

Change History:

Date:	Version Update:	Description:
2013/06/28	1.0	Document created.
2013/06/28	1.1	Purpose added
2013/06/28	1.2	Document Conventions added
2013/06/28	1.3	Project Scope added
2013/06/28	1.4	References added
2013/06/28	1.5	System Description added
2013/06/28	1.6	Overall Architecture added
2013/06/28	1.7	Architectural Pattern added
2013/06/28	1.8	Architectural Strategies/Tactics added
2013/06/28	1.9	Reliability added
2013/06/28	2.0	Scalability added
2013/06/28	2.1	Modifiability added
2013/06/28	2.2	Security added
2013/06/28	2.3	Performance added
2013/06/28	2.4	Usability added
2013/06/28	2.5	Integrability added
2013/06/28	2.6	Maintainability/Flexibility added
2013/06/28	2.7	Constraints added
2013/06/28	2.8	Reference Architecture added
2013/06/28	2.9	Technology and Framework Selection added
2013/06/28	3.0	Build Tools added
2013/06/28	3.1	Details of the Subsystem added
2013/06/28	3.2	Scenarios added
2013/06/28	3.3	Logical View added
2013/06/28	3.4	Development View added
2013/06/28	3.5	Physical View added
2013/06/28	3.6	Policies added
2013/06/28	3.7	Open Issues added
2013/06/28	3.8	Glossary added
2013/06/30	3.9	Purpose updated
2013/06/30	4.0	Document Conventions updated
2013/06/30	4.1	Project Scope updated
2013/06/30	4.2	References updated
2013/06/30	4.3	System Description updated
2013/06/30	4.4	Overall Architecture updated
2013/06/30	4.5	Architectural Pattern updated
2013/06/30	4.6	Architectural Strategies/Tactics updated
2013/06/30	4.7	Reliability updated
2013/06/30	4.8	Scalability updated
2013/06/30	4.9	Modifiability updated
2013/06/30	5.0	Security updated
2013/06/30	5.1	Performance updated
2013/06/30	5.2	Usability updated
2013/06/30	5.3	Integrability updated
2013/06/30	5.4	Maintainability/Flexibility updated
2013/06/30	5.5	Constraints updated

2013/06/30	5.7	Reference Architecture updated
2013/06/30	5.8	Technology and Framework Selection updated
2013/06/30	5.9	Build Tools updated
2013/06/30	6.0	Details of the Subsystem updated
2013/06/30	6.1	Scenarios updated
2013/06/30	6.2	Logical View updated
2013/06/30	6.3	Development View updated
2013/06/30	6.4	Physical View updated
2013/06/30	6.5	Policies updated
2013/06/30	6.6	Open Issues updated
2013/06/30	6.7	Glossary Updated
2013/08/22	6.8	Purpose updated
2013/08/22	6.9	Document Conventions updated
2013/08/22	7.0	Project Scope updated
2013/08/22	7.1	References updated
2013/08/22	7.2	System Description updated
2013/08/22	7.3	Overall Architecture updated
2013/08/22	7.4	Architectural Pattern updated
2013/08/22	7.5	Architectural Strategies/Tactics updated
2013/08/22	7.6	Reliability updated
2013/08/22	7.7	Scalability updated
2013/08/22	7.8	Modifiability updated
2013/08/22	7.9	Security updated
2013/08/22	8.0	Performance updated
2013/08/22	8.1	Usability updated
2013/08/22	8.2	Integrability updated
2013/08/22	8.3	Maintainability /Flexibility updated
2013/08/22	8.4	Constraints updated
2013/08/22	8.5	Reference Architecture updated
2013/08/23	8.6	Technology and Framework Selection updated
2013/08/23	8.7	Build Tools updated
2013/08/23	8.8	Details of the Subsystem updated
2013/08/23	8.9	Scenarios Diagrams updated
2013/08/23	9.0	Logical View Diagrams updated
2013/08/24	9.1	References updated
2013/08/24	9.2	System Description updated
2013/08/24	9.3	Overall Architecture updated
2013/08/24	9.4	Reliability updated
2013/08/24	9.5	Modifiability updated
2013/08/24	9.6	Integrability updated
2013/08/24	9.7	Usability updated
2013/08/24	9.8	Maintainability /Flexibility updated
2013/08/24	9.9	Constraints updated
2013/08/24	10.0	Details of Subsystem removed
2013/08/24	10.1	Index page updated
2013/08/24	10.2	Login Use Case Added
2013/08/24	10.3	Login Use Case: Services Contract Added
2013/08/24	10.4	Login Use Case: Functional Requirements added

2013/08/24	10.5	Login Use Case: Activity Diagram added
2013/08/24	10.6	Messaging Use Case added
2013/08/24	10.7	Messaging Use Case: Services Contract added
2013/08/24	10.8	Messaging Use Case: Functional Requirements added
2013/08/24	10.9	Messaging Use Case: Activity Diagram added
2013/08/24	11.0	View Contact List Use Case Added
2013/08/24	11.1	View Contact List Use Case: Services Contracts added
2013/08/24	11.2	View Contact List Use Case: Functional Requirements Added
2013/08/24	11.3	View Contact List Use Case: Activity Diagram added
2013/08/24	11.4	Exit Use Case added
2013/08/24	11.5	Exit Use Case: Services Contracts added
2013/08/24	11.6	Exit Use Case: Functional Requirements added
2013/08/24	11.7	Exit Use Case: Activity Diagram added
2013/08/24	11.8	Technology and Framework Selection updated
2013/08/24	11.9	Open Issues updated
2013/08/24	12.0	Glossary updated
2013/08/24	12.1	Policies updated
2013/08/24	12.2	Document Conventions updated
2013/08/24	12.3	Technology and Framework updated
2013/08/24	12.4	Login Use Case. View Contact Use Case, Messaging Use Case and Exit Use Case updated
2013/08/24	12.5	Exit Use Case: Activity Diagram updated

Table of Contents:

Subject:	Page:
1. Introduction	4
1.1 Purpose	4
1.2 Document Conventions	4
1.3 Project Scope	4
1.4 References	4
2. System Description	6
3. Overall Architecture	7
3.1 Architectural Pattern	7
3.2 Architectural Strategies/Tactics	8
Reliability	8
Scalability	8
Modifiability	8
Security	8
Performance	9
Usability	9
Integrability	9
Maintainability/Flexibility	9
Constraints	10
3.3 Reference Architecture	10
3.4 Technology and Framework Selection	10
3.3 Build Tools	10
4. System Diagrams	11
4.1 Login Use Case	11
4.1.1 Services Contract	11
4.1.2. Functional Requirements	11
4.1.3 Activity Diagram	12
4.2 View Contact List Use Case	13
4.2.1 Services Contract	13
4.2.2. Functional Requirements	13
4.2.3 Activity Diagram	14
4.3 Messaging Use Case	15
4.3.1 Services Contract	15
4.3.2. Functional Requirements	15
4.3.3 Activity Diagram	16
4.4 Exit Use Case	17
4.4.1 Services Contract	17
4.4.2. Functional Requirements	17
4.4.3 Activity Diagram	17
5. Policies	18
6. Open Issues	18
7. Glossary	18

1. Introduction

1.1 Purpose

The purpose of this document is to provide our client with a high level overview of the architectural strategies or tactics and patterns that will form a basis for the development of the Latex Chat Application. The overall outline of these concepts and how they are implemented will provide our team with a means to achieve the given set of requirements as previously agreed upon in the Requirements and Design document.

1.2 Document Conventions

- Document Formatting: LaTeX
- UML Diagrams: Diagram Designer, Visual Paradigm

1.3 Project Scope

The aim of the project is to develop an open source android XMPP chat client which supports the embedded LaTeX base equations which are rendered as images. LaTeX based equations will be rendered on the handset to produce mathematical equations. Our system will also provide the ability to edit and correct equations before sending.

The application will provide a similar functionality to yaxim. Exchange of images and mathematical expressions will be possible through our software solution. The TeXchat application will have the ability to show a preview of the entered text and also export conversations accompanied by their mathematical expressions into a LaTeX file.

1.4 References

- Mr. Will van Heerden.
- MSDN Authors. Chapter 3: Architectural Patterns and Styles.
[Online] Available at: <http://msdn.microsoft.com/en-us/library/ee658117.aspx>
[Accessed 21 June 2013].
- Android Authors, 2013. Android NDK.
[Online] Available at: <http://developer.android.com/tools/sdk/ndk/index.html>
[Accessed 18 August 2013].

2. System Description

The goal of the application is to provide a chat service that will allow the users to exchange message and also to send mathematical equations in a rendered format. The application is intended to provide a better and more usable mobile version of LaTeX chat applications.

Support for Latex and Mimetex Libraries

The final application will have to make use of a Latex based library for the rendering of equations as images on the handheld device. For this reason we have implemented support for a Mimetex library, through the use of the Android NDK (Native Development Kit), which allows us to embed the native C/C++ code of the Mimetext library, in the source code of this release.

Messaging

Our final goal is for messaging to be possible between multiple clients, and the support for sending Latex based equations, rendered on the client side and displayed as images on the handheld device. In this release however, our server should support two initial users, and have capabilities for them to send plain text messages and display these in a easy to understand manner. These messages should be stored statically on the device for later retrieval. For this purpose a SQLite database was implemented.

Login

A user should be authenticated by some type of login component. This has been implemented during the initial phases of development of this application. The server that our application uses provides the basic functionality for this authentication component, and uses a username and password based authentication method.

3.Overall Architecture

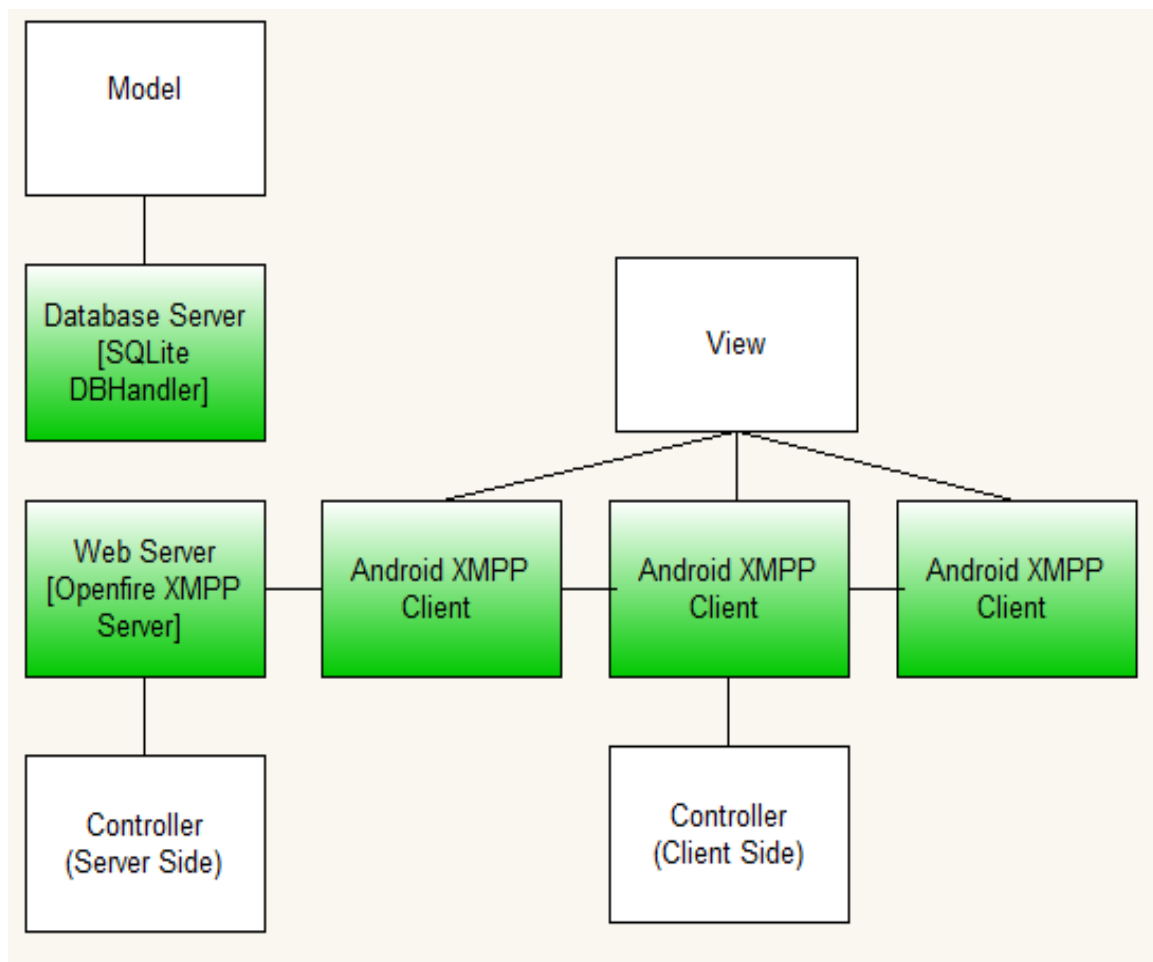
The chosen architectural frameworks chosen for the application will be the Model View Controller(MVC) architecture in combination with the Client-Server architecture. The platform on which the system will be developed in Java and XML

The MVC architecture will allow us to separate the view (the user interface of the Android application itself), controller (the actual coding that will handle the LaTeX rendering and the message sending and connection of clients to the server) and the model (the database that will be found on the server given to us).

The Client-Server architecture will be present when it comes to the actual connection of the clients to the server. This is where a lot of contention will be present. The Client-Server architecture will be encapsulated inside the MVC architecture.

3.1 Architectural Pattern

As mentioned previously our system will be using the MVC architectural pattern as foundation, and follow the client server approach. This method will allow us to separate presentation specifics from the overall control, as well as provide the support of multiple clients. The overall structure of the model is depicted below:



[Figure 1] Model-View-Controller and Client-Server Architectural Model

3.2 Architectural Strategies/Tactics

Reliability

The technology and framework we will be using, such as the jabber server, Openfire, will provide a very reliable server for hosting our clients. It is already implemented with multiple backup servers, and we will build our application using that. This component, provided by the server itself, is included in the current release for this iteration.

We will also add a component dedicated to ensuring that messages do deliver after a client has sent it, and report an error if it failed. This will use methods such as heartbeat or ping. This will make the chat more reliable in ensuring that communication lines remain available. This component is not included in the current release of the application, and will be added during the next iteration phases of the software solution.

We will implement a backup database component to ensure that data relating to clients and messages remain reliable and available while the application is running. This component needs to ensure that a fail-over database remains available. The initial stages of this component has been implemented, and will be expanded on in later iterations of the application during development.

Scalability

The application at this phase of development will support only one client chat, thus it is able to support the two users that are participating in the chat. The provided server and database will allow for more than two users, but this kind of support for multiple clients will only be expected from later releases of the software.

We also need to consider the available size for the SQLite database on the client side, to ensure that data such as client information and chat history doesn't grow too large for the space available.

This may happen with previous chat histories that a user may want to retrieve or view at a later stage. Our tactic to ensure that this memory usage doesn't grow too large will be a component to periodically delete the earliest n (some integer value greater than 0) message entries that have been in the database for long periods of time. This will remove stale data that will take up unnecessary space. At this point of development the only database functionality included, is the retrieval of all message information ordered by date and time, thus the deletion of message will be included at a later phase of development.

Modifiability

The agile methodology that we are following throughout this project will assume changes in the requirements and the design will have to be modifiable to compensate for these future changes.

For instance we will need the system to eventually be able to support more than the initial amount of users, thus supporting multiple clients participating in separate chat instances. In this release of the application we will only be supporting the one chat instance between two users, but the system's design at this point is relatively modifiable, and support for multiple chat instances may be easily implemented.

Users may want other features such as changing background colours to suite their profile or their personal preference to the display of the chat client. Thus the application has to be modifiable to later include such user specific features and also extra options and features. This component may only be expected in a later version of the design. The current iteration phase will only display necessary information, and not provide any additional design and presentation semantics.

Security

One of the nice-to-have features will be that Messages sent via the chat application will be encrypted when sent and decrypted when received. This will make them secure and unobtainable by outside sources. This will also provide privacy between two users when using the application. We will

provide a component or function to provide the encryption and decryption mechanics in a later phase of the development.

Performance

Messages need to be delivered within reasonable time therefore we will add a component to avoid contention on the network between the various clients. The performance of the application will also be tested with more and more active users, exchanging messages concurrently. Analysis for this component will be done during the testing phases of our development process, and thus will not be provided in this current version of the application.

Usability

Our final goal for this project is to develop an application that is intuitive and easy to use. Therefore we need our design to be as simple as possible while still providing the needed or required functionality as provided by our client. This version of the project is not as user friendly as we intend for the final product to be. At this point it works on a basic Android device.

User friendliness will be analyzed once the core functionality, as requested by our client, has been included, and user testing will be done at a later stage of development.

- The application should be intuitive.
- It should be easy to use.
- The design should be simple.
- It should be able to work on the most basic Android device.

Integrability

The application is currently being developed using Eclipse as the development tool, which has Android development capabilities. Thus the core programming languages are Java and XML. For our application to support the Mimetex library we have in this release started implementing components for the integration of C and C++ languages as native development languages in our application. For this integrability requirement we have used the Android NDK (Native Development Kit).

The application will also have to be integrated with the Android device basics, for example the onscreen keyboard, but this will be done using coding and prewritten functions found in the Android Development Kit. This functionality is partially implemented in this release, and will be further developed in later iterations of the application's development.

Maintainability/Flexibility

Using a set of coding standards will allow us to reuse code and increase efficiency and reducing redundancy. This will make our source code easy to read and modify if needed by parties that aim to develop the project further. This standard has been implemented from the initial phases of development and will remain consistent throughout the development process.

The application will have a low maintainability factor regarding the server since the server that is used is an open source server, and will provide all the maintenance needed.

Additional maintenance will have to be done on the client side, namely on the android application itself. These maintenance components will include database deletion for messages that have been in the database for too long periods of time. This database maintenance component will be implemented in later iterations and aims to increase performance on the client side.

Documentation for the application will also be provided as a hard copy. We also intend to include a user manual document within the application itself, to provide the user of the device with additional help and needed information while using the application. This will aim to increase user satisfaction and overall usability for the application. This component will only be implemented at a later stage of development, and thus not included in this version of the application.

Unit testing will be done during the iterations of the releases and summaries of these will also be provided with the final project in the end.

Constraints

The constraints of the physical device might be an issue, designing the application in such a manner that it is still intuitive and user friendly with regard to issues such as a person being able to see the rendered image clearly even if their device has a small or limited screen display.

Another issue will be whether the libraries such as the Mimetex library will be reliable enough for the application's needs will be analysed in the next iteration of development.

3.3 Reference Architecture

Our reference architecture will be the Model View Controller (MVC) architecture. It is a well known architecture that has proven to solve known web application problems, and will allow us to incorporate concepts of separating the representation of the data from a user's (or android client's) interaction with it. This way we can keep the data safe from the client aspect of our system, and only provide it with the allowed functions or methods to alter, view or retrieve the data as required by the user, via the control mechanism.

3.4 Technology and Framework Selection

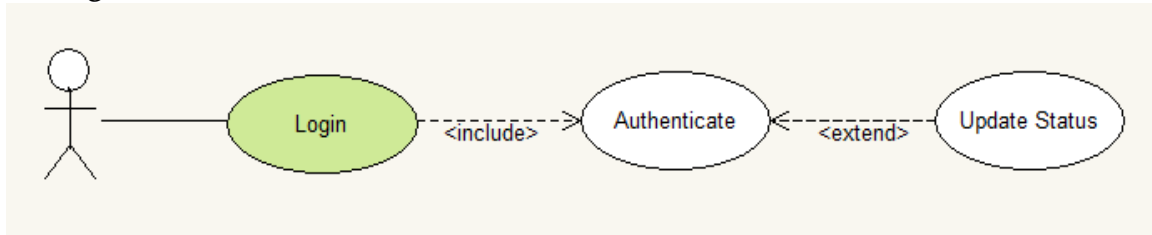
- Android SDK, NDK
- Technology: Java, XML, C
- Frameworks: MVC

3.5 Build Tools

- We will be using Eclipse IDE and Android SDK software to build the Android Application, as well as the NDK software.
- We will be using the Android ADT plugin to test the application.
- The application will then be distributed on Google PlayStore.

4. System Diagrams

4.1 Login Use Case



[Figure 2]Login Use Case

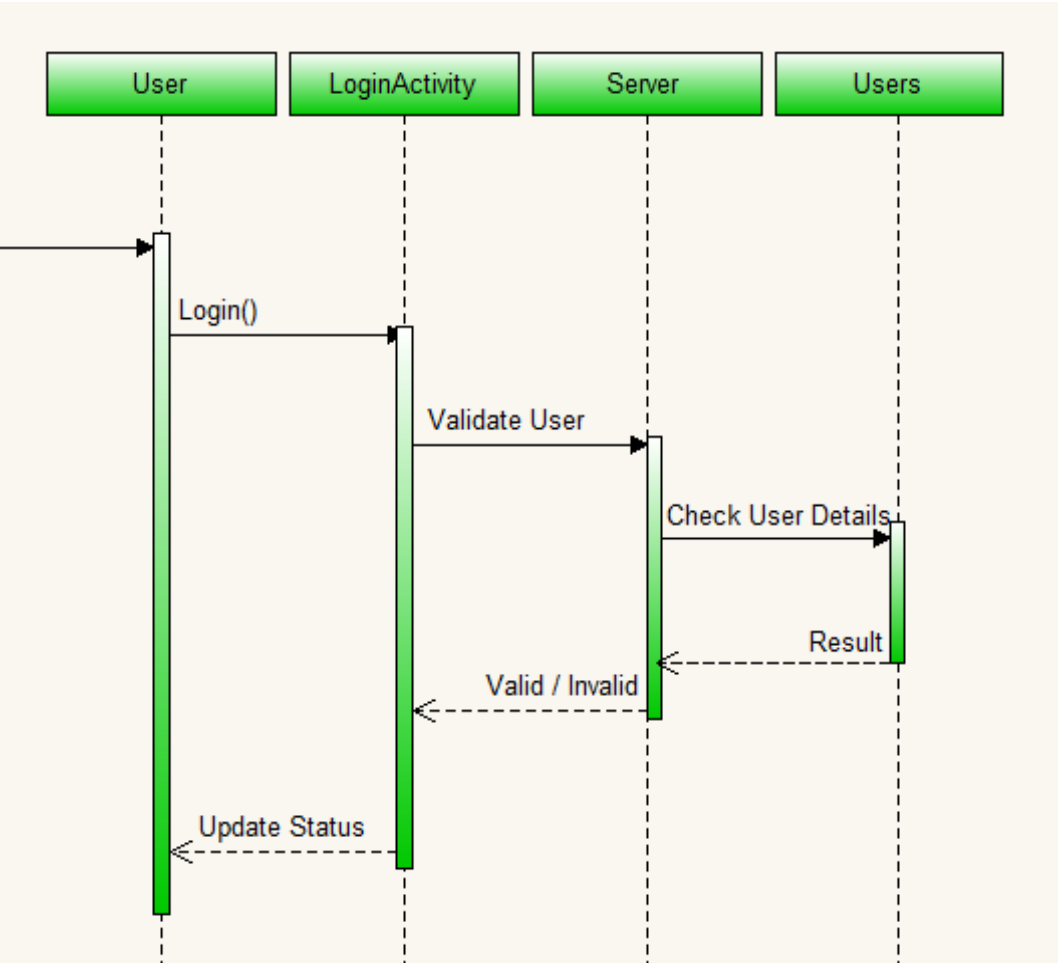
4.1.1 Services Contract

To be added.

4.1.2 Functional Requirements

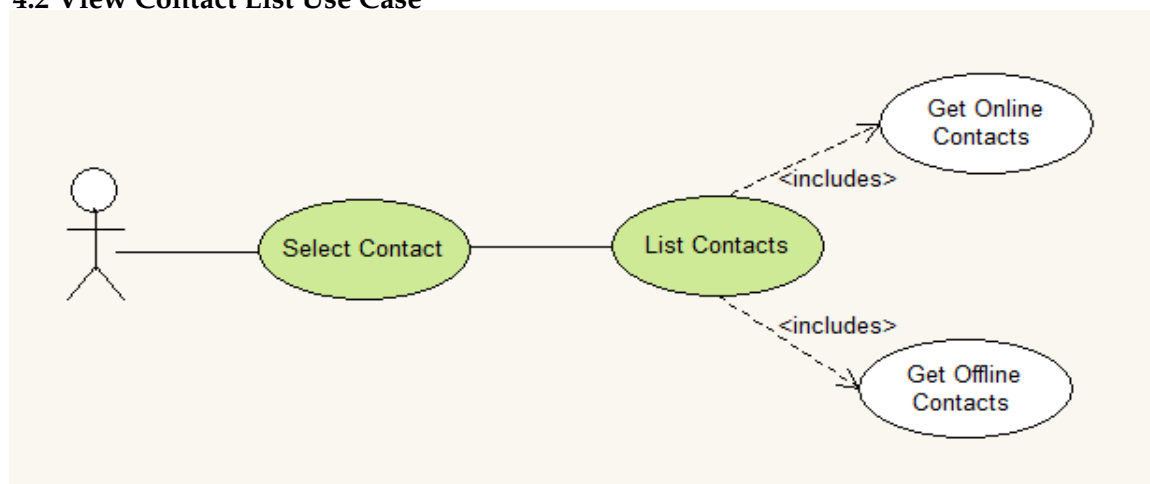
- A secure login
- Log in with username, password combination.
- Authentication is server side.
- Status stored in server database.
- Option to remember username and passwod (stored locally on device then)

4.1.3 Activity Diagram



[Figure 3]Login Activity Diagram

4.2 View Contact List Use Case



[Figure 4]View Contact List Use Case

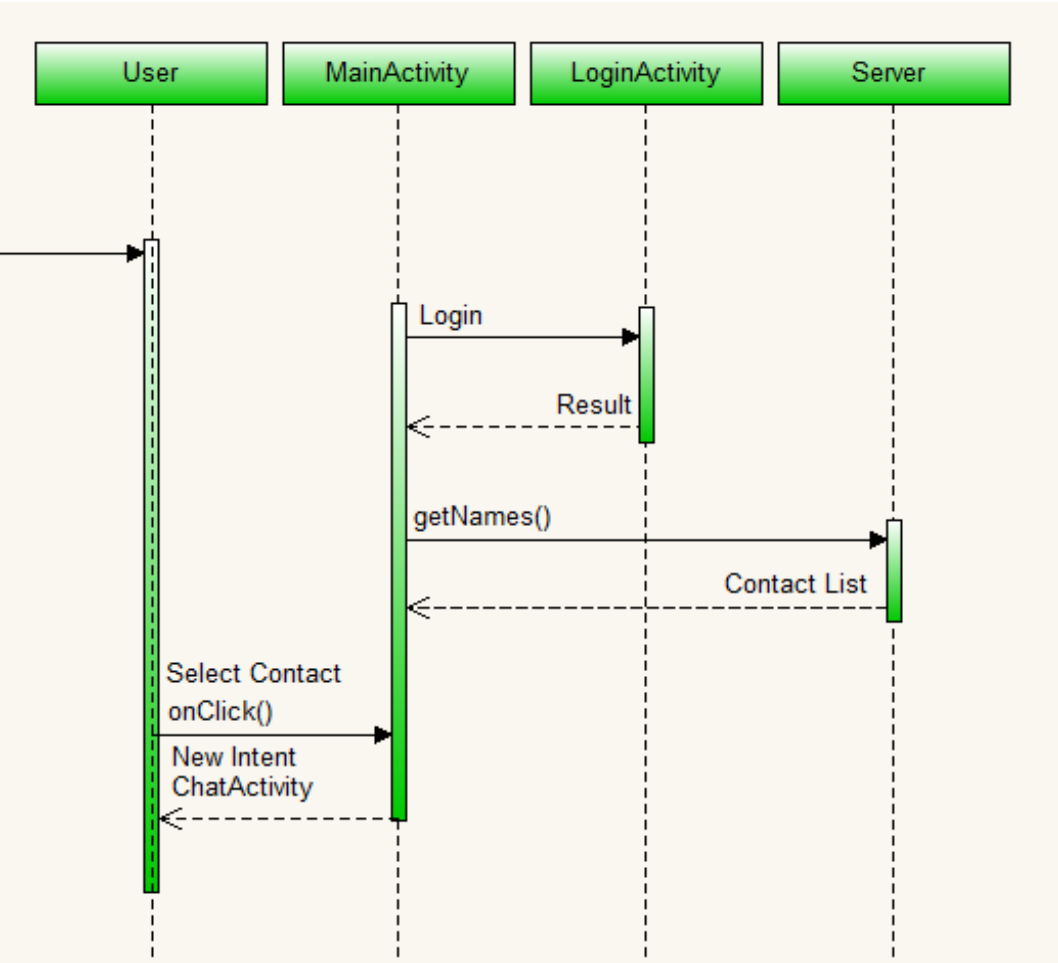
4.2.1 Services Contract

To be added.

4.2.2 Functional Requirements

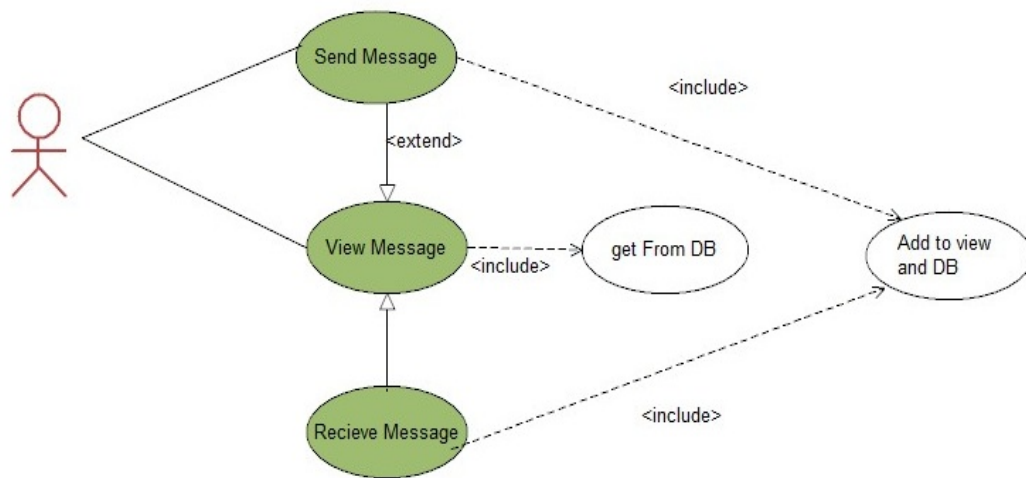
- View a contact's status.
- Click on contact to open a chat with the chosen contact.
- All contacts are shown, online and offline.
- Gets the previous messages from the database, they can also be viewed.

4.2.3 Activity Diagram



[Figure 5]View Contact List Activity Diagram

4.3 Messaging Use Case



[Figure 6]Messaging Use Case

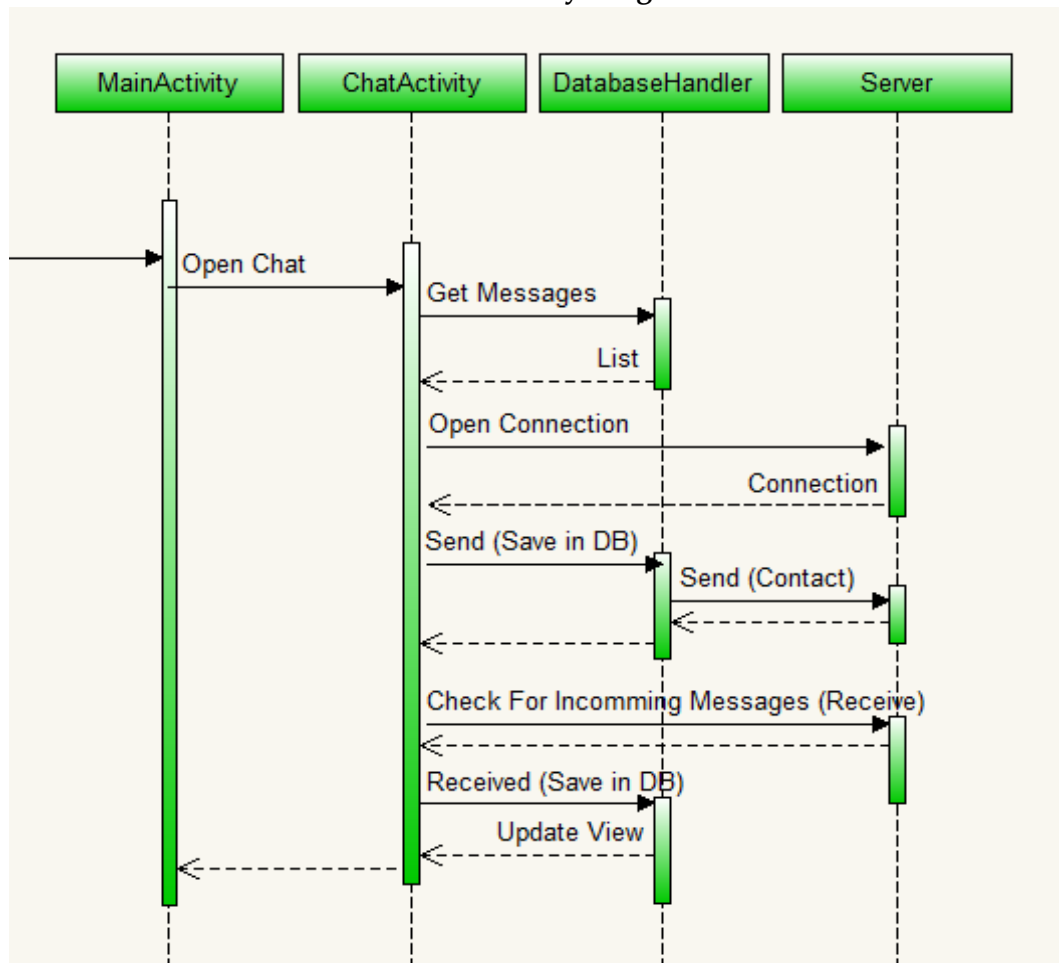
4.3.1 Services Contract

To be added.

4.3.2 Functional Requirements

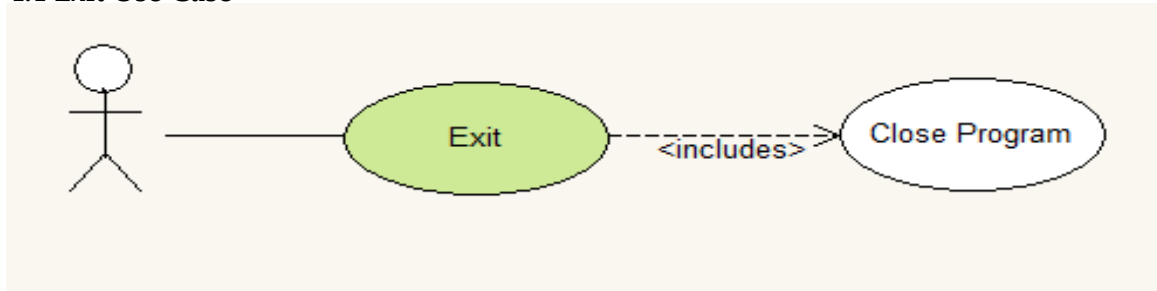
- You can view previous messages.
- Send a new messages. New message is stored in the db.
- Offline and online messages are received instantly.

4.3.3 Activity Diagram



[Figure 7]Messaging Activity Diagram

4.4 Exit Use Case



[Figure 8]Exit Use Case

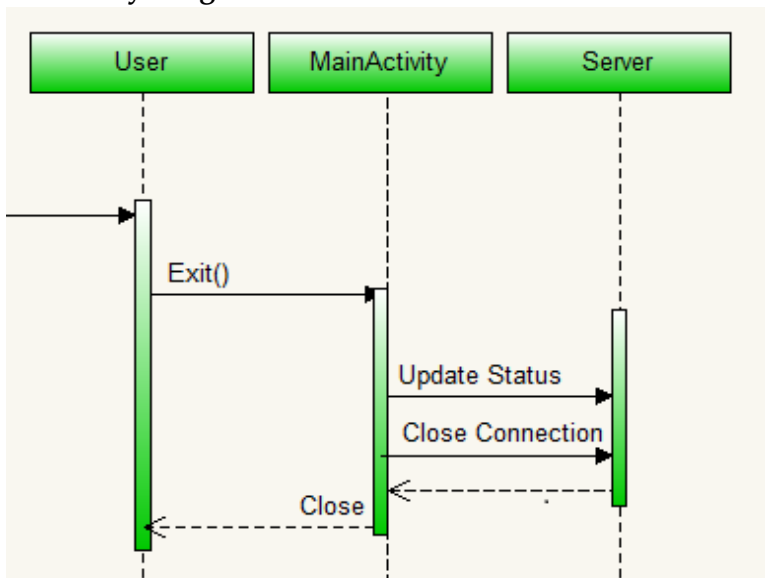
4.4.1 Services Contract

To be added.

4.4.2 Functional Requirements

- Ends the application on the device.

4.4.3 Activity Diagram



[Figure 9]Exit Activity Diagram

5. Policies

- Following Google's coding standards for the different languages used.
- Using Google's coding conventions.
- Software Development Methodology: Agile - RUP.
- Development will be mainly done using Java and XML and the Eclipse IDE, SDK, ADT and the NDK.
- The Android API will be used as a guideline and the API functions will be incorporated in the application.

6. Open Issues

- Understanding the MimeTeX library.
- Understanding how to render the LaTeX code using the MimeTeX library.
- Rendering the image inline (with the normal message text before it).

7. Glossary

- SDK - Software Development Kit
- NDK - Native Development Kit
- ADT - Android Development Toolkit plugin
- IDE - Integrated Development Environment
- RUP - Rational Unified Process
- XML - eXtensible Markup Language
- Agile - Development methodology
- MVC - Model View Controller
- UML - Unified Modelling Language
- API - Application Programming Interface
- MimeTex - LaTeX library ported to Android