Chat App API Design

Team I: Sammy The Owl

Xiao Xin, Zhijian Yao, Weiwei Zhou, Xinru Xiao, Xuyang Xiao, Wenlong Yan

1. Summary

Our ChatApp will have the following features.

- A user must initiate the creation of a chat room
- A user can only send a message to someone else in the same chat room as them
- A unblocked user should be notified that their message has been received
- A user may be in multiple chat rooms (public and private)
- A user can choose to exit one chat room or all chat rooms
- At least one of the users in the chat must be the admin
- Admin broadcast approved requests of qualified users. Admin monitor users and messages. Admin can ban users and delete messages.
    - Age
    - School (e.g. Rice, Harvard, Duke, Nanjing, Wuhan, etc…)
    - Interests (e.g. Reading, Sports, Traveling, etc...)
    - A user has a profile that includes their age, school, and interests
- A user can join a public chat room (no special interests) if they are qualified to join. Admin invite users to join private rooms.
- A user will be warned and eventually forcibly banned from all chat rooms if the user uses the word “hate” in a message.
- An unblocked user can send direct or broadcast messages to all users in the chat room
- A user can determine who is in the chat room
- A user can determine what chat rooms they have joined and what public rooms they can join
- When user1 sends a message to user2, user1's message should only appear on user1 and user2's screen. It should not appear on user3's screen.
- Messages can contain text, images (emojis), and/or links. Messages can be edited, recalled, deleted.
- If message msg2 is sent after message msg1, msg2 should appear on the screen below msg1
- 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.
- No JavaScript alerts should be used in the ChatApp since they will pause the app.

2. Use Case

a.

Use Case UC1	Enter ChatApp website
Primary Actor	User
Secondary Actor	Server
Pre condition	The user must have a good internet connection.
Post condition	The user accesses the ChatApp website successfully.
Main Flow	1. The user enters the url of our ChatApp in the browser.
Second Flow	N/A
Exceptions	N/A

b.

Use Case UC2	Log into ChatApp
Primary Actor	User
Secondary Actor	Server
Pre condition	1. The user accesses the ChatApp website successfully. (UC1)
Post condition	1. The server creates a profile for the user. 2. The user enters the main page.
Main Flow	1. The user enters his/her name. 2. The user enters his/her school (e.g. Rice, Harvard, Duke, Nanjing, Wuhan, etc…). 3. The user chooses multiple interests (e.g. Reading, Sports, Traveling, etc...). 4. The user clicks the “Enter” button.
Second Flow	N/A
Exceptions	1a. If the name is blank. Ask the user to enter the name. 1b. If the name is duplicated. Ask the user to choose a different name. 2a. If the school is empty. Ask the user to enter the

	school. 3a. If the interest. Ask the user to choose at least one interest.

c.

Use Case UC3	Create a chat room with restrictions
Primary Actor	User
Secondary Actor	Server
Pre condition	1. The user has accessed the ChatApp successfully. (UC1) 2. The user has created a profile. (UC2)
Post condition	1. The chat room is created successfully. 2. The room name is shown on the page. 3. The user becomes the admin of this room.
Main Flow	1. The user enters the room name. 2. The user sets the room as public.
Second Flow	1. The user enters the room name. 2. The user sets the room as private. 3. The user sets restricted topics.
Exceptions	N/A

d.

Use Case UC4	Send a message
Primary Actor	User
Secondary Actor	User
Pre condition	1. Both senders and receivers are in the same chat room.
Post condition	1. A new message appears in the current chat room and is only visible to the sender and the receiver.
Main Flow	1. The user clicks on other users from the attendee list in the same chat room. 2. The user enters the message. 3. The user clicks on the “Send” button.
Second Flow	N/A

Exception	2a. If the content includes “hate”. The user will be warned and banned from **all chat rooms**. 3a. If the message is not sent successfully. The sender will receive an alert. 3b. If the sender is blocked by the receiver. The sender will receive an alert.

e.

Use Case UC5	Receive notification -- “message has been received”
Primary Actor	User (Sender)
Secondary Actor	User (Receiver)
Pre condition	1. The sender is not blocked by the receiver. 2. The sender has sent a message to the receiver. (UC4 and UC8)
Post condition	1. “Your message has been received” appears on the dashboard of the sender.
Main Flow	1. The sender receives the message-received notification.
Second Flow	N/A
Exception	N/A

f.

Use Case UC6	Join a public chat room
Primary Actor	User
Secondary Actor	Server
Pre condition	1. The room has logged into ChatApp. (UC2) 2. Some rooms have been created.
Post condition	1. The name of the user appears on the attendee list of the chat room he joins.
Main Flow	1. The user chooses a room and clicks on the “join” button. 2. The user meets all the requirements of the

	room.
Second Flow	N/A
Exception	2a. If the user fails to meet all the requirements. The user will receive a notification.

g.

Use Case UC7	Exit a chat room
Primary Actor	User
Secondary Actor	User
Pre condition	1. The user is in a chat room. 2. The user is not admin.
Post condition	1. The current user name is removed from the attendee list of the current room. 2. All the other users in the chat room receive a notification. (UC9) 3. The chat room is removed from the joined room list of the user. 4. The name of the user is removed from the attendee list of the chat room he just exited.
Main Flow	1. The user clicks the “exit” button of one chat room.
Second Flow	N/A
Exception	1a. If the room becomes empty or the admin leaves the room. Dismiss the chat room.

h.

Use Case UC8	Broadcast a message
Primary Actor	User
Secondary Actor	User
Pre condition	1. The senders and the receivers are in the same room.
Post condition	1. A new message appears on the dashboard of the receivers.
Main Flow	1. The user chooses “all attendees” and sends

	a message.
Second Flow	N/A
Exception	1a. If the content includes “hate”. The user will be warned and banned from all chat rooms.

i.

Use Case UC9	Receive notification -- “A user just left the room”
Primary Actor	User
Secondary Actor	User
Pre condition	1. The user is in the same chat room as the user who just left.
Post condition	1. A notification appears on the dashboard of every user.
Main Flow	1. All the users receive a notification -- “A user just left the room”.
Second Flow	N/A
Exception	N/A

j.

Use Case UC10	Invite a user to a private room
Primary Actor	Admin
Secondary Actor	User
Pre condition	1. The admin is in a private room. 2. The user to be invited is not in the private room as the admin.
Post condition	1. The user joins the private room regardless of his interests.
Main Flow	1. Admin clicks the “invite” button and input the username of the user to be invited.
Second Flow	N/A
Exception	1a. If the user to be invited does not exist. Notify the admin. 2a. If the user has been banned to enter the room

	before. Notify the user.

k.

Use Case UC11	Edit a message
Primary Actor	User
Secondary Actor	Message
Pre condition	1. The user is in a chat room. 2. The message to be edited was sent by the user.
Post condition	1. The content of the message is updated on all dashboards of all the receivers. 2. All receivers receive a notification -- “The message has been recalled”.
Main Flow	1. The user clicks the “edit” button of a message. 2. The user edits the message and clicks the “done” button.
Second Flow	N/A
Exception	2a. If the content includes “hate”. The user will be warned and banned from all chat rooms.

l.

Use Case UC12	Delete a message
Primary Actor	User
Secondary Actor	Message
Pre condition	1. The user is in a chat room. 2. The message to be deleted was sent by the user.
Post condition	1. The content of the message is removed from the dashboard of the current user.
Main Flow	1. The user clicks the “delete” button of the message.
Second Flow	N/A
Exception	N/A

m.

Use Case UC13	Recall a message
Primary Actor	User
Secondary Actor	Message
Pre condition	1. The user is in a chat room. 2. The message to be recalled was sent by the user.
Post condition	1. The message is removed from all dashboards of other users in the same chat room. 2. The other users in the same chat room receive a notification -- “A message has been recalled”.
Main Flow	1. The user clicks the “recall” button of the message.
Second Flow	N/A
Exception	N/A

n.

Use Case UC14	Admin forces a user to leave
Primary Actor	Admin
Secondary Actor	User
Pre condition	1. The admin and the user are in the same chat room.
Post condition	1. The user’s name is removed from the attendee list of the chat room. 2. All other users receive a notification. (UC9)
Main Flow	1. The admin chooses one user and clicks the “force leave” button.
Second Flow	N/A
Exception	N/A

o.

Use Case UC15	Users report a user

Primary Actor	User
Secondary Actor	User
Pre condition	1. The reporter and the reportee are in the same room. 2. The reporter has sent at least one message in the chat room.
Post condition	1. The admin receives a reporting message.
Main Flow	1. The reporter chooses one message and inputs his/her reason. 2. The reporter clicks the “report” button.
Second Flow	N/A
Exception	N/A

p.

Use Case UC16	Receive a reporting message
Primary Actor	Admin
Secondary Actor	Server
Pre condition	1. A user reports another user.
Post condition	1. The reporting message disappears. 2. The reportee is removed from that chat room.
Main Flow	1. The admin clicks the “agree” button.
Second Flow	1. The admin clicks the “disagree” button.
Exception	N/A

q.

Use Case UC17	Exit all chat rooms
Primary Actor	User
Secondary Actor	User
Pre condition	1. The user has joined at least one chat room.
Post condition	1. The user exits all chat rooms. 2. All the other users in the chat room receive a

	notification. (UC9) 3. All chat rooms are removed from the joined room list. 4. The name of the user is removed from all the attendee list of the chat rooms he just exited.
Main Flow	1. The user clicks the “exit all” button.
Second Flow	N/A
Exception	1a. If the user has not joined any rooms. The server does nothing.

r.

Use Case UC18	Block a user
Primary Actor	User
Secondary Actor	User
Pre condition	1. The two users are in the same chat room.
Post condition	1. The user is added to the blocked list of the other user.
Main Flow	1. The user chooses to block the other user and clicks the “block” button.
Second Flow	N/A
Exception	1a. If the to-be-blocked user is currently offline. The server does nothing.

3. API Design

For this assignment, we use the Model-View-Controller design pattern.
singleton and command design patterns. All data will be in JSON-format. The data is only transferred via WebSocket protocol.

3.1 View

We use React.js and AntUI Design on the front-end. The front-end mainly consists of two parts -- a login form and a chatting page. After logging into our chatApp successfully, the user will be redirected to our main chatting page.

The login page has a login form which includes the following parts:

1) Name
2) Age
3) School
4) Interests

The main chatting page is divided into several parts:

1) A list of rooms that the user has joined. Each room has a button to exit.
2) A list of all rooms, which includes both available and unavailable rooms. The unavailable rooms are shown in grey
3) A list of all the current users in the chat room. The user can click a username in the list to send a private message. Below the user list, there is an “invite” button and a “force to leave” button for admin.
4) A chat window where the user can receive and send public or private messages.
5) A button for creating a new chat room with restrictions.
6) A button for exiting all rooms.
7) Each message sent by the owner has a delete button, a recall button and an edit button. Each message received from other users has a report button.

3.2 Model

The model mainly consists of a User Class, a ChatRoom Class, a Message Class, a dispatcher adapter and commands.

3.2.1 User

Class User has four fields -- age, name, school, interest. It also includes all getter and setter functions, as well as a constructor method.

Type	Field	Description
Private Int	age	The age of the user.
Private String	name	The name of the user.
Private String	school	The school of the user.

Private String[]	interest	The interest of the user.

3.2.2 ChatRoom

Class ChatRoom represents the chat room. It contains 3 fields. It also includes all getter and setter functions, as well as a constructor method.

Type	Field	Description
Private String	name	The chat room name.
Private User	owner	The owner of the room
Private List<String>	interestRequirement	The requirements of the chatRoom. Only the user whose interests match the requirements can join this room.

According to the use cases we mentioned above, we use field -- *interestsRequirement* to mark a chat room as private or public. If the field interestsRequirement is null, the chat room is private. Otherwise, the room is public.

3.2.3 Message

Class Message represents the message that the user sends. It contains 4 fields. It also includes all getter and setter functions, as well as a constructor method.

Type	Field	Description
Private String	text	The content of the message.
Private String	sender	The sender of the message.
Private String	chatRoom	The requirements of the chatRoom. Only the user whose interests match the requirements can join this room.

Private LocalTime (java.time.LocalTime)	time	The time when the message was sent.

3.2.4 DispatchAdapter

The adapter interfaces with the models and the controllers. It also uses a singleton design pattern in order to guarantee the consistency of the data. It contains 7 concurrent maps to store all room, user and message data and a concurrent list to store all the banned users.

Type	Field	Description
Private Static DispatchAdapter	singleton	Singleton object.
Public Static Map<Session, User>	session2user	Map a session to a user.
Public Static Map<String, ChatRoom>	name2ChatRoom	Map a room name to a chatRoom.
Public Static Map<ChatRoom, List<User>>	chatRoom2listUser	Map a room to its corresponding list of users.
Public Static Map<List<User>, ChatRoom>	listUser2chatRoom	Map a list of users to its chatRoom.
Public Static Map<User, Session>	user2session	Map a user to a session.
Public Static Map<User, List<User>>	user2blockList	Map a user to his/her blocked user lists.
Public Static List<User>	chatRoomBanList	A list of users banned from accessing all chatRooms.

3.2.5 AbstractCmd

We use an abstract class named Acmd which includes an abstract execute method. It also includes functions that allows the server to send different types of messages to the user.

Each concrete command class will implement the abstract execute method. The concrete commands use a singleton design pattern except for the ones where variables need to change. Based on the logic that each concrete command deals with, it can call sendErrorMsg / sendUserMsg / sendSystemMsg functions extended from the Class AbstractCmd to communicate with the user if needed.

According to the requirements of this assignment, we will need the following concrete classes to handle different user actions.

1) BanCmd
2) BlockCmd
3) BroadcastMsgCmd
4) CreateRoom
5) DeleteMsgCmd
6) RecallMsgCmd
7) EditMsgCmd
8) GetARoomCmd
9) LeaveRoomCmd
10) InviteCmd
11) NotifyOwnerCmd
12) LeaveAllRoomCmd
13) SendMsgCmd

3.3 Controller

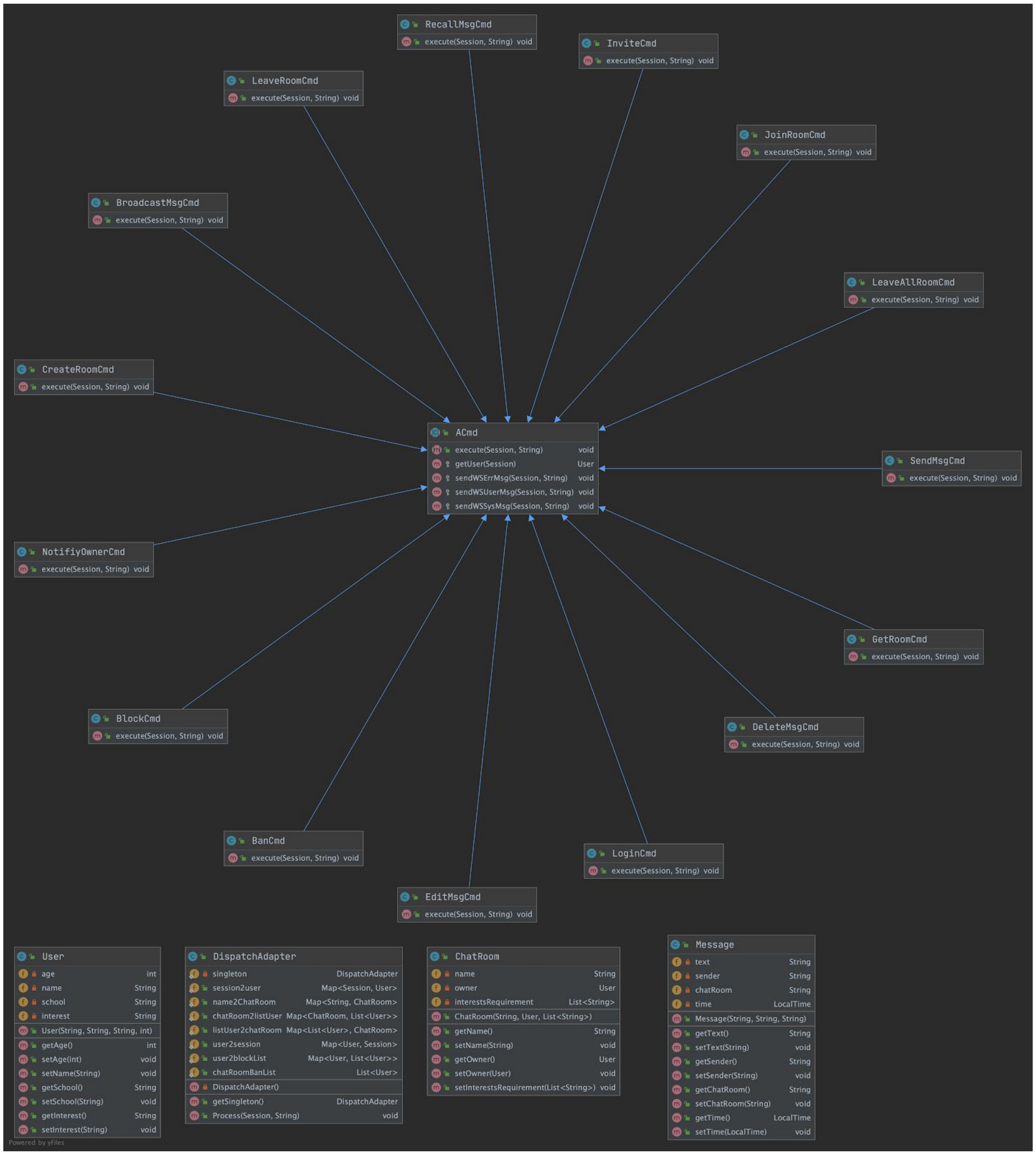
3.3.1 ChatAppController

The chat app controller communicates with all the clients on the web socket. It has a main function, which is the entry point of this chat app. It also includes methods for hosting the app on Heroku.

3.3.2 WebSocketController

The web socket controller creates a web socket for the server. It contains methods -- onConnect(), onClose() and onMessage().

4. UML Diagram


RecallMsgCmd
execute(Session, String) void
InviteCmd
execute(Session, String) void
LeaveRoomCmd
execute(Session, String) void
JoinRoomCmd
execute(Session, String) void
BroadcastMsgCmd
execute(Session, String) void
LeaveAllRoomCmd
execute(Session, String) void
CreateRoomCmd
execute(Session, String) void
ACmd
execute(Session, String) void
getUser(Session) User
sendWSErrMsg(Session, String) void
sendWSUserMsg(Session, String) void
sendWSSysMsg(Session, String) void
SendMsgCmd
execute(Session, String) void
NotifiyOwnerCmd
execute(Session, String) void
GetRoomCmd
execute(Session, String) void
BlockCmd
execute(Session, String) void
DeleteMsgCmd
execute(Session, String) void
BanCmd
execute(Session, String) void
LoginCmd
execute(Session, String) void
EditMsgCmd
execute(Session, String) void
User
age int
name String
school String
interest String
User(String, String, String, int)
getAge() int
setAge(int) void
setName(String) void
getSchool() String
setSchool(String) void
getInterest() String
setInterest(String) void
DispatchAdapter
singleton DispatchAdapter
session2user Map<Session, User>
name2ChatRoom Map<String, ChatRoom>
chatRoom2listUser Map<ChatRoom, List<User>>
listUser2chatRoom Map<List<User>, ChatRoom>
user2session Map<User, Session>
user2blockList Map<User, List<User>>
chatRoomBanList List<User>
DispatchAdapter()
getSingleton() DispatchAdapter
Process(Session, String) void
ChatRoom
name String
owner User
interestsRequirement List<String>
ChatRoom(String, User, List<String>)
getName() String
setName(String) void
getOwner() User
setOwner(User) void
setInterestsRequirement(List<String>) void
Message
text String
sender String
chatRoom String
time LocalTime
Message(String, String, String)
getText() String
setText(String) void
getSender() String
setSender(String) void
getChatRoom() String
setChatRoom(String) void
getTime() LocalTime
setTime(LocalTime) void
Powered by yFiles