HW8 – Chat App API Design

Chat App API Specification Document

Team Tofu

Xiongfeng Song (xs16), Hao Lin (hl76), Neil Chen (ndc1)
Ian Fan (yf18), Siyang Zhang (sz55), Zhaokang Pan (zp7)

November 17th, 2019

1. Summary

This project involves the implementation of a chat application (chatapp) using the design principles conveyed by COMP504: Graduate Object-Oriented Programming and Design. This chatapp allows for 1-to-1 messaging between users that are members of chat rooms. Some (but not all) of the features are:

1. Users have profiles (age, location, school)
2. Users are ephemeral (their information is deleted once they close their session)
3. A user can create a chatroom with profile restrictions
4. A user can join a chat room if they meet profile restrictions as set by the owner of a chat room
5. A user can view a list of chat rooms that they have joined and a list of chat rooms that they meet the profile restrictions for
6. A user can view the list of other users in the chat room
7. A user may be present in multiple rooms
8. Rooms may have multiple users
9. A user can choose to leave one or all of their chat rooms
10. A user will be forcibly removed from a chat room if they send a message with the word "hate"
11. A user can send a message to any other user in a chat room that they are both in
12. An owner of a room can send a message to all other users in their chat room
13. When a user sends a message, they will be notified when their message is received
14. When a user leaves a room, it is indicated to other users in the same chatroom why they left the room (voluntarily left, connection closed, forced to leave)

This project's version of the chatapp is hosted on Heroku as chatapp-team-tofu.

2. Use Cases

The use cases for the chatapp can be grouped into three main categories:

1. Creation of users and chatrooms
2. Selection of chatrooms by users
3. Message passing within chatrooms

2.1 Use Cases: Creation of Users and Chatrooms

1. User creation
   When a new session is opened, a user will be asked to create a profile with the following information:
   - Username (a non-empty string with letters and numbers)
   - Age (integer between 18 and 150)
   - Location: A continent
     - Asia, Africa, Europe, North America, South America, Oceania, Antarctica

  - School

2. User deletion
   If a user closes their session, then their information is removed from the chatapp. They will be removed from all of their currently joined chatrooms. If they re-open a session, then they will have to create a new profile, which will be unconnected to their previous profile.

3. Chatroom creation
   A user can create up to a maximum number of rooms. They will be the owner of these rooms.
   An owner must set the following information for each chatroom:
   - Room name (a non-empty string with letters and numbers)
   - A list of restrictions, if desired:
     - A range of ages permitted to enter the chatroom
     - A list of locations permitted to enter the chatroom (continents)
     - A list of schools permitted to enter the chatroom

4. Chatroom deletion
   An owner can delete a chatroom. This will remove all users from the chatroom, including the user.
   If an owner's session is closed, then all of the chatrooms for which they are owners will also be deleted.

2.2 Selection of Chatrooms by Users

1. Users viewing chatrooms
   A user can view a list of joined chatrooms they are currently a member of. A user can also view a list of joinable chatrooms for which they are not a member of, but that they meet the profile restrictions for.

2. Users joining a chatroom
   A user can choose to join a chatroom for which they are not a member of, but that they meet the profile restrictions for. When they join the new chatroom, then this chatroom will be removed from the list of joinable rooms, and added to the list of joined rooms for this user.

3. Users leaving a chatroom
   A user can choose to leave a chatroom that they are a member of. When they leave the chatroom, then this chatroom will be removed from the list of joined rooms, and will be added to the list of joinable rooms.

2.2 Message Passing Within Chatrooms

1. Users sending private messages
   A user can select a joined chatroom to view all other users in. A user can then select one

another user to be the current receiver. A user will then see all messages that have passed between them and the receiver during the current session. A user can then send a message to the receiver. When the receiver receives the message, the user will get a confirmation that their message has been received by the receiver.

2. Owner broadcasting
   An owner can select a joined chatroom. If they are the owner of the selected chatroom, then they can view all messages that have been broadcasted to this chatroom during the current session. An owner can then send a message to all users within the chatroom except themselves. When all receivers have received the broadcast, the owner will get a confirmation that their message has been received by all receivers.

3. Auto-ban
   If any user sends a message containing the string "hate", then they will be removed from all of their current chatrooms.

3. API Design Decisions

The game design follows the Model-View-Controller software design pattern. The model manages the chatrooms, users, and messages. The view displays the chatapp for the user. The controller accepts user input and passes it to the model.

A UML diagram of our model is provided at the end of this document in section 5.

The Model, View, and Controller are discussed in more detail in sections 3.1, 3.2, and 3.3, respectively. Details for the interfaces are also provided in section 3.1.

3.1 Model

The model consists of a ChatRoom class, a Message class, a User class, and a DispatchAdapter, which manages the interactions between different instances of these classes.

In our API design, we also have an IChatCmd, an interface which allows us to use commands, and several concrete commands which implement this interface (discussed later).

<u>ChatRoom</u>

We have a class that represents a chatroom. It contains fields for its own name and owner, as well as fields to store restrictions on the age, location, and school of its allowed users, as well as setters and getters for these fields.

We also have a ChatRoomBuilder which allows us to create ChatRooms; this follows the factory design pattern. It has fields similar to those of ChatRoom but use them to construct ChatRoom instances.

<u>Message</u>

We have a class that represents a message. It contains fields for the actual text within the message, a timestamp, and targeting information (a receiver and the name of the chatroom in which it is sent). It also contains getters and setters for these fields.

User

We have a class that implements PropertyChangeListener, and it represents a user. It contains fields for the user's name, age, location, and school, as well as the user's session. It also contains a list of the user's joined groups, and another list of the user's joinable groups. It also contains getters and setters for these methods.

DispatchAdapter

Our dispatch adapter communicates with the model and view. It contains methods to manage the various chatrooms, messages, and users within the model. It contains a map from sessions to users, and another map from usernames to sessions. It also contains a set of all chatroom names.

In particular, the dispatch adapter contains methods to initialize and closing of sessions; update profiles of users; manage the creation, joining, and leaving of chatrooms; and manage the sending or broadcasting of messages.

IChatCmd

We have an interface called IChatCmd which only consists of the execute method, which takes a PropertyChangeListener as its only parameter. IChatCmd is an interface that allows us to send commands within our model, currently allowing us to implement commands to join chat rooms, leave chat rooms, and send messages within chat rooms. These are implemented as JoinGroupCmd, LeaveGroupCmd, and SendMessageCmd, respectively.

3.2 View

The view displays the various elements of the chatapp, including:

- A list of rooms, with some rooms marked as **joinable** and others marked as **joined**, with options for a user to join the joinable rooms. Rooms are only displayed if the user's profile meets the requirements set by the room owner.
- The ability to expand a room's display to show a list of current users within that room.
- The ability to select another current user within a joined room, and show all message history sent between the two users during the current session. Messages change how they are displayed when they have been received by the receiving user. When users leave a chatroom (voluntarily, via session close, or via banning) this will also be broadcasted to all users within their joined rooms as a message.
- Buttons to leave chatrooms.
- Menus for chatroom owners to broadcast messages to all users.
- Menus to create profiles and new chatrooms.

The view supports user interactions with all of these displayed elements.

3.3 Controllers

The chatapp controller processes GET and POST requests and returns JSON representations of information returned by the Dispatch Adapter. It communicates with all of the clients on the web socket. The chatapp controller supports the following endpoints:

- /register is a POST request that allows users to join the chatapp.
- /createRoom is a POST request that allows users to create chat rooms.
- /join is a POST request that allows users to join chat rooms.
- /leave is a POST request that allows users to leave chat rooms.

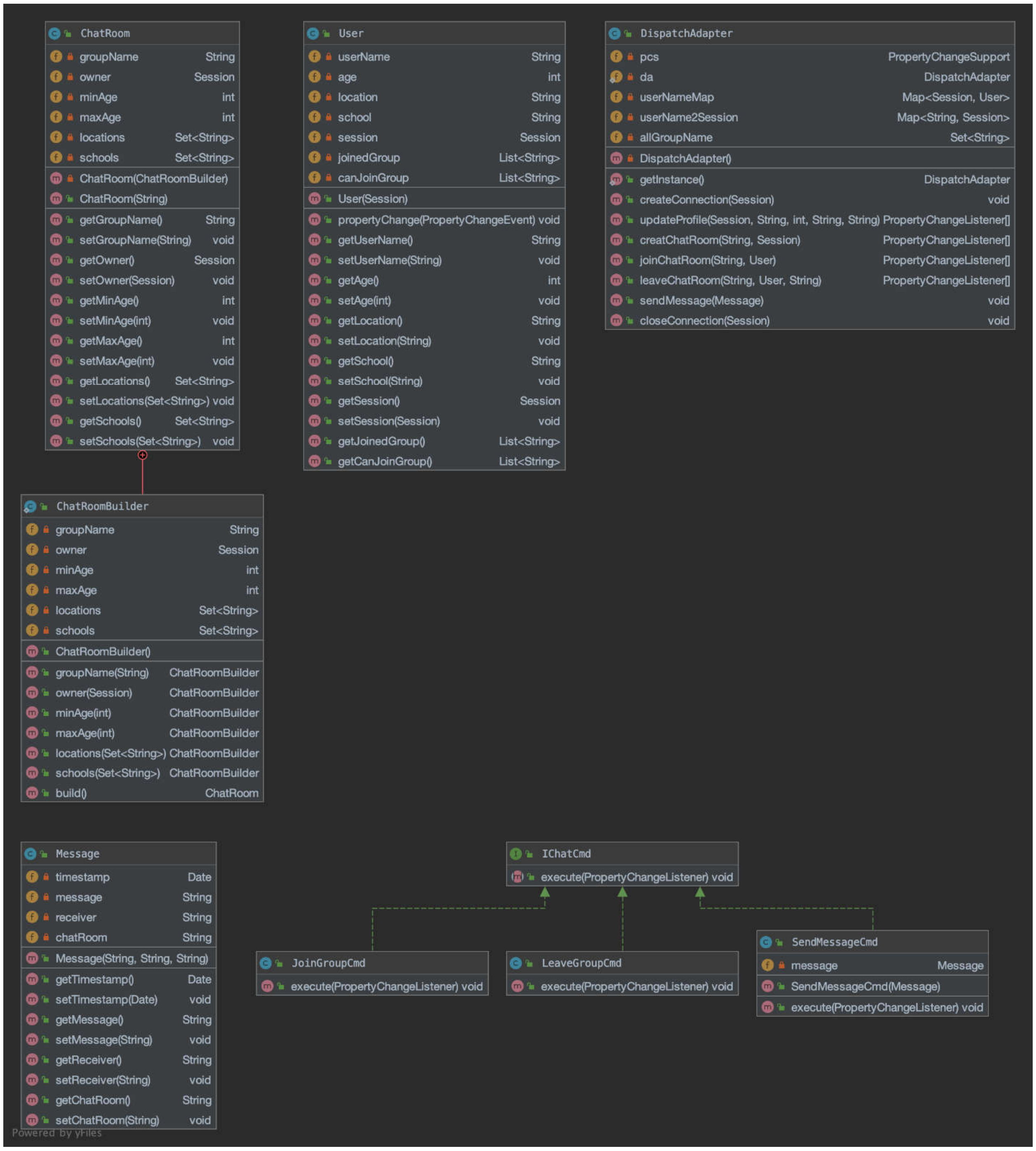
The chatapp controller also has methods to connect to Heroku and interact with the web socket controller.

The web socket controller creates a web socket for the server, can open and close user sessions, and enables message sending.

4. Additional Information

The chatapp is hosted on Heroku. Because this project is still in the API design phase, there is not a sufficient number of unit tests to ensure 85% code coverage for all lines of code in the model.

5. UML Diagram



6. Project Team Members

Six people contributed to this project:

xs16	Xiongfeng Song	Project Lead
hl76	Hao Lin	Tech Lead
ndc1	Neil Chen	Doc Lead
yf18	Ian Fan	Developer
sz55	Siyang Zhang	Developer
zp7	Zhaokang Pan	Developer

The guidelines for this project were provided by course COMP504: Graduate Object-Oriented Programming and Design, taught during the Fall 2019 semester by Mack Joyner at Rice University.