server s) {
        this.ss = ss;
        this.s = s;
    }

    @Override
    public void run() {
        try {
            //this.s.DebugLog(this.s.authenticationHandlerQueueManager.queue.toString());
            this.s.DebugLog(this.s.connectionHandlerQueueManager.queue.toString());
            //this.s.DebugLog(this.s.requestHandlerQueueManager.queue.toString());

            //for (Handler h : this.s.authenticationHandlerPool) {
            //    System.out.print(h.busy + " ");
            //}
            //System.out.println();
            for (Handler h : this.s.connectionHandlerPool) {
                System.out.print(h.busy + " ");
            }
            System.out.println();
            //for (Handler h : this.s.requestHandlerPool) {
            //    System.out.print(h.busy + " ");
            //}

            this.ss.close();

        } catch (IOException e) {
            e.printStackTrace();
        }
```

```
    }
}
```

Server main method

```
    public static void main(String[] args) {
        // Create a new server
        Server server = new Server();
        // Create a new server socket
        ServerSocket serverSocket = null;

        // Attempt to open the server socket
        try {
            serverSocket = new ServerSocket(server.port);

        } catch (IOException e) {
            server.DebugLog("Failed to create server socket! (" + e.getMessage() + ")");
            System.exit(1);
        }

        assert serverSocket != null;

        // Add a shutdown hook so that the server socket is correctly closed when the program is terminated
        Runtime.getRuntime().addShutdownHook(new ShutdownProcess(serverSocket, server));

        server.DebugLog("Server start complete, ready to receive connections!");

        Socket clientSocket = null;
        while (true) {
            try {
                // Accept any incoming connections
                clientSocket = serverSocket.accept();
                server.DebugLog("Received connection: " + clientSocket.getInetAddress().toString());

                // Push the connection to the queue
                server.connectionHandlerQueueManager.queue.Push(new CloneableObject(clientSocket));

            } catch (IOException e) {
                server.DebugLog("An error occurred when accepting a connection: " + e.getMessage());
            }
        }
    }
```

Testing

Given that some of the functions of the server require certain data to be present in the server, it makes sense that we test them in the following in order:

- Add user
- Authentication
- Get user
- Send friend requests
- Get friend requests
- Decline friend requests
- Accept friend requests

- Get friendships
- Add chat
- Get chat
- Get public key
- Send chat invite
- Get chat invites
- Decline chat invites
- Accept chat invites
- Get message queue
- Send message

Add user		
Input	***Expected output***	***Successful?***

`"data": User {`
`UserID=-1,`
`Username='12345',`
`Password='Oogle',`
`DisplayName='Herman',`
`DateCreated='Tue Feb 21 13:33:28 GMT 2023',`
`ProfilePicturePath='default.png'`
`},`
`"type": AddUser`
	`User {`
	`UserID=1,`
	`Username='12345',`
	`Password='Oogle',`
	`DisplayName='Herman',`
	`DateCreated='Tue Feb 21 13:33:28 GMT 2023',`
	`ProfilePicturePath='default.png'`
		Yes

Authenticate		
Input	***Expected output***	***Successful?***

`"data": User {`
`UserID=-1,`
`Username='dwiejfef',`
`Password='adiwdjiwad',`
`DisplayName='null',`
`DateCreated='null',`
`ProfilePicturePath='null'`
`},`
`"type": Authenticate`
	failed	Yes

`"data": User {`
`UserID=-1,`
`Username='12345',`
`Password='adiwdjiwad',`
`DisplayName='null',`
`DateCreated='null',`
`ProfilePicturePath='null'`
	failed	Yes

}, "type": Authenticate		
"data": User { UserID=-1, Username='12345', Password='Oogle', DisplayName='null', DateCreated='null', ProfilePicturePath='null' }, "type": Authenticate	User { UserID=1, Username='12345', Password='Oogle', DisplayName='Herman', DateCreated='Tue Feb 21 13:33:28 GMT 2023', ProfilePicturePath='default.png'	Yes

Get user		
Input	***Expected output***	***Successful?***
"data": User { UserID=5, Username='null', Password='null', DisplayName='null', DateCreated='null', ProfilePicturePath='null' }, "selector": "id", "type": GetUser	null	Yes
"data": User { UserID=1, Username='null', Password='null', DisplayName='null', DateCreated='null', ProfilePicturePath='null' }, "selector": "id", "type": GetUser	User { UserID=1, Username='12345', Password='Oogle', DisplayName='Herman', DateCreated='Tue Feb 21 13:33:28 GMT 2023', ProfilePicturePath='default.png'	Yes
"data": User { UserID=-1, Username='iaoaidj', Password='null', DisplayName='null', DateCreated='null', ProfilePicturePath='null' }, "selector": "username", "type": GetUser	null	Yes
"data": User { UserID=-1,	User { UserID=1,	Yes

Username='12345', Password='null', DisplayName='null', DateCreated='null', ProfilePicturePath='null' }, "selector": "username", "type": GetUser	Username='12345', Password='Oogle', DisplayName='Herman', DateCreated='Tue Feb 21 13:33:28 GMT 2023', ProfilePicturePath='default.png'	
"data": User { UserID=-1, Username='null', Password='null', DisplayName='dkdowo', DateCreated='null', ProfilePicturePath='null' }, "selector": "displayName", "type": GetUser	Empty array	Yes
"data": User { UserID=-1, Username='null', Password='null', DisplayName='Herman', DateCreated='null', ProfilePicturePath='null' }, "selector": "displayName", "type": GetUser	[User { UserID=1, Username='12345', Password='Oogle', DisplayName='Herman', DateCreated='Tue Feb 21 13:33:28 GMT 2023', ProfilePicturePath='default.png']	Yes
"data": User { UserID=-1, Username='null', Password='null', DisplayName='Her', DateCreated='null', ProfilePicturePath='null' }, "selector": "displayName", "type": GetUser	[User { UserID=1, Username='12345', Password='Oogle', DisplayName='Herman', DateCreated='Tue Feb 21 13:33:28 GMT 2023', ProfilePicturePath='default.png']	Yes

Send friend request		
Input	***Expected output***	***Successful?***
"data":FriendRequest { FriendRequestID=-1, SenderID=1, RecipientID=2, TimeSent=1677152766279 }, "type":SendFriendRequest	Done	Yes

Get friend requests		
Input	***Expected output***	***Successful?***
"data":FriendRequest { FriendRequestID=-1, SenderID=2, RecipientID=-1, TimeSent=-1 }, "selector":"senderID", "type":GetFriendRequests	[]	Yes
"data":FriendRequest { FriendRequestID=-1, SenderID=1, RecipientID=-1, TimeSent=-1 }, "selector":"senderID", "type":GetFriendRequests	[FriendRequest { FriendRequestID=1, SenderID=1, RecipientID=2, TimeSent=1677152766279 }]	Yes
"data":FriendRequest { FriendRequestID=-1, SenderID=-1, RecipientID=1, TimeSent=-1 }, "selector":"recipientID", "type":GetFriendRequests	[]	Yes
"data":FriendRequest { FriendRequestID=-1, SenderID=-1, RecipientID=2, TimeSent=-1 }, "selector":"recipientID", "type":GetFriendRequests	[FriendRequest { FriendRequestID=1, SenderID=1, RecipientID=2, TimeSent=1677152766279 }]	Yes

Decline friend request		
Input	***Expected output***	***Successful?***
"data":FriendRequest { FriendRequestID=1, SenderID=-1,	Done	Yes

RecipientID=-1, TimeSent=-1 }, "type":DeclineFriendRequest		

Accept friend request		
Input	**Expected output**	**Successful?**
"data":FriendRequest { FriendRequestID=2, SenderID=-1, RecipientID=-1, TimeSent=-1 }, "type":AcceptFriendRequest	Done	Yes

Get friendships		
Input	**Expected output**	**Successful?**

"data": Friendship {
FriendshipID=3,
UserID=-1,
FriendID=-1,
DateEstablished=''
},
"selector":"id",
"type":GetFriendship
}
	Friendship {
	FriendshipID=3,
	UserID=1,
	FriendID=2,
	DateEstablished=''
	}
		Yes

"data": Friendship {
FriendshipID=4,
UserID=-1,
FriendID=-1,
DateEstablished=''
},
"selector":"id",
"type":GetFriendship
}
	Friendship {
	FriendshipID=4,
	UserID=2,
	FriendID=1,
	DateEstablished=''
	}
		Yes

Add chat		
Input	**Expected output**	**Successful?**
"data":Chat {	Chat {	Yes

ChatID=-1, Name='Test', Description='Chat desc', PublicKeyID=-1 }, "keyPair":<>, "type":AddChat	ChatID=1, Name='Test', Description='Chat desc', PublicKeyID=410716395 }	

Get chat		
Input	***Expected output***	***Successful?***

"data":Chat {
ChatID=1,
Name=null,
Description=null,
PublicKeyID=-1
},
"type":GetChat
	Chat {
	ChatID=1,
	Name='Test',
	Description='Chat desc',
	PublicKeyID=410716395
	}
		Yes

Get public key		
Input	***Expected output***	***Successful?***
410716395	Public key to match the one given when making the chat.	Yes

Send chat invite		
Input	***Expected output***	***Successful?***
"data": ChatInvite { ChatInviteID=-1, ChatID=1, SenderID=1, RecipientID=2, TimeSent=1677243137403, PrivateKeyID=-1 }, "keyPair":<>	Done	Yes

Get chat invite		
Input	***Expected output***	***Successful?***

"data": ChatInvite {
ChatInviteID=1,
ChatID=-1,
SenderID=-1,
RecipientID=-1,
TimeSent=-1,
PrivateKeyID=-1
},
"selector":"id"
"selector":GetChatInvite
	ChatInvite {
	ChatInviteID=1,
	ChatID=1,
	SenderID=1,
	RecipientID=2,
	TimeSent=1677243137403,
	PrivateKeyID=1076397193
	}
		Yes

"data": ChatInvite { ChatInviteID=-1, ChatID=-1, SenderID=1, RecipientID=-1, TimeSent=-1, PrivateKeyID=-1 }, "selector":"senderID" "selector":GetChatInvite	[ChatInvite { ChatInviteID=1, ChatID=1, SenderID=1, RecipientID=2, TimeSent=1677243137403, PrivateKeyID=1076397193 }]	Yes
"data": ChatInvite { ChatInviteID=-1, ChatID=-1, SenderID=-1, RecipientID=2, TimeSent=-1, PrivateKeyID=-1 }, "selector":"recipientID" "selector":GetChatInvite	[ChatInvite { ChatInviteID=1, ChatID=1, SenderID=1, RecipientID=2, TimeSent=1677243137403, PrivateKeyID=1076397193 }]	Yes
"data": ChatInvite { ChatInviteID=-1, ChatID=-1, SenderID=2, RecipientID=-1, TimeSent=-1, PrivateKeyID=-1 }, "selector":"senderID" "selector":GetChatInvite	[]	Yes
"data": ChatInvite { ChatInviteID=-1, ChatID=-1, SenderID=-1, RecipientID=1, TimeSent=-1, PrivateKeyID=-1 }, "selector":"recipientID" "selector":GetChatInvite	[]	Yes

Decline chat invites		
Input	***Expected output***	***Successful?***

`"data":ChatInvite {`
`ChatInviteID=1,`
`ChatID=-1,`
`SenderID=-1,`
`RecipientID=-1,`
`TimeSent=-1,`
`PrivateKeyID=-1`
`},`
`"type":DeclineChatInvite`
	Done	Yes

Accept chat invite		
Input	***Expected output***	***Successful?***

`"data":ChatInvite {`
`ChatInviteID=2,`
`ChatID=-1,`
`SenderID=-1,`
`RecipientID=-1,`
`TimeSent=-1,`
`PrivateKeyID=-1`
`},`
`"type":AcceptChatInvite`
	<Private key>	Yes

Get message queue		
Input	***Expected output***	***Successful?***

`input: {`
`"data":1,`
`"type":GetMessageQueue`
`}`
	`com.nathcat.messagecat_database.MessageQueue`
	
	We don't have a string representation for this class so this is all we can expect to be returned.
		Yes

Send message		
Input	***Expected output***	***Successful?***
`"data":Message {`	Done	Yes

`SenderID=1,`
`ChatID=1,`
`TimeSent=1677337187116,`
`Content='Hello world'`
`},`
`"type":SendMessage`
`}`		

Furthermore, we should be able to see this message in the message queue we requested earlier.

Implementing this test by requesting the message queue and outputting its contents results in the following output:

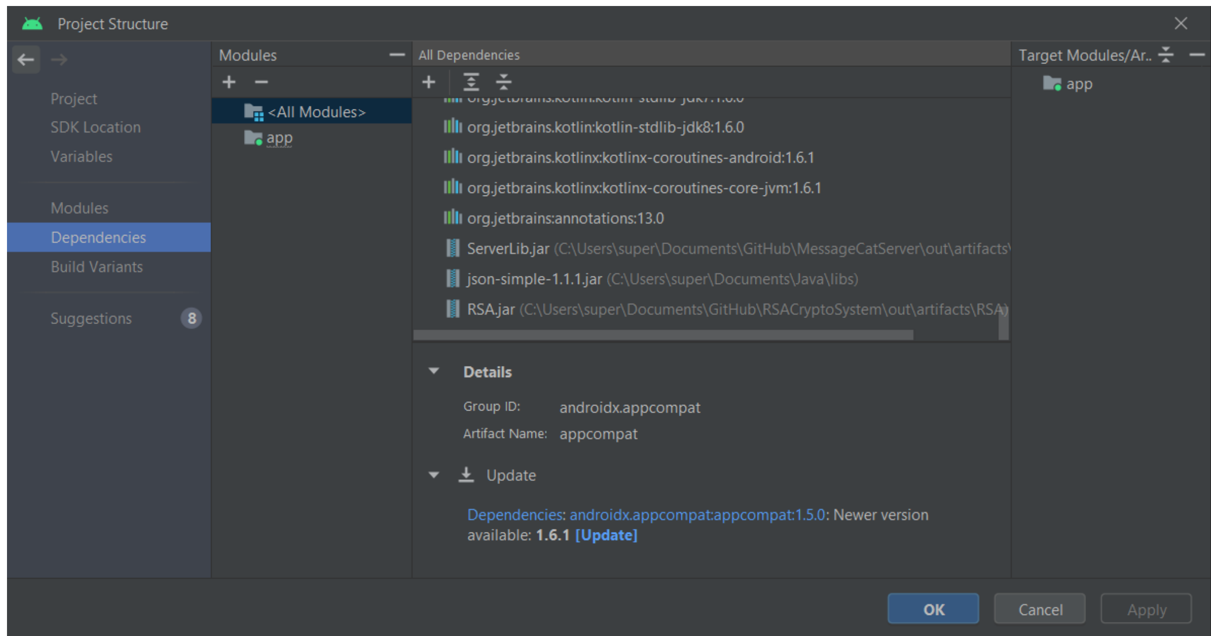
```
[{"TimeSent":1677337187116,"Content":"Hello world","SenderID":1,"ChatID":1}, null, null,
...]
```

This is the expected result and clearly shows that the message we sent with the previous request has been successfully stored on the server. In practice the *content* parameter would be an encrypted object containing the encrypted contents of the message, but for the purposes of this test a string reading *Hello world* is sufficient.

Client application

With the server and database complete we can continue with the implementation of the client application. This will be contained within a package called *com.nathcat.messagecat_client*.

Before we begin developing the application we should also include the *ServerLib.jar* library, primarily for access to the database entity classes.



As you can see, the *ServerLib.jar* library has been added to the list of dependencies for this project, and we may now proceed.

ConnectionHandler (client)

```
package com.nathcat.messagecat_client;

import android.content.Context;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.os.Message;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.List;

import com.nathcat.RSA.*;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_server.ListenRule;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

/**
* Manages a connection to the server
*/
public class ConnectionHandler extends Handler {
   public Socket s = null;                // The socket used to connect to the server
   public ObjectOutputStream oos = null;  // Object output stream
   public ObjectInputStream ois = null;   // Object input stream
   public KeyPair keyPair = null;         // The client's key pair
   public KeyPair serverKeyPair = null;   // The server's key pair
   public final Context context;          // The context this handler was created in
   public int connectionHandlerId;        // The identifier of the connection handler this handler has connected to
   public ListenRuleCallbackHandler callbackHandler;

   public class ListenRuleRecord {
       public final ListenRule listenRule;
       public final NetworkerService.IListenRuleCallback callback;
       public final Bundle bundle;

       private ListenRuleRecord(ListenRule listenRule, NetworkerService.IListenRuleCallback callback, Bundle bundle)
{
           this.listenRule = listenRule;
           this.callback = callback;
           this.bundle = bundle;
       }
   }

   public ArrayList<ListenRuleRecord> listenRules = new ArrayList<>();

   public ConnectionHandler(Context context, Looper looper) {
```

```java
    super(looper);  // Handler constructor

    this.context = context;
}

/**
 * Send an object to the server
 * @param obj The object to send
 * @throws IOException Thrown in case of I/O issues
 */
public void Send(Object obj) throws IOException {
    oos.writeObject(obj);
    oos.flush();
}

/**
 * Receive an object from the server
 * @return The object received
 * @throws IOException Thrown by I/O issues
 * @throws ClassNotFoundException Thrown if the required class cannot be found
 */
public Object Receive() throws IOException, ClassNotFoundException {
    return ois.readObject();
}

/**
 * Handles messages passed to the handler.
 *
 * @param msg The message passed to the handler
 */
@Override
public void handleMessage(Message msg) {
    // If the 'what' parameter of the message is 0, that indicates an initialisation request
    // This should only occur when the service is first started
    if (msg.what == 0) {
        try {
            // Try to connect to the server
            this.s = new Socket("13.40.226.47", 1234);
            this.s.setSoTimeout(20000);
            this.oos = new ObjectOutputStream(s.getOutputStream());
            this.ois = new ObjectInputStream(s.getInputStream());

            // Create a key pair and perform the handshake
            this.keyPair = RSA.GenerateRSAKeyPair();
            this.serverKeyPair = (KeyPair) this.Receive();

            this.Send(new KeyPair(this.keyPair.pub, null));

            this.connectionHandlerId = (int) this.keyPair.decrypt((EncryptedObject) this.Receive());
            System.out.println("Got handler id: " + connectionHandlerId);

            int port = (int) this.keyPair.decrypt((EncryptedObject) this.Receive());
            System.out.println("Got port: " + port);

            callbackHandler = new ListenRuleCallbackHandler(this, keyPair, serverKeyPair, port);
            callbackHandler.setDaemon(true);
            callbackHandler.start();
```

```
            } catch (IOException | NoSuchAlgorithmException | ClassNotFoundException | PrivateKeyException e) {
                e.printStackTrace();
            }

            assert this.s != null && this.oos != null && this.ois != null && this.keyPair != null &&
this.serverKeyPair != null;
            return;
        }
        else if (msg.what == 2) {
            try {
                this.oos.close();
                this.ois.close();
                this.s.close();
            } catch (IOException e) {
                e.printStackTrace();
            }

            this.getLooper().quit();
            return;
        }

        // This point will only be reached if the 'what' parameter of the message is not 0
        // Get the request object from the message
        NetworkerService.Request request = (NetworkerService.Request) msg.obj;

        if (((JSONObject) request.request).get("type") == RequestType.AddListenRule) {
            NetworkerService.ListenRuleRequest lrRequest = (NetworkerService.ListenRuleRequest) request;

            try {
                ListenRule rule = (ListenRule) ((JSONObject) lrRequest.request).get("data");
                ((JSONObject) lrRequest.request).put("data", rule);

                // Send the request
                this.Send(this.serverKeyPair.encrypt(lrRequest.request));
                // Receive the ID
                int id = (int) this.keyPair.decrypt((EncryptedObject) this.Receive());
                rule = (ListenRule) ((JSONObject) lrRequest.request).get("data");
                rule.setId(id);
                // Add the listen rule to the array along with it's callback
                this.listenRules.add(new ListenRuleRecord(rule, lrRequest.lrCallback, lrRequest.bundle));
                lrRequest.callback.callback(Result.SUCCESS, id);

            } catch (IOException | PublicKeyException | ClassNotFoundException | PrivateKeyException |
ListenRule.IDAlreadySetException e) {
                e.printStackTrace();
                lrRequest.callback.callback(Result.FAILED, null);
            }
        }
        else if (((JSONObject) request.request).get("type") == RequestType.RemoveListenRule) {
            NetworkerService.ListenRuleRequest lrRequest = (NetworkerService.ListenRuleRequest) request;

            try {
                // Remove the listen rule from the array
                for (int i = 0; i < this.listenRules.size(); i++) {
                    if (this.listenRules.get(i).listenRule.getId() == (int) ((JSONObject)
lrRequest.request).get("data")) {
```

```
                        this.listenRules.remove(i);
                        break;
                    }
                }
                // Send the request
                this.Send(this.serverKeyPair.encrypt(lrRequest.request));
                lrRequest.callback.callback(Result.SUCCESS, this.keyPair.decrypt((EncryptedObject) this.Receive()));

            } catch (IOException | PublicKeyException | ClassNotFoundException | PrivateKeyException e) {
                e.printStackTrace();
                lrRequest.callback.callback(Result.FAILED, null);
            }
        } else {

            try {
                // Send the request
                this.Send(this.serverKeyPair.encrypt(request.request));
                // Receive the response and perform the callback from the request object
                request.callback.callback(Result.SUCCESS, this.keyPair.decrypt((EncryptedObject) this.Receive()));

            } catch (IOException | PublicKeyException | ClassNotFoundException | PrivateKeyException e) {
                e.printStackTrace();
                // Perform callback with failure code
                request.callback.callback(Result.FAILED, null);
            }
        }
    }
}
```

NetworkerService

```
package com.nathcat.messagecat_client;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.Service;
import android.content.Intent;
import android.os.Binder;
import android.os.Bundle;
import android.os.HandlerThread;
import android.os.IBinder;
import android.os.Looper;
import android.os.Message;
import android.widget.Toast;

import androidx.annotation.Nullable;
import androidx.core.app.NotificationCompat;
import androidx.core.app.NotificationManagerCompat;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

;
import com.nathcat.messagecat_database.MessageQueue;
import com.nathcat.messagecat_database_entities.Chat;
```

```java
import com.nathcat.messagecat_database_entities.ChatInvite;
import com.nathcat.messagecat_database_entities.FriendRequest;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.ListenRule;
import com.nathcat.messagecat_server.RequestType;
import com.nathcat.messagecat_database.Result;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Date;

/**
* Background service used to handle networking tasks.
*
* @author Nathan "Nathcat" Baines
*/
public class NetworkerService extends Service implements Serializable {
   /**
    * Used to manage notifications.
    * All notifications will be sent through this class
    */
   public class NotificationChannel {
       private final String channelName;         // Name of the notification channel
       private final String channelDescription;  // Description of the notification channel

       public NotificationChannel(String channelName, String channelDescription, int importance) {
           this.channelName = channelName;
           this.channelDescription = channelDescription;

           // Create the notification channel
           android.app.NotificationChannel channel = new android.app.NotificationChannel(this.channelName,
this.channelName, importance);
           channel.setDescription(this.channelDescription);
           NotificationManager notificationManager = getSystemService(NotificationManager.class);
           notificationManager.createNotificationChannel(channel);
       }

       /**
        * Create and display a notification from a given title and message.
        *
        * @param title The title of the notification
        * @param message The message of the notification
        */
       public void showNotification(String title, String message) {
           NotificationCompat.Builder builder = new NotificationCompat.Builder(NetworkerService.this,
this.channelName)
                   .setSmallIcon(R.drawable.cat_notification)
                   .setContentTitle(title)
                   .setContentText(message)
                   .setPriority(Notification.PRIORITY_MAX)
                   .setDefaults(Notification.DEFAULT_SOUND | Notification.DEFAULT_VIBRATE);
```

```
            NotificationManagerCompat manager = NotificationManagerCompat.from(NetworkerService.this);
            manager.notify(0, builder.build());
        }
    }

    /**
     * Passed to the active ConnectionHandler, contains request object and a callback
     */
    public static class Request {
        public final IRequestCallback callback;  // The callback to be performed when a response is received
        public final Object request;              // The request object
        public final Bundle bundle;              // External data that may be needed in the callback

        public Request(IRequestCallback callback, Object request) {
            this.callback = callback;
            this.request = request;
            this.bundle = null;
        }

        public Request(IRequestCallback callback, Object request, Bundle bundle) {
            this.callback = callback;
            this.request = request;
            this.bundle = bundle;
        }
    }

    /**
     * Request callback interface.
     * Implement this and pass to the Request object to create a callback
     */
    public interface IRequestCallback {
        default void callback(Result result, Object response) {}
    }

    /**
     * Passed to the active ListenRuleHandler to create a new listen rule on the server
     */
    public static class ListenRuleRequest extends Request {
        public final IListenRuleCallback lrCallback;      // Called when the listen rule is triggered

        public ListenRuleRequest(IListenRuleCallback lrCallback, IRequestCallback callback, Object request) {
            super(callback, request);
            this.lrCallback = lrCallback;
        }

        public ListenRuleRequest(IListenRuleCallback lrCallback, IRequestCallback callback, Object request, Bundle
bundle) {
            super(callback, request, bundle);
            this.lrCallback = lrCallback;
        }
    }

    /**
     * Listen rule callback interface, called when a listen rule is triggered
     */
    public interface IListenRuleCallback {
        default void callback(Object response) {}
```

```
        default void callback(Object response, Bundle bundle) {}
    }

    /**
     * Used by the main application activity (UI thread) to get the active instance of this service
     */
    public class NetworkerServiceBinder extends Binder {
        /**
         * Get an instance of the active service
         * @return The instance of the active service
         */
        NetworkerService getService() {
            return NetworkerService.this;
        }
    }

    public NotificationChannel notificationChannel;   // The notification channel to be used to send notifications
    public NotificationChannel serviceStatusChannel;  // The notification channel used to show service notifications
    private ConnectionHandler connectionHandler;      // The active connection handler
    private Looper connectionHandlerLooper;           // The Handler looper attached to the active connection handler
    public boolean authenticated = false;             // Is the client currently authenticated
    public boolean waitingForResponse = false;        // Is the client currently waiting for a response
    private final NetworkerServiceBinder binder = new NetworkerServiceBinder();
    private boolean bound = false;                    // Is the service currently bound to the UI thread
    public User user = null;                          // The user that is currently authenticated
    public int activeChatID = -1;                     // The id of the chat that is currently being viewed, or -1 if
none are being viewed

    /**
     * Returns a binder
     * @param intent The intent to bind to
     * @return The binder for this service
     */
    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        bound = true;
        return this.binder;
    }

    /**
     * Unbind from another process
     * @param intent The intent that just unbound from this service
     * @return false, indicating to Android that the service should not try to rebind
     */
    @Override
    public boolean onUnbind(Intent intent) {
        bound = false;
        return false;
    }

    /**
     * Called when the service is created
     */
    @Override
    public void onCreate() {
        // Create the notification channel
```

```java
        notificationChannel = new NotificationChannel(
                "MessageCat",
                "Notification channel used to send notifications to the user about things that happen in the app.",
                NotificationManager.IMPORTANCE_HIGH
        );

        serviceStatusChannel = new NotificationChannel(
                "MessageCat Service",
                "Used to notify the user that the MessageCat service is running",
                NotificationManager.IMPORTANCE_NONE
        );

        startForeground(1, new Notification.Builder(this, serviceStatusChannel.channelName)
                .setSmallIcon(R.drawable.cat_notification)
                .setContentTitle("MessageCat")
                .setContentText("MessageCat service is running")
                .setPriority(Notification.PRIORITY_LOW)
                .build());

        startConnectionHandler();
    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return START_STICKY;
    }

    /**
     * Send a request to the server via the connection handler
     * @param request The request object
     */
    public void SendRequest(Request request) {
        this.waitingForResponse = true;  // Indicate that the client is waiting for a response
        // Create the message to the connection handler
        Message msg = connectionHandler.obtainMessage();
        msg.obj = request;
        msg.what = 1;

        // Send the request to the connection handler to be sent off to the server
        connectionHandler.sendMessage(msg);
    }

    @Override
    public void onTaskRemoved(Intent intent) {
        System.out.println("Service task removed");
    }

    @Override
    public void onDestroy() {
        System.out.println("Service destroyed");
        connectionHandler.sendEmptyMessage(2);
        super.onDestroy();
    }

    public void startConnectionHandler() {
        // Create the connection handler thread
        HandlerThread thread = new HandlerThread("MessageCatNetworkingHandlerThread", 10);
```

```
        thread.start();

        this.connectionHandlerLooper = thread.getLooper();
        this.connectionHandler = new ConnectionHandler(this, this.connectionHandlerLooper);

        // Initialise the connection to the server
        connectionHandler.sendEmptyMessage(0);

        // Try and find auth data
        File authDataFile = new File(getFilesDir(), "UserData.bin");
        if (authDataFile.exists()) {
            // Try to use the data in the auth file to authenticate the client
            try {
                ObjectInputStream authDataInputStream = new ObjectInputStream(new FileInputStream(authDataFile));
                Object userData = authDataInputStream.readObject();
                authDataInputStream.close();

                // Create the request and send it to the server
                JSONObject requestData = new JSONObject();
                requestData.put("type", RequestType.Authenticate);
                requestData.put("data", userData);

                // Send the authentication request
                this.SendRequest(new Request(new IRequestCallback() {
                    @Override
                    public void callback(Result result, Object response) {
                        if (result == Result.FAILED) {
                            startConnectionHandler();
                            return;
                        }

                        if (response.getClass() == String.class) {
                            authenticated = false;
                            waitingForResponse = false;
                            Toast.makeText(NetworkerService.this, "Finished auth", Toast.LENGTH_SHORT).show();
                            return;
                        }

                        // If authentication was successful...
                        if (result == Result.SUCCESS) {
                            authenticated = true;

                            // Update the data in the auth file
                            try {
                                ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new
File(getFilesDir(), "UserData.bin")));
                                oos.writeObject(response);
                                oos.flush();
                                oos.close();

                                user = (User) response;

                            } catch (IOException e) {
                                e.printStackTrace();
                            }

                            // Add the notification listen rules
```

```
                    JSONObject friendRequestRuleRequest = new JSONObject();
                    friendRequestRuleRequest.put("type", RequestType.AddListenRule);
                    friendRequestRuleRequest.put("data", new ListenRule(RequestType.SendFriendRequest,
"RecipientID", user.UserID));
                    SendRequest(new ListenRuleRequest(new IListenRuleCallback() {
                        @Override
                        public void callback(Object response) {
                            // Get the friend request from the response object
                            FriendRequest friendRequest = (FriendRequest) ((JSONObject) response).get("data");

                            // Request the user that sent the request from the server
                            JSONObject senderRequest = new JSONObject();
                            senderRequest.put("type", RequestType.GetUser);
                            senderRequest.put("selector", "id");
                            senderRequest.put("data", new User(friendRequest.SenderID, null, null, null, null,
null));

                            SendRequest(new Request(new IRequestCallback() {
                                @Override
                                public void callback(Result result, Object response) {
                                    if (result == Result.FAILED) {
                                        return;
                                    }

                                    // Cast the response into a user object
                                    User sender = (User) response;
                                    // Show the notification
                                    notificationChannel.showNotification("New friend request",
sender.DisplayName + " wants to be friends!");
                                }
                            }, senderRequest));
                        }
                    }, new IRequestCallback() {
                        @Override
                        public void callback(Result result, Object response) {
                            IRequestCallback.super.callback(result, response);
                        }
                    }, friendRequestRuleRequest));

                    JSONObject chatRequestRuleRequest = new JSONObject();
                    chatRequestRuleRequest.put("type", RequestType.AddListenRule);
                    chatRequestRuleRequest.put("data", new ListenRule(RequestType.SendChatInvite,
"RecipientID", user.UserID));
                    SendRequest(new ListenRuleRequest(new IListenRuleCallback() {
                        @Override
                        public void callback(Object response) {
                            // Get the chat invite from the request that triggered the listen rule
                            ChatInvite chatInvite = (ChatInvite) ((JSONObject) response).get("data");

                            // Create a new request to get the chat that the user has been invited to
                            JSONObject getChatRequest = new JSONObject();
                            getChatRequest.put("type", RequestType.GetChat);
                            getChatRequest.put("data", new Chat(chatInvite.ChatID, null, null, -1));

                            SendRequest(new Request(new IRequestCallback() {
                                @Override
                                public void callback(Result result, Object response) {
                                    if (result == Result.FAILED) {
```

```
                                                return;
                                            }

                                            // Cast the response to a Chat object
                                            Chat chat = (Chat) response;
                                            // Show the notification
                                            notificationChannel.showNotification("New chat invitation", "You have been
invited to " + chat.Name);
                                        }
                                    }, getChatRequest));
                                }
                            }, new IRequestCallback() {
                                @Override
                                public void callback(Result result, Object response) {
                                    IRequestCallback.super.callback(result, response);
                                }
                            }, chatRequestRuleRequest));

                            File chatsFile = new File(getFilesDir(), "Chats.bin");
                            if (chatsFile.exists()) {
                                try {
                                    // Get the array of chats
                                    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(chatsFile));
                                    Chat[] chats = (Chat[]) ois.readObject();

                                    // Create a listen rule for each of the chats
                                    for (Chat chat : chats) {
                                        JSONObject msgRule = new JSONObject();
                                        msgRule.put("type", RequestType.AddListenRule);
                                        msgRule.put("data", new ListenRule(RequestType.SendMessage, "ChatID",
chat.ChatID));

                                        Bundle bundle = new Bundle();
                                        bundle.putSerializable("chat", chat);

                                        SendRequest(new ListenRuleRequest(new IListenRuleCallback() {
                                            @Override
                                            public void callback(Object response, Bundle bundle) {
                                                if (activeChatID != ((Chat) bundle.getSerializable("chat")).ChatID) {
                                                    // Create a notification
                                                    notificationChannel.showNotification("New message", "You have a
new message in " + ((Chat) bundle.getSerializable("chat")).Name);
                                                }
                                            }
                                        }, new IRequestCallback() {
                                            @Override
                                            public void callback(Result result, Object response) {
                                                IRequestCallback.super.callback(result, response);
                                            }
                                        }, msgRule, bundle));
                                    }

                                } catch (IOException | ClassNotFoundException e) {
                                    e.printStackTrace();
                                }

                            }
                        }
```

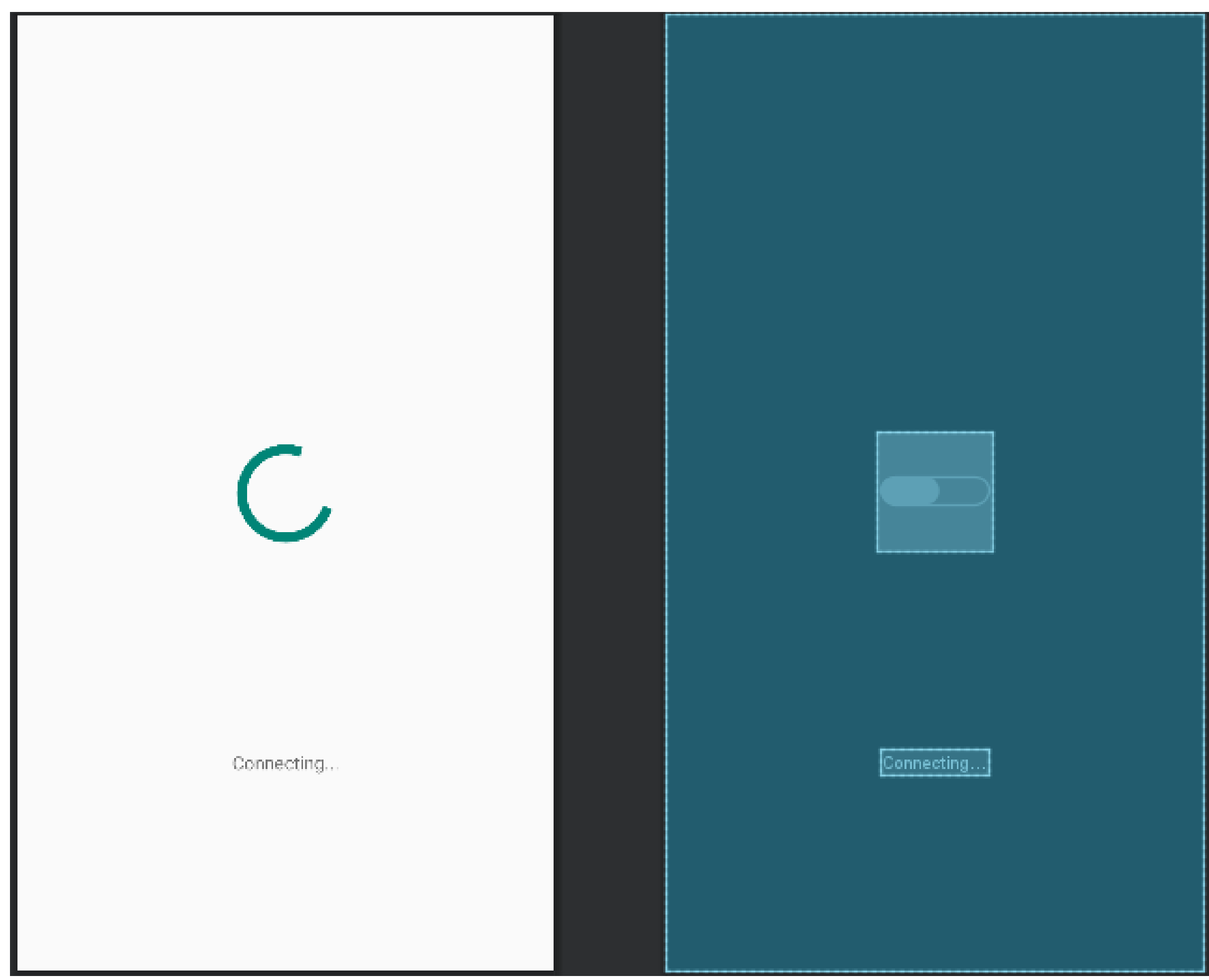
```
                        waitingForResponse = false;
                    }
                }, requestData));

            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
            }
        }
    }
}
```

AutoStartService

```
package com.nathcat.messagecat_client;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class AutoStartService extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
       context.startForegroundService(new Intent(context, NetworkerService.class));
       //context.startService(new Intent(context, NetworkerService.class));
   }
}
```

LoadingActivity

This is the first part of the actual GUI which I will develop. The aim of this part of the application is to check that everything is ready for the application to run, and attempts to rectify any missing conditions, for example it will start the Networker service if it isn't already running, and will authenticate the connection to the server if it isn't already authenticated. Here is the design and blueprint rendered by Android studio:



And the code:

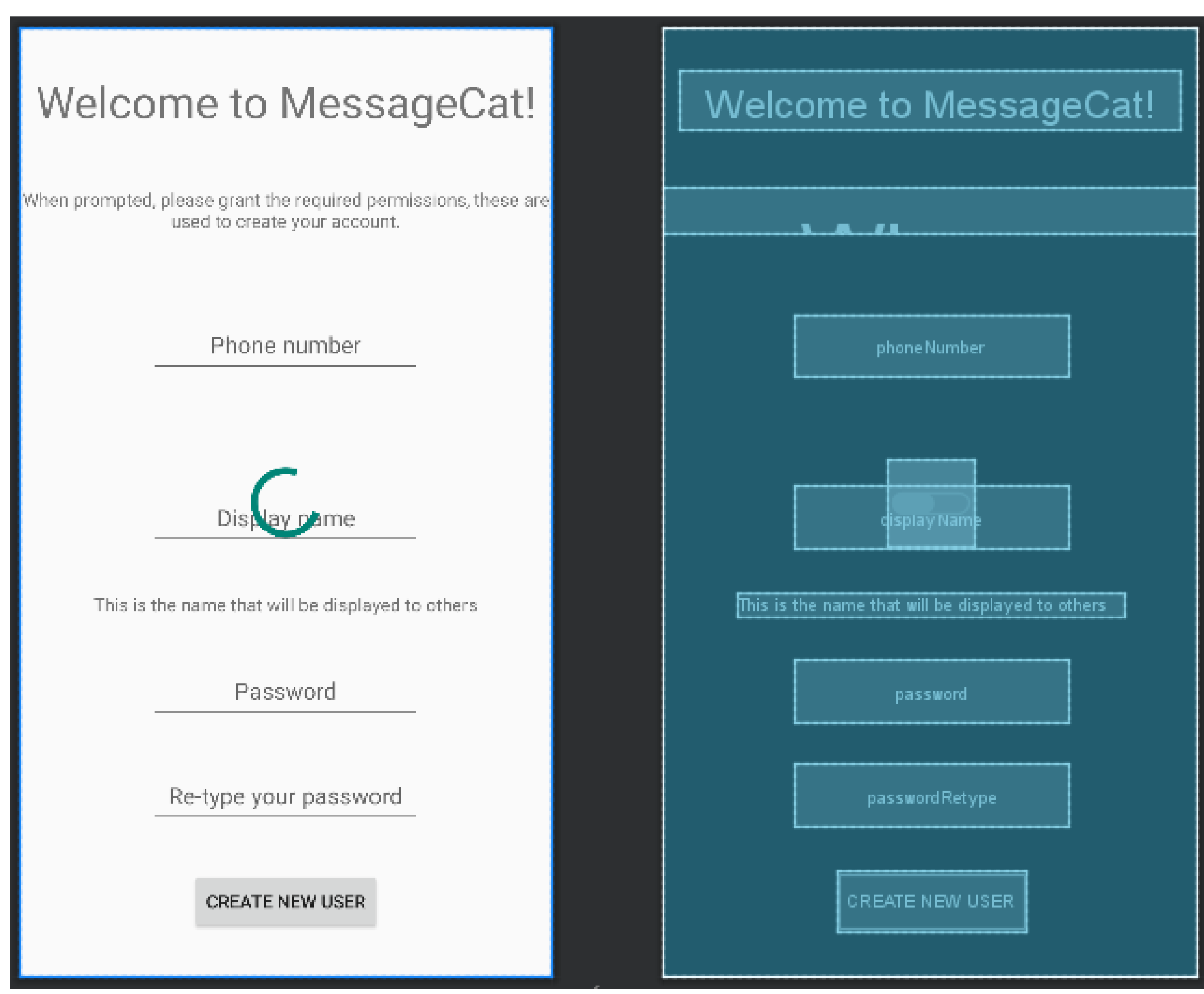
```
package com.nathcat.messagecat_client;

import androidx.appcompat.app.AppCompatActivity;

import android.app.Activity;
import android.app.ActivityManager;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.widget.Toast;

public class LoadingActivity extends AppCompatActivity {
   /**
    * Thread to wait for authentication to complete, if it is not already complete
    */
   private class WaitForAuthThread extends Thread {
       @Override
       public void run() {
           while (networkerService.waitingForResponse) {
               try {
```

```
                Thread.sleep(100);

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

        if (networkerService.authenticated) {
            startActivity(new Intent(LoadingActivity.this, MainActivity.class));
        }
        else {
            startActivity(new Intent(LoadingActivity.this, NewUserActivity.class));
        }
    }
}

// This is used to connect to the networker service
private ServiceConnection networkerServiceConnection = new ServiceConnection() {

    /**
     * Called when the service connects to this process
     * @param componentName Name for an application component
     * @param iBinder The binder returned by the service
     */
    @Override
    public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
        // Get the service instance
        networkerService = ((NetworkerService.NetworkerServiceBinder) iBinder).getService();
        bound = true;

        // Wait for authentication to complete
        new WaitForAuthThread().start();
    }

    /**
     * Called when the service disconnects from this process
     * @param componentName Name for an application component
     */
    @Override
    public void onServiceDisconnected(ComponentName componentName) {
        networkerService = null;
        bound = false;
    }
};

private NetworkerService networkerService = null;  // The instance of the networker service
private boolean bound = false;                     // Is the service currently bound to this process

@Override
protected void onCreate(Bundle savedInstanceState) {
    // This method is called when the application is opened
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_loading);

    // Check if the networker service is not running
    if (!isServiceRunning(NetworkerService.class)) {
        // Start the foreground service
```

```
            startForegroundService(new Intent(this, NetworkerService.class));
        }

        // Try to bind to the networker service
        bindService(
                new Intent(this, NetworkerService.class),  // The intent to bind with
                networkerServiceConnection,                          // The ServiceConnection object to use
                Context.BIND_AUTO_CREATE                             // If the service does not already exist,
create it
        );
    }

    /**
     * Check if a service is running
     * @param serviceClass The class of the service
     * @return boolean
     */
    private boolean isServiceRunning(Class<?> serviceClass) {
        ActivityManager manager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVICE);
        for (ActivityManager.RunningServiceInfo service : manager.getRunningServices(Integer.MAX_VALUE)) {
            if (serviceClass.getName().equals(service.service.getClassName())) {
                return true;
            }
        }
        return false;
    }
}
```

NewUserActivity

This part of the application will allow users to create a new account, which they can then login to the service with. Here is the design and blueprint, followed by the code:




```
package com.nathcat.messagecat_client;

import static android.Manifest.permission.READ_PHONE_NUMBERS;
import static android.Manifest.permission.READ_PHONE_STATE;
import static android.Manifest.permission.READ_SMS;

import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.IBinder;
import android.telephony.TelephonyManager;
import android.view.View;
import android.view.WindowManager;
import android.widget.EditText;
import android.widget.ProgressBar;
import android.widget.Toast;

import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.Chat;
import com.nathcat.messagecat_database_entities.ChatInvite;
import com.nathcat.messagecat_database_entities.FriendRequest;
```

```java
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.ListenRule;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.io.File;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Date;

public class NewUserActivity extends AppCompatActivity {

    private ServiceConnection networkerServiceConnection = new ServiceConnection() {
        @Override
        public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
            networkerService = ((NetworkerService.NetworkerServiceBinder) iBinder).getService();
            bound = true;
        }

        @Override
        public void onServiceDisconnected(ComponentName componentName) {
            networkerService = null;
            bound = false;
        }
    };

    private EditText displayNameEntry;
    private EditText phoneNumberEntry;
    private ProgressBar loadingWheel;

    private String phoneNumber;
    private EditText passwordEntry;
    private EditText passwordRetypeEntry;

    private final MessageDigest digest = MessageDigest.getInstance("SHA-256");
    private NetworkerService networkerService;
    private boolean bound = false;

    public NewUserActivity() throws NoSuchAlgorithmException {
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_new_user);

        loadingWheel = (ProgressBar) findViewById(R.id.newUserLoadingWheel);
        loadingWheel.setVisibility(View.GONE);

        bindService(
                new Intent(this, NetworkerService.class),
                networkerServiceConnection,
                BIND_AUTO_CREATE
        );

        displayNameEntry = (EditText) findViewById(R.id.displayName);
```

```
        phoneNumberEntry = (EditText) findViewById(R.id.phoneNumber);

        // Get the user's phone number
        TelephonyManager telephonyManager = (TelephonyManager) this.getSystemService(Context.TELEPHONY_SERVICE);
        // Check if the required permissions are granted
        if (ActivityCompat.checkSelfPermission(this, READ_SMS) == PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, READ_PHONE_NUMBERS) == PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, READ_PHONE_STATE) == PackageManager.PERMISSION_GRANTED) {
            phoneNumber = telephonyManager.getLine1Number();
        }
        else {
            // Try and request those permissions and try again
            requestPermissions(new String[]{READ_SMS, READ_PHONE_NUMBERS, READ_PHONE_STATE}, 100);

            if (ActivityCompat.checkSelfPermission(this, READ_SMS) == PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, READ_PHONE_NUMBERS) == PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this, READ_PHONE_STATE) == PackageManager.PERMISSION_GRANTED) {
                phoneNumber = telephonyManager.getLine1Number();
            }
            else {
                System.exit(1);
            }
        }

        assert phoneNumber != null;

        passwordEntry = (EditText) findViewById(R.id.password);
        passwordRetypeEntry = (EditText) findViewById(R.id.passwordRetype);

        phoneNumberEntry.setText(phoneNumber);
    }

    public void onSubmitButtonClicked(View v) {
        // Check if any of the fields are empty, and check that the password entries match
        if (phoneNumberEntry.getText().toString().contentEquals("") ||
passwordEntry.getText().toString().contentEquals("") || passwordRetypeEntry.getText().toString().contentEquals("") ||
displayNameEntry.getText().toString().contentEquals("")) {
            Toast.makeText(this, "One or more of the entry fields are empty!", Toast.LENGTH_LONG).show();
            return;
        }

        if (!passwordEntry.getText().toString().contentEquals(passwordRetypeEntry.getText().toString())) {
            Toast.makeText(this, "Passwords do not match!", Toast.LENGTH_LONG).show();
            return;
        }

        // Hash the password
        String hashedPassword = bytesToHex(digest.digest(
                passwordEntry.getText().toString().getBytes(StandardCharsets.UTF_8)
        ));


        // Block user interaction
        getWindow().setFlags(WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE,
                WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE);

        // Show the loading wheel
        loadingWheel.setVisibility(View.VISIBLE);

        JSONObject request = new JSONObject();
        request.put("type", RequestType.AddUser);
```

```
        request.put("data", new User(-1, phoneNumberEntry.getText().toString(), hashedPassword,
displayNameEntry.getText().toString(), new Date().toString(), "default.png"));

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    networkerService.startConnectionHandler();
                    runOnUiThread(() -> onSubmitButtonClicked(v));
                }

                // Check if the response is null
                // If this is the case then the entry had duplicate data
                if (response == null) {
                    NewUserActivity.this.runOnUiThread(() -> {
                        Toast.makeText(NewUserActivity.this, "Either your username or display name is already used,
try something else.", Toast.LENGTH_SHORT).show();
                        loadingWheel.setVisibility(View.GONE);
                        getWindow().clearFlags(WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE);
                    });
                    return;
                }

                // Write the data to the auth file
                try {
                    ObjectOutputStream oos = new ObjectOutputStream(Files.newOutputStream(new File(getFilesDir(),
"UserData.bin").toPath()));
                    oos.writeObject(response);
                    oos.flush();
                    oos.close();

                    oos = new ObjectOutputStream(Files.newOutputStream(new File(getFilesDir(),
"Chats.bin").toPath()));
                    oos.writeObject(new Chat[0]);
                    oos.flush();
                    oos.close();

                    // Now start an authentication request and start up the loading screen
                    JSONObject authRequest = new JSONObject();
                    authRequest.put("type", RequestType.Authenticate);
                    authRequest.put("data", response);

                    // Send the auth request and start the loading activity
                    networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback()
{
                        @Override
                        public void callback(Result result, Object response) {
                            if (result == Result.FAILED) {
                                System.exit(1);
                            }

                            if (response.getClass() == String.class) {
                                networkerService.authenticated = false;
                                networkerService.waitingForResponse = false;
                                System.exit(1);
                            }

                            // If authentication was successful...
                            if (result == Result.SUCCESS) {
                                networkerService.authenticated = true;

                                // Update the data in the auth file
```

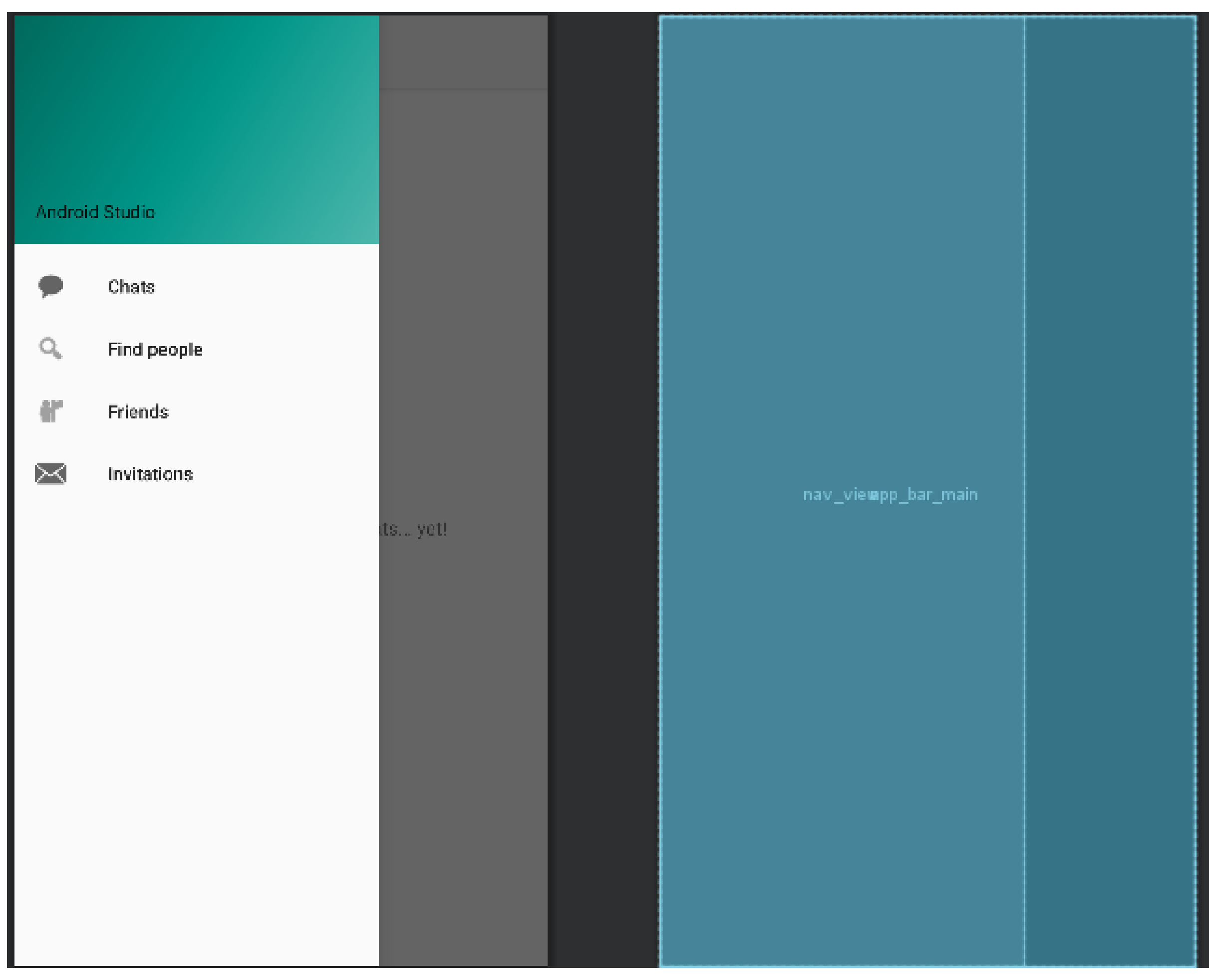
```
                                try {
                                    ObjectOutputStream oos = new ObjectOutputStream(Files.newOutputStream(new
File(getFilesDir(), "UserData.bin").toPath()));
                                    oos.writeObject(response);
                                    oos.flush();
                                    oos.close();

                                    networkerService.user = (User) response;

                                } catch (IOException e) {
                                    e.printStackTrace();
                                }

                                // Add the notification listen rules
                                JSONObject friendRequestRuleRequest = new JSONObject();
                                friendRequestRuleRequest.put("type", RequestType.AddListenRule);
                                friendRequestRuleRequest.put("data", new ListenRule(RequestType.SendFriendRequest,
"RecipientID", networkerService.user.UserID));
                                networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new
NetworkerService.IListenRuleCallback() {
                                    @Override
                                    public void callback(Object response) {
                                        // Get the friend request from the response object
                                        FriendRequest friendRequest = (FriendRequest) ((JSONObject)
response).get("data");

                                        // Request the user that sent the request from the server
                                        JSONObject senderRequest = new JSONObject();
                                        senderRequest.put("type", RequestType.GetUser);
                                        senderRequest.put("data", new User(friendRequest.SenderID, null, null, null,
null, null));
                                        networkerService.SendRequest(new NetworkerService.Request(new
NetworkerService.IRequestCallback() {
                                            @Override
                                            public void callback(Result result, Object response) {
                                                if (result == Result.FAILED) {
                                                    return;
                                                }

                                                // Cast the response into a user object
                                                User sender = (User) response;
                                                // Show the notification
                                                networkerService.notificationChannel.showNotification("New friend
request", sender.DisplayName + " wants to be friends!");
                                            }
                                        }, senderRequest));
                                    }
                                }, new NetworkerService.IRequestCallback() {
                                    @Override
                                    public void callback(Result result, Object response) {
                                        NetworkerService.IRequestCallback.super.callback(result, response);
                                    }
                                }, friendRequestRuleRequest));

                                JSONObject chatRequestRuleRequest = new JSONObject();
                                chatRequestRuleRequest.put("type", RequestType.AddListenRule);
                                chatRequestRuleRequest.put("data", new ListenRule(RequestType.SendChatInvite,
"RecipientID", networkerService.user.UserID));
                                networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new
NetworkerService.IListenRuleCallback() {
                                    @Override
                                    public void callback(Object response) {
```

```
                                        // Get the chat invite from the request that triggered the listen rule
                                        ChatInvite chatInvite = (ChatInvite) ((JSONObject) response).get("data");

                                        // Create a new request to get the chat that the user has been invited to
                                        JSONObject getChatRequest = new JSONObject();
                                        getChatRequest.put("type", RequestType.GetChat);
                                        getChatRequest.put("data", new Chat(chatInvite.ChatID, null, null, -1));

                                        networkerService.SendRequest(new NetworkerService.Request(new
NetworkerService.IRequestCallback() {
                                            @Override
                                            public void callback(Result result, Object response) {
                                                if (result == Result.FAILED) {
                                                    return;
                                                }

                                                // Cast the response to a Chat object
                                                Chat chat = (Chat) response;
                                                // Show the notification
                                                networkerService.notificationChannel.showNotification("New chat
invitation", "You have been invited to " + chat.Name);
                                            }
                                        }, getChatRequest));
                                    }
                                }, new NetworkerService.IRequestCallback() {
                                    @Override
                                    public void callback(Result result, Object response) {
                                        NetworkerService.IRequestCallback.super.callback(result, response);
                                    }
                                }, chatRequestRuleRequest));

                                File chatsFile = new File(getFilesDir(), "Chats.bin");
                                if (chatsFile.exists()) {
                                    try {
                                        // Get the array of chats
                                        ObjectInputStream ois = new
ObjectInputStream(Files.newInputStream(chatsFile.toPath()));
                                        Chat[] chats = (Chat[]) ois.readObject();

                                        // Create a listen rule for each of the chats
                                        for (Chat chat : chats) {
                                            JSONObject msgRule = new JSONObject();
                                            msgRule.put("type", RequestType.AddListenRule);
                                            msgRule.put("data", new ListenRule(RequestType.SendMessage, "ChatID",
chat.ChatID));
                                            Bundle bundle = new Bundle();
                                            bundle.putSerializable("chat", chat);

                                            networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new
NetworkerService.IListenRuleCallback() {
                                                @Override
                                                public void callback(Object response, Bundle bundle) {
                                                    // Create a notification
                                                    networkerService.notificationChannel.showNotification("New
message", "You have a new message in " + ((Chat) bundle.getSerializable("chat")).Name);
                                                }
                                            }, new NetworkerService.IRequestCallback() {
                                                @Override
                                                public void callback(Result result, Object response) {
                                                    NetworkerService.IRequestCallback.super.callback(result,
response);
                                                }
```

```
                                            }, msgRule, bundle));
                                        }

                                    } catch (IOException | ClassNotFoundException e) {
                                        e.printStackTrace();
                                    }

                                }
                            }

                            networkerService.waitingForResponse = false;
                        }
                    }, authRequest));

                    NewUserActivity.this.runOnUiThread(() ->
getWindow().clearFlags(WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE));

                    startActivity(new Intent(NewUserActivity.this, LoadingActivity.class));

                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        }, request));
    }

    /**
     * Converts a byte array to a hex string. Used for hashing passwords.
     * @param bytes The byte array to convert.
     * @return A hex string
     */
    public String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder(2 * bytes.length);  // Twice the length since we have two hex characters
per byte
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hex = "0" + hex;
            }

            sb.append(hex);
        }

        return sb.toString();
    }
}
```

MainActivity

This is the main page of the application, it contains a central view which changes depending on what the user is doing, and a drawer layout which allows them to change this central view to different pages.




```
package com.nathcat.messagecat_client;

import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.MenuItem;
import android.view.View;
import android.view.Menu;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

import com.google.android.material.snackbar.Snackbar;
import com.google.android.material.navigation.NavigationView;

import androidx.annotation.NonNull;
import androidx.fragment.app.FragmentContainerView;
import androidx.navigation.NavController;
import androidx.navigation.Navigation;
import androidx.navigation.ui.AppBarConfiguration;
import androidx.navigation.ui.NavigationUI;
import androidx.drawerlayout.widget.DrawerLayout;
import androidx.appcompat.app.AppCompatActivity;
```

```java
import com.nathcat.RSA.EncryptedObject;
import com.nathcat.RSA.KeyPair;
;
import com.nathcat.RSA.PublicKeyException;
import com.nathcat.messagecat_client.databinding.ActivityMainBinding;
import com.nathcat.messagecat_database.KeyStore;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.Chat;
import com.nathcat.messagecat_database_entities.ChatInvite;
import com.nathcat.messagecat_database_entities.FriendRequest;
import com.nathcat.messagecat_database_entities.Friendship;
import com.nathcat.messagecat_database_entities.Message;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.ListenRule;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.Date;
import java.util.HashMap;
import java.util.Random;

public class MainActivity extends AppCompatActivity {

   private ServiceConnection connection = new ServiceConnection() {
       @Override
       public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
           // Get the networker service instance
           networkerService = ((NetworkerService.NetworkerServiceBinder) iBinder).getService();

           // Set the display name in the nav header to the user's display name
           ((TextView) ((NavigationView) findViewById(R.id.nav_view))
                   .getHeaderView(0).findViewById(R.id.displayName))
                   .setText(networkerService.user.DisplayName);
       }

       @Override
       public void onServiceDisconnected(ComponentName componentName) {
           networkerService = null;
       }
   };

   public NetworkerService networkerService;
   private AppBarConfiguration mAppBarConfiguration;
   private ActivityMainBinding binding;

   // Used on the "find people page"
   private User[] searchResults;
```

```
// Used on the "friends" page
public User[] friends;

// Used on the invites page
public InvitationsFragment invitationsFragment;
public ArrayList<InvitationFragment> invitationFragments = new ArrayList<>();

// Used on the messaging page
public HashMap<Integer, User> users = new HashMap<>();
public MessagingFragment messagingFragment;

// Used on the chats page
public ArrayList<ChatFragment> chatFragments = new ArrayList<>();

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    // Bind to the networker service
    bindService(
            new Intent(this, NetworkerService.class),
            connection,
            BIND_AUTO_CREATE
    );

    // Set up the drawer and action bar (the bar at the top of the application)
    binding = ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(binding.getRoot());

    setSupportActionBar(binding.appBarMain.toolbar);

    DrawerLayout drawer = binding.drawerLayout;
    NavigationView navigationView = binding.navView;
    // Passing each menu ID as a set of Ids because each
    // menu should be considered as top level destinations.
    mAppBarConfiguration = new AppBarConfiguration.Builder(
            R.id.chatsFragment, R.id.findPeopleFragment)
            .setOpenableLayout(drawer)
            .build();
    NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main);
    NavigationUI.setupActionBarWithNavController(this, navController, mAppBarConfiguration);
    NavigationUI.setupWithNavController(navigationView, navController);

    // Set a click listener so that the navigation drawer menu correctly navigates the available pages
    ((NavigationView) findViewById(R.id.nav_view)).setNavigationItemSelectedListener(item -> {
        NavController navController1 = Navigation.findNavController(MainActivity.this,
R.id.nav_host_fragment_content_main);
        switch (item.getItemId()) {
            case R.id.nav_chats:
                navController1.navigate(R.id.chatsFragment);
                break;

            case R.id.nav_find_people:
                navController1.navigate(R.id.findPeopleFragment);
                break;
```

```
            case R.id.nav_friends:
                navController1.navigate(R.id.friendsFragment);
                break;

            case R.id.nav_invitations:
                navController1.navigate(R.id.invitationsFragment);
                break;
        }
        return false;
    });

    // Set the text in the nav header to show the app is waiting for the networker service
    // It is unlikely that the user will actually see this message
    ((TextView) ((NavigationView) findViewById(R.id.nav_view))
            .getHeaderView(0).findViewById(R.id.displayName))
            .setText("Waiting for networker service");
}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.main, menu);
    return true;
}

@Override
public boolean onSupportNavigateUp() {
    NavController navController = Navigation.findNavController(this, R.id.nav_host_fragment_content_main);
    return NavigationUI.navigateUp(navController, mAppBarConfiguration)
            || super.onSupportNavigateUp();
}

/**
 * Called when the search button is clicked on the search for people page
 * @param v The view that called this method
 */
public void onSearchButtonClicked(View v) {
    View fragmentView = (View) v.getParent().getParent();

    // Get the display name entered into the search box
    String displayName = ((EditText)
fragmentView.findViewById(R.id.UserSearchBar).findViewById(R.id.userSearchDisplayNameEntry)).getText().toString();
    // If the entry is empty, ask the user to enter a name and then end the method
    if (displayName.contentEquals("")) {
        Toast.makeText(this, "Please enter a name first!", Toast.LENGTH_SHORT).show();
        return;
    }

    // Create a request to search users by display name
    JSONObject request = new JSONObject();
    request.put("type", RequestType.GetUser);
    request.put("selector", "displayName");
    request.put("data", new User(-1, null, null, displayName, null, null));

    // Send the request
    networkerService.SendRequest(new NetworkerService.Request(
            new NetworkerService.IRequestCallback() {
```

```
                   @Override
                   public void callback(Result result, Object response) {
                       Runnable action;

                       if (result == Result.FAILED) {
                           action = () -> Toast.makeText(MainActivity.this, "Something went wrong :(",
Toast.LENGTH_SHORT).show();
                       }
                       else {
                           action = () -> {

                               User[] results = (User[]) response;

                               // Get the fragment container linear layout to add the result fragments to
                               LinearLayout fragmentContainerLayout =
fragmentView.findViewById(R.id.SearchResultFragmentContainer);

                               fragmentContainerLayout.removeAllViews();

                               // If the list of results is empty, hide the no results message
                               if (results.length == 0) {
                                   TextView message = new TextView(MainActivity.this);
                                   message.setText(R.string.no_search_results_message);

                                   ((LinearLayout)
fragmentView.findViewById(R.id.SearchResultFragmentContainer)).addView(message);
                               }

                               // Clean the results of invalid results
                               int numberRemoved = 0;
                               for (int i = 0; i < results.length; i++) {
                                   // Check if this user is the logged in user
                                   if (results[i].UserID == networkerService.user.UserID) {
                                       results[i] = null;
                                       numberRemoved++;
                                   }
                               }
                               searchResults = new User[results.length - numberRemoved];
                               int finalIndex = 0;
                               for (int i = 0; i < results.length; i++) {
                                   if (results[i] != null) {
                                       searchResults[finalIndex] = results[i];
                                       finalIndex++;
                                   }
                               }

                               // Add each of the results to the fragment container layout
                               for (User user : searchResults) {
                                   // Generate a random id for the fragment container
                                   int id = new Random().nextInt();

                                   FragmentContainerView fragmentContainer = new
FragmentContainerView(MainActivity.this);
                                   fragmentContainer.setId(id);

                                   // Create the argument bundle to pass to the fragment
                                   Bundle bundle = new Bundle();
```

```
                                    bundle.putSerializable("user", user);

                                    // Add the fragment to the fragment container view
                                    MainActivity.this.getSupportFragmentManager().beginTransaction()
                                            .setReorderingAllowed(true)
                                            .add(id, UserSearchFragment.class, bundle)
                                            .commit();

                                    // Add the fragment container view to the linear layout
                                    fragmentContainerLayout.addView(fragmentContainer);
                                }
                            };
                        }

                        // Run the predetermined action on the UI thread
                        MainActivity.this.runOnUiThread(action);
                    }

                },
                request));
    }

    /**
     * Called when the button to add a friend is clicked on the search people page
     * @param v The view that called the method
     */
    public void onAddFriendButtonClicked(View v) {
        FragmentContainerView fragmentContainerView = (FragmentContainerView) v.getParent().getParent();
        LinearLayout ll = (LinearLayout) fragmentContainerView.getParent();
        User user = null;

        for (int i = 0; i < ll.getChildCount(); i++) {
            if (fragmentContainerView.equals(ll.getChildAt(i))) {
                user = searchResults[i];
            }
        }

        assert user != null;

        // Create a request to send a friend request
        JSONObject request = new JSONObject();
        request.put("type", RequestType.SendFriendRequest);
        request.put("data", new FriendRequest(-1, networkerService.user.UserID, user.UserID, new Date().getTime()));

        // Send the request to the server
        networkerService.SendRequest(new NetworkerService.Request(
                new NetworkerService.IRequestCallback() {
                    @Override
                    public void callback(Result result, Object response) {
                        // Notify the user of the result
                        if (result == Result.SUCCESS) {
                            MainActivity.this.runOnUiThread(() -> Toast.makeText(MainActivity.this, "Friend request
sent!", Toast.LENGTH_SHORT).show());
                        }
                        else {
                            MainActivity.this.runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went
wrong :(", Toast.LENGTH_SHORT).show());
```

```
                    }
                }
            }, request
        ));
    }

    /**
     * Called when a chat invite button is clicked on the friends page
     * @param v The view that called this method
     */
    public void onInviteToChatClicked(View v) {
        // Get the friend that was clicked
        LinearLayout ll = (LinearLayout) v.getParent().getParent();
        User friend = null;
        for (int i = 0; i < ll.getChildCount(); i++) {
            if (ll.getChildAt(i).equals(v.getParent())) {
                friend = friends[i];
                break;
            }
        }

        assert friend != null;

        Intent intent = new Intent(this, InviteToChatActivity.class);
        Bundle bundle = new Bundle();
        bundle.putSerializable("userToInvite", friend);
        intent.putExtras(bundle);

        startActivity(intent);
    }

    /**
     * Called when a chat box is clicked
     * @param v The view that called this method
     */
    public void onChatClicked(View v) {
        // Get the navigation controller for this activity
        NavController navController1 = Navigation.findNavController(MainActivity.this,
R.id.nav_host_fragment_content_main);
        // Create the bundle which we will pass to the messaging fragment
        Bundle bundle = new Bundle();

        // We now need to determine which chat was clicked
        boolean found = false;
        for (int i = 0; i < chatFragments.size(); i++) {
            if (chatFragments.get(i).requireView().equals(v)) {
                bundle.putSerializable("chat", chatFragments.get(i).chat);
                found = true;
                break;
            }
        }

        assert found;  // Ensure that the chat was found

        // Navigate to the messaging fragment, passing the argument bundle
        navController1.navigate(R.id.messagingFragment, bundle);
    }
```

```
    /**
     * Called when an invite is accepted
     * @param v The view that called this method
     */
    public void onAcceptInviteClicked(View v) {
        // Get the invite from the view that was clicked
        Object invite = null;

        // Iterate over the invitation fragments
        for (int i = 0; i < invitationFragments.size(); i++) {
            // Check if the views are the same
            if (invitationFragments.get(i).requireView().equals(v.getParent())) {
                // Get the invite object and exit the loop
                invite = invitationFragments.get(i).invite;
                break;
            }
        }

        // Ensure that the invite is found
        assert invite != null;

        // Determine if the invite is a friend request or chat invite
        RequestType type = (invite.getClass() == FriendRequest.class) ? RequestType.AcceptFriendRequest :
RequestType.AcceptChatInvite;

        // Check if the user is already a member of this chat
        if (type == RequestType.AcceptChatInvite) {
            try {
                ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File(getFilesDir(),
"Chats.bin")));
                Chat[] chats = (Chat[]) ois.readObject();
                for (Chat chat : chats) {
                    if (chat.ChatID == ((ChatInvite) invite).ChatID) {
                        Toast.makeText(this, "You are already a member of this chat!", Toast.LENGTH_SHORT).show();
                        return;
                    }
                }
            } catch (IOException | ClassNotFoundException e) {
                e.printStackTrace();
                System.exit(1);
            }
        }

        // Create a request to accept the invite
        JSONObject request = new JSONObject();
        request.put("type", type);
        request.put("data", invite);

        Object finalInvite = invite;
        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
```

```
            }

            // Check if the response is a string
            if (response.getClass() == String.class) {
                // Output an appropriate message based on the response
                if (response.equals("failed")) {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Failed to accept invite :(",
Toast.LENGTH_SHORT).show());
                }
                else {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Invite accepted!",
Toast.LENGTH_SHORT).show());

                    if (finalInvite.getClass() == ChatInvite.class) {
                        JSONObject getChatRequest = new JSONObject();
                        getChatRequest.put("type", RequestType.GetChat);
                        getChatRequest.put("data", new Chat(((ChatInvite) finalInvite).ChatID, null, null, -1));

                        networkerService.SendRequest(new NetworkerService.Request(new
NetworkerService.IRequestCallback() {
                            @Override
                            public void callback(Result result, Object response) {
                                // Add a message listen rule for the new chat
                                JSONObject lrRequest = new JSONObject();
                                lrRequest.put("type", RequestType.AddListenRule);
                                lrRequest.put("data", new ListenRule(RequestType.SendMessage, "ChatID",
((ChatInvite) finalInvite).ChatID));

                                Bundle bundle = new Bundle();
                                bundle.putSerializable("chat", (Chat) response);

                                networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new
NetworkerService.IListenRuleCallback() {
                                    @Override
                                    public void callback(Object response, Bundle bundle) {
                                        if (networkerService.activeChatID != ((Chat)
bundle.getSerializable("chat")).ChatID) {
                                            // Create a notification
                                            networkerService.notificationChannel.showNotification("New message",
"You have a new message in " + ((Chat) bundle.getSerializable("chat")).Name);
                                        }
                                    }

                                }, new NetworkerService.IRequestCallback() {
                                    @Override
                                    public void callback(Result result, Object response) {
                                        NetworkerService.IRequestCallback.super.callback(result, response);
                                    }
                                }, lrRequest, bundle));
                            }
                        }, getChatRequest));
                    }
                }
                MainActivity.this.runOnUiThread(() -> invitationsFragment.reloadInvites());
            }
            else {  // In this case the response will be a key pair, and the invite must have been a chat invite
                assert finalInvite instanceof ChatInvite;
```

```
                KeyPair privateKey = (KeyPair) response;

                // Request the chat as we need it's public key id to store it in the internal key store
                JSONObject chatRequest = new JSONObject();
                chatRequest.put("type", RequestType.GetChat);
                chatRequest.put("data", new Chat(((ChatInvite) finalInvite).ChatID, "", "", -1));

                networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback()
{
                    @Override
                    public void callback(Result result, Object response) {
                        if (result == Result.FAILED) {
                            runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                            System.exit(1);
                        }

                        Chat chat = (Chat) response;

                        // Add the private key to the key store and the chat to the chats file
                        try {
                            KeyStore keyStore = new KeyStore(new File(getFilesDir(), "KeyStore.bin"));
                            keyStore.AddKeyPair(chat.PublicKeyID, privateKey);

                            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new
File(getFilesDir(), "Chats.bin")));
                            Chat[] chats = (Chat[]) ois.readObject();
                            Chat[] newChats = new Chat[chats.length + 1];
                            System.arraycopy(chats, 0, newChats, 0, chats.length);
                            newChats[chats.length] = chat;
                            ois.close();
                            ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new
File(getFilesDir(), "Chats.bin")));
                            oos.writeObject(newChats);
                            oos.flush();
                            oos.close();


                        } catch (IOException | ClassNotFoundException e) {
                            e.printStackTrace();
                            runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                            System.exit(1);
                        }

                        runOnUiThread(() -> Toast.makeText(MainActivity.this, "Invite accepted!",
Toast.LENGTH_SHORT).show());
                        MainActivity.this.runOnUiThread(() -> invitationsFragment.reloadInvites());

                        networkerService.waitingForResponse = false;
                    }
                }, chatRequest));

            }

            networkerService.waitingForResponse = false;
        }
```

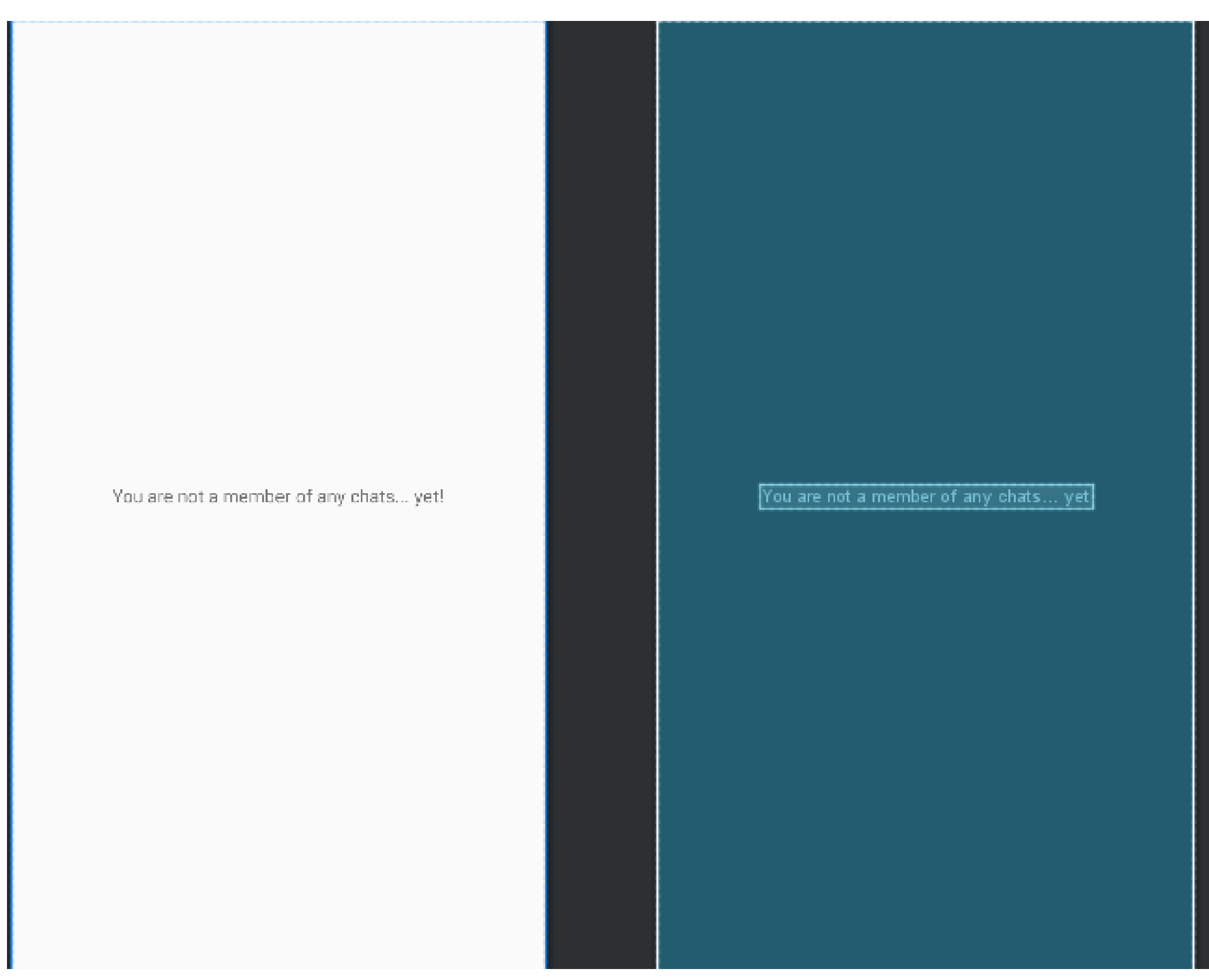
```
        }, request));
    }

    /**
     * Called when an invite is declined
     * @param v The view that called this method
     */
    public void onDeclineInviteClicked(View v) {
        // Get the invite from the view that was clicked
        Object invite = null;

        // Iterate over the invitation fragments
        for (int i = 0; i < invitationFragments.size(); i++) {
            // Check if the views are the same
            if (invitationFragments.get(i).requireView().equals(v.getParent())) {
                // Get the invite object and exit the loop
                invite = invitationFragments.get(i).invite;
                break;
            }
        }

        // Ensure that the invite is found
        assert invite != null;

        // Determine if the invite is a friend request or chat invite
        RequestType type = (invite.getClass() == FriendRequest.class) ? RequestType.DeclineFriendRequest :
RequestType.DeclineChatInvite;

        // Create a request to decline the invite
        JSONObject request = new JSONObject();
        request.put("type", type);
        request.put("data", invite);

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                // Output an appropriate message based on the response
                if (response.equals("failed")) {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Failed to decline invite :(",
Toast.LENGTH_SHORT).show());
                }
                else {
                    runOnUiThread(() -> Toast.makeText(MainActivity.this, "Invite declined!",
Toast.LENGTH_SHORT).show());
                }

                MainActivity.this.runOnUiThread(() -> invitationsFragment.reloadInvites());

                networkerService.waitingForResponse = false;
            }
        }, request));
```

```
    }

    /**
     * Sends a message to the currently active chat
     * @param v The view that called this method
     */
    public void SendMessage(View v) throws IOException {
        // Hide the send button and show the loading wheel
        v.setVisibility(View.GONE);
        ((View) v.getParent()).findViewById(R.id.messageSendButtonLoadingWheel).setVisibility(View.VISIBLE);

        // Get the active chat from the messaging fragment
        Chat chat = messagingFragment.chat;

        // Get the message from the text box
        String messageContent = ((EditText) ((View)
v.getParent()).findViewById(R.id.MessageEntry)).getText().toString();
        // Get the chat's public key from the server
        JSONObject keyRequest = new JSONObject();
        keyRequest.put("type", RequestType.GetPublicKey);
        keyRequest.put("data", chat.PublicKeyID);

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED || response == null) {
                    MainActivity.this.runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong!",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                // Encrypt the message contents using the public key
                assert response != null;
                KeyPair publicKey = (KeyPair) response;
                EncryptedObject eContent = null;
                try {
                    eContent = publicKey.encrypt(messageContent);

                } catch (PublicKeyException e) {
                    MainActivity.this.runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went wrong!",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                assert eContent != null;

                // Create the message object to send to the server
                Message message = new Message(networkerService.user.UserID, chat.ChatID, new Date().getTime(),
eContent);

                JSONObject sendRequest = new JSONObject();
                sendRequest.put("type", RequestType.SendMessage);
                sendRequest.put("data", message);

                networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
                    @Override
                    public void callback(Result result, Object response) {
```

```
                        if (result == Result.FAILED || response == null) {
                            MainActivity.this.runOnUiThread(() -> Toast.makeText(MainActivity.this, "Something went
wrong!", Toast.LENGTH_SHORT).show());
                            System.exit(1);
                        }

                        runOnUiThread(() -> {
                            // Hide the loading wheel and show the send button
                            v.setVisibility(View.VISIBLE);
                            ((View)
v.getParent()).findViewById(R.id.messageSendButtonLoadingWheel).setVisibility(View.GONE);
                        });
                    }
                }, sendRequest));
            }
        }, keyRequest));

        // Clear the text box
        ((EditText) ((View) v.getParent()).findViewById(R.id.MessageEntry)).setText("");
    }
}
```

ChatsFragment

This is a page which can be displayed on the central view of the main activity, it displays the chats the user is currently a part of.



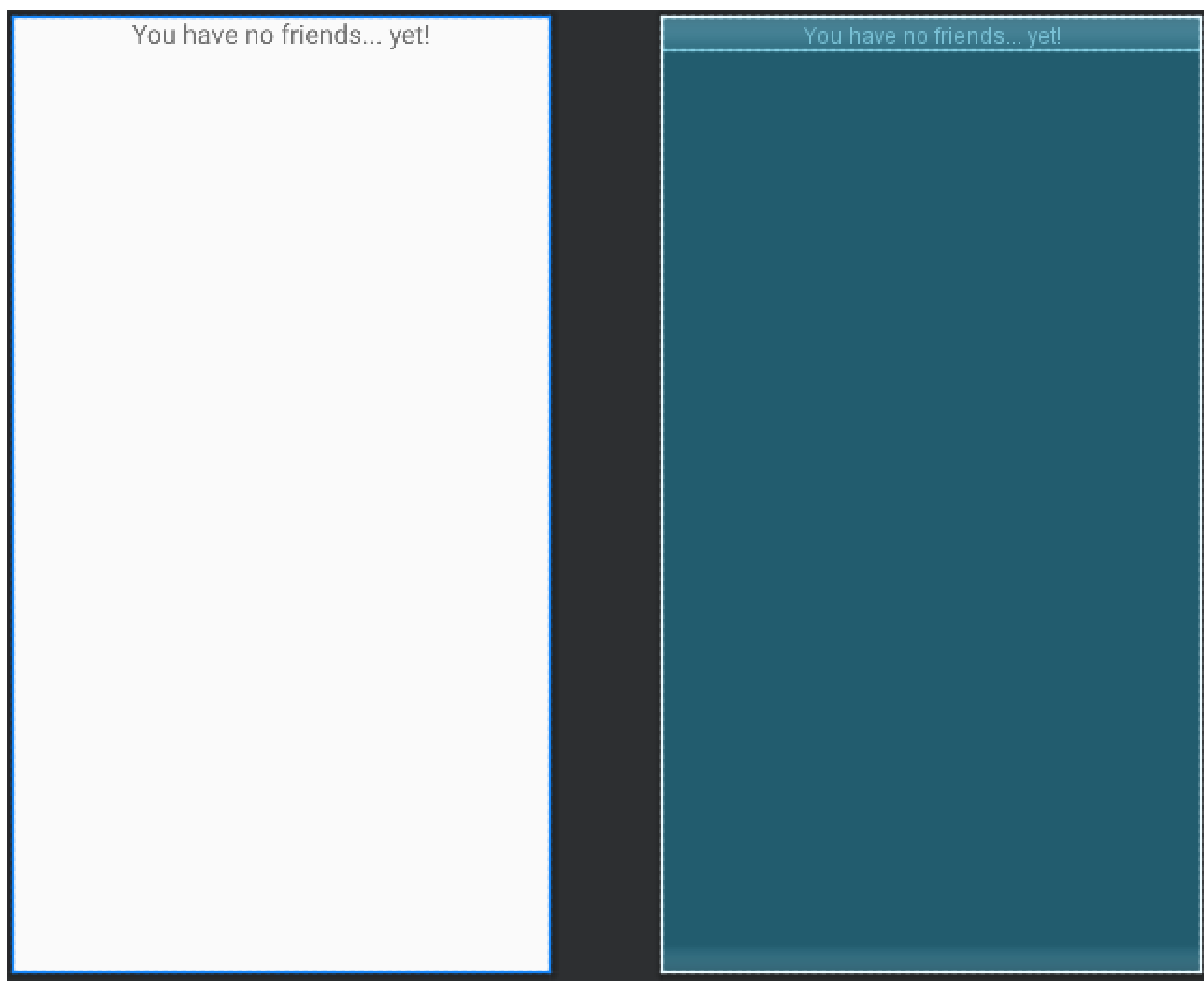

```
package com.nathcat.messagecat_client;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentContainerView;
import androidx.fragment.app.FragmentTransaction;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;

import com.nathcat.messagecat_database_entities.Chat;

import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.util.Random;

public class ChatsFragment extends Fragment {

   public ChatsFragment() {
       super(R.layout.fragment_chats);
```

```
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_chats, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();

        // Set the title on the action bar
        // Otherwise it will be "fragment_chats"
        ((AppCompatActivity) requireActivity()).getSupportActionBar().setTitle("Chats");

        Chat[] chats;

        try {
            ((MainActivity) requireActivity()).chatFragments.clear();

        } catch (ClassCastException ignored) {}

        // Get the array of chats from local storage
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new
File(requireActivity().getFilesDir(), "Chats.bin")));
            chats = (Chat[]) ois.readObject();
            ois.close();

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            return;
        }

        // If there are chats to display, hide the no chats message
        if (chats.length != 0) {
            requireView().findViewById(R.id.noChatsMessage).setVisibility(View.GONE);
        }

        // Get the linear layout widget
        LinearLayout chatsContainer = requireView().findViewById(R.id.ChatsContainer);
        chatsContainer.removeAllViews();

        for (Chat chat : chats) {
            // Get a random id for the new fragment container
            int containerId = new Random().nextInt();

            // Create a new fragment container
            FragmentContainerView fragmentContainer = new FragmentContainerView(requireContext());
            fragmentContainer.setId(containerId);

            // Create a new chat fragment inside the fragment container
            Bundle bundle = new Bundle();
            bundle.putString("chatName", chat.Name);
            bundle.putString("chatDesc", chat.Description);
```

```
            bundle.putSerializable("chat", chat);

            requireActivity().getSupportFragmentManager().beginTransaction()
                    .setReorderingAllowed(true)
                    .add(containerId, ChatFragment.class, bundle)
                    .commit();

            // Add the new fragment container to the linear layout widget
            chatsContainer.addView(fragmentContainer);
        }
    }
}
```

FriendsFragment

This is another page which can be displayed in the central view of the main activity, it shows all the users this user is currently friends with.



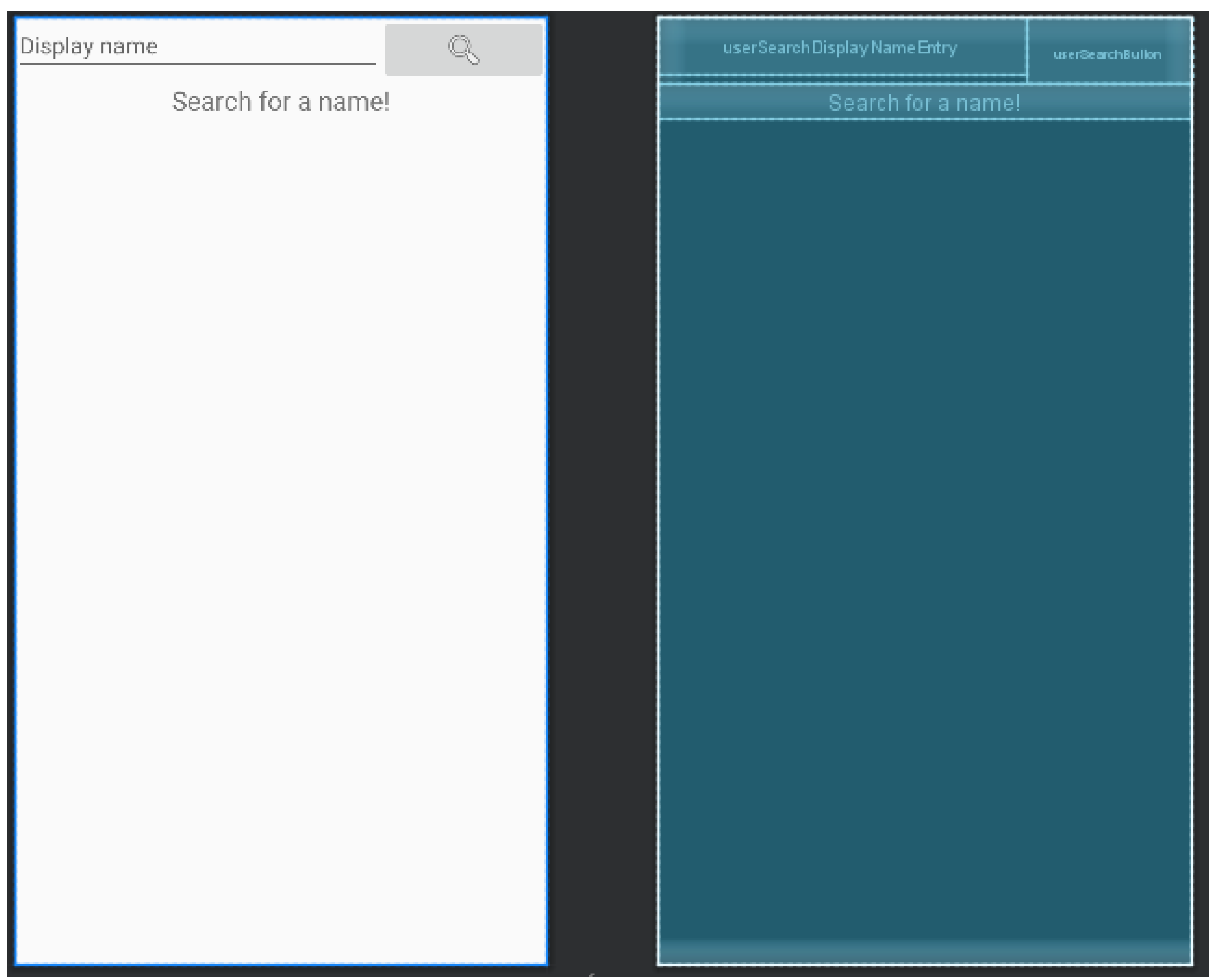

```
package com.nathcat.messagecat_client;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
```

```
import androidx.fragment.app.FragmentContainerView;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
import android.widget.Toast;

;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.Friendship;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.util.Random;

public class FriendsFragment extends Fragment {

    public FriendsFragment() {
        super(R.layout.fragment_friends);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_friends, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();

        // Set the title on the action bar
        ((AppCompatActivity) requireActivity()).getSupportActionBar().setTitle("Friends");

        LinearLayout container = requireView().findViewById(R.id.FriendsContainer);

        NetworkerService networkerService = ((MainActivity) requireActivity()).networkerService;

        // Create a request to get friendships from the server
        JSONObject request = new JSONObject();
        request.put("type", RequestType.GetFriendship);
        request.put("selector", "userID");
        request.put("data", new Friendship(-1, networkerService.user.UserID, -1, null));

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    requireActivity().runOnUiThread(() -> Toast.makeText(requireActivity(), "Something went wrong :(",
Toast.LENGTH_SHORT));
                    return;
                }
```

```
                Friendship[] friendships = (Friendship[]) response;
                ((MainActivity) requireActivity()).friends = new User[friendships.length];

                if (friendships.length != 0) {
                    requireActivity().runOnUiThread(container::removeAllViews);
                }

                for (int i = 0; i < friendships.length; i++) {
                    JSONObject friendRequest = new JSONObject();
                    friendRequest.put("type", RequestType.GetUser);
                    friendRequest.put("selector", "id");
                    friendRequest.put("data", new User(friendships[i].FriendID, null, null, null, null, null));

                    networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback()
{
                        @Override
                        public void callback(Result result, Object response) {
                            if (result == Result.FAILED) {
                                requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went
wrong :(", Toast.LENGTH_SHORT));
                                return;
                            }

                            // Add the friend to the friends array
                            for (int x = 0; x < ((MainActivity) requireActivity()).friends.length; x++) {
                                if (((MainActivity) requireActivity()).friends[x] == null) {
                                    ((MainActivity) requireActivity()).friends[x] = (User) response;
                                    break;
                                }
                            }

                            Bundle bundle = new Bundle();
                            bundle.putSerializable("user", (User) response);

                            FriendsFragment.this.getChildFragmentManager().beginTransaction()
                                    .setReorderingAllowed(true)
                                    .add(R.id.FriendsContainer, FriendFragment.class, bundle)
                                    .commit();

                            networkerService.waitingForResponse = false;
                        }
                    }, friendRequest));
                }

                networkerService.waitingForResponse = false;
            }
        }, request));
    }
}
```

FindUserFragment

This is yet another page which can be displayed in the central view of the main activity, the intention of this page is to allow users to search for other users, who they can then add as a friend.

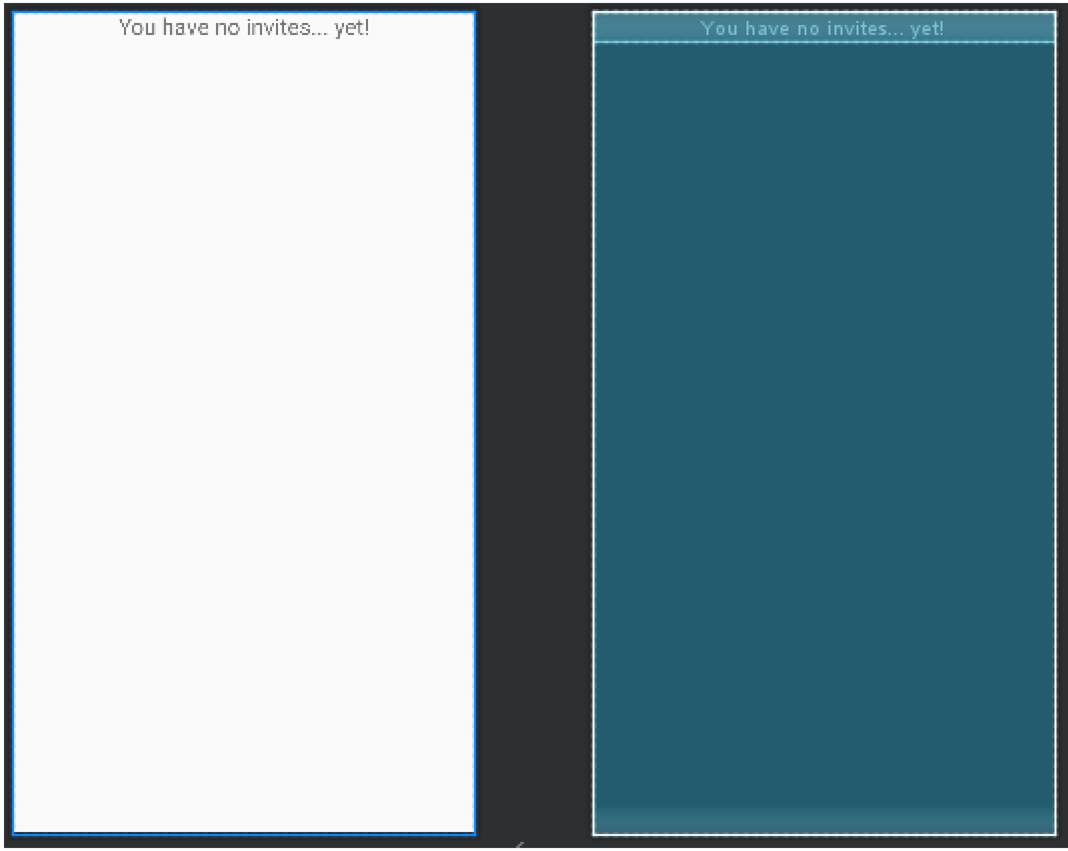



```
package com.nathcat.messagecat_client;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;
import androidx.fragment.app.FragmentContainerView;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.util.Random;

public class FindPeopleFragment extends Fragment {
```

```
    public FindPeopleFragment() {
        super(R.layout.fragment_find_people);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_find_people, container, false);
    }

    @Override
    public void onStart() {
        super.onStart();

        // Set the title on the action bar
        ((AppCompatActivity) requireActivity()).getSupportActionBar().setTitle("Find people");
    }
}
```

InvitationsFragment

This is the final page which can be displayed in the central view of the main activity, it shows pending invitations the user has received from their friends.




```
package com.nathcat.messagecat_client;
```

```java
import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.Toast;

;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.ChatInvite;
import com.nathcat.messagecat_database_entities.FriendRequest;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.util.ArrayList;
import java.util.Arrays;

public class InvitationsFragment extends Fragment {

   private ChatInvite[] chatInvites;
   private FriendRequest[] friendRequests;

   public InvitationsFragment() {
       super(R.layout.fragment_invitations);
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {



       // Inflate the layout for this fragment
       return inflater.inflate(R.layout.fragment_invitations, container, false);
   }

   @Override
   public void onStart() {
       super.onStart();

       ((MainActivity) requireActivity()).invitationFragments.clear();
       ((MainActivity) requireActivity()).invitationsFragment = this;

       // Set the title on the action bar
       ((AppCompatActivity) requireActivity()).getSupportActionBar().setTitle("Invitations");

       // Load the invites
       reloadInvites();
   }

   /**
```

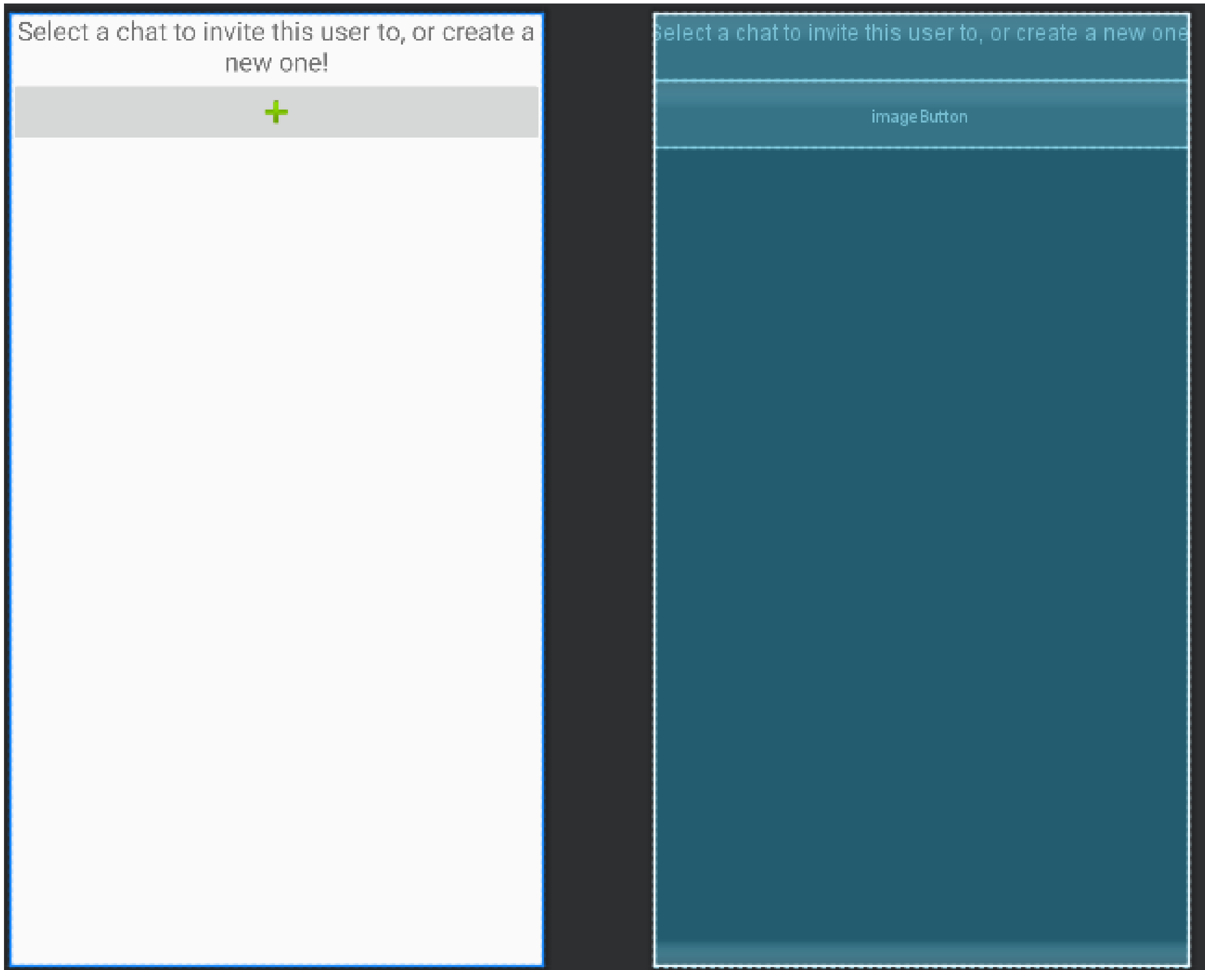
```
     * Load the invites from the server and display them onto the page
     */
    public void reloadInvites() {
        // Destroy all the currently displayed invites
        ((LinearLayout) requireView().findViewById(R.id.InvitationsContainer)).removeAllViews();
        // Remove all the invites from the main activity array
        ((MainActivity) requireActivity()).invitationFragments.clear();

        // Get the networker service from the main activity
        NetworkerService networkerService = ((MainActivity) requireActivity()).networkerService;

        // Request incoming friend requests from the server
        JSONObject request = new JSONObject();
        request.put("type", RequestType.GetFriendRequests);
        request.put("data", new FriendRequest(-1, -1, networkerService.user.UserID, -1));
        request.put("selector", "recipientID");

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                friendRequests = (FriendRequest[]) response;
                if (friendRequests.length != 0) {
                    requireActivity().runOnUiThread(() -> { try {
requireView().findViewById(R.id.noInvitesMessage).setVisibility(View.GONE); } catch (NullPointerException ignored) {}
});
                }

                for (FriendRequest fr : friendRequests) {
                    JSONObject userRequest = new JSONObject();
                    userRequest.put("type", RequestType.GetUser);
                    userRequest.put("selector", "id");
                    userRequest.put("data", new User(fr.SenderID, "", "", "", "", ""));

                    networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback()
{
                        @Override
                        public void callback(Result result, Object response) {
                            if (result == Result.FAILED) {
                                requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went
wrong :(", Toast.LENGTH_SHORT).show());
                                System.exit(1);
                            }

                            User user = (User) response;
                            Bundle bundle = new Bundle();
                            bundle.putSerializable("invite", fr);
                            bundle.putString("text", user.DisplayName + " wants to be friends!");

                            getChildFragmentManager().beginTransaction()
                                    .setReorderingAllowed(true)
                                    .add(R.id.InvitationsContainer, InvitationFragment.class, bundle)
```

```
                                    .commit();

                            networkerService.waitingForResponse = false;
                        }
                    }, userRequest));
                }

                networkerService.waitingForResponse = false;
            }
        }, request));

        // Request incoming chat requests from the server
        JSONObject chatInviteRequest = new JSONObject();
        chatInviteRequest.put("type", RequestType.GetChatInvite);
        chatInviteRequest.put("data", new ChatInvite(-1, -1, -1, networkerService.user.UserID, -1, -1));
        chatInviteRequest.put("selector", "recipientID");

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                chatInvites = (ChatInvite[]) response;

                if (chatInvites.length != 0) {
                    requireActivity().runOnUiThread(() -> { try {
requireView().findViewById(R.id.noInvitesMessage).setVisibility(View.GONE); } catch (NullPointerException ignored) {}
});
                }

                for (ChatInvite inv : chatInvites) {
                    JSONObject userRequest = new JSONObject();
                    userRequest.put("type", RequestType.GetUser);
                    userRequest.put("selector", "id");
                    userRequest.put("data", new User(inv.SenderID, "", "", "", "", ""));

                    networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback()
{
                        @Override
                        public void callback(Result result, Object response) {
                            if (result == Result.FAILED) {
                                requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went
wrong :(", Toast.LENGTH_SHORT).show());
                                System.exit(1);
                            }

                            User user = (User) response;
                            Bundle bundle = new Bundle();
                            bundle.putSerializable("invite", inv);
                            bundle.putString("text", user.DisplayName + " wants to chat!");

                            getChildFragmentManager().beginTransaction()
                                    .setReorderingAllowed(true)
```

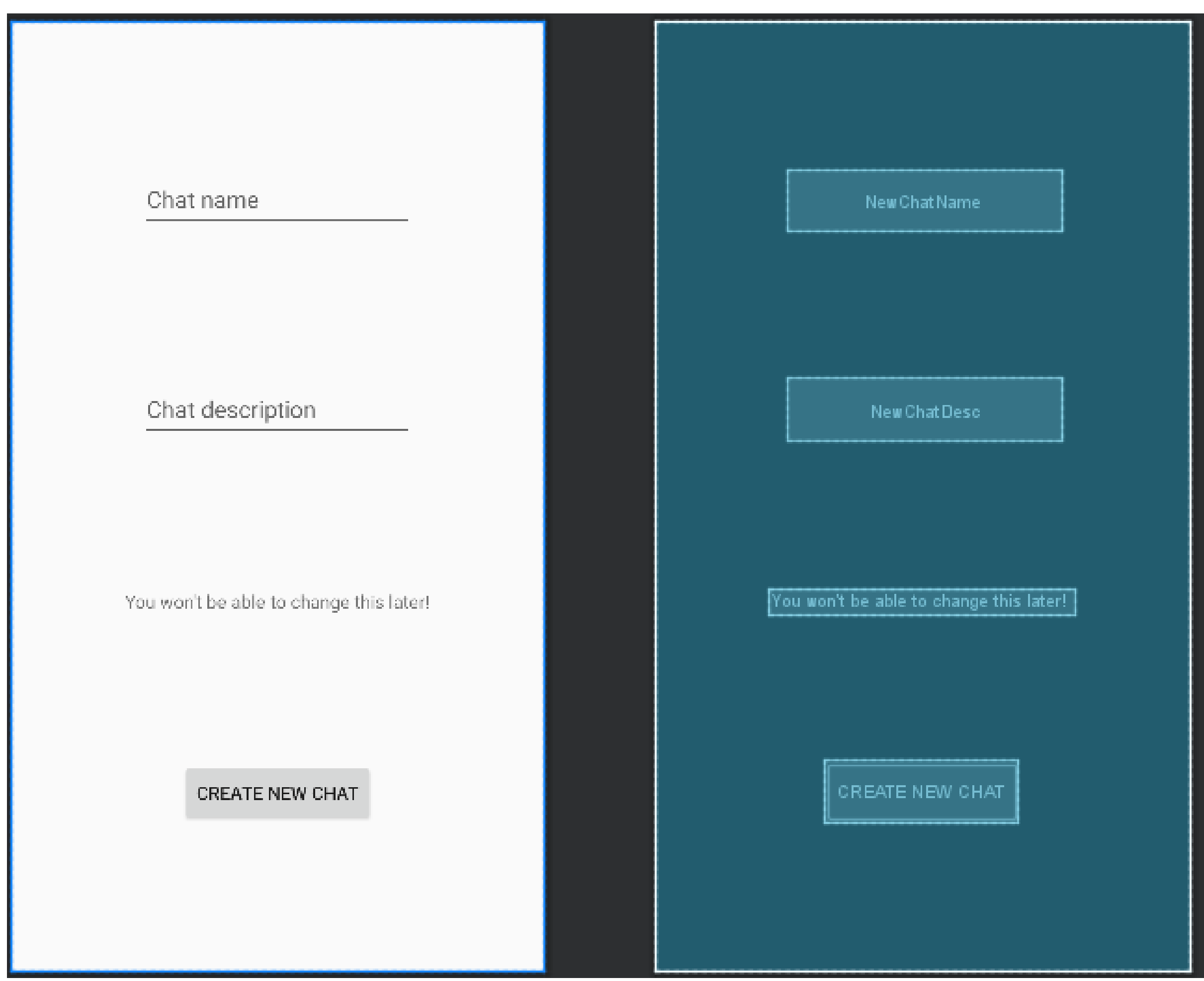
```
                                    .add(R.id.InvitationsContainer, InvitationFragment.class, bundle)
                                    .commit();

                            networkerService.waitingForResponse = false;
                        }
                    }, userRequest));
                }

                networkerService.waitingForResponse = false;
            }
        }, chatInviteRequest));
    }
}
```

InviteToChatActivity

This activity allows users to invite a friend to a chat, they can do this by either inviting them to an existing chat or creating a completely new one, and then inviting them to the new one, this activity facilitates both of these cases.



By default, they will be given the option to choose an existing chat, so the program will load the chats fragment below the existing content in the window. When the green plus button is clicked, the program will load the create new chat fragment:



In place of the chats fragment.

```
package com.nathcat.messagecat_client;

import androidx.appcompat.app.AppCompatActivity;

import android.content.ComponentName;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.widget.EditText;
import android.widget.LinearLayout;
import android.widget.Toast;

import com.nathcat.RSA.KeyPair;
;
import com.nathcat.RSA.RSA;
import com.nathcat.messagecat_database.KeyStore;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.Chat;
import com.nathcat.messagecat_database_entities.ChatInvite;
import com.nathcat.messagecat_database_entities.User;
```

```java
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;

import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.security.NoSuchAlgorithmException;
import java.util.Date;

public class InviteToChatActivity extends AppCompatActivity {

   private ServiceConnection connection = new ServiceConnection() {
       @Override
       public void onServiceConnected(ComponentName componentName, IBinder iBinder) {
           networkerService = ((NetworkerService.NetworkerServiceBinder) iBinder).getService();
       }

       @Override
       public void onServiceDisconnected(ComponentName componentName) {
           networkerService = null;
       }
   };

   private NetworkerService networkerService;
   private User userToInvite;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_invite_to_chat);

       // Bind to the networker service
       bindService(
               new Intent(this, NetworkerService.class),
               connection,
               BIND_AUTO_CREATE
       );

       // The user that is being invited to a chat should be passed to this activity as a bundle
       userToInvite = (User) getIntent().getSerializableExtra("userToInvite");

       // Add the chats fragment view
       getSupportFragmentManager().beginTransaction()
               .setReorderingAllowed(true)
               .add(R.id.InviteToChatChatsContainer, ChatsFragment.class, null)
               .commit();
   }

   /**
    * Changes the fragment to the create new chat fragment
    * @param v The view that called this method
    */
   public void onAddChatButtonClicked(View v) {
```

```java
        // Change the chats fragment to the create new chat fragment
        ((LinearLayout) findViewById(R.id.InviteToChatChatsContainer)).removeAllViews();

        getSupportFragmentManager().beginTransaction()
                .setReorderingAllowed(true)
                .add(R.id.InviteToChatChatsContainer, CreateNewChatFragment.class, null)
                .commit();
    }

    /**
     * Sends a chat invite for the chat that was clicked
     * @param v The view that called this method, this will be the chat that was clicked
     */
    public void onChatClicked(View v) {
        // Get the container object of the chat fragment that was clicked
        LinearLayout container = (LinearLayout) v.getParent().getParent();

        // Get the chats array from the file
        Chat[] chats = null;
        try {
            ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File(getFilesDir(), "Chats.bin")));
            chats = (Chat[]) ois.readObject();
            ois.close();

        } catch (IOException | ClassNotFoundException e) {
            e.printStackTrace();
            Toast.makeText(this, "Something went wrong :(", Toast.LENGTH_SHORT).show();
            System.exit(1);
        }

        assert chats != null;

        // Determine which chat was clicked
        Chat chat = null;

        for (int i = 0; i < container.getChildCount(); i++) {
            if (container.getChildAt(i).equals(v.getParent())) {
                chat = chats[i];
            }
        }

        assert chat != null;

        // Get the key pair from the key store
        KeyPair pair = null;

        try {
            KeyStore keyStore = new KeyStore(new File(getFilesDir(), "KeyStore.bin"));
            pair = keyStore.GetKeyPair(chat.PublicKeyID);

        } catch (IOException e) {
            e.printStackTrace();
            Toast.makeText(this, "Something went wrong :(", Toast.LENGTH_SHORT).show();
            System.exit(1);
        }

        assert pair != null;
```

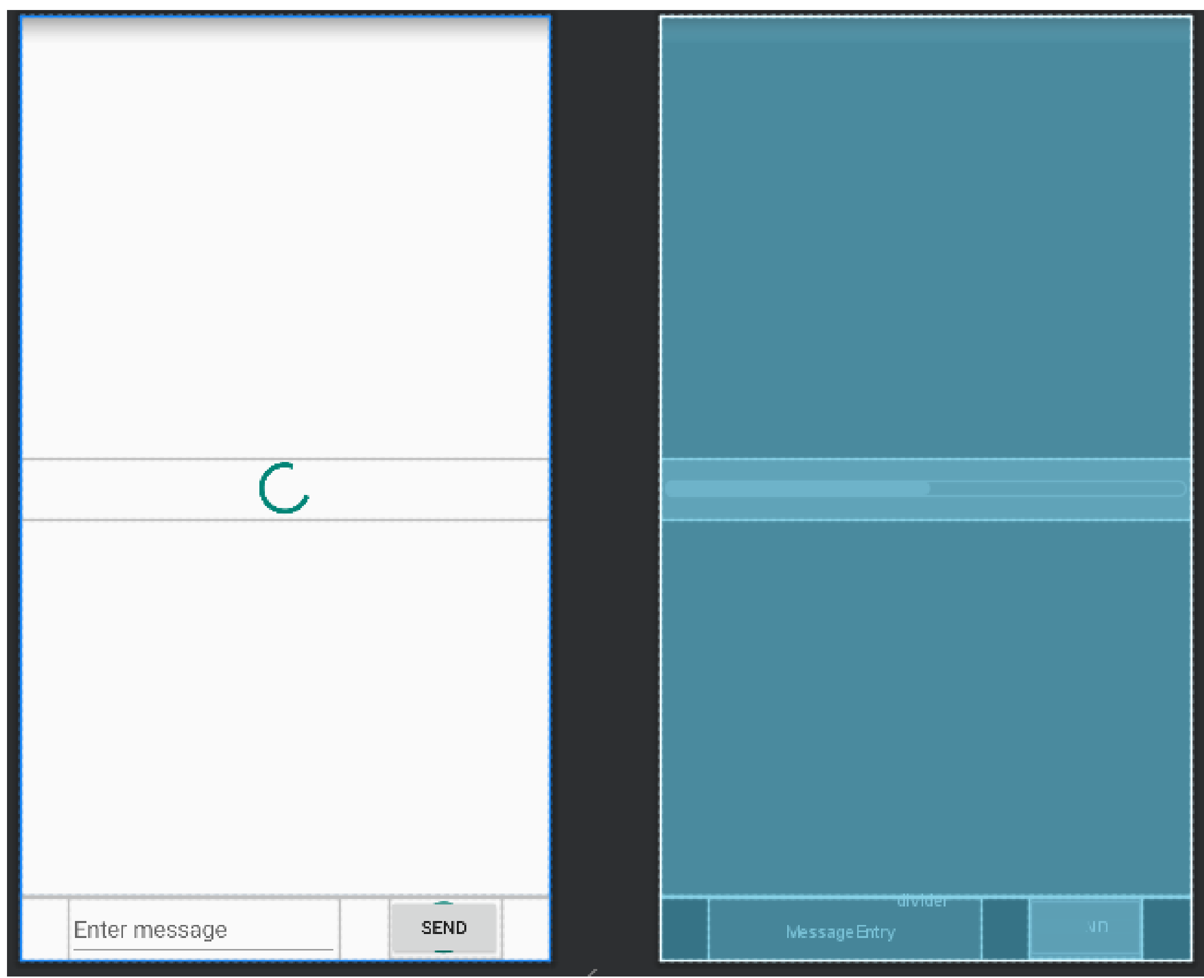
```
        // Create and send the request
        JSONObject request = new JSONObject();
        request.put("type", RequestType.SendChatInvite);
        request.put("data", new ChatInvite(-1, chat.ChatID, networkerService.user.UserID, userToInvite.UserID, new
Date().getTime(), -1));
        request.put("keyPair", new KeyPair(null, pair.pri));

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    InviteToChatActivity.this.runOnUiThread(() -> Toast.makeText(InviteToChatActivity.this, "Something
went wrong :(", Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                Toast.makeText(InviteToChatActivity.this, "Sent chat invite!", Toast.LENGTH_SHORT).show();

                // Go back to the main activity
                InviteToChatActivity.this.runOnUiThread(() -> startActivity(new Intent(InviteToChatActivity.this,
MainActivity.class)));
            }
        }, request));
    }

    /**
     * Creates a new chat
     * @param v The view that called this method
     */
    public void onCreateNewChatClicked(View v) {
        // Get the name and description of the new chat
        View fragmentContainer = (View) v.getParent();
        String chatName = ((EditText) fragmentContainer.findViewById(R.id.NewChatName)).getText().toString();
        String chatDesc = ((EditText) fragmentContainer.findViewById(R.id.NewChatDesc)).getText().toString();

        // Check that neither of the fields are empty before proceeding
        if (chatName.contentEquals("") || chatDesc.contentEquals("")) {
            Toast.makeText(this, "One or more of the required fields are empty!", Toast.LENGTH_SHORT).show();
            return;
        }

        // Generate a new key pair for the new chat
        KeyPair chatKeyPair;
        try {
            chatKeyPair = RSA.GenerateRSAKeyPair();

        } catch (NoSuchAlgorithmException e) {
            e.printStackTrace();
            return;
        }

        // Send the request to the server
        JSONObject request = new JSONObject();
        request.put("type", RequestType.AddChat);
        request.put("data", new Chat(-1, chatName, chatDesc, -1));
        request.put("keyPair", new KeyPair(chatKeyPair.pub, null));
```

```
        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    InviteToChatActivity.this.runOnUiThread(() -> Toast.makeText(InviteToChatActivity.this, "Something
went wrong!", Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                // Get the chat from the response
                Chat chat = (Chat) response;

                try {
                    // Open the chats file and read the current chat array
                    ObjectInputStream ois = new ObjectInputStream(new FileInputStream(new File(getFilesDir(),
"Chats.bin")));
                    Chat[] chats = (Chat[]) ois.readObject();
                    ois.close();

                    // Add the new chat to the array
                    Chat[] newChats = new Chat[chats.length + 1];
                    System.arraycopy(chats, 0, newChats, 0, chats.length);
                    newChats[chats.length] = chat;

                    // Write the new array to the chats file
                    ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new File(getFilesDir(),
"Chats.bin")));
                    oos.writeObject(newChats);
                    oos.flush();
                    oos.close();

                    // Open the key store and add the new key pair
                    KeyStore keyStore = new KeyStore(new File(getFilesDir(), "KeyStore.bin"));
                    if (keyStore.AddKeyPair(chat.PublicKeyID, chatKeyPair) == Result.FAILED) {
                        InviteToChatActivity.this.runOnUiThread(() -> Toast.makeText(InviteToChatActivity.this,
"Something went wrong!", Toast.LENGTH_SHORT).show());
                        System.exit(1);
                    }

                } catch (IOException | ClassNotFoundException e) {
                    e.printStackTrace();
                    InviteToChatActivity.this.runOnUiThread(() -> Toast.makeText(InviteToChatActivity.this, "Something
went wrong!", Toast.LENGTH_SHORT).show());
                    System.exit(1);
                }

                // Send a chat invite
                JSONObject inviteRequest = new JSONObject();
                inviteRequest.put("type", RequestType.SendChatInvite);
                inviteRequest.put("data", new ChatInvite(-1, chat.ChatID, networkerService.user.UserID,
userToInvite.UserID, new Date().getTime(), -1));
                inviteRequest.put("keyPair", new KeyPair(null, chatKeyPair.pri));

                networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
                    @Override
                    public void callback(Result result, Object response) {
```

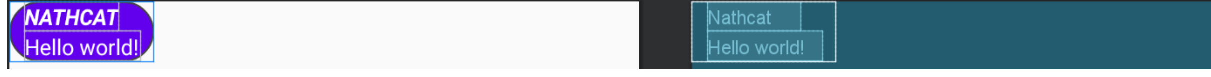
```
                        if (result == Result.FAILED) {
                            InviteToChatActivity.this.runOnUiThread(() -> Toast.makeText(InviteToChatActivity.this,
"Something went wrong!", Toast.LENGTH_SHORT).show());
                            System.exit(1);
                        }

                        InviteToChatActivity.this.runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                Toast.makeText(InviteToChatActivity.this, "Sent chat invite!",
Toast.LENGTH_SHORT).show();

                                // Go back to the main activity
                                InviteToChatActivity.this.runOnUiThread(() -> startActivity(new
Intent(InviteToChatActivity.this, MainActivity.class)));
                            }
                        });
                    }
                }, inviteRequest));
            }
        }, request));
    }
}
```

MessagingFragment

This page is loaded when a chat is clicked on the chats fragment page, it allows users to send messages to a chat, and view old messages in that chat.



Your messages will appear in purple, like this:



While other user’s messages will appear in green.

```
package com.nathcat.messagecat_client;

import android.os.Bundle;

import androidx.appcompat.app.AppCompatActivity;
import androidx.fragment.app.Fragment;

import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.ScrollView;
import android.widget.Toast;

import com.nathcat.RSA.EncryptedObject;
import com.nathcat.RSA.KeyPair;
;
```

```java
import com.nathcat.RSA.PrivateKeyException;
import com.nathcat.messagecat_database.KeyStore;
import com.nathcat.messagecat_database.MessageQueue;
import com.nathcat.messagecat_database.Result;
import com.nathcat.messagecat_database_entities.Chat;
import com.nathcat.messagecat_database_entities.Message;
import com.nathcat.messagecat_database_entities.User;
import com.nathcat.messagecat_server.ListenRule;
import com.nathcat.messagecat_server.RequestType;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.HashMap;

public class MessagingFragment extends Fragment {

   public Chat chat;
   private KeyPair privateKey;
   private NetworkerService networkerService;
   private MessageQueue messageQueue;
   private int listenRuleId = -1;

   public MessagingFragment() {
       super(R.layout.fragment_messaging);
   }

   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);

       // Get the chat argument passed to this fragment, and the networker service instance from
       // the main activity
       chat = (Chat) requireArguments().getSerializable("chat");
       networkerService = ((MainActivity) requireActivity()).networkerService;
   }

   @Override
   public View onCreateView(LayoutInflater inflater, ViewGroup container,
                            Bundle savedInstanceState) {
       // Inflate the layout for this fragment
       return inflater.inflate(R.layout.fragment_messaging, container, false);
   }

   @Override
   public void onStart() {
       super.onStart();

       // Set the title on the action bar
       ((AppCompatActivity) requireActivity()).getSupportActionBar().setTitle(chat.Name);

       // Hide the loading wheel under the send button
       requireView().findViewById(R.id.messageSendButtonLoadingWheel).setVisibility(View.GONE);
```

```
        ((MainActivity) requireActivity()).messagingFragment = this;

        try {
            KeyStore keyStore = new KeyStore(new File(requireActivity().getFilesDir(), "KeyStore.bin"));
            privateKey = keyStore.GetKeyPair(MessagingFragment.this.chat.PublicKeyID);

        } catch (IOException e) {
            e.printStackTrace();
        }

        assert privateKey != null;

        ((MainActivity) requireActivity()).networkerService.activeChatID = this.chat.ChatID;

        // Create the users hashmap and put the current user in it
        ((MainActivity) requireActivity()).users.put(networkerService.user.UserID, networkerService.user);

        networkerService.waitingForResponse = false;

        // Request the message queue from the server
        JSONObject request = new JSONObject();
        request.put("type", RequestType.GetMessageQueue);
        request.put("data", chat.ChatID);

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                // Check if the request failed
                if (result == Result.FAILED || response == null) {
                    requireActivity().runOnUiThread(() -> Toast.makeText(requireContext(), "Something went wrong :(",
Toast.LENGTH_SHORT).show());
                    return;
                }

                // Assign the message queue to the field
                System.out.println(response);
                messageQueue = (MessageQueue) response;

                // Call the update message box function on the UI thread
                // Passing the instance of the fragment class as a parameter
                try {
                    requireActivity().runOnUiThread(() ->
MessagingFragment.updateMessageBoxStart(MessagingFragment.this, privateKey));
                    // Hide the loading wheel
                    requireActivity().runOnUiThread(() ->
requireView().findViewById(R.id.messagingLoadingWheel).setVisibility(View.GONE));

                } catch (IllegalStateException e) {
                    System.out.println("Message window was closed!");
                    return;
                }
                networkerService.waitingForResponse = false;
            }
        }, request));
```

```
        // Create the listen rule for messages in this chat
        ListenRule listenRule = new ListenRule(RequestType.SendMessage, "ChatID", this.chat.ChatID);
        JSONObject listenRuleRequest = new JSONObject();
        listenRuleRequest.put("type", RequestType.AddListenRule);
        listenRuleRequest.put("data", listenRule);

        networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new NetworkerService.IListenRuleCallback()
{
            @Override
            public void callback(Object response) {
                updateMessageBox((Message) ((JSONObject) response).get("data"));
            }
        }, new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                if (result == Result.FAILED) {
                    System.out.println("Failed to create listen rule");
                    System.exit(1);
                }

                listenRuleId = (int) response;
            }
        }, listenRuleRequest));
    }

    @Override
    public void onStop() {
        super.onStop();

        JSONObject request = new JSONObject();
        request.put("type", RequestType.RemoveListenRule);
        request.put("data", listenRuleId);

        networkerService.SendRequest(new NetworkerService.ListenRuleRequest(new NetworkerService.IListenRuleCallback()
{
            @Override
            public void callback(Object response) {
                NetworkerService.IListenRuleCallback.super.callback(response);
            }
        }, new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                NetworkerService.IRequestCallback.super.callback(result, response);
            }
        }, request));

        ((MainActivity) requireActivity()).networkerService.activeChatID = -1;
    }

    /**
     * Updates the messagebox with the current contents of the message queue
     * @param fragment The instance of the current fragment class, this is necessary as this method will mostly be
called from a Runnable, which is a static context
     */
    public static void updateMessageBoxStart(MessagingFragment fragment, KeyPair privateKey) {
        fragment.networkerService.waitingForResponse = false;
```

```
        // Remove all messages currently in the messagebox
        fragment.requireActivity().runOnUiThread(() -> ((LinearLayout)
fragment.requireView().findViewById(R.id.MessageBox)).removeAllViews());


        // Add the new messages
        for (int i = 0; i < 50; i++) {
            while (fragment.networkerService.waitingForResponse) {
                System.out.println("Waiting for response");
            }

            Message message = fragment.messageQueue.Get(i);
            if (message == null) {
                continue;
            }

            // If the user that sent the message is not currently in the hashmap, request the user from the server and
add them
            // Then we can add the message
            if (((MainActivity) fragment.requireActivity()).users.get(message.SenderID) == null) {
                JSONObject request = new JSONObject();
                request.put("type", RequestType.GetUser);
                request.put("selector", "id");
                request.put("data", new User(message.SenderID, null, null, null, null, null));

                fragment.networkerService.SendRequest(new NetworkerService.Request(new
NetworkerService.IRequestCallback() {
                    @Override
                    public void callback(Result result, Object response) {
                        ((MainActivity) fragment.requireActivity()).users.put(((User) response).UserID, (User)
response);

                        // Decrypt the message before passing it to the fragment
                        String content = null;
                        try {
                            content = (String) privateKey.decrypt((EncryptedObject) message.Content);

                        } catch (PrivateKeyException e) {
                            e.printStackTrace();
                            System.exit(1);
                        }

                        // Create a new message object with the decrypted contents
                        Message decryptedMessage = new Message(message.SenderID, message.ChatID, message.TimeSent,
content);

                        // Add the message to the view as a fragment
                        Bundle bundle = new Bundle();
                        bundle.putSerializable("message", decryptedMessage);
                        bundle.putBoolean("fromOtherUser", message.SenderID != fragment.networkerService.user.UserID);

                        fragment.getChildFragmentManager().beginTransaction()
                                .setReorderingAllowed(true)
                                .add(R.id.MessageBox, MessageFragment.class, bundle)
                                .commit();

                        fragment.networkerService.waitingForResponse = false;
```

```
                }
            }, request));
        }
        else {
            // Decrypt the message contents
            String content = null;
            try {
                content = (String) privateKey.decrypt((EncryptedObject) message.Content);

            } catch (PrivateKeyException e) {
                e.printStackTrace();
                System.exit(1);
            }

            // Add the message to the view as a fragment
            Bundle bundle = new Bundle();
            bundle.putSerializable("message", new Message(message.SenderID, message.ChatID, message.TimeSent,
content));
            bundle.putBoolean("fromOtherUser", message.SenderID != fragment.networkerService.user.UserID);

            fragment.getChildFragmentManager().beginTransaction()
                    .setReorderingAllowed(true)
                    .add(R.id.MessageBox, MessageFragment.class, bundle)
                    .commit();
        }
    }

    // Scroll to the bottom of the scroll view
    ((ScrollView) fragment.requireView().findViewById(R.id.MessageBoxScrollView)).fullScroll(View.FOCUS_DOWN);
}

/**
 * Updates the message box with the current contents of the chat following the listen rule architecture
 */
public void updateMessageBox(Message newMessage) {
    // If the user that sent the message is not currently in the hashmap, request the user from the server and add
them
    // Then we can add the message
    if (((MainActivity) requireActivity()).users.get(newMessage.SenderID) == null) {
        JSONObject request = new JSONObject();
        request.put("type", RequestType.GetUser);
        request.put("selector", "id");
        request.put("data", new User(newMessage.SenderID, null, null, null, null, null));

        networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
            @Override
            public void callback(Result result, Object response) {
                ((MainActivity) requireActivity()).users.put(((User) response).UserID, (User) response);

                // Decrypt the message before passing it to the fragment
                String content = null;
                try {
                    content = (String) privateKey.decrypt((EncryptedObject) newMessage.Content);

                } catch (PrivateKeyException e) {
                    e.printStackTrace();
                    System.exit(1);
```

```
                    }

                    // Create a new message object with the decrypted contents
                    Message decryptedMessage = new Message(newMessage.SenderID, newMessage.ChatID,
newMessage.TimeSent, content);

                    // Add the message to the view as a fragment
                    Bundle bundle = new Bundle();
                    bundle.putSerializable("message", decryptedMessage);
                    bundle.putBoolean("fromOtherUser", newMessage.SenderID != networkerService.user.UserID);

                    getChildFragmentManager().beginTransaction()
                            .setReorderingAllowed(true)
                            .add(R.id.MessageBox, MessageFragment.class, bundle)
                            .commit();

                    // Scroll to the bottom of the scroll view
                    requireActivity().runOnUiThread(() -> ((ScrollView)
requireView().findViewById(R.id.MessageBoxScrollView)).fullScroll(View.FOCUS_DOWN));
                }
            }, request));
        }
        else {
            // Decrypt the contents of the message
            String content = null;
            try {
                content = (String) privateKey.decrypt((EncryptedObject) newMessage.Content);

            } catch (PrivateKeyException e) {
                e.printStackTrace();
                System.exit(1);
            }

            // Add the message to the view as a fragment
            Bundle bundle = new Bundle();
            bundle.putSerializable("message", new Message(newMessage.SenderID, newMessage.ChatID, newMessage.TimeSent,
content));
            bundle.putBoolean("fromOtherUser", newMessage.SenderID != networkerService.user.UserID);

            getChildFragmentManager().beginTransaction()
                    .setReorderingAllowed(true)
                    .add(R.id.MessageBox, MessageFragment.class, bundle)
                    .commit();
        }
    }
}
```

Testing

In order to test this application I will be using a virtual machine running android 12.

LoadingActivity

Test	***Expected outcome***	***Was this outcome achieved?***
Load the application *without* a	Redirected from the	Yes

preexisting user account.	*LoadingActivity* to the *NewUserActivity*.	
Load the application *with* a preexisting user account.	Directed straight to the *MainActivity*.	Yes
Load the application with an already authenticated connection running in the NetworkerService	Directed straight *MainActivity* immediately.	No - The program hangs on the *LoadingActivity*.

After inspecting the code I noticed I missed a condition from the *WaitForAuthThread* process in the *LoadingActivity*, which checks if the connection is already authenticated when the application binds to the service. The following condition does this.

```
if (networkerService.authenticated) {
    startActivity(new Intent(LoadingActivity.this, MainActivity.class));
}
```

This fixed the issue and the *LoadingActivity* no longer hangs.

Furthermore, I noticed that the listen rules set during the setup phase of the application were not appearing on the server, and that the listen rule sockets do not appear to be connecting properly. I figured this was due to the fact that I was having the server connect to the client, so a more appropriate way of doing this might be to have the client create a new connection to the server. This lead to the following changes in the server's *ConnectionHandler* class:

```
63  63              if (this.DoHandshake()) {
64  64                  // Open listen rule socket
65  65                  try {
66      -                   int port = (int) this.keyPair.decrypt((EncryptedObject)
                            this.Receive());
67      -                   this.lrSocket = new Socket(new Proxy(Proxy.Type.SOCKS,
                            new InetSocketAddress(this.socket.getInetAddress().getHostName(), port)));
68      -
                            this.lrSocket.connect(this.socket.getRemoteSocketAddress());
    66  +                   ServerSocket ss = new ServerSocket(0);
    67  +                   this.Send(this.clientKeyPair.encrypt(ss.getLocalPort()));
    68  +                   this.lrSocket = ss.accept();
69  69                      this.lrOos = new
                            ObjectOutputStream(lrSocket.getOutputStream());
70  70
71      -               } catch (IOException | PrivateKeyException |
                        ClassNotFoundException e) {
    71  +               } catch (IOException | PublicKeyException e) {
72  72                      e.printStackTrace();
73  73                      this.Close();
74  74                      return;
```

Hence resolving the issue.

NewUserActivity

Test	*Expected outcome*	*Was this outcome achieved?*
Entered the following data into the appropriate fields: Phone number (autofilled): +155583634682 Display name: Nathcat1234😂 Password: OogleBop!4321 Password retype: OogleBop!4321	New user created on server and app is redirected to the loading activity, which follows onto the main activity.	No
Entered the following data into the appropriate fields: Phone number (autofilled): +155583634682 Display name: Nathcat1234😂 Password: OogleBop!4321 Password retype: awadwad	Message is displayed asking the user to retype their password since the entries do not match	Yes

The *NewUserActivity* did not redirect the user to the *MainActivity*. After reading through my code for the authentication section in this class, I discovered that it was setting the *NetworkerService*'s *waitingForResponse* field to false and then loading the *LoadingActivity*, this meant that the *LoadingActivity* saw the service not waiting for a response from the server, and the fact that connection was not yet authenticated, since the request wouldn't have gone through yet, so the *LoadingActivity*, as it should in that situation, loaded the *NewUserActivity*. The authentication request in general was also slightly incorrect. Following is the list of changes I made to rectify this issue, displayed in GitHub's difference format. All changes were to *NewUserActivity*.

```
     167  +                         System.exit(1);
     168  +                     }
     169  +
     170  +                     if (response.getClass() == String.class) {
167  171                           networkerService.authenticated = false;
     172  +                         networkerService.waitingForResponse = false;
     173  +                         System.exit(1);
168  174                       }
169       -                     else {
170       -                         if (response.getClass() == String.class) {
171       -                             networkerService.authenticated = false;
172       -                         }
173       -                         else {
174       -                             networkerService.authenticated = true;
175  175
     176  +                     // If authentication was successful...
     177  +                     if (result == Result.SUCCESS) {
     178  +                         networkerService.authenticated = true;
     179  +
     180  +                         // Update the data in the auth file
     181  +                         try {
     182  +                             ObjectOutputStream oos = new ObjectOutputStream(new FileOutputStream(new
          File(getFilesDir(), "UserData.bin")));
     183  +                             oos.writeObject(response);
     184  +                             oos.flush();
     185  +                             oos.close();
     186  +
     187  +                             networkerService.user = (User) response;
     188  +
     189  +                         } catch (IOException e) {
     190  +                             e.printStackTrace();
176  191                           }
     192  +
     193  +                     /* TODO Notifier routine should be done through listen rule handler now
     194  +                     NotifierRoutine notifierRoutine = new NotifierRoutine();
     195  +                     notifierRoutine.setDaemon(true);
     196  +                     notifierRoutine.start();*/
177  197                       }
178  198
179  199                       networkerService.waitingForResponse = false;
          @@ -187,8 +207,6 @@ public void callback(Result result, Object response) {
187  207                  } catch (IOException e) {
188  208                      e.printStackTrace();
189  209                  }
190       -
191       -                networkerService.waitingForResponse = false;
```

After these changes were made, the issue was resolved and the *NewUserActivity* functioned as expected.

MainActivity

Test	***Expected outcome***	***Was this outcome achieved?***
Clicking on the chats page	Loads the *ChatsFragment*.	Yes
Clicking on the friends page	Loads the *FriendsFragment*.	Yes
Clicking on the invitations page	Loads the *InvitationsFragment*.	Yes

Clicking on the find people page.	Loads the *FindUserFragment*.	Yes

FindPeopleFragment

Test	***Expected outcome***	***Was this outcome achieved?***
Search an existing username *in full*: "I do not exist"	Shows the desired user	Yes
Search an existing username missing the end: "I do"	Shows the desired user	Yes
Search for my own username	Shows *no* results	Yes
Search for a username which does not exist.	Shows no results	Yes
Search with an empty username	Shows message asking you to enter a username first	Yes

InvitationsFragment

Test	***Expected outcome***	***Was this outcome achieved?***
Decline a request	The request is deleted with no further actions	Yes
Accept a request	The request is accepted: ● If a friend request, a new friend should be added to the *friends* page. ● If a chat request, a new chat should be added in the *chats* page.	Yes, to both friend and chat requests.

InviteToChatActivity

Test	***Expected outcome***	***Was this outcome achieved?***
Invite a user to a new chat, testing that the entry fields will not accept no values.	A message should be shown when an empty input is detected asking the user to enter a value. Past this a chat request should be sent to the recipient.	Yes, all points were met. The chat request was sent with valid input, and a message was shown when empty inputs were detected.

Invite a user to an existing chat	A chat request should be sent to the recipient.	Yes

FriendsFragment

Test	***Expected outcome***	***Was this outcome achieved?***
Click on the button on a friend record, then proceed to create an invite.	The friend that was clicked should be sent a chat invite to the chat the user just invited them to.	Yes

ChatsFragment

Test	***Expected outcome***	***Was this outcome achieved?***
Click on chats in the page	The messaging fragment should open	Yes

MessagingFragment

Test	***Expected outcome***	***Was this outcome achieved?***
Is the chat information the same as the one that was clicked? Is the past message history viewable (up to 10 messages)?	Yes	Yes
Send a message with normal characters	Should send the message into chat, displayed in purple.	Yes
Send a message with special characters	Should send the message into the chat, displayed in purple.	Yes
Send a message with no characters	Should *not* send the message.	No
Receive a message with normal characters	Should see the message displayed in green with the correct username above it.	Yes
Receive a message with special characters	Should see the message displayed in green with the correct username above it.	Yes
Receive a message with no characters	Should not see any message displayed.	No

I clearly neglected to implement a validation check for no characters when implementing this part of the application, I don't think this is a serious problem, something I can fix pretty easily.

Simply adding the following if statement to the send message method in the main activity fixed this issue, it just checks if the content of the message is empty before proceeding, if it is it displays a message to the user and ends the method, if it is not empty then it proceeds with the rest of the function.

```
// Ensure that the message is not empty before proceeding
if (messageContent.equals("")) {
    Toast.makeText(this, "Please enter a message first!", Toast.LENGTH_SHORT).show();
    return;
}
```

Notifications

Test	***Expected outcome***	***Was this outcome achieved?***
Send friend request	The user receiving the request should get a notification with the name of the user that has sent them a friend request, regardless of the page of the application they are on and whether or not the application is open.	Yes MessageCat • now New friend request Nathcat1234😂 wants to be friends!
Send chat request	The user receiving the request should get a notification with the name of the chat they have been invited to, they should receive this application regardless of whether or not the application is open.	Yes MessageCat • 1m New chat invitation You have been invited to Hello world MK2
Send message (application open)	The user should only receive a notification if the chat the notification is for is not currently open in the application, otherwise no notification should be shown. The notification should show the name of the chat that the message was sent to.	Yes, but only after the application was restarted after having accepted the chat invitation. MessageCat • now New message You have a new message in Hello world MK2
Send message (application closed)	The user should always receive notifications when new messages are sent into chats. Again, these notifications should contain the name of the chats.	Yes, same as above. MessageCat • now New message You have a new message in Hello world MK2

Evaluation

At this point I have completed the implementation of the initial application and it is time to begin to evaluate what I have accomplished during this process.

This section will begin with the end user testing of the application, this will allow us to gain a better understanding of how people respond to the application, and their unbiased opinion will give me a good insight on how I should proceed with further development in the future.

Android 13

Before I was able to begin end user testing, Google began rolling out *Android 13* to existing Android devices. Reading into some of the changes that were made in this update I felt fairly confident that the application would not suffer any functionality issues as a result, most of the changes were around services and background tasks, but from what I understood they should not affect this application, but I decided to conduct white-box testing again to ensure that this was the case.

I noticed that while the application was open and running with the GUI, the networker service functioned exactly as expected, as it did in *Android 12*, however it immediately closed when the application's GUI was closed. This may be as a result of the way that services and the UI thread work together in terms of processes, unless otherwise specified in the program, a service will execute in the same process as the UI thread, so when the UI thread is closed, this process is killed, along with the service. I cannot say that this is certainly the case, since as far as I am aware this was the case in *Android 12* but it did not appear to affect the application, perhaps *Android 13* simply enforces this rule more ruthlessly upon applications.

In any case this leads to the user not receiving any notifications while the application is closed, which is clearly a problem, since they will then not be aware of any incoming messages or friend / chat requests, so this is a problem I must fix.

The obvious fix is to explicitly state that the networker service should execute in another process; this can be done in the Android manifest file, which describes various properties of the application for the OS' use. However this then means that I would have to create an Inter-process communication interface, which is a lot of hassle and would likely involve a lot of changes to the already existing code base, which is something I would like to avoid wherever possible, since that was the point of using a modular design in the application.

However, in investigating this approach further, I discovered that the Android SDK provides a method for Inter-process communication without having to write an *AIDL* interface, which makes use of *Messengers* and *Handlers*. I implemented this in a test application and tested it in an Android 13 Virtual Machine and the service did appear to run even after the GUI was closed, so this approach may be viable. The only problem is that it will require significant changes to the networker service at the very least, potentially some of the rest of the application.

I believe that this is an acceptable sacrifice, especially since there are other potential benefits to this approach which can be explored in future development, although I won't do anything with them here as my priority in this project is to have the application functioning as expected.

After attempting to implement this fix, I have realised that the changes required to make this work would fundamentally change the application and at this point in the project I don't believe it would be viable to attempt to continue the implementation on the existing code base. Implementing this fix would require near enough a completely new implementation, which is something I don't have time for at this stage.

The problem that made me realise this was a serialisation error. I started implementing the fix by separating the networker service into its own process, and then re-creating the interface between the UI process and the networker service process to use Android's inter process communication methods. I did manage to complete this, and the service was able to function properly, connecting to the server as expected, and continuing to execute even when the UI process was killed. However, when I began to change the UI process' code to use the new interface I encountered a significant number of issues.

The system I had in place originally used a system of callbacks, these callback functions were passed to the networker service along with the request data, and once the request was completed the connection handler would call the callback function, passing in the result code and the response data from the server. These callback functions often use the context of the class they were created in to perform various tasks. For example, in the new user activity, when the user clicks the submit button, the callback function which is called when the new user request completes loads a new activity, using a method which is declared in the parent class of *NewUserActivity*. This kind of context management is handled by Java for us. This works in the original implementation since the context data does not have to be passed over a process, requiring *serialisation*. Now that the networker service is in a different process, in order to pass the request data and callback to the service for it to be handled, it must be *serialised*. During this process, Java takes the context the callback method executes in and serialises it with the rest of the data. So in the new user activity example, the entire *NewUserActivity* class is serialised, this is a problem since the *NewUserActivity* class, along with all of my activity classes, is not serialisable, and contains data which is not serialisable, hence the program throws a *NotSerializableException*.

One thought I had to fix this would be to create an area of shared memory, which could contain all of the contextual data required for the callback function, but because of the way I have declared the callback function I cannot avoid the *NewUserActivity* being serialised with it.

In any case, the changes to the code base that fixing this problem would require is simply too much to complete within the time frame. It would require fundamental changes to the way the application functions, which is something that I do not have the time for at this stage in the project.

Despite this I feel I should point out that the application still functions as expected on Android 12, and it is only new versions of Android that it does not work with.

Here is an example of the problem:

```
networkerService.SendRequest(new NetworkerService.Request(new NetworkerService.IRequestCallback() {
    @Override
    public void callback(Result result, Object response) {
        if (result == Result.FAILED) {
            networkerService.startConnectionHandler();
```

```java
                    runOnUiThread(() -> onSubmitButtonClicked(v));
                }

                // Check if the response is null
                // If this is the case then the entry had duplicate data
                if (response == null) {
                    NewUserActivity.this.runOnUiThread(() -> {
                        Toast.makeText(NewUserActivity.this, "Either your username or display name is already used,
try something else.", Toast.LENGTH_SHORT).show();
                        loadingWheel.setVisibility(View.GONE);
                        getWindow().clearFlags(WindowManager.LayoutParams.FLAG_NOT_TOUCHABLE);
                    });
                    return;
                }

                // Write the data to the auth file
                // ...
```

This is a sample of code from the *NewUserActivity* class, it sends a network request through the *NetworkerService* to create a new user account, then writes the new user data to the authentication data file on the local storage system.

All of the highlighted lines are examples of *contextual execution*. This code is located in a callback method, which is executed by the *ConnectionHandler* when the network request has completed and a response has been received from the server, the highlighted lines contain references to data which is present *in an instance of the NewUserActivity class*, this means that to access this data, the callback would have to have access to an instance of the *NewUserActivity* class. The way it does this is by *capturing* the instance it was created in, This is fine in the original version of the application, since the callback is not serialised at any point, but in order to pass it between processes, the callback will need to be serialised, and since this capture is required for the callback to execute properly, it must be included in this serialisation, and the *NewUserActivity* class is not marked as serialisable, and its instances contain data which is not serialisable. This results in the serialisation errors I saw in implementing this fix.

Using a class called *SharedData* which stored common data in static fields which can be accessed throughout the program was my first thought to solve this problem, however upon researching this I discovered that it was not possible to share static fields between processes, since each process has a separate memory space. This is not something that is specific to Java, all processes share this behaviour, but in other languages like C/C++ which allow more control over the use of memory it is possible to share memory between processes, however this is not possible in Java, or at least not practical.

So to conclude, this is not a problem I can solve while retaining the majority of my existing solution, I would most likely have to completely rethink my approach, which is something I don't have time to implement at this late stage in the project.

End-user testing

Due to the fact that the application does not function properly on devices with Android 13, and my existing testing group consists of users whose device's are running Android 13, it is not really possible to conduct full scale end user testing as discussed in the design phase of the application. However we

can show the product to the stakeholder on an Android 12 virtual machine to get their feedback on the product as it is, and this can then be used in future development to re-develop this application for Android 13.

Following is the general user stakeholder's response to a testing questionnaire:

What was your experience of the actual messaging system? I.e. did you find it easy to use, was it simple to look at, was there too much information displayed, not enough information displayed?

I thought the actual messaging interface was simple to look at and use, although I feel like there were some elements that could be improved, such as the send button disappearing when you send a message. I would also appreciate having timestamps displayed next to the actual messages so that you can see when someone sent them. Another thing which could be useful is changing the colours that user's messages are displayed as, while the current colours are fine, it might provide a better user experience, and it would be clearer whose messages are whose.

What was your experience of the contact management system? (the ability to see your friend's list, send friend and chat invites, and manage invitations you have received from others).

I found the contact management system easy enough to use, and it functioned well. Although I did notice that when a chat invitation is accepted I had to restart the application before I got notifications for messages in that chat. I also would appreciate the ability to remove people as friends, rather than it just being permanent after you have accepted the friend request.

How did you find the performance of the application, was it fast, slow, did it crash a lot?

I didn't really notice any crashes, but the process of sending messages could be fairly slow at times. The rest of the application's performance was okay, just the sending of messages that was slower than I expected it to be.

How easy was the application to use, were there any areas where you felt that usability could be improved?

The application was simple as I requested, and I was able to use it pretty easily, but there might be some people who don't find it as easy to use so perhaps some kind of tutorial would help this.

How secure do you feel your data is when using this application? Do you think the application should be more transparent about how it stores your data and why it requires that data?

I'm not really aware of the technical measures taken to secure my data, but the way you described them to me makes me feel happy that my data is secure, although I think it might be a good idea to put that into the application as well, because I didn't see anywhere that said that my data was being stored securely or offered any kind of transparency as to how this was done.

Do you feel that there were any missing features that you expected to be in an application like this?

I know I said that the application should be as simple as possible, without any of the extra features that other messaging applications have, I do feel like the lack of profile pictures leaves something to

be desired. I also mentioned this earlier but timestamps on messages would also be a good addition, and potentially different colours for different user's messages, not just different colours for you and other people.

Any other comments?

I felt that the GUI was slightly underwhelming, it wasn't bad per say, but I just thought it could be nicer to look at. Nonetheless I do feel like my criteria for a simple messaging application were met, albeit with some potential improvements.

Here is the Programmer's response to a testing questionnaire designed for them:

How secure do you feel your data is given the measures that were taken to secure it?

The measures that were taken are, in my opinion, sufficient for me to feel safe enough to use the application regularly, although there are some potential improvements that could be made, such as using a Quantum safe encryption algorithm, rather than RSA, and implementing salting when hashing passwords. Perhaps even client specific hashing?

Following a review of the code, does its level of maintainability meet your standards?

Yes, the code includes a lot of comments which explain the process, and is designed in a modular and layered manner, which meets my original expectations. Although if the code base is to be replicated on other platforms, it might be a good idea to create some formal documentation of the server program so that other programmers know how to use it and create their own clients without having to analyse the code themselves.

What are your thoughts on the design of the overall application?

The design is effective for Android and other Java based applications, although it might be difficult to write clients for programs that are not written in Java, or at the very least a language capable of understanding and working with Java objects. Java is a cross platform language so this might not be a huge problem, with the exception of development on iOS devices, which might require the development of a kind of wrapper which ports Java objects to Swift or Objective-C objects.

Final evaluation

Evaluation of success criteria

- The application should allow users to communicate text based messages through the internet.

I believe the evidence shows that this criteria has been met. Development testing showed that the application was able to send and receive messages successfully and without issues during the encryption and decryption process.

The stakeholder mentioned that the performance of this part of the application left something to be desired, this is a fair criticism, I'm not sure at this point in time how I would go about fixing this problem, although I would start by attempting to optimise the existing code as much as I could, perhaps there are some unnecessary network requests in there which could be removed or done more efficiently, which would improve performance. I think generally network requests are a sort of bottleneck for performance in this application, I'm not sure there's a lot I could do about that beyond optimising the encryption process and minimising the amount of data transferred by the client and server.

Nonetheless I do believe this criteria is met by my product.

- Minimal data is stored on the client device

Again I believe that this criteria is met, I looked at the data that the application would require to be on the local storage to function properly and included only that data in the local storage, that being user data for authentication, chats that the user is a member of, and the private keys of the chats the user is a member of. I don't think I could reduce the amount of data stored any further, although I might be able to improve performance in some areas by including more data, perhaps a sort of cache for network requests, or more information about a chat, like links between users and their user IDs, which is something that the application has to request from the server every time it opens a chat, regardless of whether or not it has been opened before.

The application and its related data takes up roughly 13.25 MB on my device, which I used for development testing. This is below the target size of 14.6 MB so this criteria is most definitely met. Although the amount of data stored by the application will of course increase over time, it is unlikely it will increase significantly, as it stands the data section of the application takes up 28.67 kB on my device.

- End-to-end encryption between clients and the server, and between clients

This criteria is clearly a success, the stakeholder said that they felt their data was secure (although they were not aware of the technical parts of the implementation), and I am satisfied with the level of security I implemented in transferring data between the client and the server. Messages sent into chats are also encrypted separately using a key pair specific to that chat.

The only security feature I would aim to improve in the future is the use of salting in password hashing. This would be a relatively simple implementation, and one that would improve the security of the application. As the programmer stakeholder mentioned this might also be made client specific, although this does potentially involve some concessions in terms of portability.

- Scalable server design

I believe that the Server design I have created is scalable to a reasonable degree. Although I am not able to conduct full scale end user testing to completely test this claim, I can test how the server handles a large number of handlers on idle, all waiting for connections. The application managed to get through the setup phase with 10, 100, 1000, and 10000 handlers, although I made to decision to stop before it finished setting up with 100000 handlers, since the time it was taking to start the handlers had jumped *significantly*, and my systems memory usage was slowly building past 90%.

With a more powerful system with more memory, and potentially a number of server systems, this could prove to be a scalable design, although I would have to test how the system holds up when these handlers are actually tasked with something.

- Scalable client application

Again I should clarify what I mean by this. I am not necessarily implying that the client application needs to perform well under high traffic, because it doesn't, at least not on the scale of the server. I mean that the application should facilitate future developments. I think that the fact that I was unable to implement a fix for the Android 13 issue doesn't support this, although I do feel that the code I have written is maintainable for Android 12 at the very least, in order to fix the Android 13 issue I would have to rethink a lot of my approach, which is not necessarily a comment on the maintainability of the code but on my approach to the problem.

Code review

I attempted to follow a modular design approach throughout development, and ensured that all of my code was commented sufficiently to illustrate its function. Furthermore, I attempted to follow a similar code style throughout, regarding naming conventions and structure of code.

I believe that I was successful in my efforts, this is the largest project I have undertaken and I believe that some of my best code has gone into creating this system. Looking back at the code there are certainly some places I would like to improve, but these are all things that I can address in later development. For example, throughout the client application I received warnings from the compiler about adding items to *JSONObject*s, I chose not to investigate it further since it didn't appear to affect functionality, but perhaps in future development I should attempt to investigate and address these warnings.

The Android 13 issues also provide a good opportunity to make a lot of these changes, since it would involve re-writing a lot of the application anyway.

I think one of the most significant issues in this application is the fact that I have made it very Java specific. While in some ways this is good, since Java is generally a cross platform language, it does limit me to some degree. Maybe future development should include a new request / response model for the server and client, i.e. rather than sending Java objects they should send data in a more general format. There is also the option of developing a kind of wrapper, as mentioned by the programmer stakeholder. This is a good idea and would not require any significant changes to the server, but may become tedious as the application is expanded into other languages.

Usability

The stakeholder stated in the questionnaire that they found the application to be simple, as they requested, but that other users who are not as technologically proficient might find it difficult to use. This is a fair criticism, there are some parts of the application which I can see could be difficult to understand for such people. A potential solution for this could be to include a tutorial on how to use the application, maybe we don't force users to go through it, but we give them the option to, and highlight where it is if they ever want to use it.

Furthermore, something that was not highlighted during testing was how people with disabilities such as visual impairments might use the application. Android does provide features within the SDK that

allow you to set the values read out by the audio descriptor built into the operating system, and while I did fill these out I did not put a large amount of effort into them. As the application grows this might become more of a problem so perhaps in the future I should work on this and perhaps attempt testing with someone that has such a disability.

While this next point does not necessarily fall under *usability*, it is related to the user experience so this might be the most appropriate category. The stakeholder mentioned that they would want to know more details about how their data is secured. Perhaps we should detail this in the privacy policy, and mention in the application that all chats are secured end-to-end, such as in Whatsapp, a message is displayed at the top of a chat saying that all messages are end-to-end encrypted. I could implement a similar message in this application.

Perhaps also there is the option of adding a default chat for every user which is controlled by a chatbot prepared to answer frequently asked questions, like a live FAQ. This is a feature similar to the *Team Snapchat* chat in *Snapchat*, which is used for distributing information about the application, and for holiday well wishes and such.

Limitations

Following is a list of limitations of the final product, as revealed by testing, or ones I suspect from analysing my code.

- Limited functionality on Android 13 (and potentially newer versions, the newest being Android 14 at the time of writing).
  - The application functions as expected on Android 12, and while the application is open on Android 13, but the *NetworkerService* fails to fulfil its full functionality when the application is closed.
  - This is because the *NetworkerService* and UI thread run in the same process.
  - While this was acceptable in Android 12, it is no longer acceptable as of Android 13.
- Limited portability to other platforms.
  - The server receives and responds with Java objects, other platforms might not be able to use Java and as such implementing a client application on them would be more difficult compared to a pure Java implementation of a client, such as this Android application.
- Limited performance
  - This is most likely due to my implementation of RSA asymmetric encryption, perhaps I should look into more efficient ways to implement this part of the system.
  - This may also be down to the time lag in making a network request, although there isn't really anything I can do to improve that significantly.
- Lacking features in the contact management system
  - The stakeholder mentioned that they thought the application was missing the ability to remove someone as a friend once they had already been added, or the ability to leave a chat once a member of it. This is something that could be addressed in future development.

These limitations will be addressed in future development where possible, although some of them don't necessarily have a definitive fix, for example the limited performance can only really be improved by performing profiling testing on the application to see which parts of the system are taking the longest to execute and look into ways to optimise those processes in particular.

Future development plan

This section will cover things I was unable to implement here, but may still be useful to this project, and should be designated for future development.

1. Android 13 fix
    a. Split *NetworkerService* and UI thread processes
    b. Implement a new method of communication between these processes (using Android SDK's *Messengers* and *Handlers.*
    c. Implement a new callback structure, which does not involve serialisation.
2. Create a new GUI design, potentially using Android's Material API rather than the View API. This would involve a Kotlin implementation but has significant benefits over the View API.
3. New response / request model for the client and server, using a more general format as opposed to straight Java objects.
4. The ability to remove friends / leave chats.
5. Profile pictures
6. Moderation system
    a. This may be quite complex to implement as it would be necessary to decrypt the contents of the messages, perhaps some kind of reporting system which reports the decrypted content of messages from the client side would be a good approach here.
7. Multimedia messages
8. VoIP calling (such as that seen in Whatsapp)

Each of these developments does something to improve the application, whether that be improving the general user experience or the function of the application, or perhaps the portability and ability to develop into other platforms. Regardless, they are all features which I was unable to implement here which are generally key parts of existing messaging systems. The stakeholder mentioned that the lack of profile pictures was disappointing, I also think that if I were to publicly release this application that people would be disappointed by the lack of other features such as a moderation system and multimedia messages, so you will find such features on the list for future development.

Summary

In summary, I believe that this initial stage of development has been a success. The stakeholder feels that I have met their requirements, with some justifiable criticism which can be addressed relatively easily with future development. The application functions as expected on Android 12, and while it may not function completely on Android 13 and newer versions of Android, we cannot deny that the system itself does work well, the server functions well and appears to be scalable, and the encryption process and security of data appears to be well implemented. The current version of the client application could be seen as a precursor to future versions, the development plan for which has been laid out in the previous section.

There is a clear path forward in future development, and I feel optimistic that I can fix the existing problems in the application. I am satisfied with the product I have produced here and the stakeholder appears to be as well. While I may not have been able to create an application that I could release publicly, I am confident that in the future I will be able to develop this into an application worthy of public use.