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 strategy design pattern, union 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 Method Summary:

```
public void handleCreateRoom(Session session, int ageMin, int ageMax, ArrayList<String> areas, ArrayList<String> s
chools)
```

- Handler for the client request to create a chat room. The parameter `session` is the current session, `ageMin` is the room's minimum age constraint, `ageMax` is the room's maximum age constraint, `areas` is an array of areas, `schools` is an array of schools.

```
public void handleCreateUser(Session session, String userId, int roomId, int age, String area, String school)
```

- Handler for the client request to create a user. The parameter `session` is the current session, `userId` is the user's ID, `roomId` is the room, `age` is user's age, `area` is user's area, `school` is user's school.

```
void handleJoinRoom (Session session, int roomId)
```

- Handler for the client request to join a room. The parameter `session` is the current session, and `roomId` is the room's ID.

```
void handleLeaveAllRooms (Session session)
```

- Handler for the client request to leave all the rooms. The parameter `session` is the current session.

```
void handleLeaveRoom (Session session, int roomId)
```

- Handler for the client request to leave a room. The parameter `session` is the current session, and `roomId` is the room's ID.

```
void handleRetrieveUser (Session session, String userId)
```

- Handler for the client request to retrieve a user. The parameter `session` is the current session, and `userId` is the user's ID.

```
void handleSendMessage (Session session, int roomId, String receiverId, String text)
```

- Handler for the client's request to send a message. The parameter `session` is the current session, and `roomId` is the room's ID, `receiverId` is the receiver's ID, and `text` is the text to send.

```
void handleSendMessageToAll (Session session, String text)
```

- Handler for the client request to send message to all users in the room. The parameter `session` is the current session and `text` is the text to send.

```
void handleUpdateRoom (Session session, int roomId, int ageMin, int ageMax, ArrayList<String> areas,ArrayList<Stri
ng> schools)
```

- Handler for the client request to update a room's constraint. The parameter `session` is the current session, `roomId` is the room's ID, `ageMin` is the room's minimum age constraint, `ageMax` is the room's maximum age constraint, `areas` is an array of areas, `schools` is an array of schools.

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

- Chat App entry point.

1.2 WebSocketController Method Summary:

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

2.1.1 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.

```
 private String time ()
```

- A private member variable that represents the timestamp of the message.

```
private boolean isGlobal
```

- A private member variable that represents if the message is visible to all users.

```
public Message(String time, boolean isGlobal)
```

- The constructor of `Message` class. There are two parameters involved: `time` and `isGlobal`.

```
public String getTime()
```

- A public method that is used to get the timestamp of the message.

2.1.2 ChatMessage

This is a concrete class that extends the abstract class **Message**. It represents all the messages that are sent from one user to another.

```
private User from
```

- A private member variable that represents the sender of the message.

```
private User to
```

- A private member variable that represents the receiver of the message.

```
private String text
```

- A private member variable that represents the context of the message.

```
public ChatMessage(User from, User to, String time, String text)
public ChatMessage(User from, String time, String text)
```

- Constructor summary: the `time` and `isGlobal` is Inherited from the parent class. `from`, `to`, and `text` are members that are different from the parent class. The first constructor is for non-global messages, and the second constructor is for global messages.

```
public User getUserFrom()
```

- A public method that is used to get the sender of the message.

```
public User getUserTo()
```

-A public method that is used to get the receiver of the message.

```
public String getText()
```

- A public method that is used to get the content of the message.

2.1.3 System Message

This is a concrete class that extends the abstract class Me**ssage.** It represents all the system messages. Several events will trigger a system message, such as:

1. A new user enters the chatroom.
2. A user creates a new chat room,
3. A user leaves the chat room
4. A user is kicked out by the room owner.

```
private User user
```

- A private member variable that represents the user who is involved in the message.

```
private Event event
```

- A private member variable that represents the event, for example, leave, enter, kick out, receive a message.

```
public SystemMessage(User user, Event event, String time)
```

- Constructor summary: the `time` and `isGlobal` is Inherited from the parent class. `user` and `event` are new members that are different from the parent class.

```
public User getUser()
```

- A public method that is used to get the user involved in the message.

```
public Event getEvent()
```

- A public method that is used to get the event happened in the message.

2.2 User

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

```
public User(String id, ArrayList<Room> myRoomList, ArrayList<Room> availableRoom, int age, String area, String sch
ool)
```

- The six parameters specify the properties of the user upon initialization. `id` is the User ID. `myRoomList` is the rooms that the user is already in. `availableRoom` is the rooms available for the user to join. `age` , `area` , and `school` are the user's personal information.

```
public String getId()
```

- Get User's Id.

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

- Get myRoomList.

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

- Get availableRoom.

```
public void setAvailableRoom(ArrayList<Room> rList)
```

- Set availableRoom. The parameter `rList` is a list of rooms.

```
public int getAge()
```

- Get age

```
public String getArea()
```

- Get area.

```
public String getSchool()
```

- Get school.

2.3 response

2.3.1 IResponse

This is an interface that specifies all the responses sent from the server to the client and what the client should do.

```
void send()
```

- A public method that is used to send the message from server to client.

2.3.2 InitResponse

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

```
public InitResponse(ArrayList<Room> rooms, User receiver)
```

- `rooms` is the list of rooms that can be seen by the user and `receiver` is the receiver of this response.

```
public void send()
```

- A public method that is used to send a message to the client.

2.3.3 AddRoomResponse

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

```
public InitResponse(ArrayList<Room> rooms, User receiver)
```

- The constructor. `rooms` is the room to be added and `receiver` is the owner of the room.

```
public void send()
```

- A public method that is used to send a message to the client.

2.3.4 RemoveRoomResponse

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

```
public RemoveRoomResponse(Room room, ArrayList<User> receivers)
```

- The constructor. `room` is the room to be removed and `receivers` is the list of users who is in the chat room.

```
public void send()
```

- A public method that is used to send a message to the client.

2.3.5 RetrieveUserResponse

This is a concrete class that implements the interface **IResponse**. It represents the response to tell if the client can retrieve the user.

```
public RetrieveUserResponse(Session session, User user)
```

- The constructor. `session` is the session between the client and server and `user` is the user to be retrieved.

```
public void send()
```

- A public method that is used to send a message to the client.

2.3.6 SendMessageResponse

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

```
public SendMessageResponse(Room room, Message message, ArrayList<User> receivers)
```

- The constructor. `room` is where the message is delivered, and `message` is the message delivered by the user. `receivers` is the list of the users who should receive the message.

```
public void send()
```

- A public method that is used to send a message to the client.

2.3.7 UpdateRoomResponse

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

```
UpdateRoomResponse(Room room, ArrayList<User> receivers)
```

- The constructor. `room` is the room whose information needs to be updated and `receivers` is a list of all the users in the room.

```
public void send()
```

- A public method that is used to send a message to the client

2.4 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<String> users, HashSet
<String> msg, User owner) {
```

- 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.

```
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 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.

2.5 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> rooms
```

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

```
private Map<String, User> userMap
```

- A private member variable that maps user id to User.

```
Map<Integer, Room> roomMap
```

- A private member variable that maps room id to Room

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

- A public static method to get a singleton class.

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

- A public method to get all chat rooms.

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

- A public method to get all users.

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

- A public method to add a user to the user list, `u` represents the user to be added.

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

- A public method to add a room to the room list, `r` represents the room to be added.

```
public void removeUser(User u)
```

- A public method to remove a user from the user list, `u` represents the user to be removed.

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

- A public method to remove a room from the room list, `r` represents the room to be removed.

```
public Room joinRoom(int roomId, String userId)
```

- A public method for a user to join a room. This method should call the `sendSystemMessage` method, the parameter `roomid` represents the room to join and the string `userId` represents the user who joins the room.

```
public Room sendMessage(int roomId, String senderId, String receiverId, String text)
```

- A public method to perform sending a message in a chat room. This method should also call `detectHate` to detect hate words. The parameter `roomid` represents the room where the message is delivered, `senderId` represents the sender of the message and `text` represents the message.

```
public Room leaveRoom(int roomId, String userId)
```

- A public method to perform leaving a room, then this method should call `deleteRoomIfEmpty` , and `sendSystemMessage` . The parameter `roomId` represents the room and the `userId` represents the user who is going to leave.

```
public Room updateRoom(int roomId, String userId, int ageMin, int ageMax, ArrayList<String> areas, ArrayList<Strin
g> schools)
```

- A public method to update the room constraint, then this method should call `kickUnqualifiedUsers` , `deleteRoomIfEmpty` , and `sendSystemMessage` . The parameter `roomId` represents the room, `ageMin` represents the minimum age requirement, `ageMax` represents

the max age requirement, `areas` is the restriction of the area and the `schools` is available schools.

```
public Room createRoom(String userId, int ageMin, int ageMax, ArrayList<String> areas, ArrayList<String> schools)
```

- A public method to create a room with constraint. The parameter `userId` represents the user ID, `ageMin` represents the minimum age requirement, `ageMax` represents the max age requirement, `areas` is the restriction of the area and the `schools` is available schools.

```
public ArrayList<User> getQualifiedUsers(int roomId)
```

- A public method to get the qualified users of a room, the parameter `roomId` represents the room.

```
private void detectHate(String text, Room room, User sender)
```

- A public method to detect if there are hate words in the text. If there is such a word, the sender will be kicked from the room. The parameter `text` is the message, `room` is the room to detect for hate words, and `sender` is the message sender.

```
private void deleteRoomIfEmpty(Room room)
```

- A public method to delete the room from the room list if it's empty. The parameter `room` represents the room.

```
private void sendSystemMessage(Room room, User sender, Event event)
```

- A public method to generate a system message in the room. The parameter room represents the current room, the parameter represents the sender of system message and the parameter event represents the event of this message.

```
private void kickUnqualifiedUsers(Room room)
```

- A public method to kick out unqualified users from the room. The parameter `room` represents the room.

3. Use Cases

3.1 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, School.

3.2 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.

3.3 Edit a room

- A user can edit the constraints for an existing room

3.4 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 the chatroom.
- 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.

3.5 Exit a 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”, or “forced to leave”)

4. Design Decisions

4.1 Union Design Pattern

4.1.1.Message

We created an abstract class **Message** as the superclass, and `ChatMessage`, `Event`, `SystemMessages` as the subclass. In this case, we can show the inheritance and polymorphism relationships between “concrete message” and “abstract message”.

4.1.2.Response

We constructed an interface **IResponse** first, and used union design pattern to show the inheritance relationship between “concrete” entities `AddRoomResponse`, `InitResponse`, `RemoveRoomResponse`, `RetrieveUserResponse`, `SendMessageResponse`, `UpdateRoomResponse` and the Interface **IResponse**.

4.2 Strategy Design

We know that all the responses sent from server to client only differs in the behavior. In this case, we decide to isolate different responses in separate classes in order to have the ability for the server to send different messages to the client. Here are all the strategies we have designed: `AddRoomResponse`, `InitResponse`, `RemoveRoomResponse`, `RetrieveUserResponse`, `SendMessageResponse`, `UpdateRoomResponse`, and all of them implements the interface **IResponse**.

4.3 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.

4.4 MVC Design Pattern

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

4.5 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.