Chat App API Design Document

**API design decisions**

1. MVC Design Pattern

Model: User, Room, Lobby...

View: GUI

Controller: ChatAppController

1. Union Design Pattern

PrivateRoom, PublicRoom

1. Singleton and Factory Design Pattern

RoomFactory, UserFactory, MessageFactory, CmdFactory...

1. Command design Pattern

User: RegisterCmd, LoginCmd, getProfileCmd

Room: JoinRoomCmd, JoinRoomListCmd, QuitRoomCmd...

Messages: SendMsgCmd, RecallMsgCmd, EditMsgCmd, DeleteMsgCmd

1. Strategy Pattern

RoomStrategy

**Chat app Use Cases**

1. Create a user

A user can sign up for an account with its username, password, age, school and interests. Once the user finish the sign up, it will create an account and login with the username and password

1. Create a room

A user can create a public or private chat room with selected maximum capacity. The user who creates the chatroom will automatically become the admin of this room. The other users can join the public rooms while the private room can only be joined by the invitation.

1. Send messages

The user can send messages to any other users that are not blocked in the group

The user will receive the messages other users sent in the chat room

If the user sends the message privately, it will not shown on other users’ screen

If the user sends some “hate” word, the message will not be shown on the screen and the sender will be warned

The latest messages will appear at the bottom

The messages can be recalled and deleted by the sender

1. Block a user

The administrator of the room can decide whether a user in the chat room should be banned or not

The other users can report to the administrator of the chat room that some users need to be banned

1. Quit a room

The user can choose to exit one or more chat rooms. After the user leaves the room, the reason why the user exits the room will be notified to other users.

**Classes and Methods Details**

**Adapter**:

1. WebSocketAdapter
   1. Class: WebSocketAdapter

Detail: Web socket for the server

* + 1. Methods:

public void onConnect(Session session)

Description: Open a session for a new connected user.

public void onClose(Session session, int statusCode, String reason)

Description: Close the user’s session.

Parameter session: The user whose session is closed.

public void onMessage(Session session, String message)

Description: Function is called while the user sends a message on the client side.

Parameter session: The session user sending a message.

Paranetter message: The message to be sent

public static ICmdFac getCmdFactory()

Description: Function to get the CmdFactory

Return: A CmdFactory

**Controller**:

1. API

Description: Following is the HTTP GET and POST request we decided to use for the communication between client side and server side. Besides those APIs, we would also use websocket to send and receive messages.

POST /register: Register for a new account; need userName, password, school, age, interests from frontend

POST /login: login for an account; need userName and password from frontend

POST /getRoomUserList: get the user list for the room; need roomId from frontend

**Model**:

We implemented our room, user and other classes in the model folder, with most of our logic functions that will help us achieve our use cases. There are five components: cmd, message, room, user, and one additional class called lobby that contains functions we will use in our lobby page.

1. User
   1. Detail: This class is the abstract class of Users, which can be extendable for creating different users. Our team will use these classes to implement user features in the chatroom.
      1. Variables

private int userId; : user Id .

private String name; : user name

private String pwd; : user password

private int age; : user age

private String school; : user school

private Session session; : the websocket session

private List*<*String*>* interest; : interest list the user interested in.

private List*<*RoomAbs*>* adminRooms; : room list the user can manage.

private Map*<*RoomAbs, Boolean*>* isBlocked; : whether the user is blocked in some rooms

private List*<*RoomAbs*>* joinRooms; : room list the user has already joined

private int curRoom; current room the user is watching

* + 1. Constructor:

*public User(int userId, String name, String pwd, int age, String school, List<String> interest)*

Description: Constructor of this abstract class.

Parameter:

* int userId: userId.
* String name: user name.
* String pwd: user password.
* int age: user age.
* String school: user school.
* List<String> interest: user interest
  + 1. Methods:

public int getUserId*()*

Description: Get user id.

Return: User id.

public String getName*()*

Description: Get username.

Return: Username.

public void setName*(*String name*)*

Description: Set username.

Parameter name: Input username.

public String getPwd*()*

Description: Get user password.

Return: User password.

public void setPwd*(*String pwd*)*

Description: Set password.

Parameter name: Input password.

public int getAge*()*

Description: Get user age.

Return: User age.

public void setAge*(*int age*)*

Description: Set user age.

Parameter name: User age.

public String getSchool*()*

Description: Get user school.

Return: User school.

public void setSchool*(*String school*)*

Description: Set user school.

Parameter name: User school.

public List*<*String*>* getInterest*()*

Description: Get user interest list.

Return: User interest list.

public void addInterest*(*String interest*)*

Description: Add one user interest into the list.

Parameter name: User interest.

public void removeInterest*(*String interest*)*

Description: Remove one user interest into the list.

Parameter name: User interest.

public List*<*RoomAbs*>* getAdminRooms*()*

Description: Get user rooms list as admin.

Return: User admin room list.

public void addAdminRooms*(*RoomAbs adminRoom*)*

Description: Add one user admin into the room list.

Parameter name: User room.

public void removeAdminRooms*(*RoomAbs adminRoom*)*

Description: Remove one user admin room in the list.

Parameter name: User room.

public Map*<*RoomAbs, Boolean*>* getIsBlocked*()*

Description: Get user block room map and block status.

Return: User block map of rooms and status.

public void addIsBlocked*(*RoomAbs room, Boolean block*)*

Description: Add user block room information.

Parameter name: User room, and block status.

public void setIsBlocked*(*RoomAbs room, Boolean block*)*

Description: Set user block room with block status.

Parameter name: User room, and block status.

public void removeIsBlocked*(*RoomAbs room*)*

Description: Remove user block room from map.

Parameter name: User room.

public List*<*RoomAbs*>* getJoinRooms*()*

Description: Get user joined chat room list.

Return: User joined chat room list.

public void addJoinRooms*(*RoomAbs joinRoom*)*

Description: Add user joined chat room into the list.

Parameter name: User room.

public void removeJoinRooms*(*RoomAbs joinRoom*)*

Description: Remove user joined room from the list.

Parameter name: User room.

*public int getCurRoom()*

Description: Get the current room the user is watching.

return: current room

*public void setCurRoom(int room)*

Description: Set the current room the user is watching

Parameter name: current roomId.

* 1. Interface Class: IUserFac

Detail: The interface class for the user factory.

* + 1. Method:

*User make(int userId, String name, String pwd, int age, String school, List<String> interest)*

Description: Make a new User.

Parameter:

* int userId: userId.
* String name: username.
* String pwd: user password.
* int age: user age.
* String school: user school.
* List<String> interest: user’s interests
  1. Concrete Class: UserFactory

Detail: This is the factory class of users that help us handle the implementation details of creating a user.

* + 1. Variable:

private static UserFactory *ONLY*; : the singleton variable of UserFactory;

* + 1. Methods:

public static UserFactory getOnly()

Description: Get the singleton UserFactory.

Return: The singleton UserFactory.

1. Room
   1. Abstract Class: RoomAbs

Detail: This class is the abstract class of Rooms, it includes functions shared with public and private rooms. Our team will use those classes to implement our chat rooms functions.

* + 1. Variables:

private int id; : the current room id;

private RoomType type; : the current room type;

private final int capacity; : the current room capacity;

private final String interest; : the current room interest;

private final User admin; : the current room admin;

private List<User> members; : the current room members list;

private List<Message> msgList; : the current room message list;

private Map<Integer, Integer> msgMapping; : the current room message and message index map;

private List<User> bannedUser; : the current room banned user list;

private Map<User, Message> userMessageMap; : the current room user and their message map;

* + 1. Constructor:

public RoomAbs(int id, RoomType type, int capacity, String interest, User admin)

Description: We will not use this constructor to initialize abstract RoomAbs class but we will use it in our two concrete classes, PublicRoom and PrivateRoom, that extend this abstract class.

Parameter:

* int id: roomId.
* RoomType type: the room type of the current room.
* int capacity: room capacity.
* String interest: room interest.
* User admin admin user.
  + 1. Methods:

public int getId()

Description: Get current room ID.

Return: The current RoomID.

public void setId(int id) :

Description: Set room id.

Parameter int id: input room id.

public RoomType getType()

Description: Get room type.

Return: Current RoomType.

public void setType(RoomType type)

Description: Set Room Type.

Parameter RoomType type: the input RoomType.

public int getCapacity()

Description: Get room capacity.

Return: room capacity.

public User getAdmin()

Description: Get Admin of the current room.

Return: the admin of the current room.

public String getInterest()

Description: Get the interest of the current room.

Return: the interest of the current room.

public List<User> getMembers()

Description: Get members in the room.

Return: All members of the room.

public void addMembers(User user)

Description: Add user to room.

Parameter User user: The input user.

public List<Message> getMsgList()

Description: Get the list of messages in the current room.

Return: The current message list.

public void addMsgToList(Message msg)

Description: Add a message to list.

Parameter Message msg: The new message.

public Map<Integer, Integer> getMsgMapping()

Description: Get the MsgMap.

Return: The MsgMap of the current room.

public void setMsgMapping(Map<Integer, Integer> msgMapping)

Description: Set the MsgMap of the current room.

Parameter Map<Integer, Integer> msgMapping: Map of message id and its index in msgList.

public List<User> getBannedUser()

Description: Get BannedUser list.

Return: The list of the banned users in the current room.

public void addBannedUser(User user)

Description: Add the user to the Ban list.

Parameter User user: The banned user in the current room.

public Map<User, Message> getUserMessageMap()

Description: Get the user message map.

Return: The user message map.

public void setUserMessageMap(Map<User, Message> map)

Description: Set the user message map.

Parameter Map<User, Message> map: The new user message map.

public User removeFromBannedUser(User user)

Description: Remove a user from Banned User.

Parameter User user: the removed user from banned list.

Return: The removed user.

public boolean isInRoom(User user)

Description: check whether a user is in a room or not.

Parameter User user: the user.

public Message getMessageFromList(int msgId)

Description: get the message from the message list based on msgId.

Parameter int msgId: Id of the message.

public void recallMessage(Message m)

Description: recall a message.

Parameter Message m: The message that needs to be recalled.

public void deleteMessage(Message m)

Description: delete a message.

Parameter Message m: The message that needs to be deleted.

public Message editMessage(int msgId, String s)

Description: edit a message.

Parameter msgId: The Id of the message that needs to be deleted. String s: The new message

public boolean equals(Object obj)

Description: check whether two rooms are equal or not.

* 1. Interface Class: IRoomFac

Detail: The interface class for room factory.

* + 1. Methods:

RoomAbs make(int id, RoomType type, int capacity, String interest, User admin);

Description: Method to make a new room.

Parameter:

* int id: roomId.
* RoomType type: the room type of the current room.
* int capacity: room capacity.
* String interest: room interest.
* User admin admin user.
  1. Concrete Class: PublicRoom extends RoomAbs

Detail: This is the public room class. In this kind of room all users who share the same interest with the room can join it.

* + 1. Constructor:

public PublicRoom(int id, RoomType type, int capacity, String interest, User admin)

Description: The constructor of the public room class.

* 1. Concrete Class: PrivateRoom extends RoomAbs

Detail: This is the private room class. In this kind of room only the admin can invite other users to join it.

* + 1. Constructor: public PrivateRoom(int id, RoomType type, int capacity, String interest, User admin)

Description: The constructor of the private room class.

* + 1. Methods:

public User inviteUser(User user)

Description: Invite a new user.

Parameter User user: The user who will be invited.

Return: The invited user.

* 1. Concrete Class: RoomFactory

Detail: This is the factory class of rooms that help us handle the implementation details of creating a room.

* + 1. Variable:

private static RoomFactory *INSTANCE*; : the singleton variable of RoomFactory;

* + 1. Methods:

public static RoomFactory getInstance()

Description: Get the singleton RoomFactory.

Return: The singleton RoomFactory.

* 1. Enum set: RoomType

Detail: This contains the types of rooms we need for the project.

1. RoomStrategy
   1. Interface Class: RoomStrategy

Detail: This is the interface class used for implementing the behavior of rooms

* + 1. Methods:

String getName()

Description: Get the strategy name of the room.

Return: The strategy name.

void updateMessage(RoomAbs room)

Description: Update the message in the current room.

Parameter RoomAbs room: The current room.

1. Message
   1. Concrete Class: Message

Detail: The Message class is designed to organize information about a message. Each message would contain sender, receiver, status, etc. With different status and type of message, the client side would have a different display method.

* + 1. Variables:

private int id; : Message id;

private User sentFrom; : Message sender;

private User sentTo; : Message receiver;

private String content; : Message content;

private MessageType type; : Message type;

private MessageStatus status; : Message status;

* + 1. Constructor:

public Message(int id, User sentFrom, User sentTo, String content, MessageType type)

Description: Constructor of the message.

* + 1. Methods:

public int getId()

Description: Get the id of message

Return: An id of message

public User getSentFrom()

Description: Get the message sender

Return: A user who is the sender

public User getSentTo()

Description: Get the message receiver

Return: A user who is the receiver

public String getContent()

Description: Set the content of message

Return: A string of message content

public void setContent(String content)

Description: Get the message content

Parameter content: : A string of message content

public MessageType getType()

Description: Get the type of message

Return: A MessageType

public MessageStatus getStatus()

Description: Get the status of the message

Return: A MessageStatus

public void setStatus(MessageStatus status)

Description: Set the status of the message

Return: : Status MessageStatus

public boolean equals(Object obj)

Description: override equals to make comparison with two messages

Return: true or false whether two messages are equal

* 1. Interface: IMessageFac

Message make(int id, User sentFrom, User sentTo, String content, MessageType type)

Description: Function to make messages. Concrete class that implements the interface could override it.

Return: : A message

* 1. Concrete Class: MessageFactory implements IMessageFac

Detail: This is the factory class of messages that help us handle the implementation details of creating a message.

* + 1. Variables:

private static MessageFactory *INSTANCE*;: Singleton instance;

* + 1. Constructor:

private MessageFactory()

Description: Constructor of the MessageFactory.

* + 1. Methods:

public static MessageFactory getInstance()

Description: Get the Singleton MessageFactory.

Return: : A MessageFactory

public Message make(int id, User sentFrom, User sentTo, String content, MessageType type)

Description: Override the make function in IMessageFac. Make a message based on input information.

Return: : A Message

1. Cmd
   1. Interface: ICmd

Detail: It is an interface used to pass command and call functions in MsgToClientSender

execute(Session session, Map<String, String> body)

Description: Function to execute ICmd. Concrete class that implements the interface could override it.

Parameter : A Map for input variables and a session

* 1. Implementation of ICmd for userCmd

1. RegisterCmd

Register the user based on the username, password, age, school and interests

The key of body requires userName, pwd, age, school and interests

1. LoginCmd

Login the user to chat room based on username and password

The key of body requires userName and pwd

1. getProfileCmd

Get the profile of user

The key of body requires eventName

1. SignoutEvent

Sign out the user

* 1. Implementation of ICmd for roomCmd

1. createRoomCmd

The user in current session creates a room based on room capacity, public or private, interests

The key of body requires capacity, publicOrPrivate, interest and eventName

1. JoinRoomCmd

The user in current session joins a room

The key of body requires roomId and eventName

1. JoinRoomListCmd

All rooms that the user in current session has already joined

The key of body requires eventName

1. QuitRoomCmd

The user in current session quits the room that the user is watching

The key of body requires eventName

1. InviteCmd

The user in current session invite other user to join the room

The key of body requires eventName and inviteUserId

1. InviteUserListCmd

The users that the user in current session can invite to join the room

The key of body requires eventName

1. AcceptInvitationCmd

The user in current session determines whether or not accept the invitation to join the room

The key of body requires eventName and invitationRoomId

1. BanUserCmd

The user in current session wants to ban another user in the current chatroom

The key of body requires eventName, banUserId and currentRoomId

1. BanUserListCmd

The list of users that the user in current session can ban in the current chatroom

The key of body requires eventName and currentRoomId

1. ReportUserCmd

The user in current session want to report another user in the current chatroom to the admin of the chatroom

The key of body requires eventName and reportedUserId

1. ReportUserListCmd

The list of users that the user in the current session can report to the admin

The key of body requires eventName and currentRoomId

1. SwitchRoomCmd

The user in the current session wants to switch the current room to another room

The key of body requires eventName and roomId

1. SwitchRoomListCmd

The list of rooms that the user can switch to

The key of body requires eventName

1. SendPrivateMsgListCmd

The user in the current session wants to send a private message

The key of body requires currentRoomId and eventName

* 1. Implementation of ICmd for messageCmd

1. sendMsgCmd

The user sends a message in the current chatroom

The key of body requires currentRoomId, privateReceiver (empty for public message), content, messageType and eventName

1. sendPrivateMsgCmd

The user sends a private message in the current chatroom

The key of body requires currentRoomId, privateReceiver, content, messageType and eventName

1. sendPublicMsgCmd

The user sends a public message in the chatroom

The key of body requires currentRoomId, content, messageType and eventName

1. editMsgCmd

The user edits a message in the current chatroom

The key of body requires RoomId, privateReceiver(empty for public message), MessageId, content, messageType and eventName

1. RecallMsgCmd

The user recalls a message in the current chatroom

The key of body requires RoomId, privateReceiver(empty for public message), MessageId and eventName

1. DeleteMsgCmd

The user deletes a message in the current chatroom

The key of body requires roomId, MessageId and eventName

* 1. Interface: ICmdFac

ICmd make(String type)

Description: Function to make ICmd. Concrete class that implements the interface could override it.

Return: : A ICmd

* 1. Concrete Class: CmdFactory implements the ICmdFac

Detail: This is the factory class of ICmd that helps us handle the implementation details of creating an ICmd.

* + 1. Variables:

private static CmdFactory *INSTANCE*; : The singleton variable of CmdFactory;

* + 1. Constructor:

private CmdFactory()

Description: Constructor of the CmdFactory.

* + 1. Methods

public static CmdFactory getInstance()

Description: Get the Singleton CmdFactory

Return: : A CmdFactory

public ICmd make(String type)

Description: Override the make function in ICmdFac. Make a message based on input type.

Return: : A ICmd

1. Lobby

This is the class that is used to build a bridge between users and rooms.

1. Make the lobby to manage the chat app

private static Lobby *singleton* = null;

public static Lobby makeLobby()

1. Variable Definition

private ArrayList<User> users; : A private variable to store the list of users

private ArrayList<RoomAbs> rooms; : A private variable to store the list of rooms

private ArrayList<Message> msgs; : A private variable to store the list of messages

private Map<Integer, User> userMap; : A private variable to store the userId of the users

private Map<Integer, RoomAbs> roomMap; : A private variable to store the roomId of the rooms

private Map<Session, User> sessionUserMap;: A private variable to store the session of the user

private Map<Integer, Session> userIdSessionMap;A private variable to store session of the user based on userId

private Map<Integer, List<RoomAbs>> userJoinedRoomsMap;A private variable to store the user and rooms it has already joined

1. Function Definition for Users

public ArrayList<User> getUsers()

Descriptions: Get all users in the user list

Return: user list

public ArrayList<User> addUser(User u)

Descriptions: Add a user in the list, u is the user you want to add

Return: user list after you add a user

public ArrayList<User> removeUser(User u)

Descriptions: Remove a user in the list, u is the user you want to delete

Return: user list after you delete a user

public ArrayList<RoomAbs> getAvailableRooms(User u)

Get the available rooms the user u can join in

Return: room list the user can join

public User findUser(String userName, String pwd)

Get the user in the userMap based on userName and password

Return: user if it exists; null if it doesn’t exist

public User createUser(String name, String pwd, int age, String school, List<String> interest)

Create a user with userName, password, age, school and interest

Return: user if it creates successfully; null if it fails.

public void blockUser(int userId, int roomId)

Block the user with userId in room with roomId

public void removeBlockUser(int userId, int roomId)

Remove the block of the user with userId in room with roomId

public Map<Session, User> getSessionUserMap()

Get the map between the Session and User

public void addSessionUser(Session session, User user)

Add a (session, user) pair

public void removeSession (Session session)

Remove a session

public Map<Integer, Session> getUserIdSessionMap()

Get the map between userId and its session

public Map<Integer, User> getUserMap()

Get the map between userId and user

public Map<Integer, List<RoomAbs>> getUserJoinedRoomsMap()

Get the map between userId the the rooms the user has already joined

public List<RoomAbs> getUserJoinedRoomList(int userId)

Get the list of rooms the user with userId has already joined.

1. Function Definition for Rooms

​​public ArrayList<RoomAbs> getRooms()

Descriptions: Get all rooms in the room list

Return: room list

public ArrayList<RoomAbs> addRoom(RoomAbs r)

Descriptions: Add a room in the list, r is the room you want to add

Return: room list after you add a room

public ArrayList<RoomAbs> removeRoom(RoomAbs r)

Descriptions: Remove a room in the list, r is the room you want to delete

Return: room list after you delete a room

public void joinRoom(int userId, int roomId)

Descriptions: A function to let the user with userId join the room with roomId

Return: none

public void quitRoom(int userId, int roomId)

Descriptions: A function to let the user with userId quit the room with roomId

Return: none

public ArrayList<User> getAvailableUsers(RoomAbs r)

Descriptions: Get the available users the room r can invite

Return: user list the room can invite

public Map<Integer, RoomAbs> getRoomMap()

Get the map between roomId and room

public RoomAbs createRoom(String type, int capacity, String interest, User admin)

Create a room based on type (public or private), capacity, interest and the admin who creates the room

public boolean switchRoom(int userId, int roomId)

switch the room the user with userId is currently watching to the room with roomId

1. Function Definition for Messages

public void sendMessage(int userId, String roomId, Message msg) Descriptions: Send a message; userId is the user that sends the message, r is the room that the message sends in and msg is the message information

Return: none

public ArrayList<Message> getMsgs()

Descriptions: Get the message list

Return: message list

public ArrayList<Message> addMsgs(Message msg)

Descriptions: Add a message, msg is the message you want to send

Return: message list after you add a message

public ArrayList<Message> removeMsgs(Message msg)

Descriptions: Remove a message, msg is the message you want to delete

Return: message list after you remove a message

7. MsgToClientSender

This is a class to send message to client by websocket, which will be called by Cmd

public static void loginEventRes(Session session, User newUser)

Login of the user to current session

public static void createRoomEventRes(String eventName, Session session, Lobby lobby, RoomAbs room)

An event to let the user in current session to create a room

public static void joinRoomEventRes(String eventName, RoomAbs room, User curUser)

An event to for curUser to join a room

public static void joinRoomUserListEventRes(String eventName, Session session, Lobby lobby, List<RoomAbs> roomList)

Get the list of rooms that user has already joined

public static void switchRoomUserListEventRes(String eventName, Session session, Lobby lobby, List<RoomAbs> roomList)

Get the list of rooms that the user can switch to

public static void switchRoomEventRes(String eventName, RoomAbs room, User curUser)

An event for curUser to switch to another room in his lists of joined rooms

public static void inviteEventRes(String eventName, RoomAbs room, User curUser, User invitedUser, Lobby lobby)

An event to let curUser to invite other users to join the room

public static void inviteUserListEventRes(String eventName, Session

session, Lobby lobby, List<User> userList)

Get the list of users that the user in current session can invite to join the room

public static void quiteRoomEventRes(Session session, String eventName, RoomAbs room, User curUser)

An event for curUser to quit the room

public static void banUserEventRes(String eventName, RoomAbs room, User curUser, User banUser)

An event for curUser to ban other users

public static void banUserListEventRes(String eventName, Session session, Lobby lobby, List<User> userList)

Get the list of users that the user can ban in the chatroom

public static void reportUserEventRes(Session session, String eventName, Lobby lobby, RoomAbs room, User curUser, User admin, User reportedUser)

An event for curUser to report another user to admin of the chatroom

public static void reportUserListEventRes(String eventName, Session session, Lobby lobby, List<User> userList)

Get the list of users that the user in the current session can report to the admin of the chatroom

public static void sendPublicMsgEventRes(String eventName, RoomAbs room)

An event for current user to send public message to the chatroom

public static void sendPrivateMsgEventRes(String eventName, User privateReveiver, Message message)

An event for current user to send private message to the chatroom

public static void recallMsgEventRes(String eventName, RoomAbs room, Message message)

An event for current user to recall messages

public static void editMsgEventRes(String eventName, RoomAbs room, Message message)

An event for current user to edit messages

public static void getProfileRes(String eventname,User curUser,Session session)

An event to get the profile of current user

public static void deleteMsgEventRes(String eventName, RoomAbs room, Message message)

An event for current user to delete messages

public static void errorMessageRes(Session session, String errorMessage)

Send error messages if the event failed