

ChatApp API Specification

Team Houston

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

We used the command design pattern for this Chat App. We have two controllers, the ChatAppController and the WebSocketController, and a DispatchAdapter to handle the requests and communications. We also have three top-level classes, ChatRoom, User, and Message. All the objects use commands to act in the chat app. We also have an abstract class AResponse to pass information to send information back to the view.

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

- Login
 - Create user and map session to user
- Create ChatRoom
 - Send request message with chat room restrictions
 - Create chat room with user as owner
 - Add chat room to the room lists
- Join ChatRoom
 - User can only see chat rooms that it can join
 - Add user to the user lists of the room
- Leave ChatRoom
 - Exit one or all chat rooms
 - Send message about why user left
 - Destroy chat room 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

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. These requests are parsed by the model to handle the appropriate changes and send a response. We also regularly send a ping request to the server to keep the session from going idle.

3 DispatchAdapter and Controller

We have one DispatchAdapter and two controllers: ChatAppController and WebSocketController.

- DispatchAdapter has the following fields and methods:
 - int nextUserId: the id of new user.
 - int nextRoomId: the id of new room.
 - int nextMessageId: the id of new message.
 - Map<Integer, User>users: maps user id to the user.
 - Map<Integer, ChatRoom>rooms: maps room id to the chat room.
 - Map<Integer, Message>messages: maps message id to the message.
 - Map<Session, Integer>userIdFromSession: maps session to user id.
 - newSession(Session session): Create a new session and enlarge the id of user by 1.
 - getUserIdFromSession(Session session): Return the id of user from corresponding session.
 - loadUser(Session session, String body): Load a user into the environment and return that loaded user object.
 - loadRoom(Session session, String body): Load a chat room into the environment and return that loaded chat room object.
 - unloadUser(int userId): Remove a user with given userId from the environment.
 - unloadRoom(int roomId): Remove a room with given roomId from the environment.

- joinRoom(Session session, String body): Add a user to chat room specified by room id in body.
 - leaveRoom(Session session, String body): Remove a user from chat room specified by room id in body.
 - sendMessage(Session session, String body): A sender sends a string message to a receiver with room id, receiver id and message information in the body.
 - ackMessage(Session session, String body): Acknowledge the message from the receiver specified by message id in the body .
 - notifyClient(User user, AResponse response): Notify the client for refreshing.
 - notifyClient(Session session, AResponse response): Notify session about the message.
 - query(Session session, String body): Send query result from controller to front end, based on query request information in body.
 - getUsers(int roomId): Return the names of all chat room members.
 - getNotifications(int roomId): Return notifications in the chat room.
 - getChatHistory(int roomId, int userAId, int userBId): Return chat history between user A and user B (commutative).
- ChatAppController is the main controller.
 - main(String[] args): ChatApp entry point.
 - getHerokuAssignedPort(): Get the heroku assigned port number.
 - WebSocketController creates a web socket for the server.
 - onConnect(Session User): Defines the action when a new user session is connected.
 - onClose(Session user, int statusCode, String reason): Defines the action when a user session is closed.
 - onMessage(Session user, String message): Defines the action when a user session sends messages to the server.

4 Model

4.1 Command

We have one interface for the commands: IUserCmd. This command will be executed by the Users, which are observers of the DispatchAdapter 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. We have these concrete commands:

- addRoomCmd: is used when a chatroom is created by a user.
- joinRoomCmd: is used when a user wants to join a chatroom.
- leaveRoomCmd: is used when a user wants to leave a chatroom.
- removeRoomCmd: is used when a chat room is deleted by its owner.

4.2 Objects

We have three concrete objects: ChatRoom, User, and Message.

- ChatRoom: initializes a new chat room with required attributes: age, owner, qualification. It also has methods to add/remove users and provide interface to owner to control the chat room. It is an observable observed by users in the chat room.
- User: initializes a new user with required attributes: session, name, age, joined chat room. It also has methods to chat with other users or create/join/leave chat room. It is an observer to the DispatchAdapter and the ChatRooms that the user is in.
- Message: initializes a new message with required attributes like: sender's id, receiver's id, message content, etc. It also has methods to get/set these info.

4.3 Response

Response has one abstract class AResponse. Responses are used to send information about model changes back to the view. It has one method: public String toJson() which converts the object to json string. It has these concrete classes:

- NewRoomResponse: covers the information that a chat room is created by a user.
- NewUserResponse: covers the information that a user is created by a user.
- NullResponse: covers no information, which is ba default message.
- RoomNotificationResponse: covers the information when some notification need to be broadcasted in the chatroom.
- RoomUsersResponse: covers the information of all users in the chatroom.
- UserChatHistoryResponse: covers the information of all history messages in the chat room.
- UserRoomResponse: covers the information of all chat rooms of one user.

5 Other Design Decisions

We also list here a summary of some of the design decisions for the chat app.

- Show latest notification regardless of room currently open
- Received messages are shown in green
- Only show eligible rooms
- Can't modify room restriction, no empty restrictions
- Delete room when owner leaves