ChatApp Design API Specification

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

We used the command, composite, and singleton design patterns for this Chat App. We have two controllers, the ChatAppController and the WebSocketController to handle the requests and communications. We also have two top-level classes, the ChatRoom and the User. The ChatAppController is an observable, observed by ChatRooms, and the ChatRoom is also an observable, observed by Users. All the objects use commands to act in the chat app.

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

1 Use Cases

- Enter ChatApp
  - Map session to new or existing user by username
- Create ChatRoom
  - Send request message with chat room restrictions
  - Create chat room with user as owner
  - Add chat room as observer of ChatAppController for users to determine relevant rooms
- Join ChatRoom
  - User can only see chat rooms that it can join
  - Add user as observer of each room it joins so chat room know who is in room
- Leave ChatRoom
  - Exit one or all chat rooms
  - Send message about why user left
  - Update owner if owner leaves

- Send message
  - Regular users send message to one user in chat room
  - Owner can send message to all users in chat room
  - Notify message has been received
  - Remove user if message contains 'hate'
  - Display messages appropriately for appropriate users
- Check ChatRooms User is in/can join, Users in ChatRoom

2 View

We have several sections in our ChatApp - a login, a chat room list, the chat room, and a create chat room form. Each interaction in the view makes a request to the model by having the websocket send a request message via the WebSocketController.

3 Controller

We have two controllers: ChatAppController and WebSocketController.

- ChatAppController is an observable observed by chat rooms. It is also a singleton, so that the notifyObservers method can be accessed from WebSocketController. WebSocketController will call the ChatAppController instance to interact with the model.
  - sessionUsernameHashmap: a ConcurrentHashMap maps the session to the username.
  - usernameUserHashmap: a ConcurrentHashMap maps the username to the user object.
  - chatAppController: a singleton 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 is not created or retrieves an existing user. Updates hashmaps accordingly.
  - getEligibleChatRooms(Session user, String request): get all chatrooms user is in or can join. Send the lists 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 of the end points by parsing messages from the view. It then calls the appropriate methods in ChatAppController to interact with the model. 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 important design decision here - all requests from the view go to the WebSocketController via the websocket send interface. The WebSocketController calls the appropriate methods in ChatAppController, which handles backend actions and sends a response back to the view as a message, allowing 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 Users and ChatRooms. It has one function: execute(Object context), which has the receiver (context) execute the command. All interactions between observables and their observers will be handled by the command design pattern.

4.2 Objects

We have two concrete objects: ChatRoom and User. ChatRoom is an observable and an observer that observes the ChatAppController. User is an observer that observes each ChatRoom it is in. CompositeUser and CompositeChatRoom extend each class to handle sending message to all users or leaving all chat rooms.

5 Design

We include two diagrams to show how the view and model communicate using the WebSocketController and ChatAppController. This flow is the primary design decision for the API.