**COS 491 Senior Project I**

**Android Chat Application**

**Ralitsa Spasova, ID#100086884**

**Author: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , Date: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Supervisor: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ , Date: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**Blagoevgrad, 2016**

**Declaration of authorship**

**Title**: Android Chat Application

**Author**: Ralitsa Spasova

**Abstract:**

**Declaration of authorship:**

“The Senior Project presented here is the work of the author solely, without any external help, under the supervision of Volin Karagiozov. All sources, used in development, are cited in the text and in the Reference section.”

Author:

**Table of Content**

1. Introduction
   1. Overview of project
   2. Overview of technologies
2. Specification and Analysis of the Software Requirements

2.1 Functional Requirements

2.2 Non-functional Requirements

2.3 Use Cases

2.4 Main UML Class Diagram

1. Design of the software solution

3.1 User Interface  
3.2 MySQL  
3.3 ER Diagram  
3.4 Queries  
3.5 Subsystems  
3.6 UML component diagram  
3.7 UML deployment diagram  
3.8 Security Features

1. Implementation

4.1 Technologies used  
4.2 Installation Requirements  
4.3 Other Features

1. Results and Conclusion

5.1 Screenshots  
5.2 Project Success  
5.3 Schedule  
5.4 Problems  
5.5 Future Development  
5.6 Conclusion

1. References

**Introduction**

Project Overview

For my senior thesis in Computer Science I chose to develop an Android application. During my years of study my interest in mobile computing grew and I wished to gain a better understanding of the material. Unfortunately, due to circumstances I was not able to enroll in the offered course, which introduces the students to the concept. Since the senior project can be on any appropriate topic, I decided to use this opportunity gain valuable practical experience, despite the difficulty of starting with a minimal knowledge base on Android development. At the beginning I wanted to design a mobile game of the style “flight shooter”. However, after short research I discovered that the scope would be too difficult if it is built without the help of a game engine and too simplistic with one. Therefore I decided on an instant messaging client instead.

The main reason for choosing this project is that at the current time mobile applications are rapidly growing in popularity and there is a particularly high demand on the market for programmers, experienced in the field. Furthermore, writing software for an Android operating system is a better choice for a beginner than iOS, which has much stricter requirements, the very least of which is access to a MAC laptop to work on.

When browsing through the Google Play Store, one can come across plenty of chat applications, similar in concept to Facebook Messenger or WhatsApp. And that is perfectly natural. In this age of global communication, it is of great importance for groups to have a channel of communication over the Internet while at the same time to preserve their privacy. For example, institutions can use their own chat application, available only to affiliates. But this is far from the only way, in which the chat can be utilized.

When choosing the topic I also had taken into consideration a much more specific benefit of such software. Millions of people worldwide are diagnosed with social anxiety, a psychological disorder, which can make interaction with others incredibly difficult. However, it has been reported that texting is not as stress-inducing as a phone or video call. Furthermore, people with these problems can overcome them easier with practice through a platform, which offers them a sense of anonymity.

Overview of Technologies Used

In order to build an instant messaging application several tools are absolutely crucial to the success of the project. The program was developed on Eclipse Mars environment with installed Android SDK. Work on the project started on Android Studio. However, after experiencing some technical difficulties, I decided to switch to Eclipse. I was more familiar with it from previous coding exercises. As it is already implied, I used Java as the programming language and XML as the markup language. The Java code is responsible for the functionality of all user actions, as well as the interaction between the network connection and the user interface. The layout was written in XML and modified through the graphical interface, which come with the Android library.

An instant messaging application requires a connection to a XMPP server. While there were many options to choose from, I finally decided on Openfire, which I further linked to a MySQL database on my old WAMP server. Both servers are installed locally on my laptop. Openfire also provides its users with a web admin console for managing connections, accounts, groups and chats over a secure port 9090 or 9091.

For the client side, I imported Smack libraries, which allow me to perform XMPP connection actions. Methods range from configuring rooms with specific attributes to sending and receiving messages to managing the user’s roster list. It also helps that Smack API is released by the same online community as the Openfire server.

I also installed Spark chat client, in order to run tests throughout the development stages of the project.

**Specification and Analysis of the Software Requirements**

Functional Requirements

* Visitors should be able to register a new account by supplying a valid username, password and e-mail address.
* Visitors should be able to log into their account with a username and a password, if such exists already
* Logged in users should be able to log out of their accounts
* Users should be able to reset their password through the provided email in the case that they have forgotten it
* Users should have access to a list of contacts, which displays their username, nickname and presence
* Users should be able to add a new contact to the list by entering a valid username and specifying a nickname
* Users should be able to delete a selected contact from the list
* Users should be able to start a private chat instance with a selected contact from the list
* Users should be able to receive an in-app notification when a contact wishes to start a private conversation
* Users should be able to see a list of available public rooms
* Users should be able to create a new public room by specifying a name, subject and a short description
* Users should be able to join public rooms and open a new chat window by selecting from the list
* Users should be able to receive the recent chat history from a room upon joining the room
* Users should be able to delete a public room if they created it
* Users should be able to see a list of available private groups
* Users should be able to request an invitation to join a selected group
* Users should be able to create new groups by providing a name, subject and a short description
* Users should be able to invite other users to a group if they are already a member of it
* Users should be able to receive invitation requests and choose whether to accept or decline them
* Users should be able to delete a group, of which they are the owner
* Users should be able to kick members of a group out, if they have owner or admin privileges

Non-functional Requirements

**Availability**

After small reconfigurations of the network connection, the project can be uploaded to an Android device to be run. For lack of resources I relied on the emulator to test my code.

**Security**

The Openfire XMPP server provides the backbone of the application’s security management. A connection from a client to the server must first be authenticated and authorized before it can be accepted and finalized.

The passwords, used in user log in, are saved in the database in an encrypted form for extra protection. A password is not retrievable by anyone and that is why the user must reset it and then input a new one.

**Compatibility**

The minimum Android API level, which would support the application, is chosen to be compatible with relatively new releases of the OS.

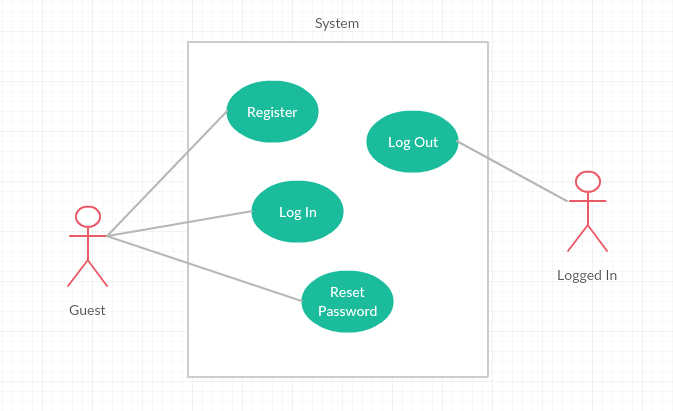
**Usability**

The user interface is simple and easy to navigate. Instructions in the form of pop-up messages are provided in case of errors to point the users to a solution.

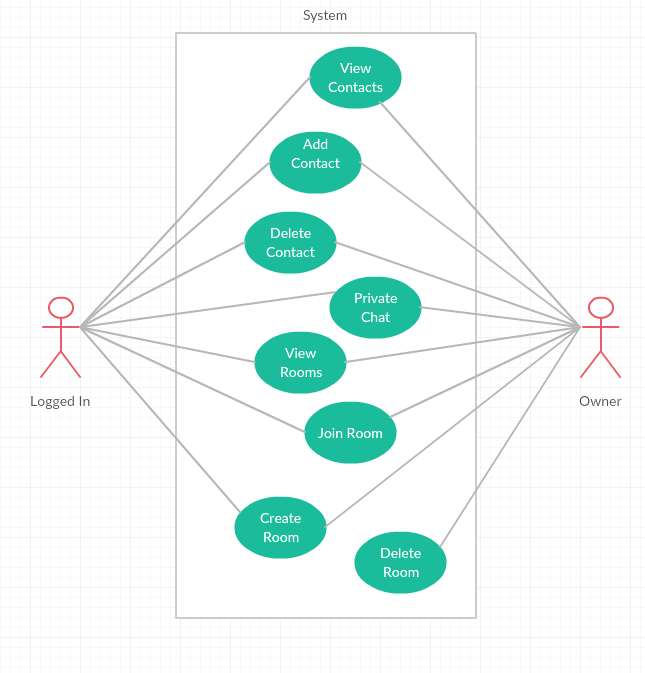
Use Cases

In order to better illustrate the features, which are offered by this project, I have separated the users into specific roles. A guest is someone who is not currently logged into an account. A normal user is the default currently open account on the application. A title of owner is given to the users who have created either a public room or a private group and it is only for that particular room. An admin is the default moderator of all available rooms and has the same privileges as an owner. Owners and admins have access to all of the features as a normal user and added to that is the option to delete rooms and ban members.

Below is a use case diagram with the basic cases, related to account management.

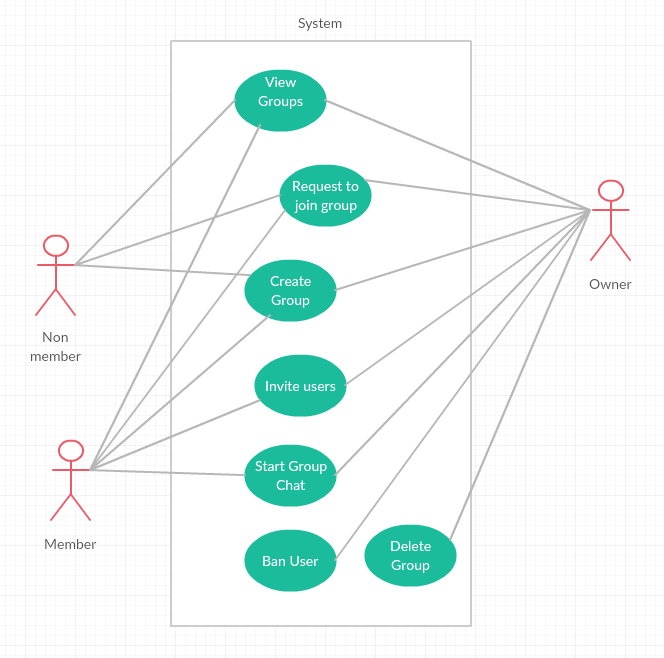


Use case diagram for a normal default user, who is logged into his account, and a user who has created a room. Both have the same privileges with the exception that only an owner of a room or an administrator can delete it.

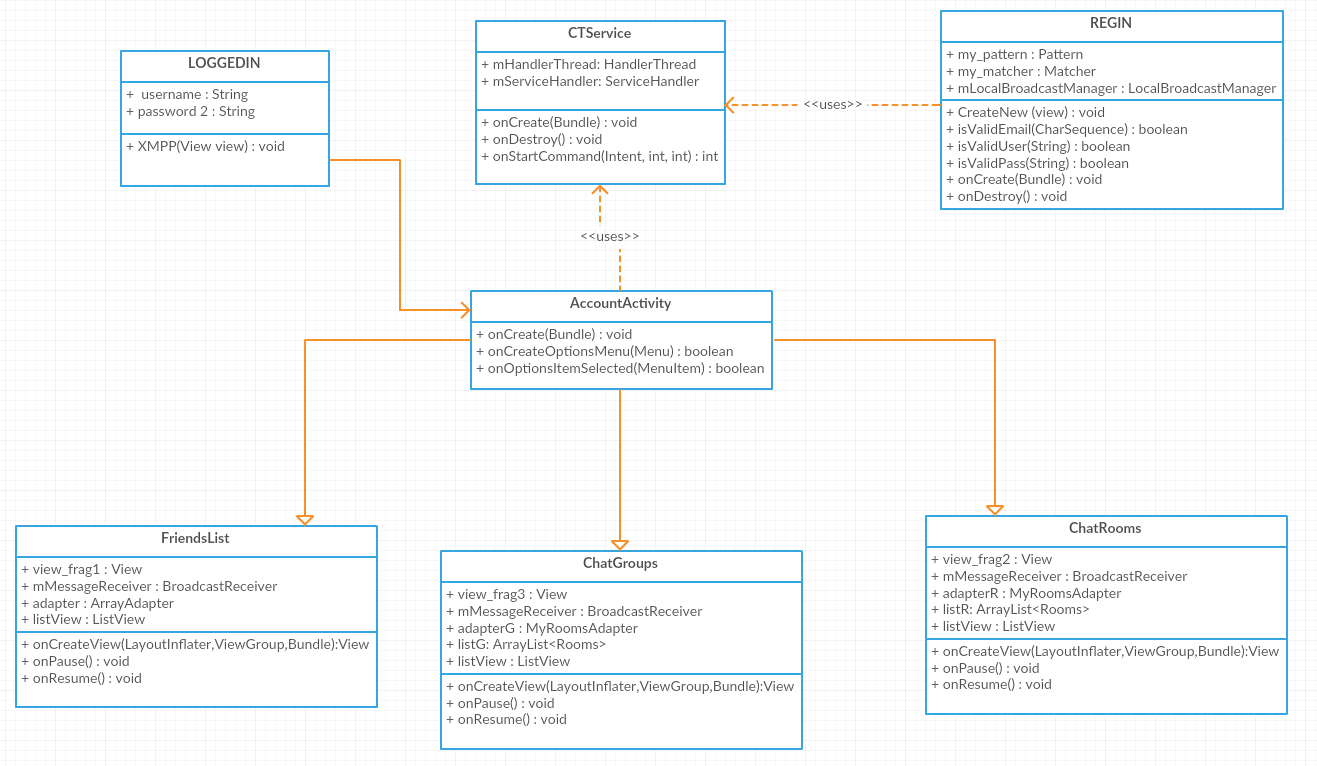


Another case diagram for group management

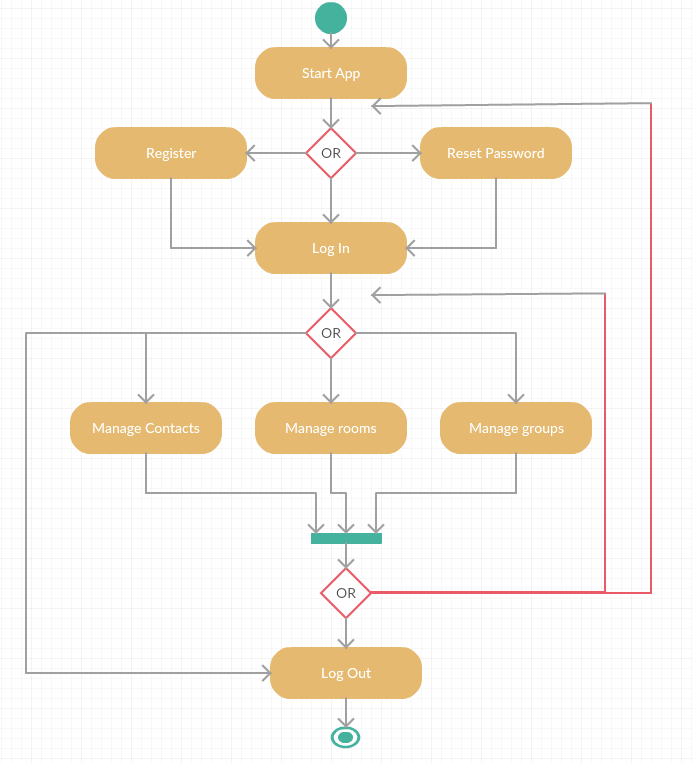
In order to enter a group chat, a user has to be invited and become a member. Owners and admins can not only destroy the groups but also kick a current member out.



Main UML Class Diagram



Activity Diagram



**Design of the software solution**

User Interface

In this section I will go through the design of the user interface. I tried to keep the design simple and the navigation straightforward.

When the user first starts the application, they will see a Welcome screen. The user can then choose to either log into their already existing account or to register if they not have an account.

Log In

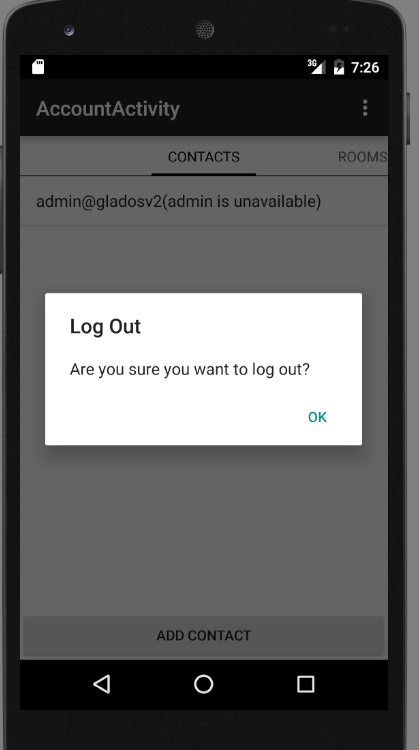
On this screen the user is presented with two fields, where he input his username and password. If the credentials are correct and the username exists on the server, the user will be redirected to their main account activity. If not, an error message will be displayed, providing further information about the error. In addition to that, there is a link in case of a forgotten password, through which the user can request an email with the reset link to the address with which he had registered.

Register

If the user wants to register a new account, he needs to provide a valid username, password and email. If the creating of the account is successful, the user will be redirected to the Log In activity, so that he can enter. If one or more of the entered values is not valid, a Toast message will appear with the details. An error will also occur if the password and the confirm password are not identical.

Account Activity

After the user successfully logs into his account, he is redirected to the main screen, which contains three fragments. The tabs menu is at the top, through which the user can navigate their account and switch between the contacts, rooms and groups fragments. Above the tabs, in the right corner, is the expandable options menu. There is only one item present, the log out action. When the user clicks on it, a dialog box will pop up, asking for confirmation before the account session is terminated. Once that happens, the user is brought back to the Welcome screen.



Contacts

The first tab of the account activity contain a list of contacts, each item containing the username of the contact, as well as a nickname, assigned to them by the user. It also shows the current presence status of the person. A contact can either be available (online) or unavailable (offline). The list dynamically updates whenever the presence of a contact changes. The user can click on an item from the list. A pop-up with two options appears, one to start a private chat with the contact and the other- to delete them from the list. When the user selects the former, a new activity starts, representing the chat window. When he clicks on the second button, the user has to confirm the removal of the contact before it is executed. Aside from the list, there is also a button at the bottom for adding a new contact. When it is clicked, a box appears asking the user to enter the person’s username, accompanied by a nickname of his choice. An error occurs if the given username does not correspond to an account on the server. After successful addition or removal of a contact, the list updates itself.

One-to-one Chat

The private chat window is opened in a new activity when the user selects the “chat” option on a contact from the list. The design of the page resembles the normal basic chats, which can be found in other applications. Most of the screen space is dedicated to displaying the messages between the chat participants. At the bottom there is a text field, in which the user writes the message, and a button next to it, which sends it to the partner and displays it in the chat log. The user does not have any other options available and if he wishes to return to the main account activity, he can simply use the back button.

Rooms

The second fragment contains a list of available rooms, which can be joined by any account. Each item on the list has information about the room, or more specifically, the room’s subject, full JID and a short description. Clicking on an item will directly open a chat window. As with the Contacts tab, there is also a button at the bottom of the page. Once clicked, a dialog box appears and asks the user to enter a name for the new room. This name serves as identification and is the first part of t

he room’s full JID. The user must also enter a subject and a short description. Clicking the “add” button will create a new room on the server and it will appear in the list. After creating the room, the user automatically joins it and is assigned the affiliation of “owner”.

Groups

The third and final fragment is incredibly smilar to the other two in its layout, having once again a list and a button. The list displays the available groups, along with the same information as the rooms’ list. The button performs the same functions as well. However, in this case the user will not be able to immediately enter a group upon selecting it. Since groups are restricted to only members, a user who wishes to join one first needs to send a request. Only after the user has been officially invited can he enter and send messages to the group.

Public Chat

The public chat window is opened when a user enters either a room or a group. Its appearance is the same as the one of the private chat, though there are a few more extra features. Upon joining a public chat, the user will see the recent discussion history. That means that he can read the most recent messages, which have been posted by other members. Depending on the affiliation of the user, he can expand the options menu in the top right corner. If he is an owner or an admin, he will see an option to delete the room or group. If he has opened a group chat specifically, he will also have the option to kick a member out. When he selects it, a pop-up is displayed with a list of the current members from which to choose. In addition to that, the owner can send invitations to others, thus allowing them access to the chat.

The Network Service

The most complicated part of the project was implementing the network service, through which all other actions would be processed. After adding the necessary Smack libraries to the dependency list and setting up the Openfire server, a connection needs to be made for the client session to start. Managing the network was more complicated than I had initially anticipated. Unlike connections between a browser and a WAMP database, here it needs to remain open for the entirety of the user session. If it were static, there would be possibility of having a real chat with another account. Instant messaging relies on packets being transmitted constantly between the server and clients.

There are several consideration that need to be taken into account, in order to maintain a persistent connection. First of all, the methods related to network and database operations cannot be executed on the main thread of the Android application, since it si reserved for control of the user interface. The developer’s guide points to a few soltions to that problem. Either a new thread needs to be instantiated, which would run the tasks in the background and post the results on the UI thread, or a developer can use AsyncTask. The second option executes asynchroniously to the main processes, so it doesn’t obstruct the executing of other functions. And although it sounds like a good choice, AsyncTask is not designed to perform long-running tasks.

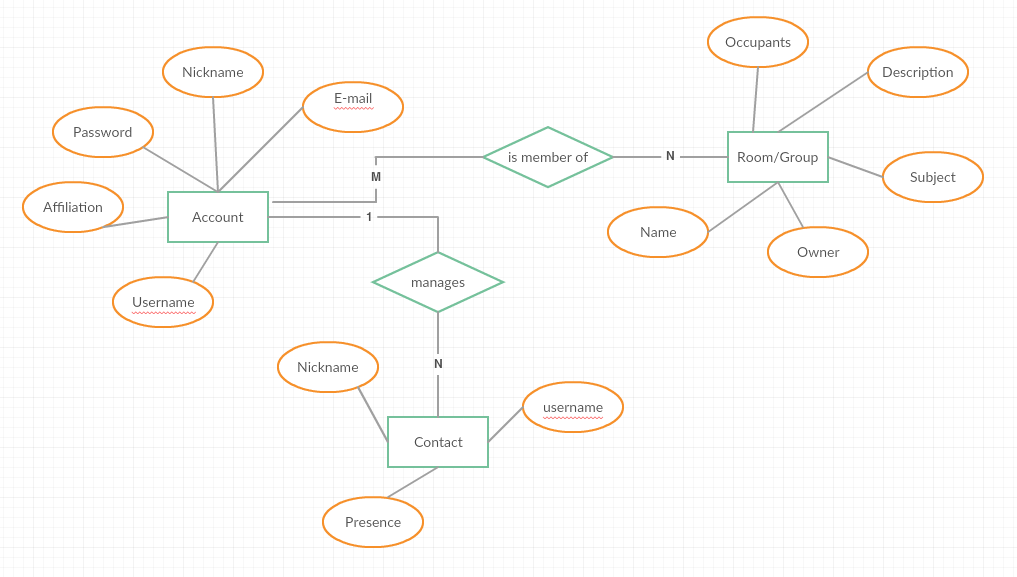
Anotehr thing, the background thread needs to maintain itself even if the user avigates in an dout of different activities. That kind of processing independence is achived with a Service, which can be modified to keep its instance alive even if the application closes unexpectedly. Unfortunately, the code in services is still run on the UI thread.

With these points in mind, I chose to combine the two elements. I created a Sercive class and in it I created a new thread. A handler needs to be defined and set up with the thread’s Looper, in order to move tasks to it. In this way, the thread will run until the user logs out of their account.

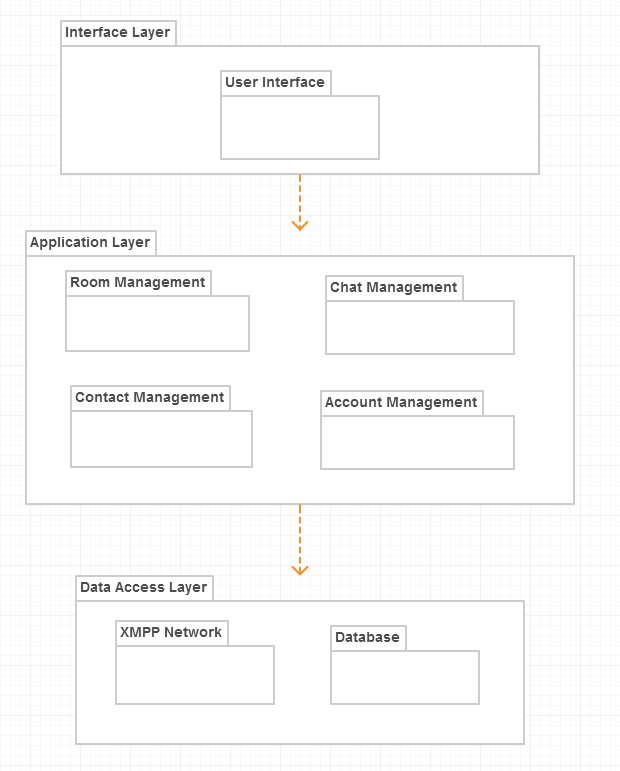
After we have the thread set up, operations need to be included in the run method. This is where the Smack API Before executing anything else, I have described the configuration details, which will be used to connect to the server. When the client tries to connect with username and password, the server must first check the credentials and authenticate the session. Only after that can the user have access to the other features.

Coomunication between the acivities an dteh network service is a little tricky. For this purpose, I used broadcasts , which serve to send data and intents between the threads. Registered broadcast receivers are listening for a intent that matches their filter. Whenever the user selects a certain action, a broadcast is sent to the service, where the appropriate receiver encapsulates the code to be run. The communication works both ways and is the best solution when dealing with asynchronous processes.

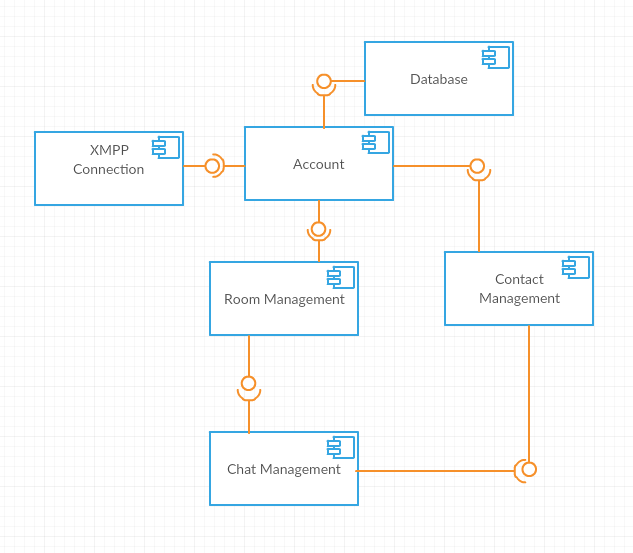
ER Diagram



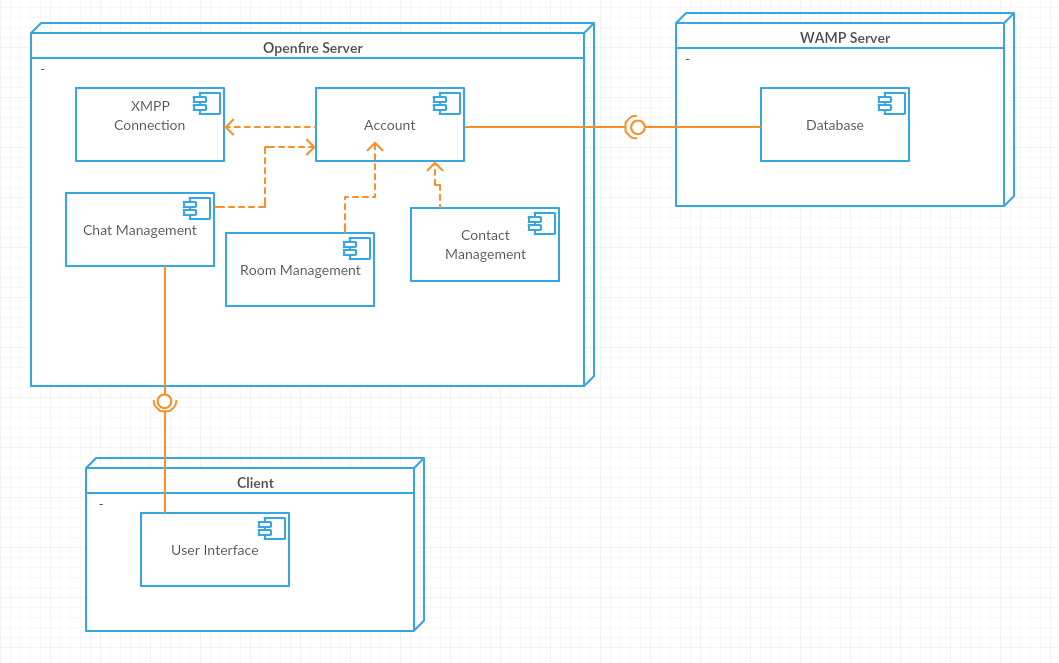
Subsystems Diagram



Component Diagram



Deployment Diagram



Security Features

**TLS/SSL**

The openfire server provides support for the Transport Layer Security and Secure Sockets Layer protocol. These certificates ensure a safe transfer of packets over the network. The connection is reliable because each message transmitted includes a message integrity check using a message authentication code to prevent undetected loss or alteration of the data during transmission. The identity of the communicating parties can be authenticated using public-key cryptography. The connection is private because symmetric cryptography is used to encrypt the data transmitted. When configuring the connection, Smack offers the developer to choose the security mode to be used when a client attempts to connect to the server:



**Password Encryption**

When a user registers a new account, their desired password is saved in the database in an encrypted format. The hash prevents anyone from retrieving the password, not even the administrator. When a user forgets his password, he can request to the server, who then sends an email with a link to reset the pass to the address, which the user provided when signing up their account initially.

**Authorization**

This is the process, in which the server compares the input to the database of currently registered accounts.

**Authentication**

The client connection needs to be authenticated by the server before being given a final approval and access to information.

**Implementation**

Technologies Used

**Eclipse IDE & Android SDK**

There are two popular choices for an Android development environment: Eclipse and Android Studio. In the beginning of the semester I chose to use Android Studio, since it provides more support and a more native approach to integrating elements into an application. I spent a certain amount of time experimenting with the software and following online tutorials, I was starting to doubt my decision. Eventually I switched to Eclipse. The downside of it is that Eclipse is not specifically targeted for mobile computing. Therefore, I had to additionally install an Android SDK and add support libraries to the dependencies of the project.

**Java**

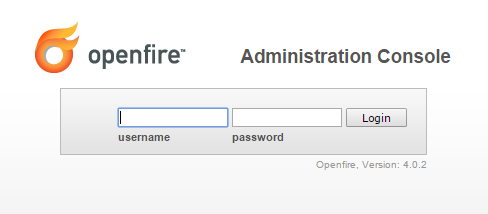
The programming language that is used to develop the application is Java. The business logic layer of the project consists of different classes, which interact with each other. The connection and maintenance of the client session is also performed by a java class. It has been some time since I have used Java code and admittedly it took some time to remember the structure of the language. Luckily, it is object-oriented the same way that C languages are and the logic behind the operations was not particularly hard to understand or apply to this specific situation

XML

While Java is used to write the functional part of the code, this markup language is responsible for designing the user interface. The elements on each screen, including dialog boxes have been arranged and customized with XML. The environment has an option to modify the graphical layout through without the need to write out the XML markup completely manually.

**Openfire 4.0.2**

One of the core ingredients in making a chat application is the XMPP server, which will be responsible for passing data to and from the database and client. Openfire is a real time collaboration (RTC) server licensed under the Open Source Apache License. It uses the only widely adopted open protocol for instant messaging. When researching the technologies I came across many backend service providers, which offered developers many ready-to-go features. And although that sounded appealing, they were not suited for the project, as it eliminated a big chunk of the work-load and rendered the idea of a thesis null. I chose Openfire because it was best suited for my needs. not only does it allow me to locally host a chat server on my machine, but gives me access to a web administration console. The admin can enter it through the address with his username and password:



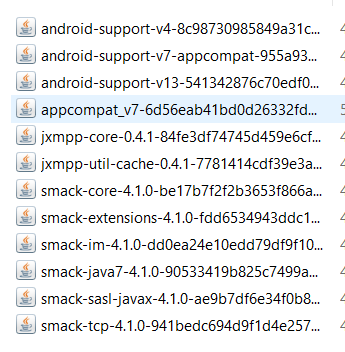
Once logged in, the admin has access to information about the configuration of the server and certain settings, which can be modified. The admin can search for registered accounts and update some of the attributes, assigned to them. The same is true for rooms. Through the console active client sessions can be monitored. Openfire has a plug-in interface, thorugh which the admin can add extra features to the server. For example, there is a plug-in, which monitors and archives one-to-one conversations.

**MySQL**

When setting up the Openfire server on my laptop, I was presented with two options fro a database. I could either use an embedded one, hosted on the server itself, or I could link an external database. I went with the latter, choosing to use the database I had already on my WAMP server.

**Smack API 4.1**

Smack is an Open Source Java library for writing XMPP based clients. It provides access to the core XMPP functionality as well as many of the XMPP extensions. The libraries have actually been released by the same software community as the Openfire server, so it felt like the obvious choice to use them in unison. A lot of important methods from the application are imported from the interface, such as ones managing the XMPP connection session, the roster manipulation, the creation of single or multi user chats. Below is the full list of Smack and Android support files I had to include to the project build.

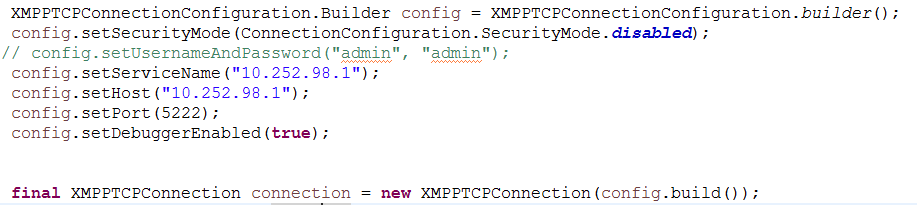


**Spark**

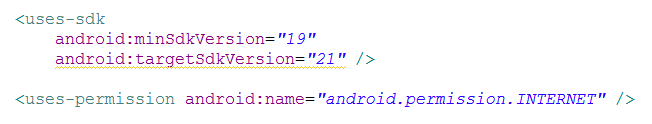
As with Smack and Openfire, it is developed and updated by the same group. Spark is a ready client desktop application, which I used in order to test my code.

Installation Requirements

Setting up the application on an Android device is relatively simple. The first step of the process is to double-check the configuration of the XMPP connection in the code. I have used my public IP address, instead of the generic “localhost”. It should look like this:



Next, we need to find out if the physical device we want to run it on will support the application. During the semester I used a virtual Nexus One emulator to test the output of the code. More information on the minimum target APIs can be located and edited in the Android manifest file:



Test the program on a real device can be performed by plugging it in to the host machine and enabling USB debugging. Once the project has been successfully built, Eclipse generates an .apk file in its directory folder. Transferring that onto the device should allow the user to access the app from anywhere, provided there is an active internet connection.

All of the necessary files for running and testing the application are located on the attached CD.

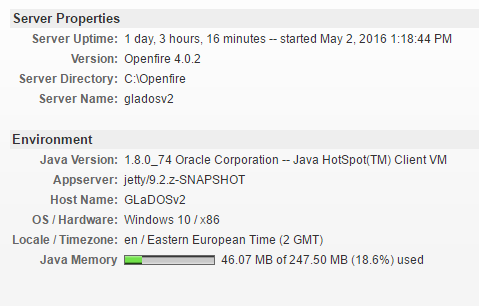
Extra Features

Openfire Admin Panel

As stated earlier, one benefir of choosing Openfire is the web console, which comes with the installation onto the host machine. It is accessed though a special port. In my case, the URL address looks like this:



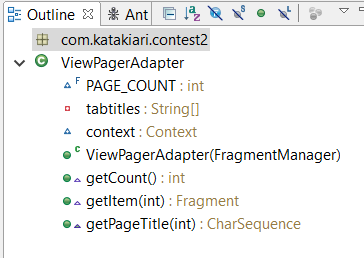
The admin needs to log in with correct credentials. He is granted access to a panel, thorugh which he can manage a variaety of settings and data. The admin can manage certificates, look up accounts, change room settings, monitor conversations, and much more. He can also get a summary of active sessions, alog with their resources. The panel allows the admin freedom to safely and quickly customize the network to individual preference or project specifications.



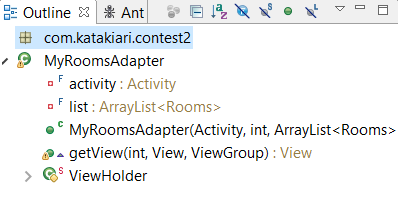
Alongside the default features, the console uses a plug-in interface. The admin can browse through a list of available plug-ins, which offer more control over a specific part of the network. I have implemented some of these plug-ins, such as the monitoring service. It allows archiving of conversations, something that is not supported by the server naturally. Implementing the feature on the client-side with Smack is not as simple, since there are currently no reliable extensions.

View Adapters

Sometimes communication between the View and Controller requires the use of a special class of the Adapter. In the project I have used two main types of adapters: ViewPagerAdapter and ArrayAdapter. The first extends FragmentPageAdapter and as the name suggests, it serves as an interface between the main account activity and the three child fragments. The view for the account activity is essentially an action bar with a title and an options menu and a Pager, which creates the tabs for navigating the fragments.



MyRoomsAdapter and MyChatsAdapter both extend an ArrayAdapter but the arrays are of two different user-defined types. The chat adapter is used when displaying messages from a conversation or group chat in the list view, while the adapter for rooms is used for teh custom list view in the rooms and groups fragments.



**Results and Conclusion**

Work Schedule

* **Week 1-2**  
  Choosing topic
* **Week 3-4**

Topic research  
Technologies research

* **Week 5-6**

Technologies research

Getting acquainted with Android Studio, Eclipse

Smack API

* **Week 7-8**

Setting up Openfire server  
Setting up MySQL database

Setting up connection

* **Week 9-10**

Account registration/login

Roster/Contacts management

* **Week 11**

Room/Group management

* **Week 12**

Private Chat

Public Chat

* **Week 13-14**

Writing documentation

Adding finishing touches

Results

At the beginning of the semester when I was choosing the topic of the project I had underestimated the amount of work a fully functioning chat app would require. What’s more, I had never developed anything for Android before this. The initial scope of the desired project was too big and complex. I spent the first part of the semester searching the ewb for tutorials and examples and videos. It was a necessary process, although it took more time than I expected. Some of the features that I had initially included in my plan were eventually dropped or simplified, of fear that the project would not be completed nor be functional by the deadline. The feature for looking up a contact’s GPS location was not implemented. Although I had to adjust the requirements several times to fit my schedule, I believe the end result is still complex enough to be regarded as a successful completion of the project.

Problems and Solutions

As I had already mentioned. The initial idea for the application turned out to be beyond my capabilities for the given time period. And as I was still utterly inexperienced with mobile computing, there were bound to be some obstacles along the way.

The first bigger challenge came when I was doing research into the different technologies, which I could use. I was amazed at the number of companies that provided back-end services for instant messaging. Usually, this would be a good thing, though for an academic project it is useless, as using them would require very little background knowledge of computer science. And because of the IM app’s popularity, There are dozens of features that could be included. I had to spend extra time on choosing the technologies, which would best suit my needs.

Another problem occurred during the process of setting up the server. Openfire seems to be experiencing a bug, which does not allow for an admin to log into the web console even if the credentials are correct. I had to go through this particular step several times to configure it properly.

The most time-consuming and nerve-wrecking obstacle, however, was connecting to the server from the client application. I had to scrap my existing project and start fresh, since the network connectivity error persistent after a week of trying to find a solution. In the end, I went with a different approach to handling the necessary operations and it appeared to have fixed the issue.

Future Development

Even though I am satisfied with the final result, I wish I could have implemented some more functionalities to the program. If I were to continue work on this application later on, I would include those features, which I had to leave out now, such as a profile page for accounts, which would show their geolocation and a profile picture of their choice. Push notifications was also something I wanted to add but the it is Google Messaging that provides the service and it is not inherently integrated into an openfire server. The notifications would be easier to implement if I had chosen other back-end Google services to go along with it. It is something I would keep in mind for future projects.

Despite the monitoring plug-in, Smack does not posses the capabilities for easy access to archived private conversations between two users. The chat window does not provide any discussion history like with the groups but it is definitely something I would like to add in the future, possibly after more research into accessing the MySQL database directly.

Conclusion

While working on the project during the semester, I had several issues with the development process, which I had to overcome. Trying my hand at Android computing and building a chat application from scratch as my first mobile project was definitely not an easy task. Yet I do not regret choosing this topic, as it taught me valuable experience, which will be necessary for a future career in software development. In the process of intense learning, I gained vast knowledge on many components of an Android application, including specific methods and algorithms, dedicated to combining long-running background processes with a simple user interface. All in all, this project was not only interesting to work on but was instrumental to teaching me the basics of networking and Android development.

**References**

StackOverflow

Android Developers

Ignite Realtime

Smack Documentation