
Software Design Document

for

Instant Messaging System

Version 1.0 approved

Prepared by Christopher Sellers, Jerry Zhu, Aaron Lam

Instructor: Sara Farag

Course: CS 410

Date: 05-20-2019

Table of Contents

Revisions

1	Introduction
1.1	Purpose
1.2	Scope
1.3	Document Overview
1.4	Definitions, Acronyms and Abbreviations
1.5	Document Conventions
1.6	References and Acknowledgments
2	System Overview
2.1	Architectural Design
2.2	Design Decomposition
2.3	Design Rationale
3	Data Design
3.1	Data Description
3.2	Data Dictionary
4	Component Design
5	Human Interface Design
5.1	Overview of User Interface
5.2	Screen Images
6	Programming Tools
7	Requirements Traceability Matrix

Revision History

Primary Authors	Date	Reason For Changes	Version
Jerry Zhu	5/18	Added section 2.1 & 2.3	0.1
Chris Sellers	5/19	Added section 1.x	0.2
Jerry Zhu	5/19	Added section 3.1	0.3
Aaron Lam	5/20	Added section 5.x & 6.x	0.4
Chris Sellers	5/20	Added section 7	0.5
Jerry Zhu	5/20	Revised section 3.1	0.6
Aaron Lam, Chris Sellers, Jerry Zhu	5/20	Final revisions to all sections	0.7
Jerry Zhu	6/10	Class diagram added to section 4	1.1
Chris Sellers	6/13	Sequence diagram added to section 4	1.2
Aaron Lam	6/13	DFD model added to section 4	1.3
Aaron Lam	6/14	DFD model revised	1.4
Aaron Lam, Chris Sellers, Jerry Zhu	6/14	Section 4 diagrams revised	1.5

1. INTRODUCTION

1.1 Purpose

The purpose of this SDD is to provide an overview of the design decisions for this project. It will describe the general structure of the system as well as individual component attributes. The general structure will demonstrate how each component is connected within the system. Component attributes will also be discussed to provide clarity for specific design choices. This document is directed towards the product owner and developers as a reference document in all stages of the project.

1.2 Scope

This project is focused on providing a reliable, maintainable, and secure messaging platform for all Android users. The scope of this project is to build a functioning product that can support additional components in the future. Our goal is to build the core components required to send, receive, store, secure, and view messages. As a benefit, the application will have a simple and intuitive design.

1.3 Overview

Section two describes the system's structure and components. The third section indicates how the data will be stored, converted, and retrieved from the system. Component design will describe each component of the system and the necessary attributes. Section five shows how the user will be able to interact with the product. The sixth section discusses all development tools being used for this project. This section also provides a brief description of the purpose for each tool. Section seven provides a traceability matrix that will cross reference the requirements from the SRS to the design components that will be later discussed.

1.4 Reference Material

"How Should a Model Be Structured in MVC?" *Stack Overflow*,
stackoverflow.com/questions/5863870/how-should-a-model-be-structured-in-mvc.

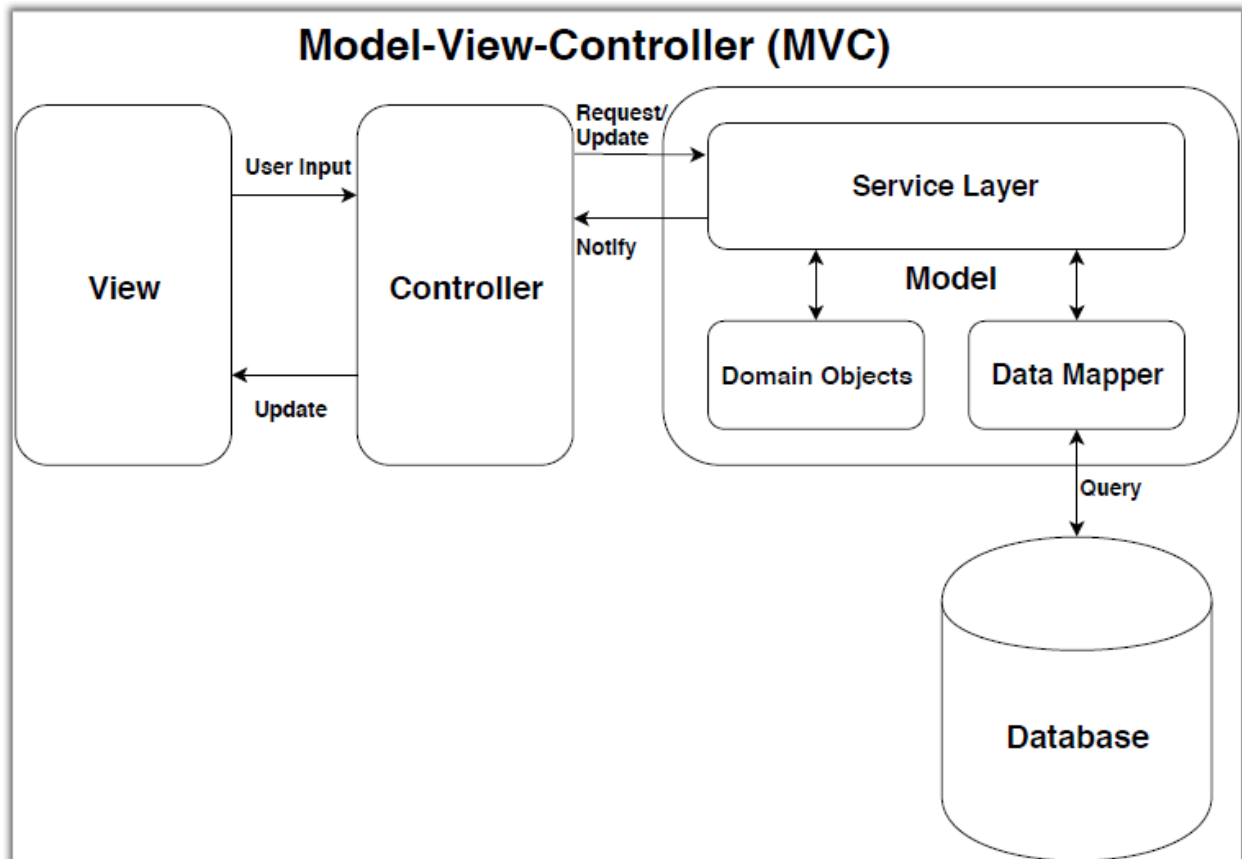
1.5 Definitions and Acronyms

SDD - Software Design Document
SRS - Software Requirements Specification
GUI - Graphical User Interface
CRUD - Create, Read, Update, Delete
MVC - Model-View-Controller

2. SYSTEM OVERVIEW

SYSTEM ARCHITECTURE

2.1 Architectural Design



The instant messaging application will be designed using Model-View-Controller (MVC), an architectural design that partitions an application into three coherent components.

Model

The model component will be responsible for retrieving and manipulating data. It will consist of domain objects, a data mapper, and a service layer. The domain objects are storage classes that will contain information regarding users and chat instances. The data mapper is a data access layer that allows for bidirectional transfer of data between the domain objects and the database. It is responsible for creating, reading, updating, and deleting (CRUD) business logic in the database. The service layer provides a public interface for the controller component to call. It is responsible for interactions between the domain objects and data mapper. This increases the overall cohesion of the model and controller components by enforcing the single responsibility principle: every module should have a single part of the functionality and that responsibility should be entirely encapsulated.

View

The view component will be responsible for displaying the GUI to allow for user interaction. Any user inputs will be sent to the controller component to handle.

Controller

The controller component will act as a mediator between the model and view component. It will handle user inputs sent from the view and use the interface provided by the service layer of the model. After the model performs data manipulation to business logic, it will notify the controller of the changes made. The controller will then update the view to inform the user. It should have minimal to no knowledge on business logic; it just needs to be calls the public methods provided by the service layer.

2.2 Decomposition Description

2.3 Design Rationale

Advantages

MVC is a cross-platform development model; it allows the model and controller to be kept the same for all views. In the future, there may be plans to support other mobile platforms like IOS and Windows. The same model and controller can be reused.

Not only does it allow for high portability, but also supports multiple views. The user will need to navigate through different views to perform desired tasks. For example, to send a message to a friend the user would need to transverse from the login screen, to the chats interface, and then to the specific chat. By using MVC, the same model and controller can be used for each of the different views.

MVC partitions the application into three logical components, making it easier for multiple developers to simultaneously work on model, view, and controller. There is low coupling between each component and high cohesion within each one. Any refactoring or modifications will have minimal impact on one another.

Drawbacks

Despite being a suitable design for our application, MVC does have several drawbacks. There can be lot of data abstractions and layers within the model component causing the complexity to increase as the application scales in features. Maintainability can thus become an issue in the future.

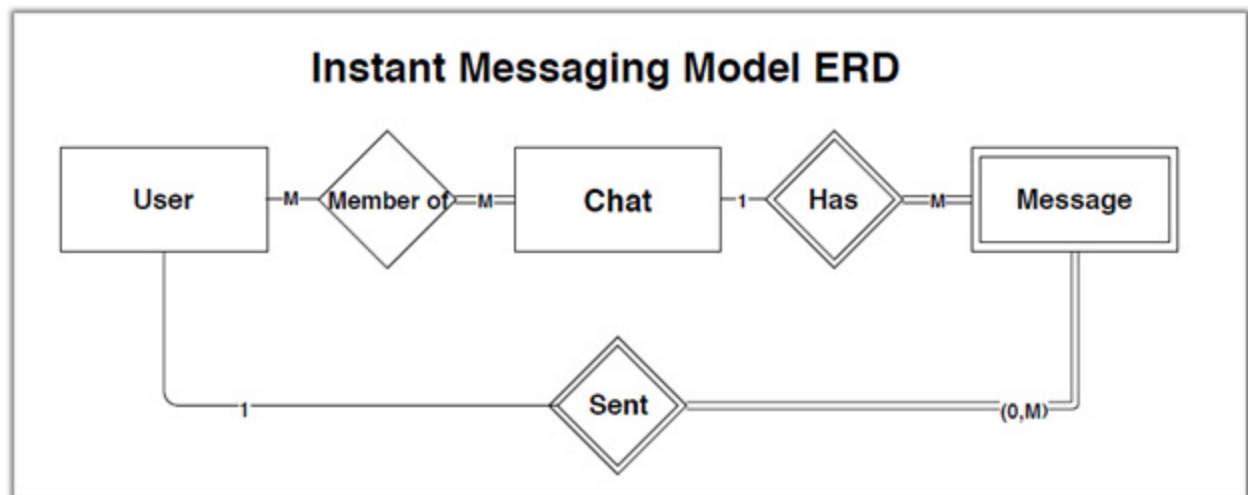
There is also low efficiency of data access in the view. The view needs to communicate with the controller and the controller needs to request or update data within the model. It can be lengthy process especially as more layers and abstractions get made.

3. DATA DESIGN

3.1 Data Description

Domain information in our application will be located strictly within the model component of the system. The model will contain in-memory storage, domain objects, to store information related to the user, chats, and messages. In order to retrieve additional information from the persistent memory, database, a data mapper will be created to allow for data access. The data mapper provides query commands to perform CRUD operations on the database. Business logic retrieved from the database will then be stored in the domain objects for easier access.

Specifically, the domain objects in the model will be user, chat, and message classes. The user class will not only store information regarding the user, but also hold references to all of the chat groups the user belongs to. The chat class will contain information about the chat settings and hold references to all of the messages in the chat. The message class will store the text message and reference to the creator (user class).



The entity-relationship diagram will consist of three entities: user, chat, and message. As a weak entity, message has two owner entities: user and chat. If either of the owner entities gets deleted, the deletion will cascade to the message entity. A chat group must have users, a message must belong to a chat, and a message must be sent by a user. Hence there must be total participation between those relationships.

Relational Schema

Relation Name	user					
Column Names	user_id	password	phone_number	first_name	last_name	gender
Constraint	PK					
Data Type	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	CHAR(1)

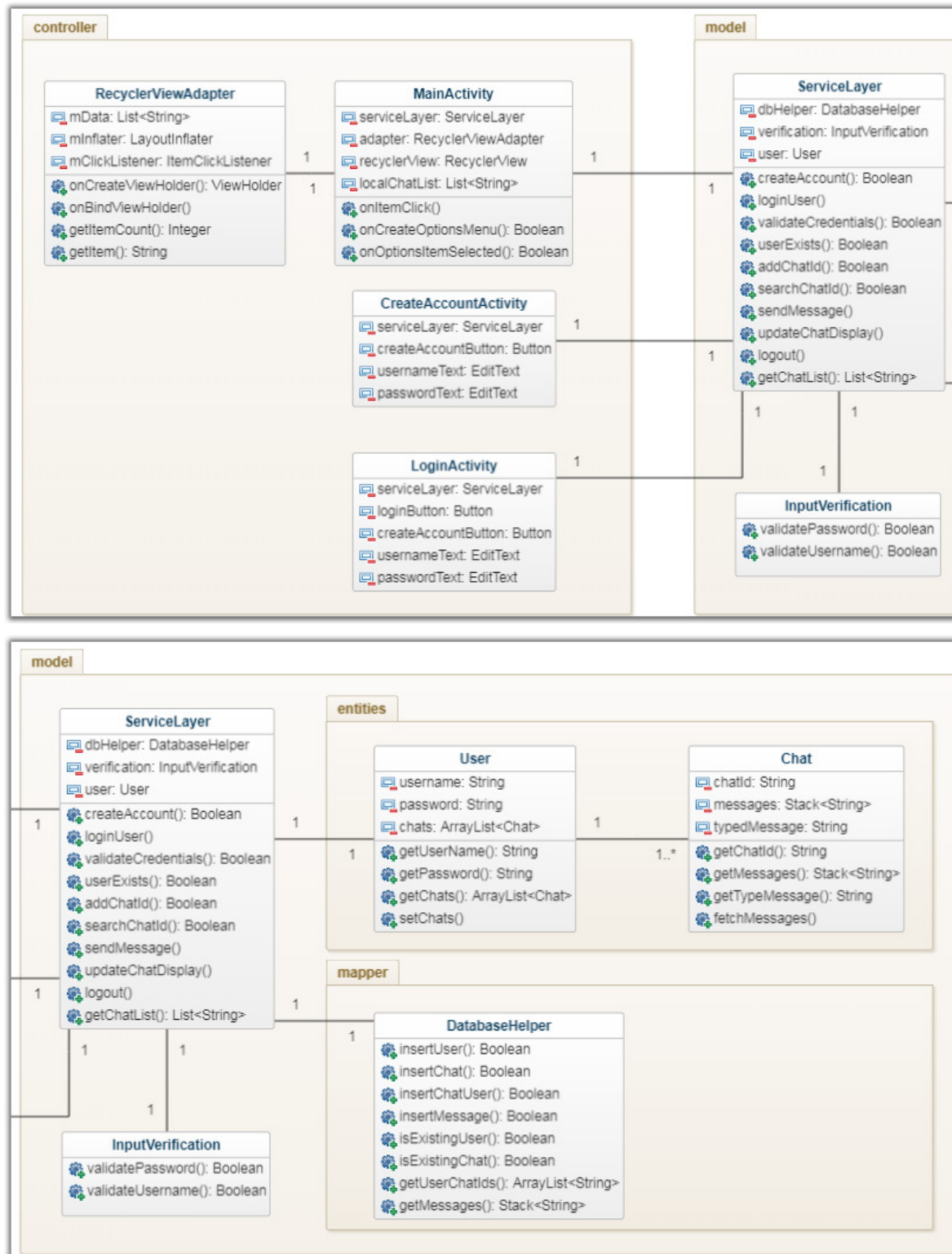
Relation Name	chat		
Column Names	chat_id	chat_capacity	chat_name
Constraint	PK		
Data Type	VARCHAR(20)	INT(10)	VARCHAR(20)

Relation Name	chat_member	
Column Names	user_id	chat_id
Constraint	PK/FK(user)	PK/FK(chat)
Data Type	VARCHAR(20)	VARCHAR(20)

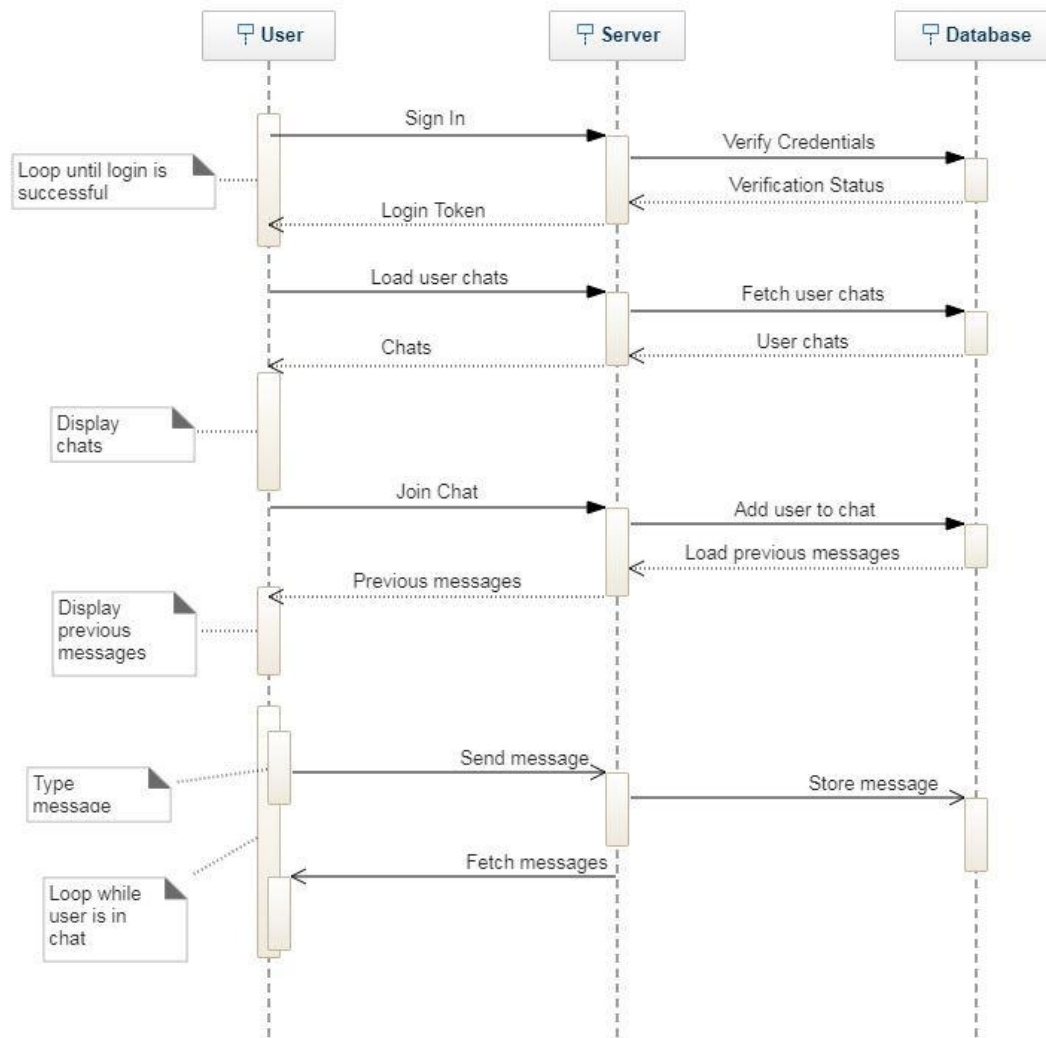
Relation Name	message			
Column Names	user_id	chat_id	message_id	text
Constraint	PK/FK(user)	PK/FK(chat)	PK	
Data Type	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)	VARCHAR(20)

4. COMPONENT DESIGN

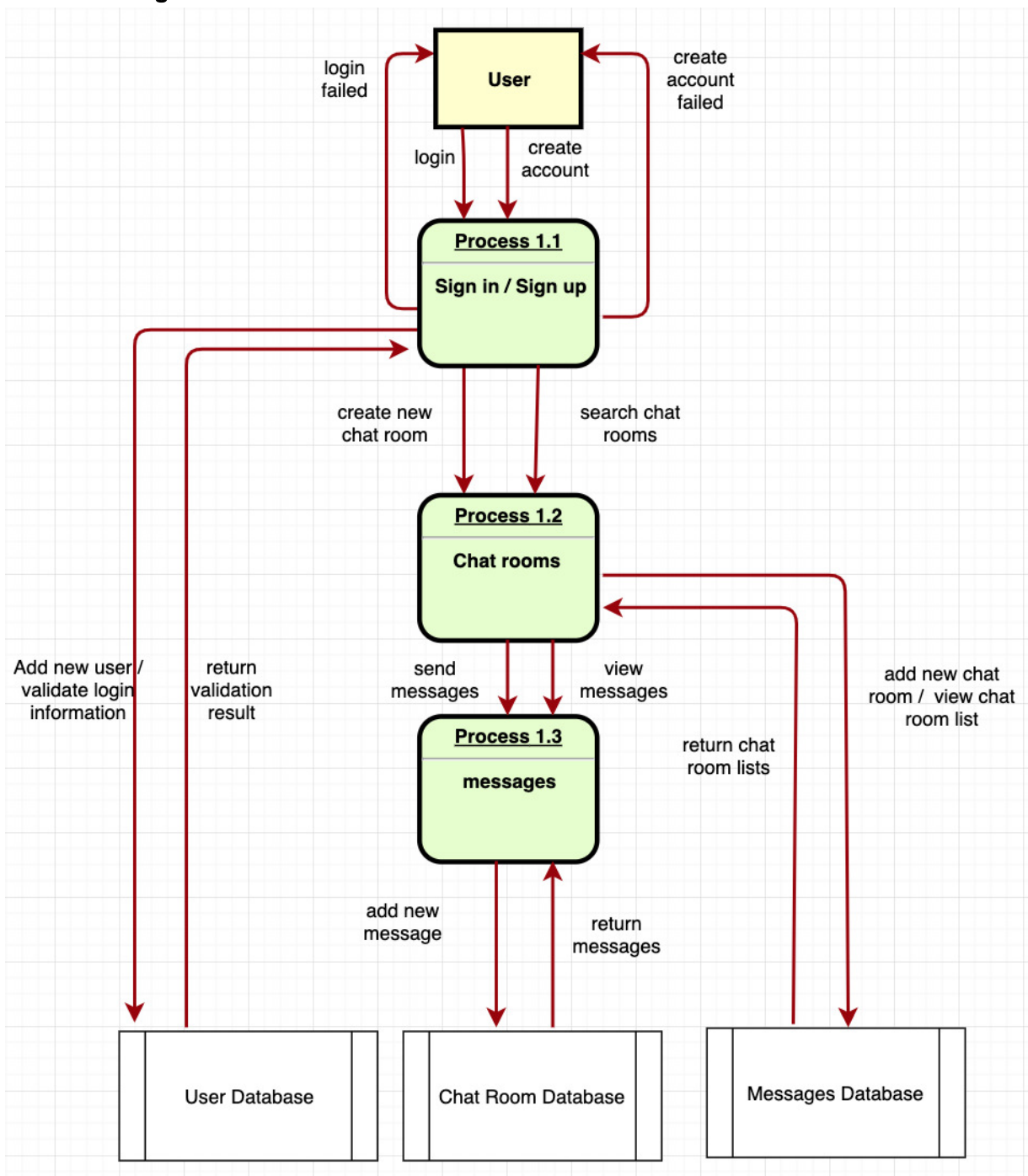
Class diagram



Sequence diagram



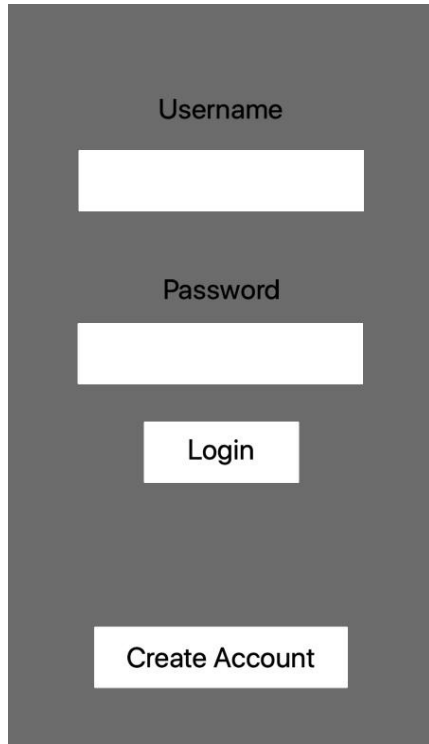
Data flow diagram



5. HUMAN INTERFACE DESIGN

5.1 Overview of User Interface & Screen Images

Login Page



A vertical UI mockup for a login page. It features a dark gray background. At the top, the label "Username" is centered above a white rectangular input field. Below this, the label "Password" is centered above another white rectangular input field. Under the password field is a white rectangular button with the text "Login" centered. At the bottom of the form is a white rectangular button with the text "Create Account" centered.

When users open the Android app, the login page will be popped up. Users could either create a new account or login with existed account.

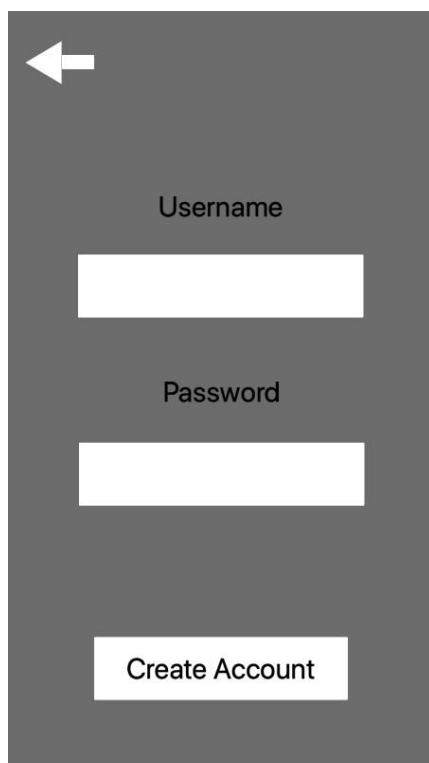
Create a new account:

After clicking the "Create Account" button, users will be redirected to the "Create Account" Page.

Login with existed account:

After typing correct username and password, users could click the "login" button to sign in. After that, users will be redirected to the Main Page.

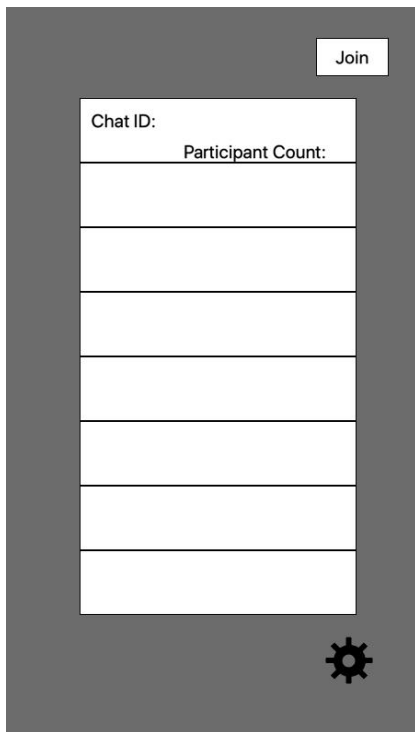
Create Account Page



A vertical UI mockup for a create account page. It features a dark gray background. In the top-left corner, there is a white left-pointing arrow. Below the arrow, the label "Username" is centered above a white rectangular input field. Below this, the label "Password" is centered above another white rectangular input field. At the bottom of the form is a white rectangular button with the text "Create Account" centered.

After typing desired username and password, users could click the "Create Account" button to create an account. Then, the app will redirect users to the "Main Page".

Chats Page



The Chats Page mockup features a dark gray background. In the top right corner, there is a white rectangular button labeled "Join". On the left side, there is a white rectangular panel. At the top of this panel, it says "Chat ID:" followed by a text input field. Below this, it says "Participant Count:" followed by another text input field. The main body of the panel consists of eight horizontal white bars, representing a list of chat items. In the bottom right corner of the dark gray background, there is a black gear icon representing settings.

There are three main components in Main page: a list of chats, a “join” button, and an “account settings” button.

List of chats:

Users can click one of the list item, and the app will redirect them to the regarding “Chat Panel”.

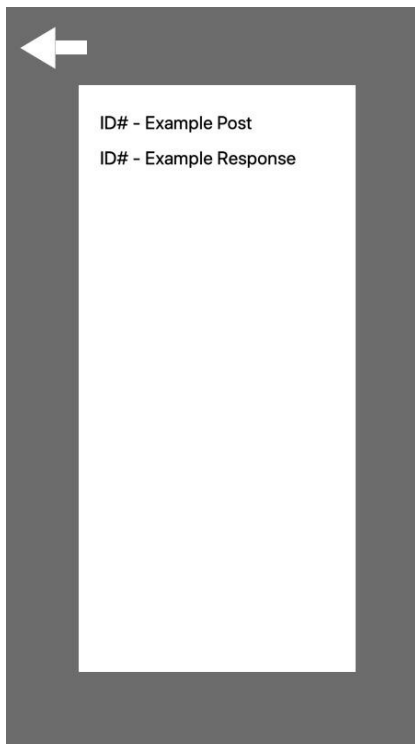
“Join” button:

A textbox will be popped up after pressing the “Join” button. Users need to type the correct chat id in order to redirect to the regarding “Chat Panel”.

“Account Settings” button:

Users will be redirected to the “Account Settings page” after pressing the “Account Settings” button.

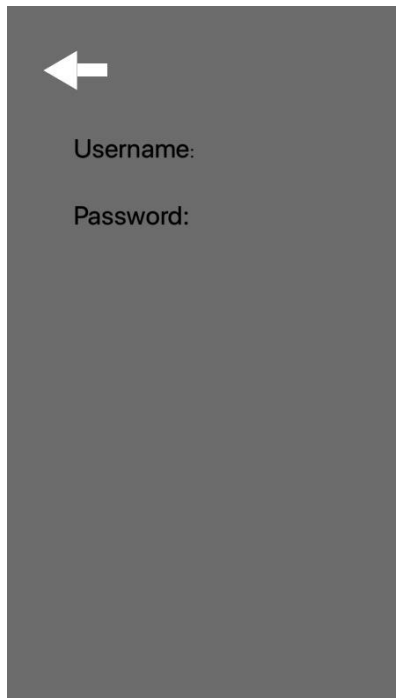
Chat Page



The Chat Page mockup has a dark gray background. In the top left corner, there is a white left-pointing arrow. Below the arrow, there is a white rectangular area. Inside this area, the text "ID# - Example Post" is displayed above "ID# - Example Response". The rest of the white area is empty, representing a space for chat messages.

Users can view chat messages and type new messages using keyboard to chat with others in real time.

Account Settings



Users can view account information on “Account Settings” page.

6. Programming Tools

Java: Most of the Android applications are developed in Java, Kotlin, or C++. Java is the most popular language in Android development, so we choose Java as our major programming language in the project.

XML (Extensible Markup Language) : Most of the Android projects use XML to design the user interface.

Android Studio: Android Studio is the official IDE for Android Development. Developers can test Android application on the laptop using Android Emulator, build local unit tests, sync gradle file automatically, etc.

7. REQUIREMENTS TRACEABILITY MATRIX

Requirement-ID	Requirement Description	Design Component	Data Design Component	Interface Design Component
3.2.1	The user needs to be able to create an account	Registration Page	User Table	Create account button
3.2.2	The user needs to be able to login	Login Page	User Table	Login button
3.2.3	The user needs to be able to logout	Account panel	None	Logout button
3.2.5	The user needs to be able to start a chat	Main Page	Chat Table	Start chat button
3.2.8	The user needs to be able to view a specific chat	Chats Page	Chat Table	Chat panel
3.2.10	The user needs to be able to send messages in the chat	Chats Page	Message Table	Send button