TeaTalk

Final Design Document

COMP 504

Rice University

Team Team

Jiaxin Wu, Lee-Hsun Hsieh, Shuai He,

Soham Pajwani, Xiangnan Chen, Yi Zhang

Contents

Section 1 - Project Description

Section 2 - Overview
- 2.1 Purpose
- 2.2 Requirements

Section 3 - Use Cases
- 3.1 Register & Login
- 3.2 Create a Chat Room
- 3.3 Before Joining a Chat Room
- 3.4 Chat After Joining a Chat Room
- 3.5 Leave or Quit a Chat Room

Section 4 - User Interface Design
- 4.1 User Interface Design and Navigation Flow

Section 5 - System Architecture

Section 6 - Software Domain Design
- 6.1 UML Diagram
- 6.2 Software Application Domain
  - 6.2.1 Controller
  - 6.2.2 Dispatch Adapter
  - 6.2.3 Interfaces
    - 6.2.3.1 IChatRoomCmd
    - 6.2.3.2 IMessageCmd
    - 6.2.3.3 IUserCmd
  - 6.2.4 Abstract Classes
    - 6.2.4.1 AChatRoom
    - 6.2.4.2 AMessage
  - 6.2.5 Concrete Classes
    - 6.2.5.1 User
- 6.3 Design Patterns

Section 7 - Data Design

Section 8 – References

Section 1 - Project Description

ChatApp Heroku link: https://chatapp-api-team-team.herokuapp.com/

The objective of this project is to design and implement an online chat application called ***TeaTalk***, which allows users with similar interests to chat with each other in various chat rooms. The target users are COMP 504 students, teaching staff, and anyone else who wants to chat with others online. The motivation of this project is to demonstrate and improve students' knowledge of object-oriented design, Javascript, and Java.

Section 2 - Overview

2.1 Purpose

The purpose of this project is to design and implement a user-friendly online chat application that satisfies all customer requirements listed below by utilizing OOD strategies learned from the COMP 504 course.

2.2 Requirements

Requirements are cited from the rubric of assignment 6.

1. A user must initiate the creation of a chat room with a room size
2. A user can only send a message to someone else in the same chat room as them
3. An unblocked user should be notified that their message has been received
4. A user may be in multiple chat rooms (public and private)
5. A user can choose to exit one chat room or all chat rooms
6. At least one of the users in the chat must be the admin
7. Admin broadcast approved requests of qualified users. Admin monitors users and messages. Admin can ban users and delete messages.
8. A user creates an account with a profile that includes their age, school, and interests
    a. Age
    b. School name (e.g. Rice, Harvard, Duke, Nanjing, Wuhan, etc…)
    c. Interests (e.g. Reading, Sports, Traveling, etc...)
9. A user can join a public chat room (with no special interests) if they are able to join. Admin invites users to join private rooms. The user does not see the chat history of messages sent before they entered the room.
10. A user will be warned and eventually forcibly banned from all chat rooms if the user uses the phrase "hate speech" in a message.
11. An unblocked user can send direct or broadcast messages to all users in the chat room
12. A user can determine who is in the chat room
13. A user can determine what chat rooms they have joined and what public rooms they can join
14. When user1 sends a message to user 2, user1's message should only appear on user1 and user 2 's screen. It should not appear on user 3's screen.

15. Messages can contain text, images (emojis), and/or links. Messages can be edited, recalled, deleted. Sending files is not supported.
16. If message msg2 is sent after message msg1, msg2 should appear on the screen below msg1
17. If a user leaves a chat room, it should be clear to the other users why the user left (voluntarily left, connection closed, forced to leave). Users can report other users.
18. No JavaScript alerts should be used in the ChatApp since they will pause the app.
19. Host the application on Heroku.

Section 3 - Use Cases

Use cases diagram below gives a brief review of how users will interact with the application. Detailed use case analysis of *ChatApp* can be categorized into the following categories.

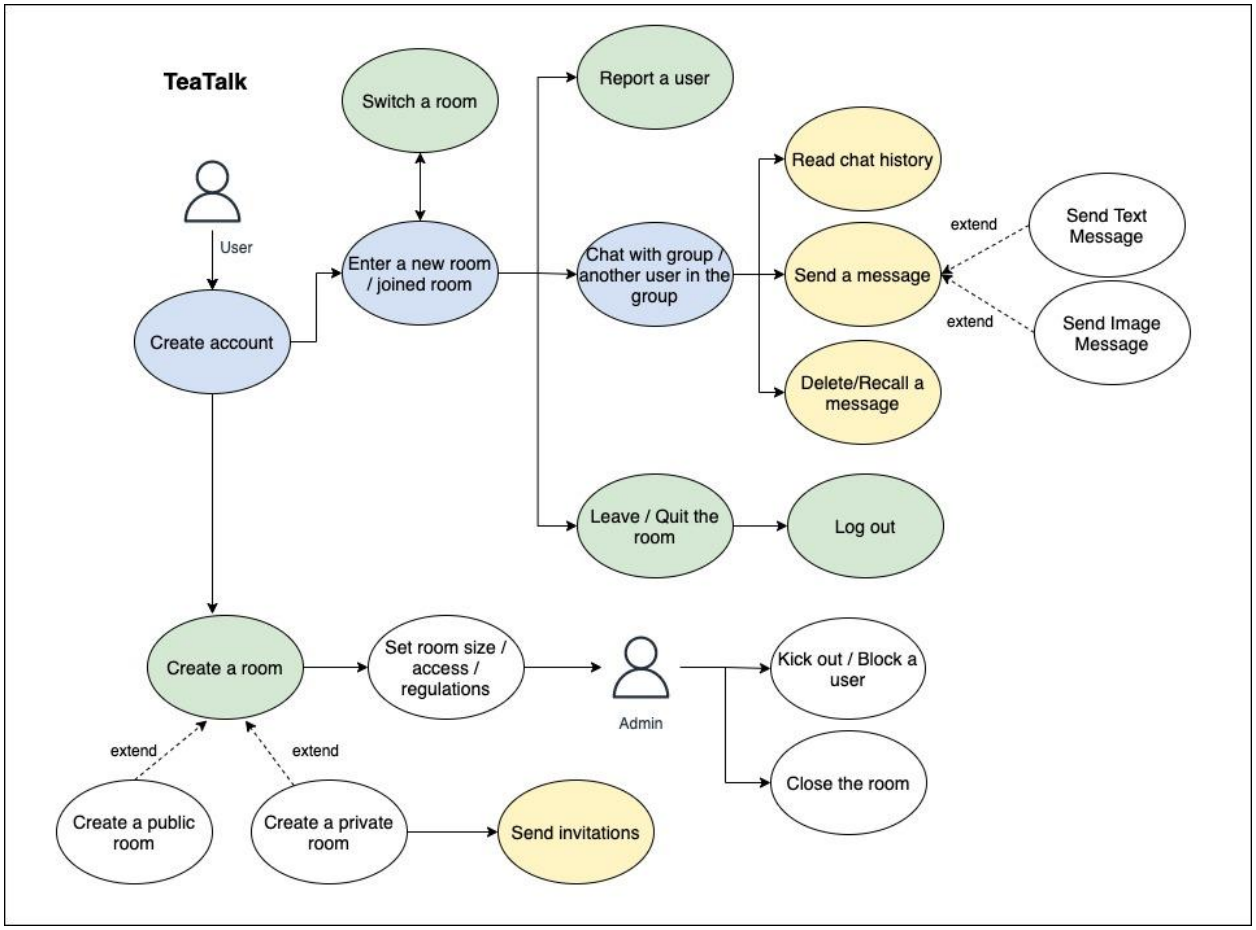

Figure 1. Use Case Diagram.

3.1 Register & Login

- Create an account with a unique id/username and password.
- Enter personal information including age, school, interest during registration.
- Login with id/username and password.

3.2 Create a Chat Room

- Create a public or private chat room(s).
- Restrict access to each created chat room.
- Set the name of each chat room.
- Set the maximum capacity of each chat room.
- Set a password to join the chat room.
- Write descriptions and rules of each chat room.
- Send invitations to other users to join a private chat room.

3.3 Before Joining a Chat Room

- Search for a chat room to join.
- Join a chat room by entering the invitation link.
- Read chat room descriptions and rules.
- Edit personal information.

3.4 Chat After Joining a Chat Room

- View other users' profiles.
- Read chat history after joining (messages should include sender and timestamp).
- Send a text or image message in multiple rooms.
- Send a text or image message to another user in the chat room.
- Read messages in multiple rooms.
- Edit a message.
- Delete a message.
- Recall a message.
- Report other users.
- Admin can block or kick out users.

3.5 Leave or Quit a Chat Room

- Exit a chat room.
- Delegate Admin role before leaving the room.
- Permanently quit a chat room.
- Broadcast reasons for leaving.

Section 4 - User Interface Design

4.1 User Interface Design and Navigation Flow

The first page a user will see after opening the website is a sign-in or sign-up page. The user can log in with email and password or create an account by entering personal information and password.

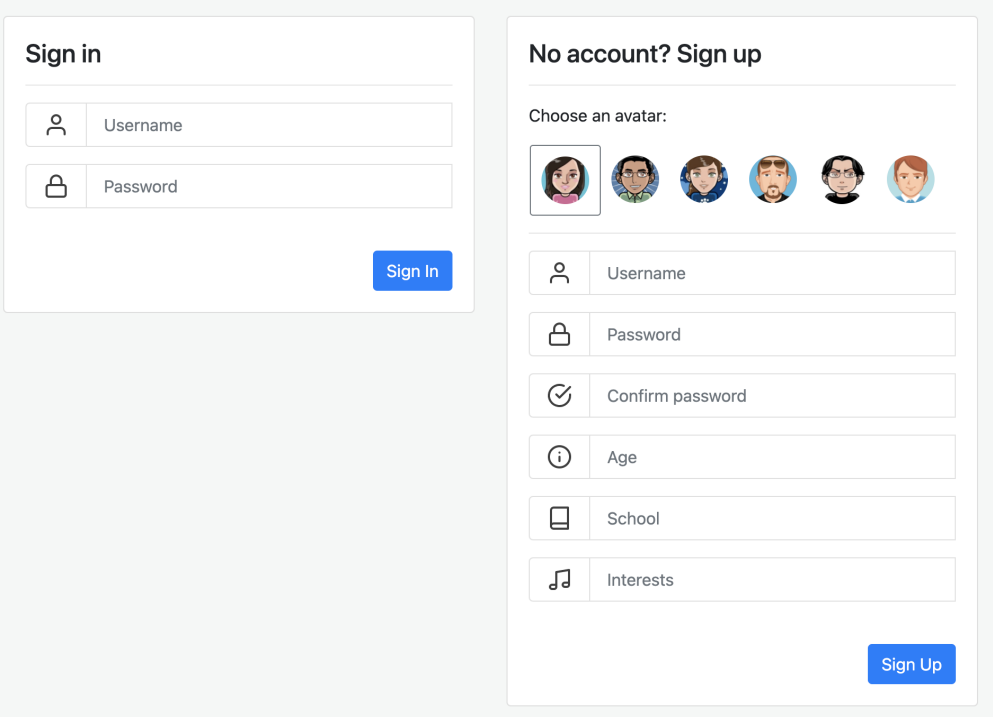

Figure 2. Sign-in & Register Page.

Next, the user will be directed to a home page showing all joined chat rooms and the current chat window (Demo in the following pictures). On the left-hand side, users will be able to search for a chat room to join, create a chat room on this page, and switch between all joined chat rooms. To create a chat room, users can set the room name, room size, room type, and room description. Users can also find a room to join by searching a room. The middle part is the current chat window, which allows users to send different types of messages, delete a message, and recall a message. The chat window will also show notifications such as “someone joined the chat”, “someone left the chat”, and “someone deleted a message”. The right-hand side of the screen will show all users in the current chat room, and a user can report another user if they violate the chat room regulations. If the user is the admin of the chat room, they will also be able to block or kick out a user. Finally, users can log out of their account by clicking the Logout button on the top right corner of the screen.

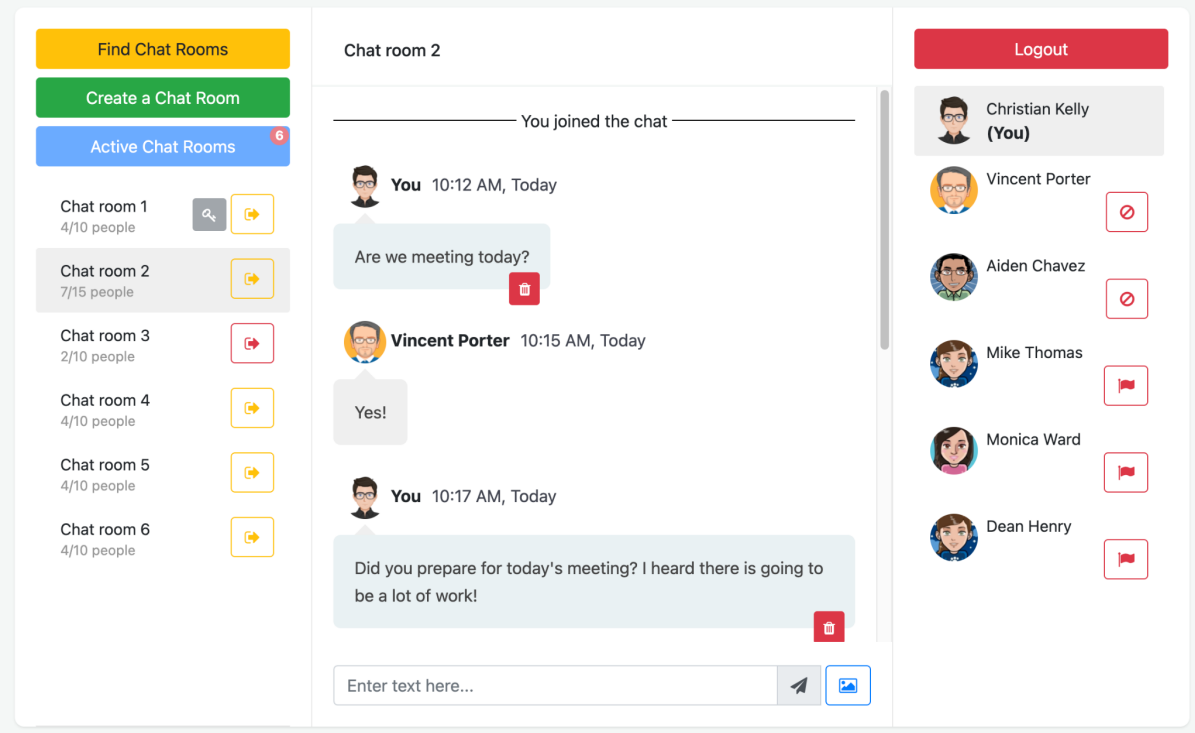

Figure 3. Main Homepage.

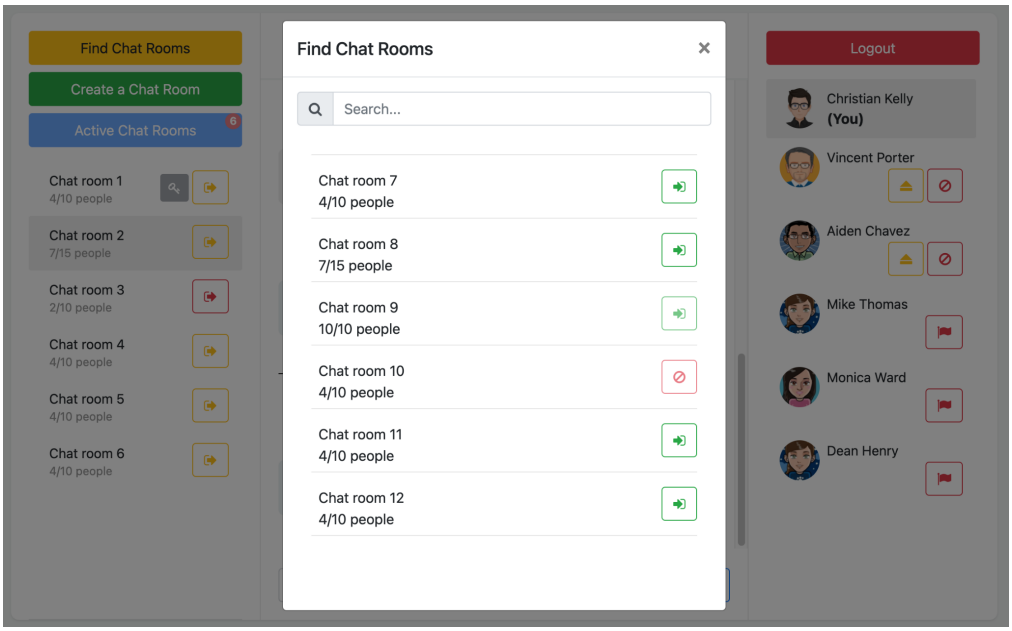

Figure 4. Find a Chat Room.

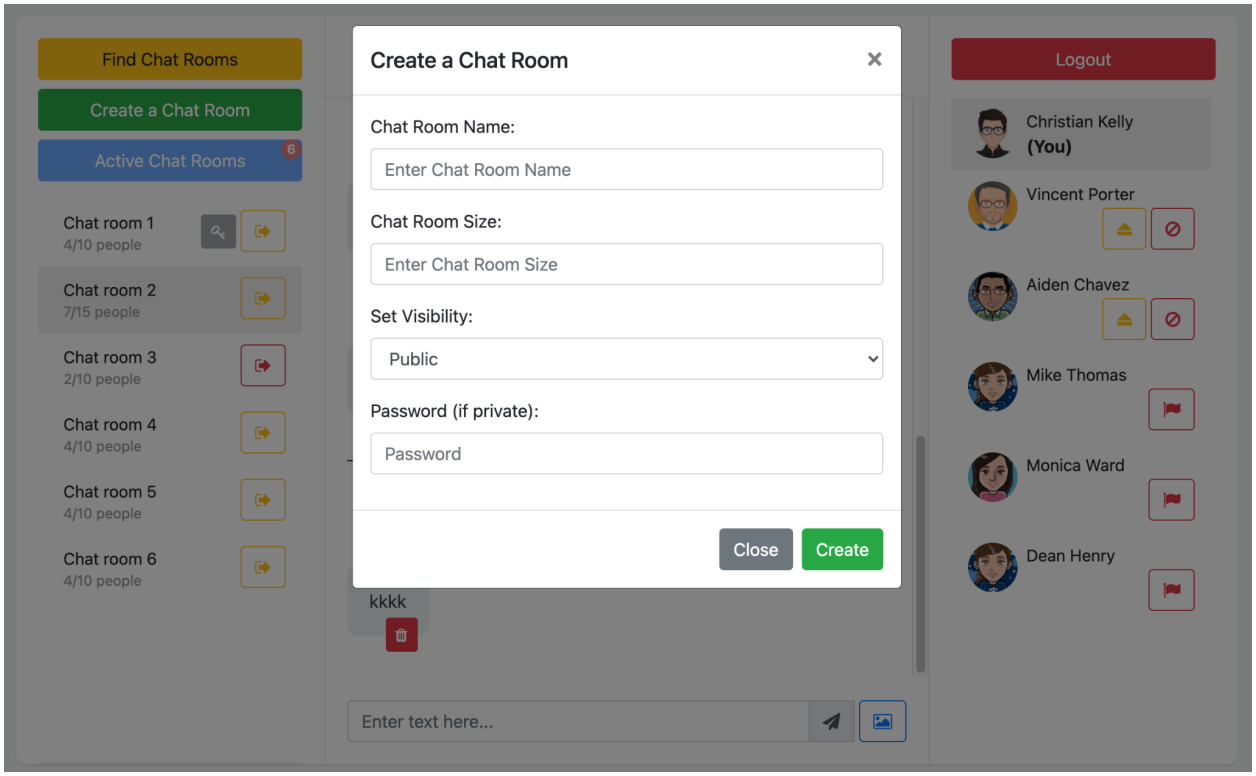

Figure 5. Create a Chat Room

Section 6 - Software Domain Design

6.1 UML Diagram

Attached is a UML diagram that shows the connections between all interfaces and classes. Detailed explanations are in the following sections.

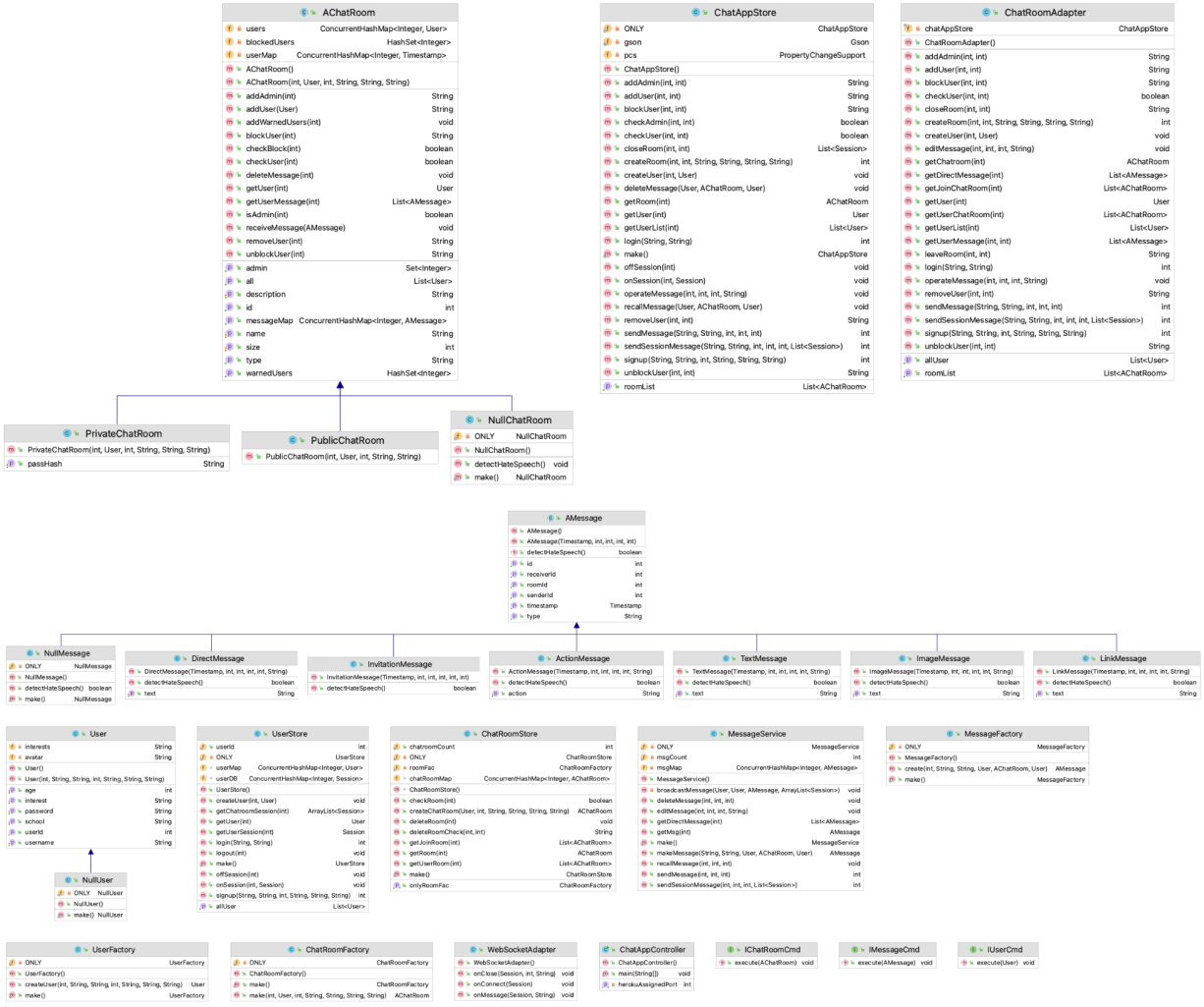

Figure 6. UML Diagram

6.2 Design Patterns

Team Team decided to take advantage of some design patterns learned from the course to make the application more robust and scalable. Five main design patterns are used: Union Design Pattern, Factory Design Pattern, Singleton Design Pattern, Command Design Pattern, and Observer Design Pattern.

- **Union design pattern** is widely used in the project especially when lots of concrete classes are sharing common fields and methods. For example, *DirectMessage, ActionMessage, TextMessage,*

and *ImageMessage* are all inherited from the abstract class *AMessage* since they share many parameters and methods.

- **Factory design pattern** is used for creating *ChatRoom*, *Message*, and *User* objects because it adds an extra layer of abstraction to hide the logic and implementation of creating those objects from the client.
- **Singleton design pattern** is used to prevent other objects from instantiating their own copies of the *Factory* and *Store* object.
- **Command design pattern** is used for updating user information, chatroom data, and chat history. There are three interfaces doing corresponding work.
- **Observer design pattern** is used where *ChatRoom* objects were treated as observers. This allows sending data to other objects effectively without any change in *ChatRoom* class and makes frequent updating *ChatRoom* more efficient.

Updates: we decided to remove Command design pattern and observer design pattern since they are not necessary.

6.3 Software Application Domain

This section describes the design and functionalities of each back-end module of the application.

6.3.1 Controller

ChatAppController will be in charge of communicating with the dispatch adapter based on requests received from view. It includes the following endpoints.

Method	**End Point**	**Parameters**	**Description**
GET	/downloadInfo/userlist	None	Get the list of all available users
GET	/downloadInfo/roomlist	None	Get the list of all chatrooms
GET	/loadUserChatRoom	:userId	Get the list of chat room an user is in
POST	/login	{ username, password }	For users to log in.
POST	/signup	{ username, password }	For users to sign up.
POST	/leaveChatroom		For users to leave a chat room.
POST	/joinChatroom	{chatroomId}	For users to join a chatroom.
POST	/uploadInfo	{ type, info }	For users to upload messages or other info.
POST	/operationToChatroom	{ adminUserId, chatroomId, operation }	For the admin to manage the chat room.
POST	/operationToUser	{ adminUserId,	For the admin to manage users.

		targetUserId, chatroomId, operation}	
POST	/operationToMsg	{ userId, chatroomId, msgId, operation }	For users to manage messages (delete, recall, edit .etc).

6.3.2 Dispatch Adapter

A *chatRoomAdapter* is responsible for communicating between the frontend and the *ChatAppStore* class. *ChatAppStore* contains *ChatRoomStore, UserStore and MessageService. ChatRoomStore* contains a concurrent hashmap that stores all the *AChatRoom* by its id. *UserStore* contains a concurrent hashmap that stores all the *User* by their id. The MessageService is responsible for sending, receiving, and recalling messages in the corresponding user/chat room. All the commands will first go to *ChatRoomAdapter*, then the *ChatAppStore*. After that, it would call the components that are required to finish the command. *chatRoomAdapter* has the following methods.

Method	**Description**
addUser(int chatRoomId, int userId)	Adds a user to a designated chat room.
removeUser(int chatRoomId, int userId)	Removes a user from a designated chat room.
blockUser(int chatRoomId, int userId)	Blocks a user from a designated chat room.
setAdmin(int chatRoomId, int userId)	Sets a user as admin of a designated chat room.
checkUser(int chatRoomId, int userId)	Checks if a user is in a certain chat room.
sendMessage(String type, String data, User sender, AChatRoom chatRoom, User receiver)	The user sends a message in a room or to another user.
recallMessage(String type, String data, User sender, AChatRoom chatRoom, User receiver)	The user recalls a message.
createUser(int userId, User user)	Adds a new user to the store.
login(String userName, String password)	Checks if the username and password are correct.
getUser(int userId)	Gets a user based on user ID.

6.3.3 Interfaces

ChatApp has three interfaces, *IChatRoomCmd, IMessageCmd, IUserCmd.* They are in charge of passing commands to their corresponding stores.

6.3.3.1 IChatRoomCmd

IChatRoomCmd is an interface used to pass commands to chat rooms in the *ChatRoomStore.* All chat rooms must execute the command.

Method	Description
execute(AChatRoom context)	Pass command to a *AChatRoom* object.

6.3.3.2 IMessageCmd

IMessageCmd is an interface used to pass commands to messages in the *MessageStore*. All messages must execute the command.

Method	Description
execute(AMessage context)	Pass command to an *AMessage* object.

6.3.3.3 IUserCmd

IUserCmd is an interface used to pass commands to users in the *UserStore*. All users must execute the command.

Method	Description
execute(User context)	Pass command to a *User* object.

6.3.4 Abstract Classes

There will be two abstract classes. They are determined to be abstract classes because there will be different types of messages (direct messages, group messages, .etc) and different types of chat rooms (private and public).

6.3.4.1 AChatRoom

AChatRoom is an abstract class for chat rooms that contains room id, room admin, room users, and other room data. *AChatRoom* contains the following fields and methods.

Field	Type	Description
id	int	A unique ID of the chat room.
admins	User	The admin user of the chat room.
size	int	Chat room capacity.
description	String	Chat room description.
name	String	Chat room name.
type	String	Chat room type (private or public).
userMap	HashMap	Stores the user and their first enter timestamp.

userDB	userDB	Stores all users and their sessions.
messageMap	HashMap	Stores messages in this chatroom and their IDs.
users	HashMap	Stores users in this chatroom and their IDs.

Method	**Description**
addUser(User)	Adds a user to the chat room.
receiveMessage(AMessage)	Adds a message to the chat room.
getAll()	Gets all users in the chat room.
checkUser(User)	Checks if a user is in the chat room.
checkBlock(User)	Checks if a user is blocked in the chat room.

6.3.4.2 AMessage

AMessage is an abstract class for messages that contains message timestamp, message-id, message sender, and message receiver of a message. *AMessage* contains the following fields.

Field	**Type**	**Description**
timestamp	Timestamp	The time when a message is sent.
id	int	The message's unique ID.
sender	String	The user who sent the message.
receiver	String	The user/room that receives the message

6.3.5 Concrete Classes

Since concrete class design is not in the scope of this assignment. This document only briefly talks about some concrete classes.

6.3.5.1 User & NullUser

User class is a concrete class that stores all data for individual users. *User* is decided to be a concrete class because we will only have one type of user, unlike messages and chat rooms. There will not be an admin user and a regular user, because we will store the admin of the chat room in the *ChatRoom* object. *NullUser* is a concrete class to make the program more robust.

Field	**Type**	**Description**
userId	int	A unique ID for the user.
username	String	The username of the user.
password	String	The password of the user.
age	int	Age of the user.
school	String	School of the user.
interests	String[]	Interests of the user.

6.3.5.2 ActionMessage, DirectMessage, ImageMessage, LinkMessage, TextMessage, & NullMessage

There will be six concrete classes extending the *AMessage* abstract class including *DirectMessage, TextMessage, ImageMessage, ActionMessage, LinkMessage*, and *NullMessage*. *DirectMessage* is the message sent from a user to another user. *TextMessage* is the message sent from a user to a chat room. *ImageMessage* is the emoji message sent by a user. *LinkMessage* is for link messages such as invitation links. *TextMessage* is for regular string messages, and *NullMessage* is a null message object that makes the application more robust by helping to avoid null pointer exceptions.

6.3.5.3 PublicChatRoom, PrivateChatRoom, NullChatRoom

There will be three concrete chat room classes extending the *AChatRoom* abstract class. PrivateChatRoom is a room that needs invitation links to join, and PublicChatRoom is a chat room that all users can join.

6.3.6 Factories

Three factory classes *UserFactory, ChatRoomFactory,* and *MessageFactory* are used to create various concrete users, chat rooms, and messages mentioned in the previous session.

Section 7 - Store

Team Team decided to use in-memory storage to store user and chat room data. There will be *ChatAppStore*, *ChatRoomStore*, and *UserStore* to store data of all chatrooms and users. More detailed data design is beyond the scope of this assignment, so it is saved for the next step.

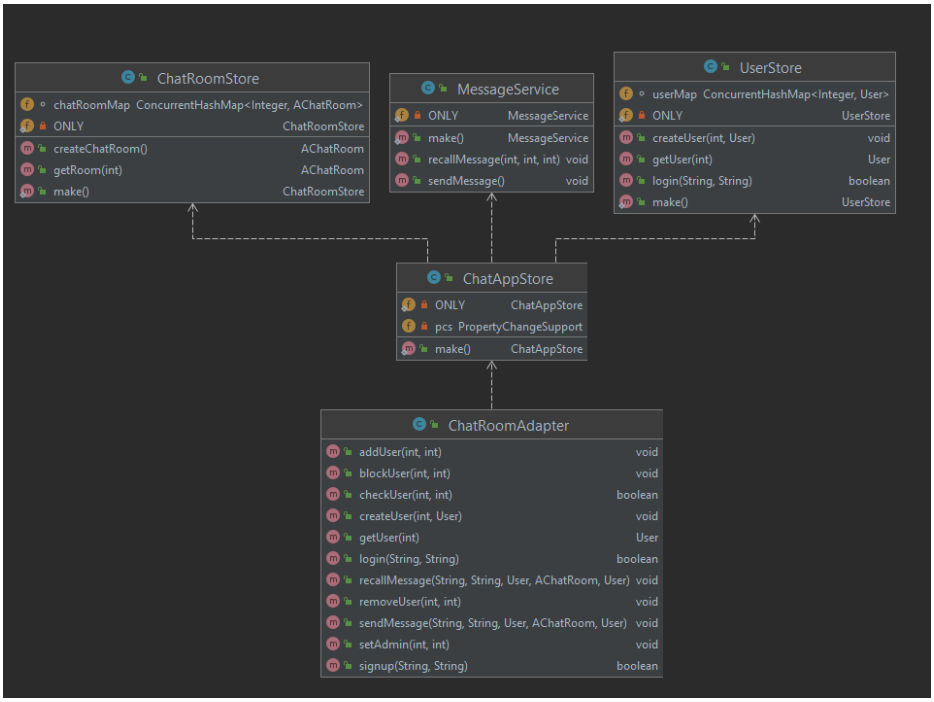

Figure 7. Data Storage Design.

7.1 ChatAppStore

ChatAppStore stores all chat room operations such as addUser, removeUser, closeRoom, blockUser, unblockUser, addAdmin, checkUser, checkAdmin, sendMessage, recallMessage, etc.

7.2 ChatRoomStore

ChatRoomStore stores all chat rooms and its corresponding IDs in the chat app. The store class also contains multiple functions for chat room operations such as *createChatRoom, getRoom, makeRoom, deleteRoom, getUserRoom, getJoinRoom*.

7.3 UserStore

UserStore stores all users and their sessions and IDs. It also contains all user operations including *createUser, signup, login, getUser, getUserSession, getAllUser, getChatroomSession, onSession, offSession.*

7.4 MessageService

MessageService stores all messages and their IDs and message operations including *makeMessage, getMessage, sendMessage, recallMessage, deleteMessage, broadcastMessage, getDirectMessage.*

Section 8 – References

- Course Rubric -- https://www.clear.rice.edu/comp504/#/assignments