ChatApp Design API Specification

Team Houston
Jonathan Wang, Manu Maheshwari, Yanjun Yang,
Tianqi Ma, Youyi Wei, Jiang Lin, Chengyin Liu

We used command pattern, composite pattern and singleton pattern in this chat app. We have two controllers ChatAppController and WebSocketController to handle the requests and communications. All the objects use commands to act in the chat app.

In this document, we will discuss the use cases for the chat app. Then we will break down the interfaces and abstract classes in the model.

1 Use Cases

- Open ChatApp
  - Return internal user id counter to view
  - Start in login screen: create new user or map session to existing username/user
- Create ChatRoom
  - POST(restrictions, username)
  - ChatAppController: create chat room, add room to observers, add user to room
- Join ChatRoom
  - Select room to join from list of rooms user can join
  - POST(user, room to join)
  - ChatAppController: add user as observer of room, add room to user list
- Leave ChatRoom
  - User can choose to leave the ChatRoom.
  - POST(user, room to leave)
  - ChatAppController: delete user from the room, remove room from the user list

- Send message
  - Select user(s) in chat room to send message to, type message
  - Websocket sends JSON(sender, recipient, chat room, message)
  - WebSocketController: onMessage - use commands, notify observer to send message to recipient
- Check ChatRooms User is in/can join, Users in ChatRoom

2 View

We have a login screen, a chat room list, chat room, create chat room and buttons for send message, join/leave chat room etc. in the view.

3 Controller

We have two controllers: ChatAppController and WebSocketController.

- ChatAppController is an observable observed by chat rooms, and singleton pattern is also implemented here. WebSocketController will call the ChatAppController instance to communicate. It has three fields and eleven methods:
  - sessionUsernameHashmap: a ConcurrentHashMap maps the session and username.
  - usernameUserHashmap: a ConcurrentHashMap maps the username and user object.
  - chatAppController: an instance of itself.
  - ChatAppController(): constructor.
  - getInstance(): get singleton instance of ChatAppController.
  - main(String[] args): ChatApp entry point.
  - logIn(Session user, String request): creates a new user if user not created or retrieves an existing user.
  - getEligibleChatRooms(Session user, String request): get all chatrooms user is in or can join. Send as JSON to session.
  - createChatRoom(Session user, String request): create a chat room. Send updated lists as JSON to session.
  - joinChatRoom(Session user, String request): join a chat room. Send updated lists as JSON to session.
  - getChatRoom(Session user, String request): get users in chat room and chat history. Send as JSON to session.

  - leaveChatRoom(Session user, String request): exit one or all chat rooms. Send updated lists as JSON to session.
  - sendMessage(String request): send a message command to all the chat room observers.
  - getHerokuAssignedPort(): get the heroku assigned port number.

- WebSocketController creates a web socket for the server, it also manages all the end points. We have seven end points:
  - "log_in": create or retrieve user in hash map.
  - "get_eligible_chat_rooms": get list of chat rooms that user is in or can join.
  - "join_chat_room": join an existing chat room.
  - "get_chat_room": get users in chat room and message history when entering chat room.
  - "exit_chat_room": exit one or all chat rooms.
  - "create_chat_room": create chat room.
  - "send_message": send message to user in chat room.

We have an import design decision here: All requests from view go to WebSocketController via websocket send interface. ChatAppController handles backend actions and sends response back to view as a message, which allows us to use Session as the single identifier from the view.

4 Model

4.1 Command

We have one interface for the commands: ISndMsgCmd. This command will be executed by the user and chat room. It has one function: execute(Object context), which executes the command by the receiver (context). We will have UserSendMsgCmd etc implement this interface. ChatAppController uses command to update chat rooms.

4.2 Objects

We have two concrete objects: ChatRoom and User. They are all observer/observables.

- ChatRoom represents a chat room in the ChatApp. It observes the ChatAppController and is observed by the users in the chat room. It has five fields and twelve methods:

- User represents a ChatApp user that can exist across sessions. The User observes each chat room it is in. It has five fields and twelve methods:

- Composite User and composite ChatRoom inherit User and ChatRoom respectively. We use composite User for sending messages to multiple users and composite ChatRoom for user to exit all chat rooms.