Chatroom API Design Document

Team J

For this chat room project, our team used the MVC programming paradigm to structure our code. We also used command design pattern, union design pattern and singleton design pattern to support the logic that flows within this chat room.

1. Controller

The **ChatAppController** acts as a coordinator between the view and the model and handles the users' requests. It is responsible for the following jobs:

1. Communicate with all the clients on the web socket.
2. Start the web socket for the server.
3. Specify the endpoint for the web socket.
4. Connect to a defined web socket on the server side.
5. Initialize to begin listening for messages.

1.1 ChatAppController:

```
public void handleRegisterUser(Session session, UserParse pUserParse)
```

- Handler for the client request to create a user. The parameter `session` is the current session, `pUserParse` is the user's parsed data.

```
public static void handleCheckUser(Session session, String username)
```

- Handler for the client request to check whether a user is valid. The parameter `session` is the current session, `username` is the username to be checked.

```
public static void handleClose(Session session)
```

- Handler for closing a session. The parameter `session` is the current session.

```
public static void handleSendMsg(Session session, SendMsgRequest smr)
```

- Handler for the client's request to send a message. The parameter `session` is the current session, `smr` is the request to send a message.

```
public static void handleCreateRoomRequest(Session session, RoomParse newRoom)
```

- Handler for the client request to create a chat room. The parameter `session` is the current session, `newRoom` is the parsed new room's data.

```
public static void handleJoinRoomRequest(Session session, String roomname)
```

- Handler for the client request to join a room. The parameter `session` is the current session, and `roomname` is the name of the room to join.

```
public static void handleExitRoomRequest(Session session, String roomname, String reason)
```

- Handler for the client request to leave a room. The parameter `session` is the current session, and `roomname` is the name of the room, and `reason` is the reason for leaving the room.

```
public static void handleEditRoomRequest(Session session, UpdateRoomRequest request)
```

- Handler for the client request to adit a room. The parameter `session` is the current session, `request` is the update room request.

```
public static void handleHeartbeatRequest(Session session, long requestTime)
```

- Handler for the heartbeat request. The parameter `session` is the current session, and `requestTime` is the request time.

```
private static int getHerokuAssignedPort()
```

- Get the Heroku assigned port number.

```
void main (java.lang.String[] args)
```

- Chat App entry point.

1.2 WebSocketController:

The **WebSocketController** is used to define the web socket on the server side.

```
void onConnect (Session user)
```

- Called when navigating to the app. The parameter is the user.

```
void onClose (Session user, int statusCode, String reason)
```

- Called when closing the web socket. The parameter `user` is the user, `statusCode` is the status code, `reason` is the reason for leaving.

```
void onMessage (Session user, String message)
```

- Called when sent a message from a client instance. The parameter `user` is the user, `message` is the message to send.

2.Model

The model contains most of the logic in the chatroom. There are four packages inside: message, response, room, and user.

2.1 Message

This class is an abstract class, which extracts all the features of the message that happened in the chat room, like system message and chat message.

```
public Message(String from, String to, boolean toAll, String text, boolean isSysMsg, long timestamp)
```

- The constructor of `Message` class. The parameter `from` is the message sender, `to` is the message receiver, `toAll` indicates whether this message is sent to all users, `text` in the text content, `isSysMsg` indicates whether this message is a system message, and `timestamp` is the timestamp of the message.

```
public Message(String data)
```

- The constructor of `Message` class. `data` is the data to be parsed.

```
public String getFrom()
```

- Get the sender of the message.

```
public String getTo()
```

- Get the receiver of the message.

```
public String getText()
```

- Get the content of the message.

```
public String getToAll()
```

- Get if the message is a toAll message.

```
public String getIsSysMsg()
```

- Get if the message is a system message.

```
public String getTimestamp()
```

- Get the timestamp of the message.

2.2 User

This class represents the users in the chatroom and contains all the information regarding users.

```
public User(Session session, String id, int age, String area, String school)
```

- The six parameters specify the properties of the user upon initialization. `session` is the current session. `id` is the user's ID. `age`, `area`, and `school` are the user's personal information.

```
public String getId()
```

- Get User's Id.

```
public int getAge()
```

- Get age.

```
public String getArea()
```

- Get area.

```
public String getSchool()
```

- Get school.

```
public Session getSession()
```

- Get current session.

```
public void setSession(Session s)
```

- Set the session. The parameter `s` is the session to be set.

```
public void sendToSession(String s)
```

- Send the String message to the session. `s` is the message to be sent.

2.3 Room

This class represents the rooms in the chatroom and contains all the information regarding rooms.

```
public Room(int ageMin, int ageMax, HashSet<String> areas, HashSet<String> schools, HashSet<User> users, HashSet<M
essage> msg, User owner, String roomId)
```

- The seven parameters specify the `ageMin`, `ageMax`, `areas`, `schools`, `users`, `msg`, and `owner` of the users. These parameters specify the constraint properties of the users upon initialization. `ageMin` is the minimum age constraint to join the room. `ageMax` is the maximum age constraint to join the room. `areas` specifies the areas in the chatroom. `schools` specifies the schools in the chatroom. `users` specifies the users in the chatroom. `msg` is History messages in the chat room. `owner` is the owner of the chat room. `roomId` is the ID of the room.

```
public boolean checkAge(int age)
```

- check whether the age is valid. The parameter `age` is the age to be checked.

```
public boolean checkArea(String area)
```

- check whether the area is valid. The parameter `area` is the area to be checked.

```
public boolean checkSchool(String school)
```

- check whether the school is valid. The parameter `school` is the area to be checked.

```
public boolean check(User user)
```

- check the user's age, area, and school together. `user` is the user to be checked.

```
public int getAgeMin()
```

- get min age constraint.

```
public int getAgeMax()
```

- get max age constraint.

```
public HashSet<String> getAreas()
```

- return valid areas.

```
public HashSet<String> getSchools()
```

- return valid schools.

```
public HashSet<String> getUsers()
```

- return users in the chat room.

```
public HashSet<String> getMessages()
```

- return messages in the chat room.

```
public void addUser(String u)
```

- add a new user in the chat room. `u` is the user to be added.

```
public void removeUser(String userId)
```

- remove a user in the chat room. `userId` specifies the user to remove.

```
public String getRoomId()
```

- Get the room ID.

```
public User getOwner()
```

- Get the room's owner.

```
public void setOwner(User owner)
```

- Set the room's owner. `owner` is the owner to be set.

```
public boolean isEmpty()
```

- Get if the room is empty(has no users).

```
public User getFirstUser()
```

- Get the first user.

```
public HashSet<String> getAllUserIds()
```

- Get all the users' IDs.

```
public String[] getAllUserIdList()
```

- Get all users' IDs as an array of strings.

```
public void addMessage(Message msg)
```

- Add a message. `msg` is the message to be added.

```
public HashSet<Message> getMessageForUser(String userId)
```

- Get messages for a specific user. `userId` is the user's ID.

```
public void setAreas(HashSet<String> areas)
```

- Set the areas passed as a HashSet of strings. `area` is a HashSet of areas.

```
public void setSchools(HashSet<String> schools)
```

- Set the schools passed as a HashSet of strings. `schools` is a HashSet of schools.

```
public void setAreas(String[] areas)
```

- Set the areas passed as an array of strings. `area` is an array of areas.

```
public void setSchools(String[] schools)
```

- Set the schools passed as an array of strings. `schools` is an array of schools.

```
public void setAgeMin(int minAge)
```

- Set minimum age constraint. `minAge` is the minimum age.

```
public void setAgeMax(int minAge)
```

- Set maximum age constraint. `maxAge` is the maximum age.

```
public String[] getAreaName()
```

- Get area names.

```
public String[] getSchoolName()
```

- Get school names.

3. Cmd

We used the command design pattern in this application. Operations such as creating/exiting a room, sending messages, are all processed here.

3.1 AbstractCmd

The abstract command class for executing some logic.

```
public abstract void execute(User user)
```

- The userModel to be executed on. `user` execute this cmd on this userModel.

3.2 CloseCmd

The concrete command class for closing a session.

```
public CloseCmd(AppContext app)
```

- The constructor. The parameter `app` is the app context.

3.3 CreateRoomCmd

The concrete command class for creating a chatroom.

```
public CreateRoomCmd(AppContext app, RoomParse newRoom)
```

- The constructor. The parameter `app` is the app context, and `newRoom` is the room to be created.

3.4 EditRoomCmd

The concrete command class for editing a chatroom.

```
public EditRoomCmd(AppContext app, UpdateRoomRequest request)
```

- The constructor. The parameter `app` is the app context, and `request` is the unique room request.

3.5 ExitRoomCmd

The concrete command class for exiting a chatroom.

```
public ExitRoomCmd(AppContext app, String roomname, String reason)
```

- The constructor. The parameter `app` is the app context, and `roomname` is the room to exit, and `reason` is the reason for leaving.

3.6 JoinRoomCmd

The concrete command class for joining a chatroom.

```
public JoinRoomCmd(AppContext app, String roomname)
```

- The constructor. The parameter `app` is the app context, and `roomname` is the room to join.

3.7 CreateRoomCmd

The concrete command class for sending messages.

```
public SendMsgCmd(AppContext app, SendMsgRequest smr)
```

- The constructor. The parameter `app` is the app context, and `smr` is the send message request.

4. Parse

The parse package is used to parse different requests. Then, the parsed requests can be interpreted and to be applied to the chatroom.

4.1 CheckUserRequest

This is used to check if the user has already registered.

```
public CheckUserRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getUsername()
```

- Return the username.

4.2 CreateRoomRequest

This is used to parse the request to create a room.

```
public CreateRoomRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getRoom()
```

- Return the room info.

4.3 ExitRoomRequest

This is used to parse the request to exit a room.

```
public ExitRoomRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getRoomname()
```

- Return the room name.

4.4 HeartBeatRequest

This is used to parse the request to handle timestamps.

```
public HeartBeatRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getTimestamp()
```

- Return the time stamp.

4.5 JoinRoomRequest

This is used to parse the request to join a room.

```
public JoinRoomRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getRoomname()
```

- Return the room name.

4.6 RegisterUserRequest

This is used to parse the request to register a new user.

```
public RegisterUserRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public UserParse getUserParse()
```

- Return the user.

4.7 RoomParse

This is used to parse data related to the chatroom.

```
public public RoomParse(String owner, String roomname, int ageMin, int ageMax, HashSet<String> areas, HashSet<Stri
ng> schools, HashSet<String> users, HashSet<Message> msgs)
```

- The Constructor. `owner` is the room owner. `roomname` is the chatroom's name. `ageMin` is the minimum age constraint of the room. `ageMax` is the maximum age constraint of the room. `areas` is the area constraint of the room. `schools` is the school constraint of the room. `users` is a HashSet of users. `msgs` is a HashSet of messages.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getMsgs()
```

- Get the messages in the room.

```
public String getMsgsHash()
```

- Get the messages HashSet in the room.

```
public String getAreas()
```

- Get the areas of the chatroom.

```
public String getSchools()
```

- Get the schools in the chatroom.

```
public String getUsers()
```

- Get the users HashSet in the chatroom.

```
public String getAgeMax()
```

- Get the max-age constraint in the chatroom.

```
public String getAgeMin()
```

- Get the minimum age constraint in the chatroom.

```
public String getOwner()
```

- Get the owner of the chatroom.

```
public String getRoomname()
```

- Return the room name.

```
public void setMsgs(HashSet<Message> msgs)
```

- Set all the messages in the chatroom.

4.8 SendMsgRequest

This request is used to parse the request to send messages.

```
public SendMsgRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getType()
```

- Return the request type.

```
public String getRoomname()
```

- Return the room name.

```
public String getMsg()
```

- Return the messages.

4.9 UpdateRoomRequest

This request is used to parse the requests to update the chatroom.

```
 public UpdateRoomRequest(String data)
```

- The Constructor. `data` is the data to be parsed.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getAreas()
```

- Get the areas of the chatroom.

```
public String getSchools()
```

- Get the schools in the chatroom.

```
public String getAgeMax()
```

- Get the max-age constraint in the chatroom.

```
public String getAgeMin()
```

- Get the minimum age constraint in the chatroom.

```
public String getRoomname()
```

- Return the room name.

```
public String getType()
```

- Return the request type.

4.10 UserParse

This request is used to parse the user-related data.

```
 public UserParse(String userId, int age, String school, String area)
```

- The Constructor. `userId` is the user's ID. `age` is the user's age. `school` is the user's school. `area` is the user's area.

```
public void parse(String data)
```

- Parse the data passed in as a String. `data` is the data to be parsed.

```
public String getArea()
```

- Get the user's area.

```
public String getSchool()
```

- Get the user's school.

```
public String getUsername()
```

- Get the username.

```
public String getAge()
```

- Get the user's age.

5. Response

5.1 ResponseAdapter

An interface to get a json representation for response to clients.

```
public String getJsonRepresentation(Gson gson)
```

- get a JSON string to respond. `gson` is an Gson instance.

5.2 InitResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client to initialize.

```
public InitResponse(UserParse user, HashSet<RoomParse> availableRooms)
```

- `user` is the list of rooms that can be seen by the user and `availableRooms` is available rooms.

```
public String getType()
```

- Get type.

```
public UserParse getUser()
```

- Get user.

```
public HashSet<RoomParse> getRooms()
```

- Get rooms.

5.3 AddRoomResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client to add a new room.

```
public AddRoomResponse(RoomParse room)
```

- The constructor. `rooms` is the room to be added.

```
public String getType()
```

- Get type.

```
public RoomParse getRoom()
```

- Get room.

5.4 RemoveRoomResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client to remove a room.

```
public RemoveRoomResponse(String roomname)
```

- The constructor. `room` is the room to be removed.

```
public String getType()
```

- Get type.

```
public String getRoomname()
```

- Get roomname.

5.5 EditRoomResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client to edit a room.

```
public EditRoomResponse(String roomname, int ageMin, int ageMax, String[] areas, String[] schools)
```

- The constructor. `room` is the room to be removed. `ageMin` is the minimum age. `ageMax` is the maximum age. `areas` is the location. `schools` is the school.

```
public String getType()
```

- Get type.

5.6 SendMessageResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client that a message is sent to the user.

```
public SendMsgResponse(String roomname, Message msg)
```

- The constructor. `roomname` is where the message is delivered, and `msg` is the message delivered by the user.

```
public String getType()
```

- Get type.

```
public String getRoomname()
```

- Get room name.

```
public Message getMsg()
```

- Get message.

5.7 JoinRoomResponse

This is a concrete class that implements the interface **ResponseAdapter** It represents the response to tell the client that the user has joined a room.

```
public JoinRoomResponse(RoomParse room)
```

- The constructor. `room` is where the message is delivered.

```
public String getType()
```

- Get type.

```
public String getRoom()
```

- Get room.

5.8 HeartbeatReponse

This is a concrete class that implements the interface **ResponseAdapter** It represents the response of heartbeat which controls the time limit of the session.

```
public HeartbeatResponse()
```

- The constructor.

5.9 NoUserResponse

This is a concrete class that implements the interface **ResponseAdapter** It represents the response to tell the client that there is no one else in the room.

```
public NoUserResponse(String username)
```

- The constructor. `username` is where the message is delivered.

```
public String getType()
```

- Get type.

```
public String getUsername()
```

- Get username.

5.10 UpdateRoomUserResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client to update the room user information.

```
public UpdateRoomUsersResponse(String owner, String[] users, String roomname)
```

- The constructor. `roomname` is the room whose information needs to be updated. `owner` is the owner of the room. `users` is the members in the room.

```
public String getType()
```

- Get type.

```
public String getUsers()
```

- Get users.

```
public String getOwner()
```

- Get owner.

5.11 UserExistResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client that there are other users existing in the room.

```
public UserExistResponse(String username)
```

- The constructor. `username` is the users' name in the room.

5.12 UserinUseResponse

This is a concrete class that implements the interface **ResponseAdapter**. It represents the response to tell the client the users in use.

```
public UserInUseResponse(String username)
```

- The constructor. `roomname` is the users' name who are in use.

```
public String getType()
```

- Get type.

```
public String getUsername()
```

- Get username.

6. AppContext

This is a concrete class that is used to manage the context of the chat app.

```
private static AppContext only = null
```

- A public static member variable that represents a singleton variable.

```
private ArrayList<User> users
```

- A private member variable that represents the list of existing users, should remove the head when the list is too long.

```
private ArrayList<Room> roomModels
```

- A private member variable that the list of existing rooms, should remove the head when the list is too long.

```
private HashMap<String, Room> roomMap
```

- A HashMap that contains the rooms.

```
private Map<Session, User> sessionUserMap
```

- A private member variable that maps a session to User.

```
public static AppContext getOnly()
```

- A public static method to get a singleton class.

```
public ArrayList<Room> getRoomModels()
```

- Get all the roomModels.

```
public void addSessionUser(Session s, User user)
```

- A public method to add a user to the `sessionUserMap` , `s` is the session, `u` represents the user to be added.

```
 public void closeSession(Session s)
```

- Close the session. `s` is the session to be closed.

```
public User findUser(Session s)
```

- Find a user in the `sessionUserMap`

```
public ArrayList<User> getUsers()
```

- Get all the userModels.

```
public void addUser(User u)
```

- Add a user to the user list. `u` represents the user to be added.

```
public void addRoom(Room r)
```

- Add a room to the room list. `r` is the room to be added.

```
public void removeRoom(Room r)
```

- Remove a room from the room list. `r` is the room to be removed.

```
public Room getRoomModel(String roomId)
```

- get room model. `roomId` is the room's ID for look up.

```
public boolean isExistedRoom(String roomId)
```

- return whether the given room exists. `roomId` is the room to be checked.

7. Use Cases

7.1 Create A room:

- A user can initiate the creation of a chat room.
- A user must create a profile that includes age, location, and school.

- The creator of the room will become the owner of the chat room automatically.
- The owner will restrict the type of the user who can join the chat room, the restrictions are Age, Location, and School.

7.2 Join A room:

- A user can join in one or multiple chat rooms.
- A user must create a profile that includes the age, location, school.
- If the user has already created a profile, the profile will be retrieved and the user can enter the chat room directly.

7.3 Send messages:

- After entering the chat room, a user can choose to send a message to anyone else in the chatroom.
- A user will be notified if the message has been received.
- The user can not send a message with the word “hate”, otherwise, the user will be removed from all chatrooms.
- If the message is sent privately, the message is not visible to other users.
- Messages will appear on separate lines on the chat app screen.
- All the messages should be displayed in the order of timestamp.
- The chatroom owner can choose to send a message to all users.

7.4 Exit Room:

- Users can choose to exit one chat room or all chat rooms.
- If the user exits the chat room, the reason should be displayed in the chat room (“voluntarily left”, “connection closed”, “forced to leave”, or “kicked out due to sensitive message”)

8. Design Decisions

8.1 Union Design Pattern

8.1.1 Command

We created an abstract class `AbstractCmd` as the superclass, and `CloseCmd` , `CreateRoomCmd` , `ExitRoomCmd` , `EditRoomCmd` , `joinRoomCmd` , and `SendMsgCmd` as the subclass. In this case, we can show the inheritance and polymorphism relationships between “concrete commands” and “abstract command”.

8.1.2 Response

We constructed an interface `ResponseAdapter` first, and used union design pattern to show the inheritance relationship between “concrete” entities `AddRoomResponse` , `InitResponse` , `RemoveRoomResponse` , `EditRoomResponse` , `HeartBeatResponse` , `JoinRoomResponse` , `NoUserResponse` , `SendMsgResponse` , `UpdateRoomUsersResponse` , `UserExistResponse` , and `UserInUseResponse` .

8.2 Command Design Pattern

We used the Command Design Pattern to implement basic operations in the Chatapp. The operations include `CloseCmd` , `CreateRoomCmd` , `ExitRoomCmd` , `EditRoomCmd` , `joinRoomCmd` , and `SendMsgCmd` .

8.3 Singleton Design

We keep only one context for the application and it uses the singleton design pattern. It contains member functions that operate on the context.

8.4 MVC Design Pattern

We use the MVC design pattern to separate the frontend and the backend to achieve better extensibility.

8.5 User Data Record

When a new user enters the chat app, he/she will be asked personal information including name, age, area, and school. Then, the data will be saved such that this user won’t need to enter personal information next time. Instead, he/she can log in and start chatting directly.

8.6 Front End

We used React-Redux in the frontend to handle dynamic communication with back end and timely state change. The data passed between the frontend and the backend is in JSON format. To ensure a unified format, we also included `interface_guideline.ts` in the resources directory.

8.7 Back End

We deployed our back end on Amazon AWS to support our running Chatroom on Heroku.