Linnéuniversitetet

Messaging systems

Author: Anastasiia Zhadan, Eyuel Tesfaye Belay, Luyuan Zhou
Supervisor: Mauro Caporuscio
Date: *27-3-2018*
Course code: 4DV608

ABSTRACT

In recent years, since the continuous maturity of Internet technology and the increasingly powerful mobile terminal functions, messaging applications which based on mobile intelligent terminal platforms have become a hot demand in people's daily lives. The most operating system used by mobile intelligent terminal users is Android. The application software developed on this platform have more user groups and use value.

For this project a system, that consists of two subsystems, was designed and implemented: Mobile Application and Messaging Server. The main work of the project was to design messaging application software based on Android platform and Messaging Server, including the overall framework design and the division of functional modules.

The mobile application has three-layer architecture: UI layer - top presentation layer. BL layer as the layer that mostly consists of classes, that form specific data (registration data, login data, message data) using model classes. In other words, BL layer consists of management classes that manages user and message. Persistence (or Data Access - DA) layer consists of modules, that are responsible for accessing local database, which stores account and message information, and and for retrieving data from server.

The Messaging Server consists of three layers, including Data Access and Business Logic Layers. Their responsibility is similar to corresponding layers of the mobile application. The top layer us API layer, which provides functionality to the clients, using REST API architectural style.

The server database is separated from the server itself, so it can be stored on the separate database server, and thus, the Messaging System is designed and implemented as three-tier architecture.

The various functions of the messaging application software designed, implemented and tested.

***Keywords**: Android, Messaging Server, Mobile application, Three-layer architecture, Three-tier architecture, Application Programming Interface (API).*

OUTLINE OF THE DESIGN

Messaging systems are widely used by users who has mobile internet and enables them to connect and share staff with other users. Basic messaging system include the clients (users that communicate) and the server, that enables user to communicate each other. The core functionality of messaging system is to enable user to send messages to a specific user and receive messages from a particular user. Depending on user requirements, the messaging system might incorporate additional functionality such as creating group user, sending message to a specific user group and receiving group message, voice and video calls, sending and receiving picture and other files etc. But the functional and nonfunctional requirements are basically affected by user requirement for that specific messaging system.

Introduction: In a nutshell Messaging Systems are systems that enable a user to perform some functionality like creating a user account, login and logout, sending a message from a particular user or receiving from a particular user, create a user group and send a message to them, receive a message from a particular group and others.

User needs and the problem that underlines the development)

Users need fast and reliable communication via designed service. They want messages to be delivered in time. Response time, reliability and securities are the main concern of the system. For messaging, a server should response in no time, the probability of sending and receiving a message correctly must be at least 99.999% and communication must be secured, which means response time, reliability and security are the issues that the system must have to fulfill.

Users need privacy, which means that their data has to be secure and not accessible to external intruders. Communication channels should also be secure and reliable; the service should ensure users that the message will be delivered to the specified recipient. The server should also be available and reachable most of the time when the user tries to access it.

User requirements describe the tasks that the user must be able to accomplish using the system. For this project, for Messaging Application, our target group should be all users who use Android smartphones. We will also provide server with REST API to enable users to access the system from desktop and web. However, due to the limitations of various conditions, we set the target users as those who use Android smartphones.

In the division of user roles, the target group is all users of Android 5.1 and higher. However, individual differences could be very large. We can divide them by purpose of using the system into those, who want to send a text message only, make voice and video call, transmit files. But for this project we consider only text messaging.

The background of using the mobile messaging system can be various, and the trigger factor is that the user wants to contact with other people. The expectation is that their message can be delivered to the designated receiver in a timely and safe manner. We use the following table to manage user needs.

User stories	Value	The problem that underlines the development
Users want their message to be sent in a timely manner.	Save time.	Users need fast and reliable communication through system services.
Users do not want others to see their message.	Security protection.	Users need privacy, which means that their data is secure and cannot be accessed by outside intruders.
Users want to send a message to their designated person.	Goal realization.	Communication channels should also be secure - this service should ensure that users can send a message to the designated receiver.
Users want to use the messaging system at any time.	Provide services.	The server should be available and accessible most of the time.
Users want to create a chat group and send/receive a group message.	Goal realization.	Communication channels should also be secure and reliable to ensure that user can send and receive a group message.

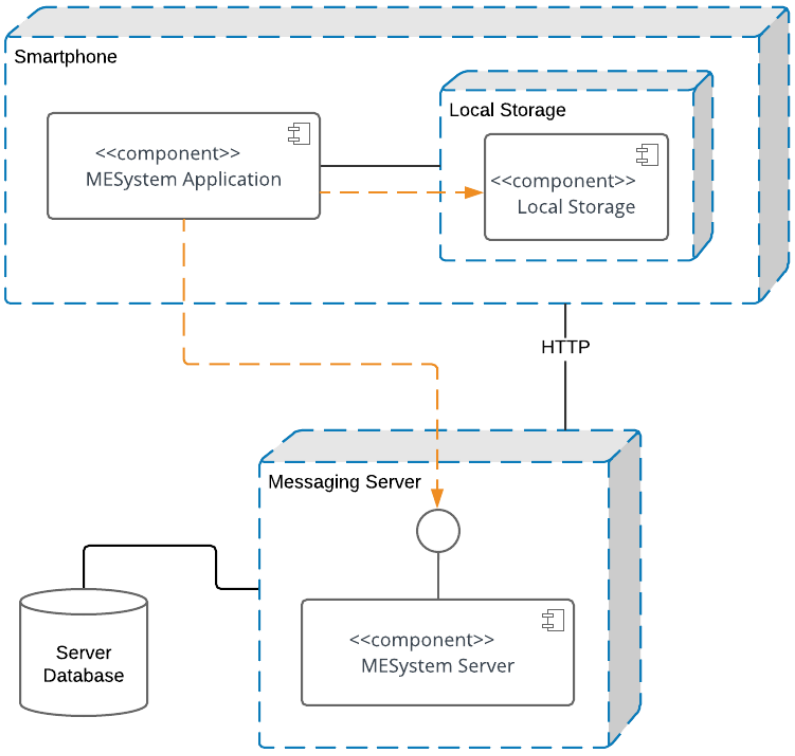
High-level Description and Alternatives

Our messaging system consists of two subsystems: mobile applications and message servers. We have two level deployment, so these two subsystems are physically distributed. The mobile app is a client that is used on Android smartphones and Messaging Server (will be deployed to the cloud). Also we consider database physically divided from the server in order to provide three-tier architecture (see Deployment Diagram).

We considered two architecture approaches: two-tier and three-tier architectures. Two-tier architecture is close to traditional client-server, but three-tier architecture is easy to maintain, with high scalability and security and single tiers can be replaced if needed.

The other important design decision was about multilayer architecture both for mobile application and server.

The application has three layer architecture with UI, BL and DA layers. Each feature of the system is provided by separate modules. And we separate cryptography facility classes and models into separate reusable modules, that can be then used by different modules. We consider server BL layer as a bridge between API and Persistence layer. It is responsible for extracting data from the message and then sending the data to the database, which stores the data. These messages will also be stored in the Android database of the user's mobile phone.



Deployment Diagram

In details, UI layer consists of layouts and UI controllers. BL layer (Business Logic layer) consists of: (1) Account Management Module. (2) Messaging Module. (3) Group Messaging Module. (4) UserList Management Module. DA layer (Data Access or Persistence layer) provides access to local storage and facilities to communicate with server.

In order to increase performance, the messages will also be stored locally.

There is also auxiliary module, that provides additional facilities for different purposes. Main modules are Object Models and Security Module (encryption and decryption), but also the module provides classes for Data Handling and Server Communication. These modules can be then reused by different modules.

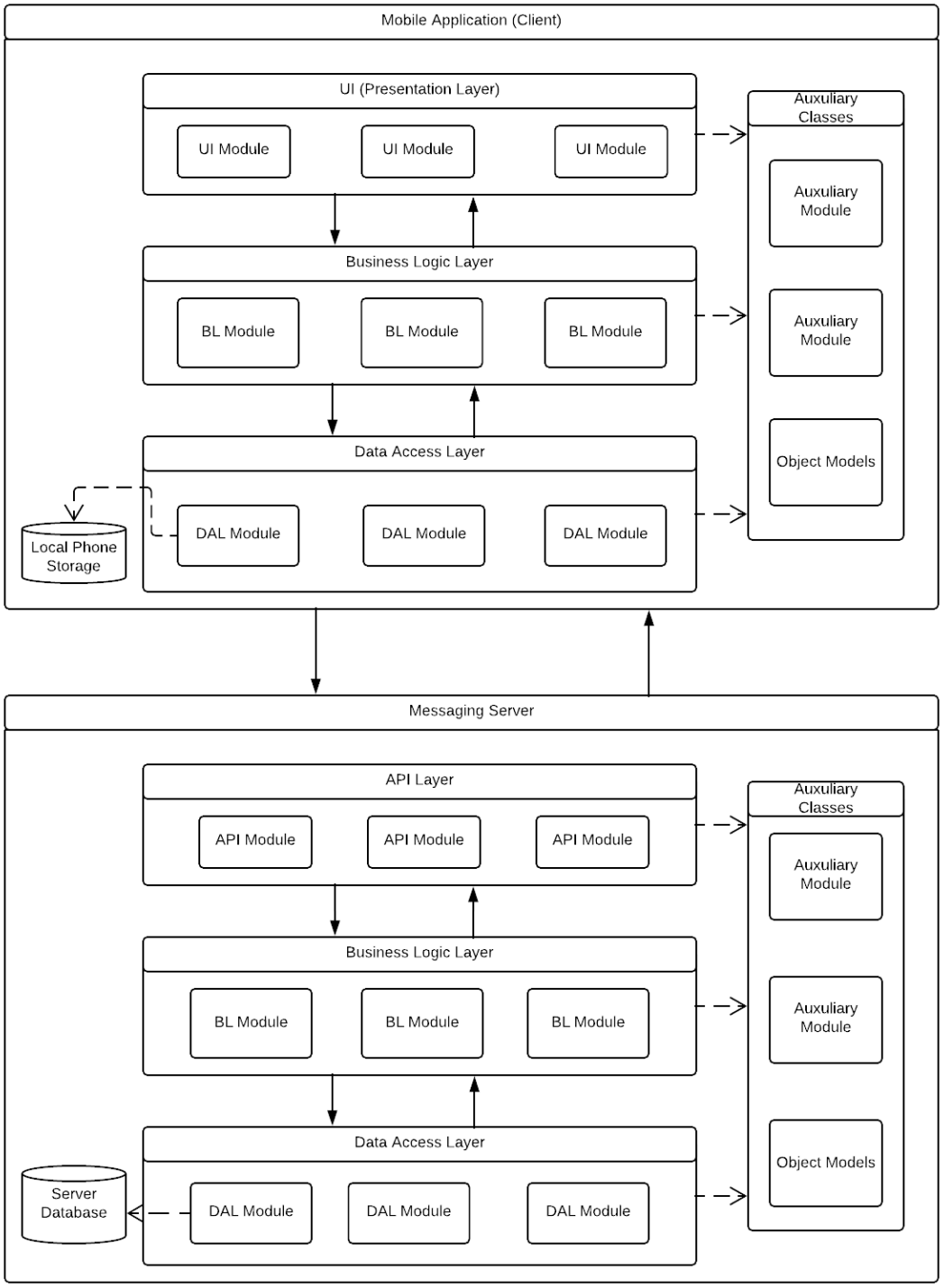
Similar approach will be used for server side. In order to provide asynchronous messaging, the messages will be stored in the server database, which accessed through Persistence layer.

The alternative was two-layered application, where UI and Persistence layer share the functionality of BL layer since the BL layer does not really provide any specific logic, but in order to keep client scalable, it was decided to keep three-layered architecture.

Each layer communicates through interfaces, so the implementation and set of provided functions are divided, and parts of the system can be replaced and re-implemented so it is good for maintenance. For Messaging Server we also decided to store group chats in the database as separate models in order to simplify the application implementation and provide asynchronous messaging for groups.

The other major design issue is sharing part of responsibilities between client and server. In particular, checking for new incoming messages is responsibility of a client. This designing decision also allows the system to provide asynchronous messaging, since all of the messages will be stored in database and retrieved on demand.

For abstract representation of system see Architecture Diagram below. For more detailed design see Component diagram.



Architecture Diagram

Ways to solve (or circumvent) the problems

Secure messaging can be provided via message encryption. Also, the database can be secured by storing only encrypted data. User's privacy needs to be protected so that the user can rely on the software and use it. In order to protect the privacy of users, we use a method of encrypting messages to ensure the security of messages. The user's messaging data will be stored in the database and user's mobile phone. The security of the user's mobile phone needs to be protected by themselves. What we can do is to make the data stored in the database to be encrypted data. After the user sends the message, the system encrypts the message data and stores it in the database. Then the system decrypts the message data and sends it to the receiver, which greatly improves the security of the database.

The purpose of providing software for users is to bring convenience to users. If the software's service efficiency is slow, it violates the most fundamental purpose. Therefore, in order to improve the efficiency of the service, we store the user's communication history in the local database of her/his mobile phone so that the user can access more quickly when he or she wants to browse his/her chat history.

The most important and basic requirement is to establish reliable message delivery. The so-called reliable message delivery means that the user's message can be sent to the receiver on time, including when the user is offline, messages are delivered immediately when the user logs in. In order to achieve such a function, we store all the message data in the service database. And even if users are offline, the unread message will not be lost.

Constraints inherent to the product

We only provide text messaging. i.e users are restricted to send only text messaging for a particular user or to a particular chat group. The user can also receive a textual message from a specific user or from a chat group. Video, voice, picture, and file messaging are out of the scope of the system.

The messaging system is an Android application, so there are restrictions on the system of the mobile phone. It must be an Android system. We are developing Android applications using Java. The messaging system sends messages over the Internet. There is no cost for the user to send a text message, but only a flow fee (such as for a GRPS cellular or wifi), but the person accepting the message also needs to use this application.

By securing the channel, through which users are communicating, we increase the probability to achieve confidentiality and integrity of a message that has to be delivered to the other user.

Alterations and refinements to the product (Ideas for possible extension)

The system we designed this time only implements the most basic functionality for sending and receiving the message, but in order to enhance the user experience, we can also increase the ability to send stickers and GIF messages in the future.

Users may need more features to bring them convenience. For example, when there are no computers around them, they can also send files, pictures, videos, and other types of data

with their phones. In order to enrich the functionality of the software, we should increase the ability to make voice/video calls.

In order to make the software using more friendly, we consider adding smart reply function to the software later.

The intelligent software function can also be implemented in the future in order intelligently respond to the question sent by the user. For example, the question “Are you arriving at the destination?”, if the user has arrived, the software will reply yes. If not, the software will estimate the time based on the current driving way and distance. And if the user is driving a car, the software will automatically switch to the mute state to prevent the user from being affected.

Functional and Nonfunctional requirements

Specific functional requirements

The messaging application has three-layer architecture and the expected functional requirements that have to be satisfied in different modules will be mentioned below. First, the user should signup to create an account and then the system only process requests from authenticated users.

Account Module:

➢ This module supports the functionality that enables users to create an account.

➢ This module supports the functionality that helps the user to login to the system and authenticates the user when she/he tries to log in.

Message Module:

➢ This module allows a user, that has already logged in, to send a message to a particular user or receive from a particular user.

➢ This module allows a user, that has already logged in, to check if s/he has got new messages.

Group Message Module:

➢ This module allows a user, that has already logged in, to create chat group.

➢ This module allows a user, that has already logged in, to send a message to a chat group and receive a message from a particular chat group.

UserList Module

➢ This module allows a user, that is already logged in, to see a list of users.

Auxiliary Module

➢ This module encrypts and decrypt message for security reason.

➢This module encrypts and decrypt user account for security reason & stored it in the database.

Non Functional requirements

Few of the non-functional requirement that messaging system has to implement are listed below. They are usually the constraint placed on the system.

✔ The Messaging system shall validate an authorized user in less than a second.

✔ The client session saves required messaging in case the client is not available online. The client receives the message when s/he available online and the client session inform the server about the status. The server changes the status from suspended to receive state.

✔ The probability of sending/receiving correctly a message must be at least 99,999% so as to say the system is reliable.

DESIGN DETAILS

Software Architecture (Architecture Description)

The ***Messaging system*** consists of two subsystems: *Mobile Application* and *Messaging Server*. We consider two-tier architecture, so these two subsystems are physically distributed: Mobile Application is a client which will be deployed on mobile phones under Android OS, and Messaging Server will be deployed to the cloud.

We considered two architecture approaches: two-tier and three-tier architectures. Two-tier architecture is close to traditional client-server, but three-tier architecture is more scalable and reliable and single tiers can be replaced if needed. For our solution, client, server, and database server are distributed physically.

Mobile Application is developed for smartphones under Android OS. The application has three-layer architecture, which includes: (1) UI layer - top presentation layer. (2) Business Logic (BL) layer, which implements data processing and is a middle-ware between UI and Data Access layer. (3) Persistence layer, which provides access to local storage and facilities to communicate with the server.

For design and implementation of a prototype of this module, we consider *BL layer* as the layer that mostly consists of classes, that form specific data (registration data, login data, message data) using model classes, then encrypt them and transfers to DA layer and decrypt data received from the server. At this point, BL layer acts as a bridge and does not provide any extra logic.

Persistence layer consists of modules, that are responsible for accessing the local database, which stores account and message information. Also, there is a module responsible for client communication with the server, i.e. establishing a connection and transferring data between client and server.

The system also has auxiliary modules, that provide facilities for cryptography and object models. Each layer is divided into modules (classes), that provide specific functionality, so the modules can be reusable. Each layer communicates through interfaces, so the implementation and set of provided functions are divided, and parts of the system can be replaced and re-implemented.

The approach described for Mobile Application is also used for **Messaging server**. The three-layer architecture includes following layers:

1. *API* layer - the top layer.
2. *Business Logic* (BL) layer, which implements data processing and is a middle-ware between API layer and Data Access layer.
3. *Persistence* layer, which provides access to the server database.

Just like for the Mobile Application, each layer is divided into modules (classes), that provide specific functionality, so the modules can be reusable.

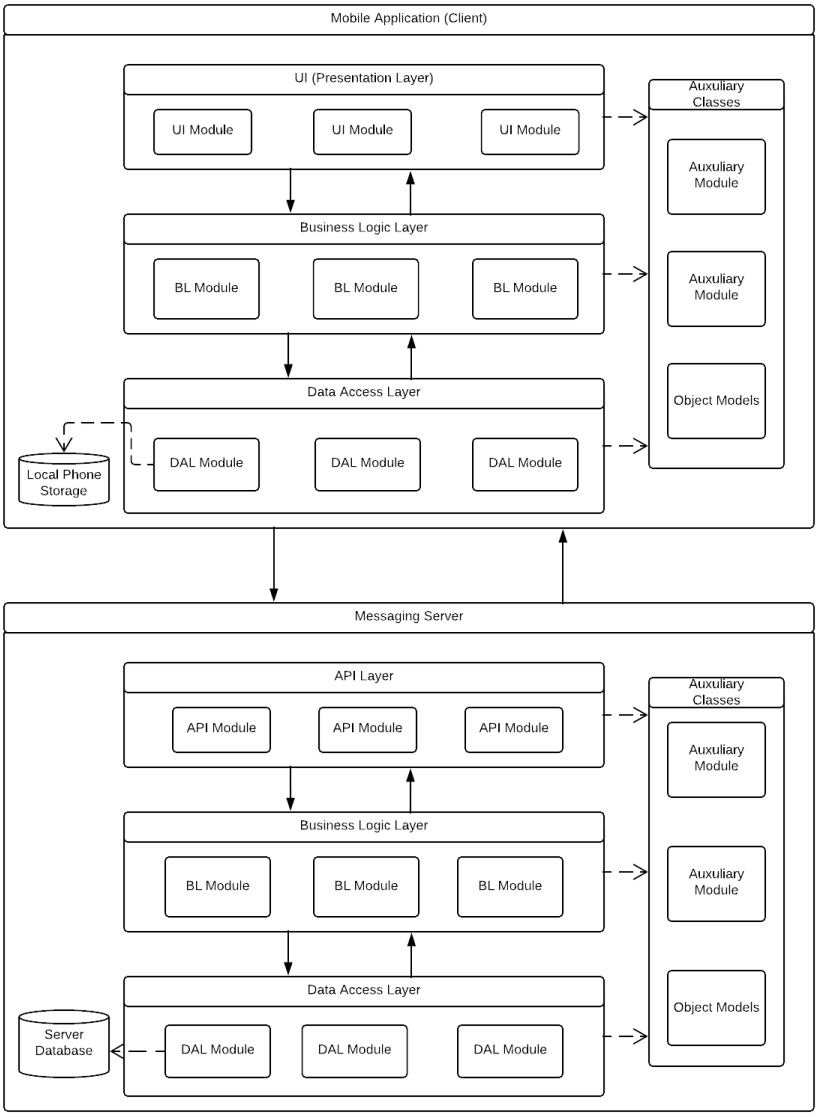
As previously, we consider server BL layer as a bridge between API and Persistence layer. Its duty is to extract data from the message and then pass it to the database if the data is supposed to be stored. Modules of the BL layer also take responsibility for managing messages, i.e., sending the messages to recipients.

Persistence layer consists of modules, that are responsible for accessing server database, which stores accounts and messages information.

The system also has auxiliary modules, that provide facilities for initializing and establishing a connection with the client in order to deliver sent messages. Each layer communicates through interfaces, so the implementation and set of provided functions are divided, and parts of the system can be replaced and re-implemented.

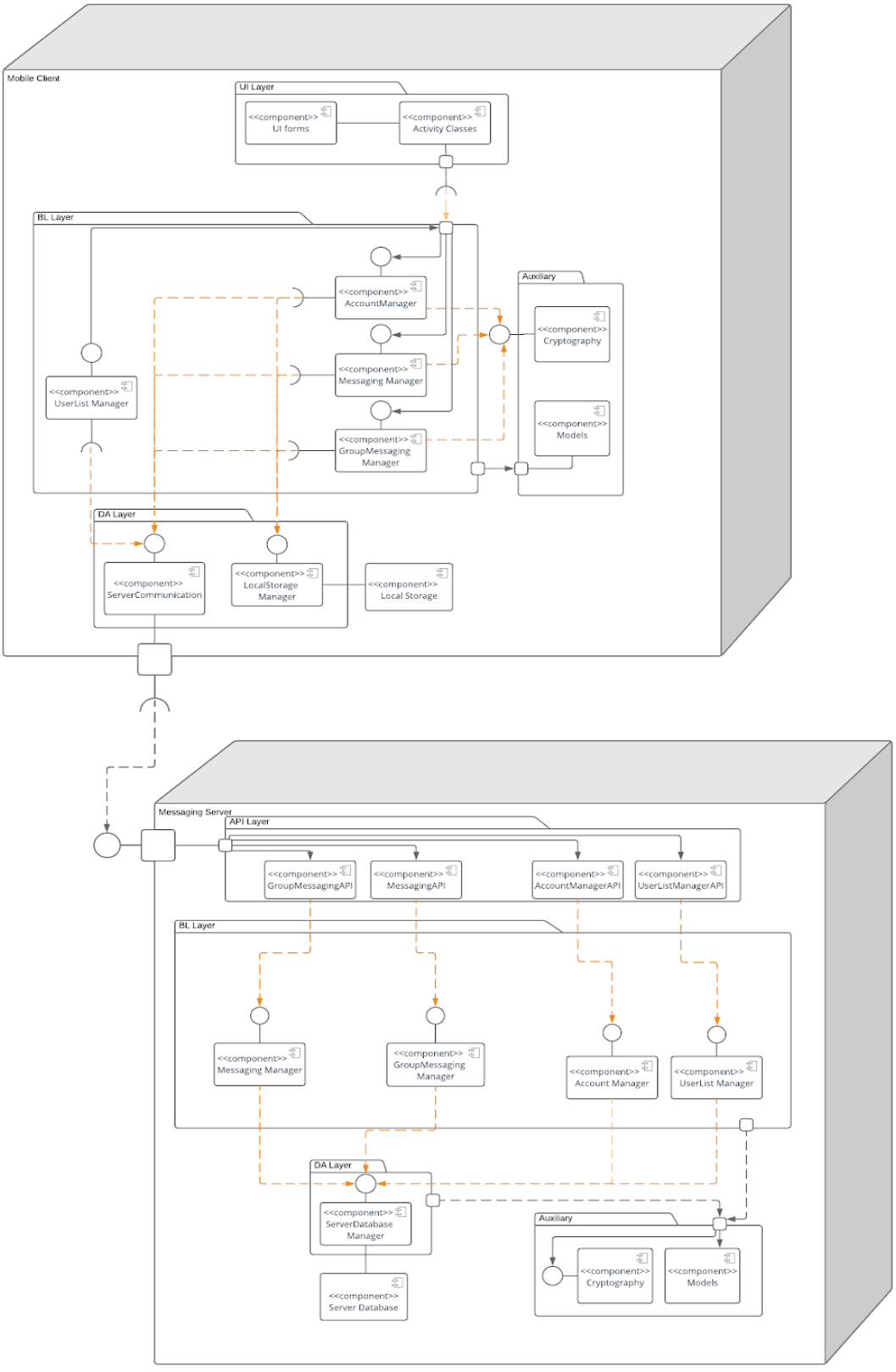
Architecture

Abstract system design described above can be seen on the architecture diagram below.



Architecture diagram

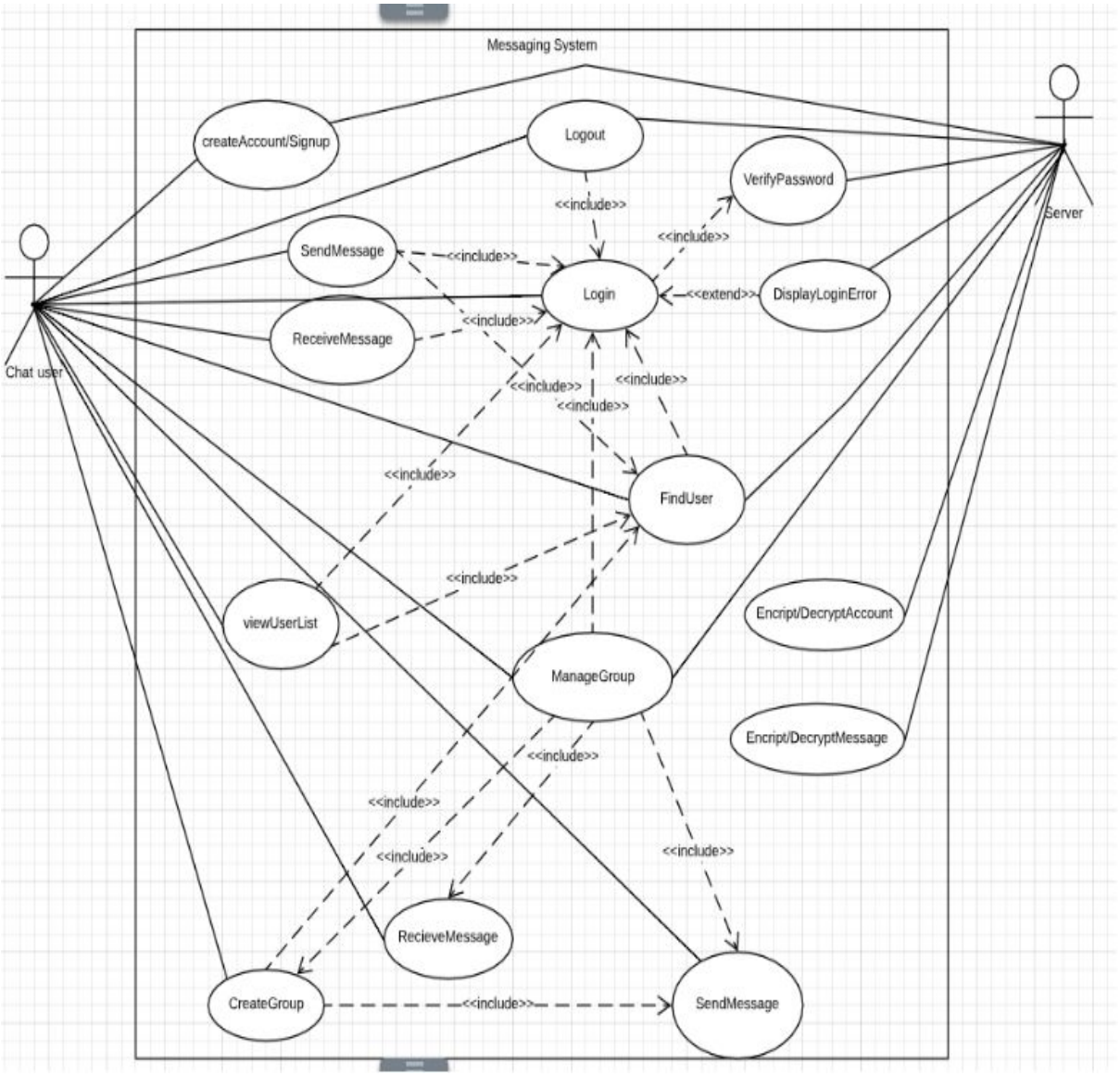
In more details the architecture is shown on the component diagram below.



Component diagram

Use Case Diagram and Use Case Specification

Use Case Diagram



No	SUC ID	SUC Name	SUC Description
1	SUC-1	CreateAccount(SignUp)	In this use case, users can sign up to create an account.
2	SUC-2	Encrypt/DecryptAccount.	In this use case, the system encrypts/decrypts user accoun nformation.

3	SUC-3	Login	In this use case, a user can be able to login to the system.
4	SUC-4	VerifyAccount	In this use case, a system verifies an account of the user every time the user tries to log in.
5	SUC-5	DisplayLoginError	In this use case, a system displays a login error when the account a user is not verified when the user tries to log in.
6	SUC-6	Logout	In this use case, a user can log out from a system when they are needed.
7	SUC-7	FindUsers	In this use case, a user can search to find other users.
8	SUC-8	ViewUserList	In this use case, a user can view a list of users.
9	SUC-9	SendMessage	In this use case, a user can send a message to a particular user.
10	SUC-10	ReceiveMessage	In this use case, a user can receive a message from a particular user.
11	SUC-11	ManageGroup	In this use case, a user can manage a chat group.
12	SUC-12	CreatChatGroup	In this use case, a user can create a chat group.
13	SUC-13	SendGroupMessage	In this use case, a user can send a message to a group.
14	SUC-14	ReceiveGroupMessage	In this use case, a user can receive a message from a group.
15	SUC-15	Encrypt/DecryptMessage	In this use case the system encrypts & decrypts message.

Use Case Specification

Use Case ID:	SUC-1
Use Case Name:	CreateUserAccount(SignUp)
Actors:	User, Server
Description:	In this system process, user information can be registered in the database and user account is created.
Trigger:	When user click the button "Sign up"
Preconditions:	
Normal Flow:	1. The use case starts when a user selects "create user" option or link. 2. The system displays user registration or a sign-up form. 3. While(Until) user details are valid 1.1. The system requires the user to fills all the required Information again for confirmation. 1.2. The system validates the input data (or user detail). 1.3. The user clicks on "createUser" button. 4. The system creates a new account for the user.
Postconditions:	A new account has been created and information about user has been saved.
Alternative Flows:	InvalidUserName and Password Cancel
Special Requirements:	User-friendly interface

Use Case ID:	SUC-1.1
Alternative Flow:	CreateUserAccount(SignUp):InvalidUserName/Password
Actors	Server.
Description:	In this use case, the system informs the user that s/he has entered invalid account (username or password).
Preconditions:	The user entered username or password or both.
Alternative Flows:	1. The alternative flow starts with step 3.2 of the main flow. 2. The system informs the user that s/he entered the invalid account.
Postconditions:	None

Use Case ID:	SUC-1.2
Alternative Flow:	CreateUserAccount(SignUp): Cancel
Actors:	User,Server
Description:	User cancels the account creation or the sign-up process.
Preconditions:	None.
Alternative Flows:	1. The alternative flow begins at any time. 2. The user cancels the account creation. 3. The server response for a user request(maybe close the page or other action).
Postconditions:	A new account has not been created for the user.

Use Case ID:	SUC-2
Use Case Name:	Encrypt/DecryptUserAccount
Actors:	Server
Description:	In this system process, the system encrypts and decrypts user account (username & password) for security reason.
Trigger:	When a user account is created or when a user tries to log in.
Preconditions:	None
Normal Flow:	1. The use case starts when a user account is successfully created or when the user tries to log in to the system. 2. The system encrypts account and store in the database when a user account is created. 3. The system decrypts user account when the user tries to log in to the system.
Postconditions:	The user account has been encrypted/decrypted.

Use Case ID:	SUC-3
Use Case Name:	Login
Actors:	User, Server
Description:	In this system process, the user tries to log in.
Trigger:	When the user clicks the link to the login button.
Preconditions:	The user should have a user account.
Normal Flow:	1. The user selects “login” option or link. 2. The system displays the login form. 3. The user enters username and password. 4. The user clicks on “login” button. 5. (include VerifyAccount) 6. The system enables the user to log in.
Postconditions:	The user has been successfully logged in.

Alternative Flows:	A4.1: Verify Account A4.2: If the network connection is lost 1. Generate error
Special Requirements:	· User-friendly interface

Use Case ID:	SUC-4
Use Case Name:	VerifyAccount
Actors:	Server
Description:	In this use case, a user account is verified by the system.
Trigger:	When a user clicks a login button.
Preconditions:	The user should enter input data (username and password).
Normal Flow:	1. Server check if account (username & password) entered by the user is valid. 2. Allow the user to be logged in or rejected. 3. If rejected 3.1 the system generates an error message. 3.2 The user asks to retype data. 3. 3 Back to normal flow step 2 in SUC-3. 4. If logged in the system display successful message.`
Postconditions:	The user account has been verified.

Use Case ID:	SUC-5
Extension Use Case Name:	DisplayLoginError
Actors:	Server
Description:	Segment 1: the server display login error.
Trigger:	When the user clicks the login button.
Segment 1 Preconditions:	The user has to enter user account(username & password) and has to be invalid account.
Segment 1 Flow:	1. The system has displayed the login error.
Postconditions:	Login Error has been displayed to the user.

Use Case ID:	SUC-6
Use Case Name:	Logout
Actors:	User, Server
Description:	In this use case, a user can log out from the system when they required.
Trigger:	When user click the link to “logout”
Preconditions:	The user has logged on to the system.
Normal Flow:	1. The user selects “logout” option or link. 2. The system allows the user to log out.
Postconditions:	The user has logged out from the system.
Alternative Flows:	None

Use Case ID:	SUC-7
Use Case Name:	FindUsers
Actors:	User, Server
Description:	The user finds other users.
Trigger:	When user click the link to “find user”
Preconditions:	The user has logged on to the system.
Normal Flow:	1. The user enters the required criteria to find other users 2. If the system found the user 2.1 The system displays the user. 3. Else 3.1 The system tells the user no matching is found.
Postconditions:	Either user has found or not.
Alternative Flows:	None

Use Case ID:	SUC-8
Use Case Name:	ViewUserList
Actors:	User, Server
Description:	In this system process, the user can view a list of users.
Trigger:	When user click the link to “view user list”
Preconditions:	The user has logged on to the system.
Normal Flow:	1. include(FindUsers) 2. The system displays a list of users.
Postconditions:	The system has display list of users.
Alternative Flows:	None

Use Case ID:	SUC-9
Use Case Name:	SendMessage
Actors:	User, Server
Description:	In this system process, the user can send a message to a particular user.
Trigger:	When user click the link to “Send Message”
Preconditions:	The user has logged on to the system and has to find a user.
Normal Flow:	1. The use case starts when a user selects “send a message” option or link. 2. The system displays message form. 3. User write Message 4. User click send button 5. The system sends a message.
Postconditions:	A new message has sent or received.
Alternative Flows:	

Use Case ID:	SUC-10
Use Case Name:	RecieveMessage
Actors:	Server, User
Description:	In this system process, the user can receive a message from a particular user.
Trigger:	When user click the link to “view Message”
Preconditions:	The user has logged on to the system.
Normal Flow:	1. The use case starts when a user selects “view message” option or link. 2. The system displays all message that has sent from different users.
Postconditions:	A new message has received.
Alternative Flows:	

Use Case ID:	SUC-11
Use Case Name:	ManageGroup
Actors:	User, Server
Description:	In this use case, the user can manage a user group so as to send/receive a group message.
Trigger:	When user click the link to “send message to croup” or “create a group”
Preconditions:	The valid user has logged on to the system.
Normal Flow:	1. The use case starts when a user selects “manage group” option or link. 2. include(creatChatGroup) 3. include(SendGroupMessage) 4. include(ReceiveGroupMessage). 5. User click manage group button.
Postconditions:	The user group has been managed.
Alternative Flows:	

Use Case ID:	SUC-12
Use Case Name:	CreateChatGroup
Actors:	User, Server
Description:	In this system process, the user can create a chat group.
Trigger:	When the user clicks the link to “create a chat group.”
Preconditions:	The user has logged on to the system.
Normal Flow:	1. The use case starts when a user selects “create chat group” option or link. 3. include(FindUser) 2. The system adds users to the chat group. 3. User click addChatGroup button.

Postconditions:	Chat group has been created.
Alternative Flows:	

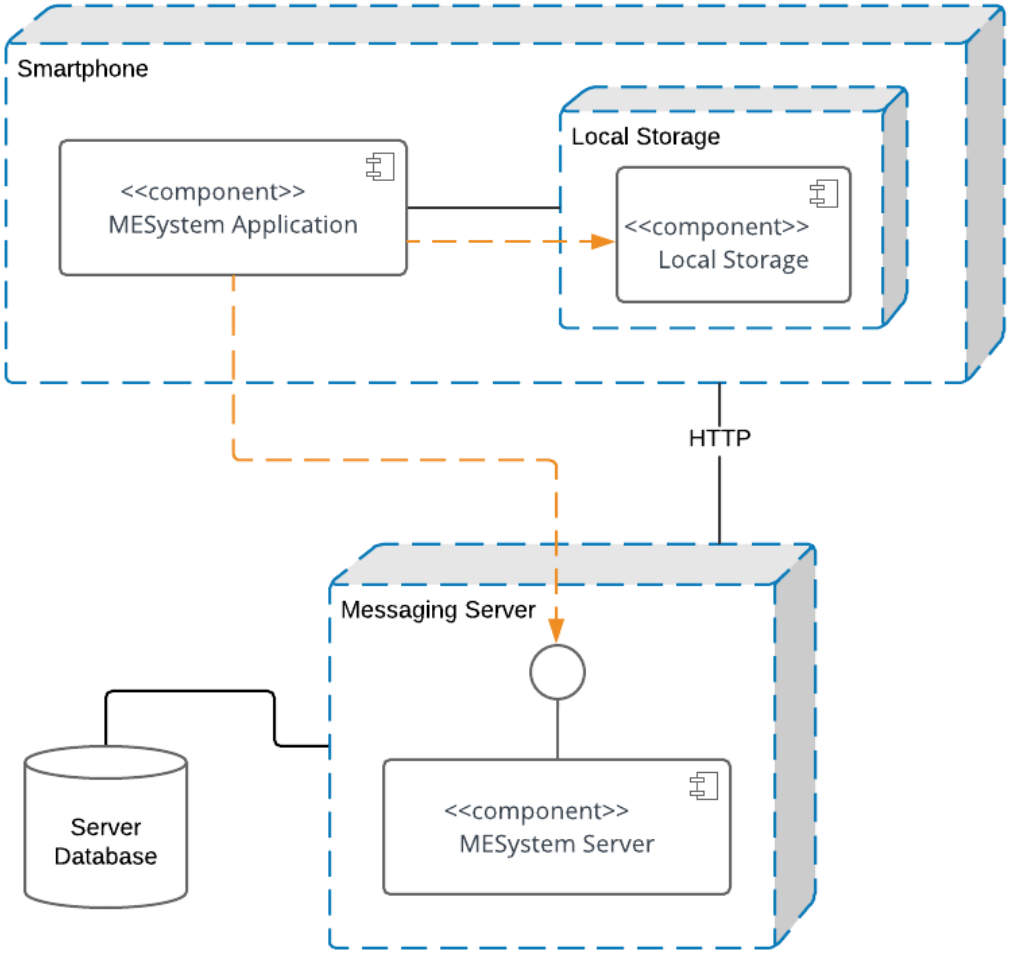
Use Case ID:	SUC-13
Use Case Name:	SendGroupMessage.
Actors:	User, Server
Description:	In this system process, the user can send a group message.
Trigger:	When the user clicks the link to “send a group message.”
Preconditions:	The user has logged on to the system.
Normal Flow:	1.The use case starts when a user selects “send a group message ” option or link. 2.include(createGroup) 3. The system shows a group message form. 4. User click writes a group message. 5. The user clicks send message button. 6. The system sends a message to each user found in the group.
Post conditions:	Group message has been sent.
Alternative Flows:	A5. Reliability Error 1. If the message has not sent to users found in a group with the probability less than 99.999%

Use Case ID:	SUC-14
Use Case Name:	ReceiveGroupMessage.
Actors:	User, Server
Description:	In this system process, the user can receive a group message.
Trigger:	When the user clicks the link to “view a group message.”
Preconditions:	The user has logged on to the system.
Normal Flow:	1.The use case starts when a user selects “view a group message ” option or link. 2. The system show messages received from a group.
Postconditions:	Group message has been received.
Alternative Flows:	

Use Case ID:	SUC-15
Use Case Name:	Encrypt/DecryptMessage
Actors:	Server
Description:	In this system process, the system encrypts and decrypts message for security reason.
Trigger:	When a user sends a message/receive the message.
Preconditions:	None
Normal Flow:	1. The use case starts when users send/receive a message. 2. The system encrypts a message and store in the database. 3. The system decrypts a message when the user receives.
Postconditions:	The message has been encrypted/decrypted.

Deployment Diagram

Deployment diagram representing three-tier architecture described above can be seen below.



Deployment diagram

TIMELOG

Task	Anastasiia	Eyuel	Luyuan
Discussion	5:00	5:00	5:00
Designing	3:00	5:00	4:00
Mobile application implementation	10:00	2:00	2:00
Server implementation	12:00	5:00	3:00
Writing the report	3:00	6:00	8:00