ChatApp Use Cases + API Documentation:

Team: Ghostbusterss
Kexi Dang, Samantha Gilmore, Siri Manjunath, Ce Liu, Po-Kai Chang, Serena Yang

Summary

The table below displays all use cases regarding chat room usage, creation, and interaction between users. Full API documentation showing the implementation of these use cases can be found at the following link: http://chatapp-ghostbusters-documentation.surge.sh/**.** Our application has been hosted on Heroku at the following link: https://chatapp-team-ghostbusters.herokuapp.com/**.**

Design Decisions:

1. We use the following design patterns in our implementation: observer, command, union, singleton.
2. When a user leaves the web page or the websocket times out, the user is deleted and chatroom information is not maintained.
3. If a user is on the homepage and a user from a chatroom they belong to starts an individual chat, the chat will show on the home page.
4. There is no option for group messages.
5. Any messages broadcasted by the owner will appear under the “Announcements” heading in the chatroom.
6. If a user leaves the room for any reason, a system message will appear under the “Announcements” heading in the chatroom with the specific reason.
7. If the owner of a room leaves the room, a new user will be randomly chosen to be the owner of the room.
8. Why we decided not to use j2html
    a. It’s a good way to create HTML for a complex login solution which has many different forms but very little actual HTML per page. We are not in this situation.
    b. It will combine java logic with HTML and CSS, which is not good to maintain code and find problems, especially in team development.
    c. Backend developers may not be familiar with HTML and CSS. If it is implemented by front-end developers, there’s a good chance there will be conflicts because they need to edit the same file.
    d. j2html is just a personal project and not so popular in the community (only 514 stars for this Github repo).

Backend Design Decisions

9. For the Dispatch Adapter class, the class/singleton instance for handling requests is given by both controllers. For the WebSocket request, the method handleMsg is used to handle them. There is a PropertyChangeSupport, in which all the users listen to the “user” property, and all the members in a chatroom listen to the chatroom’s roomID property. To send a message to all members in a chatroom, only file an event with a property of that roomID. To send a message to a user, file an event with the “user” property.

10. The ResponseAdapter interface defines a uniform way to get a JSON representation to be sent as a response. All classes under the response package implement this interface.
11. The AbstractCmd abstract class is for executing a cmd on a user. Usually, a concrete cmd will be sent in the pcs and be executed by a user.
12. The user class implements the PropertyChangeListener interface.

Frontend Design Decisions

13. In this project, we used websockets to communicate changes between the frontend and backend.
    a. Since the websocket cannot keep the connection when the application loads a new page, we will implement the chat app as a single page application (SPA). Without any framework, we implement it based on the hash change.
    b. There are two details. First, the change of hash will not refresh the page, which is a suitable way to make a SPA. Second, hashchange event will be fired if we change the hash of the url. According to the current url, we can render different pages or templates, and also insert the corresponding javascript file dynamically.
14. We have two global variables ‘websocket’ and ‘username’ that can be accessed by all the pages, since they are used in almost every page.
    a. For the websocket, it should always keep a connection with the backend when a user logs in to the app. The app will go to its index (login) page if users refresh the page because the websocket becomes disconnected, and the user will have to log in again.
15. Furthermore, we always need to distinguish between different users and different rooms. In our design, users are distinguished by the username and rooms are distinguished by the room id.

ChatApp Use Cases

ID	Description	Endpoint	WebSocket Message Type	Trigger Event
Joining / Leaving a Chat Room				
AU-01	User is brought to registration page upon load	N/A	OnWebSocketConnect method	User loads web page
AU-02	User fills in their Username, Age, School, Location and clicks “Register” button	/register	N/A	User completes registration form and clicks “Register”
AU-03	New Page loads with information about “Your Rooms”, “Rooms You Can Join”, “Rooms You Own”, “Your Profile” and “Create	/joined_rooms/:username /possible_rooms/:username	N/A	User clicks “Register” and is brought to home page

	Room” button			
AU-04	User clicks “Your Profile” button and pop-up is displayed with user profile information, including age, school, and location.	N/A	N/A	User clicks “Your Profile” button on the home page.
AU-05	User chooses to create a new room. Form pop-up is displayed for room details.	N/A	N/A	User clicks “Create New Room”
AU-06	User enters room name and selects restrictions, including age, location, and school and clicks “Create” button	/create_room	N/A	User completes form and clicks “Create” button
AU-07	New chatroom page loads displaying the users in the room, “Leave Room” button	GET /member_list/:roomID	enter_room	User has created a new room
AU-08	User is removed from chatroom automatically if the word ‘hate’ is sent in a message. Message displayed in dashboard saying “User X has been removed due to hate speech”	N/A	remove	User sends a message involving ‘hate’
AU-09	User is removed from chatroom by the Owner. Message displayed in dashboard saying “User X has been removed by the owner”	GET /member_list/:roomID	remove	Owner clicks “Remove” button next to a listed user
AU-10	User joins a room they did not belong to	/possible_rooms/:username /joined_rooms/:username /member_list/:roomID	enter_room	User clicks “Join” button next to a listed room under the “Rooms you can join” heading
AU-11	User enters a room they belong to	N/A	N/A	User clicks “Enter” button next to a listed

				room under the “Your rooms” heading
AU-12	User leaves chatroom voluntarily. Message displayed in dashboard saying “User X has left voluntarily.” User’s list of possible rooms and joined rooms is updated accordingly.	/possible_rooms/:username /joined_rooms/:username /member_list/:roomID	leave_room	User clicks “Leave Room” button
AU-13	User is removed from chatroom automatically if the websocket connection is closed. Message displayed in dashboard saying “User X has left due to closed connection.” the other user attempts to send a message to the user that left. User is removed and room information is deleted.	/member_list/:roomID	OnWebSocketClose method	User times out or closes webpage
AU-14	User is removed from all chatrooms. No rooms are listed under “Your rooms” heading.	/possible_rooms/:username /joined_rooms/:username		User clicks “Leave All Rooms” on the home page
Individual User Chats				
U-01	User 1 is taken to the Chat Room	/member_list/:roomID	enter_room	User 1 has clicked the “Enter” or “Join” buttons
U-02	User 1 opens chat with User 2. Chat box opens for User 1.	N/A	chat	User 1 clicks “Chat” button next to User 2 under “People in this room” heading
U-03	User 1 sends a message to User 2. Chat box opens for	N/A	chat	User 1 types a message and

	User 2. Message displays for both User 1 and User 2			clicks “Send”
U-04	User 1 can see the sent message has been delivered.	N/A	chat	User 1 has clicked “Send”
U-05	User 1 leaves the Chat Room. Message displayed in Dashboard that User 1 left voluntarily.	/possible_rooms/:username /joined_rooms/:username /member_list/:roomID	leave_room	User 1 clicks “Leave Room” button
U-06	User 2 attempts to send a message to User 1 after User 1 has left the chatroom. A message is displayed in the chat box that “User 1 has left the chatroom”	N/A	chat	User 2 already has a chat open with User 1 and User 1 leaves the chatroom.
U-07	User 1 receives message from Owner		announcement	Owner types their message and clicks “Announcement” button
U-08	User 2 sees “User 1 Has Left” message in “Announcements” section. User 1 is no longer shown in the list of people in this room.	/member_list/:roomID	leave_room announcement	User 1 clicks “Leave This Room” button
Room Owner				
O-01	Owner sends a message to all the users in their chat room. Message displays in chatroom dashboard.		announcement	Owner types their message and clicks “Announcement” button
O-02	Owner removes user from chatroom. Message displays in chatroom dashboard.		remove announcement	Owner clicks “Remove” button next to user listed in chatroom.

O-03	Owner leaves the chatroom. If there is still at least one member in that chat room, a user is chosen randomly to become the new owner. Message displays under “Announcements” heading as follows, “User X is the new owner”			Owner clicks “Leave Room” button when there is at least one other user in the room.
O-04	Owner leaves the chatroom when there are no other users present. The chatroom is deleted in this case.			Owner clicks “Leave Room” button when there are no other users in the room.