COMP 504 Graduate Object-Oriented Programming and Design

Final Chat App API Specification

Group Dog

Team lead: Lize Chen

Tech Lead: Huaminghui Ding

Documentation Lead: Tingting Li

Developer: Daolun Chen

Developer: Ruimin Li

Developer: Yichen Sun

Developer: Junhao Yu

Rice University November 5, 2021

Content

1 Overview

1.1 introduction

People talk to each other for numerous reasons. To know more about your friends, to get information about your interests, to avoid inconvenience of using a phone, etc. Web chat is a great way to offer real-time chat with efficiency. To provide a place for people to share interests and to connect with friends, our team decided to build an online web chat room app.

1.2 project description

The object of our team is to design and implement a user-friendly web app to allow people with shared interests to communicate. Users visit our Heroku app link and start chatting under our guidance. Talking need lots of users. Each online user is simultaneously running their own instance of a chat room and able to create, join, chat, and leave a chat room. There are two roles of users and two types of chat rooms, which satisfy multiple use cases. We apply Object-oriented Programming, Web Sockets, design patterns to structure our code and meet requirements. We neither use a database nor front end frameworks. Technologies include Java, HTML, CSS, JavaScript, bootstrap, and Spark Java framework.

2 Use cases

2.1 use cases included

(1) log in, register, and log out

• Create an account with a profile including the user name, school, age, interests, and password.
• See notice if the registration info is invalid.
• Log into the system with correct user name and password after creating an account.
• Back to the login page when log out from the system.
• Log into the system again if they already have an account.

(2) create group chat room
• A user can create a chat room and become the admin of it
• Set the unique room name for a chat room
• Initiate the creation of a chat room with a room size (1-50)
• Set a chat room as public or private
• Set the optional interest for a chat room
• Set the password for a private chat room
• See notice if the creation info is invalid
• Record the admin of the room. One of the users in the chat is the admin.
• Set restrictions on who can join the chat room (imposed by owner)
• Create rules of the chat room before joining it, for example, no hate speech, invite yourself after joining, etc.
• Write description about the chat room for other people want to join

	admin	member
Create a chat room	1	0
Join a chat room	1	1
Send messages	1	1
Mute others	1	0
Kick others	1	0
Send notifications	1	0
Block others	1	0
Chat with one user	1	1

Invite users	1	0
See chat history	1	1
Leave a chat room	1	1

	Group chat room	User chat room
create	Created by an owner	Created when user A chats directly with user B in the same group
Number of people	>=2	2
message	broadcast	direct
feature	Chat, Mute, kick, block, invite, history, notification, leave	Chat, leave

(3) join chat room

• See the room name, privacy, current number of people, room size of chat rooms, which have been created and the user is not in

• Select from available chat rooms and attempt to join one of them

• A user can determine what public rooms they can join

• A user able to join a private room by accepting a invitation

• Read the room name, description, rules of a room a user currently select

• A user may be in multiple chat rooms (public and private)

• See what group chat rooms and user chat rooms they have joined (ought to be empty when users join a new room at first)

• See the members and their role of a room a user currently select

• Users able to see and edit his/her selection

• See the chat history of a chat room when a user wants to (click a button)

(4) chat with a user

• See all user names in the same room except for himself/herself

• Select one user and create a user chat room, which maximum size is two and have two users
• Users able to select and send direct messages with one of the users in the current chat room, and messages only appear on these two users' screen
• See user chat room name, user list, and related information on the page
• Able to chat and leave a user chat room

(5) send and receive messages
• Send a broadcast message when a user enters the chat room
• Receive a message when someone join the chat room
• Type out some words or select emojis
• Edit, recall or delete a message
• Send files is not supported. Send links is supported.
• Send a broadcast message by enter or clicking on the dog clows image
• Users able to see when a message was sent (timestamp)
• Users only see the messages they are suppose to view
• Users able to see who sent a message (Username)
• Users able to see real-time messages with the newest one below the others
• Users able to receive notifications when invited, blocked, muted by admins, or kicked by admins
• Users able to accept or not accept notifications
• An unblocked user can send a direct or broadcast message to all users in the chat room
• Users able to see update of chat page and current people of the room when switch to a different room
• A user will be warned and eventually forcibly banned from all chat rooms if the user uses the phrase "hate speech" in a message

(6) manage group
• Owner able to set the members of the chat room by sending invitations
• Users able to accept or reject invitations from the admin to join a chat room

• Admin able to select and block some members

• Blocked users of a chat room cannot enter the room

• Admin able to select and mute some members

• Muted users cannot send messages

• Admin able to select and kick some members and decide who is in the chat room

• Kicked users will leave this group chat room

• Admins broadcast approved requests of qualified users. Admins monitor users and messages. Admin can ban users and delete messages.

(7) leave a chat room

• Read the confirmation page and choose to stay or leave the current chat room when click on a button

• Stop receiving messages

• Delegate his/her (e.g. admin) role before leaving the room.

• A user able to choose leave the current one chat room or leave all chat rooms when log out.

• A user sends a broadcast message in the group about when and why he/she leaves the chat room

• Users in the current chat room receive the message when a user leave

• The owner is able to delete the room if there is only one user in the room

• Clear to remaining users why a user left the room

2.2 use cases not included

• Set different names for display on chat rooms

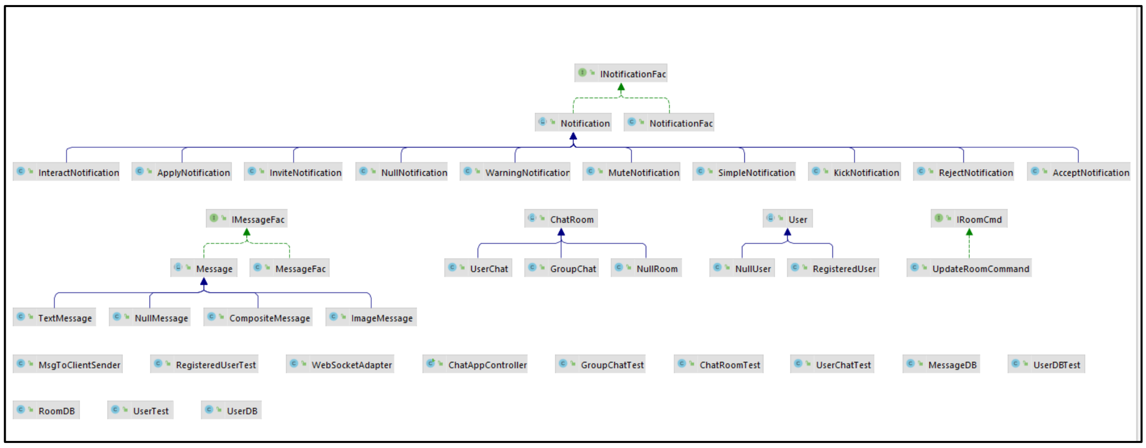
• Join a group by sending a request to the admins

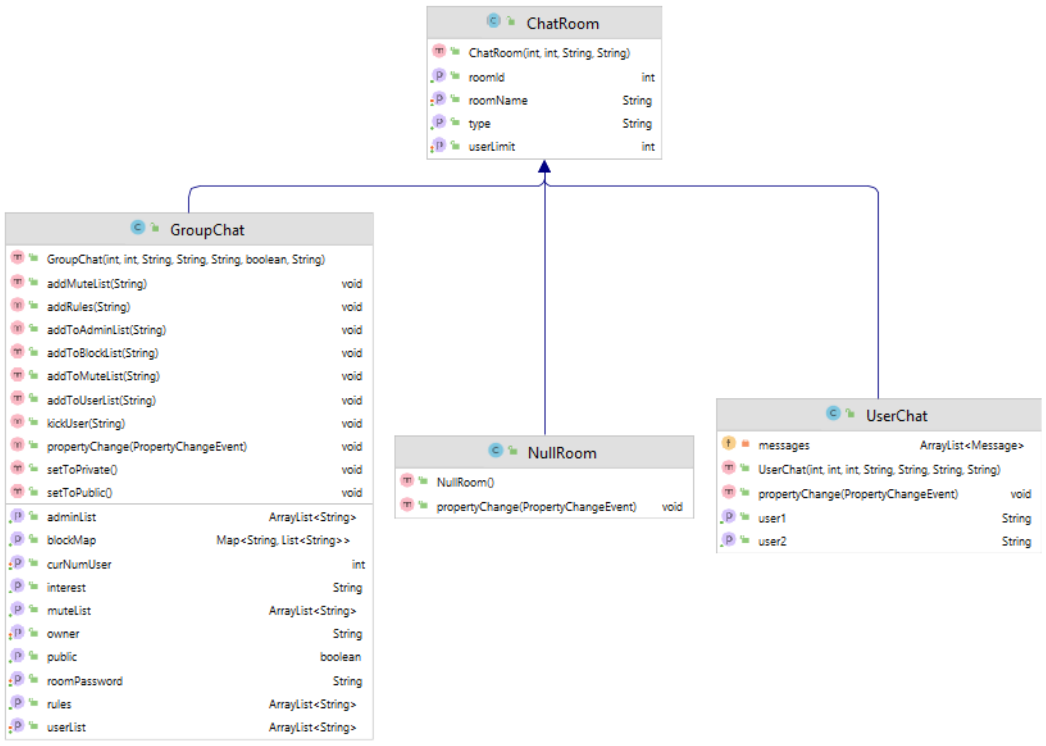
• Format the message (font, size, color)

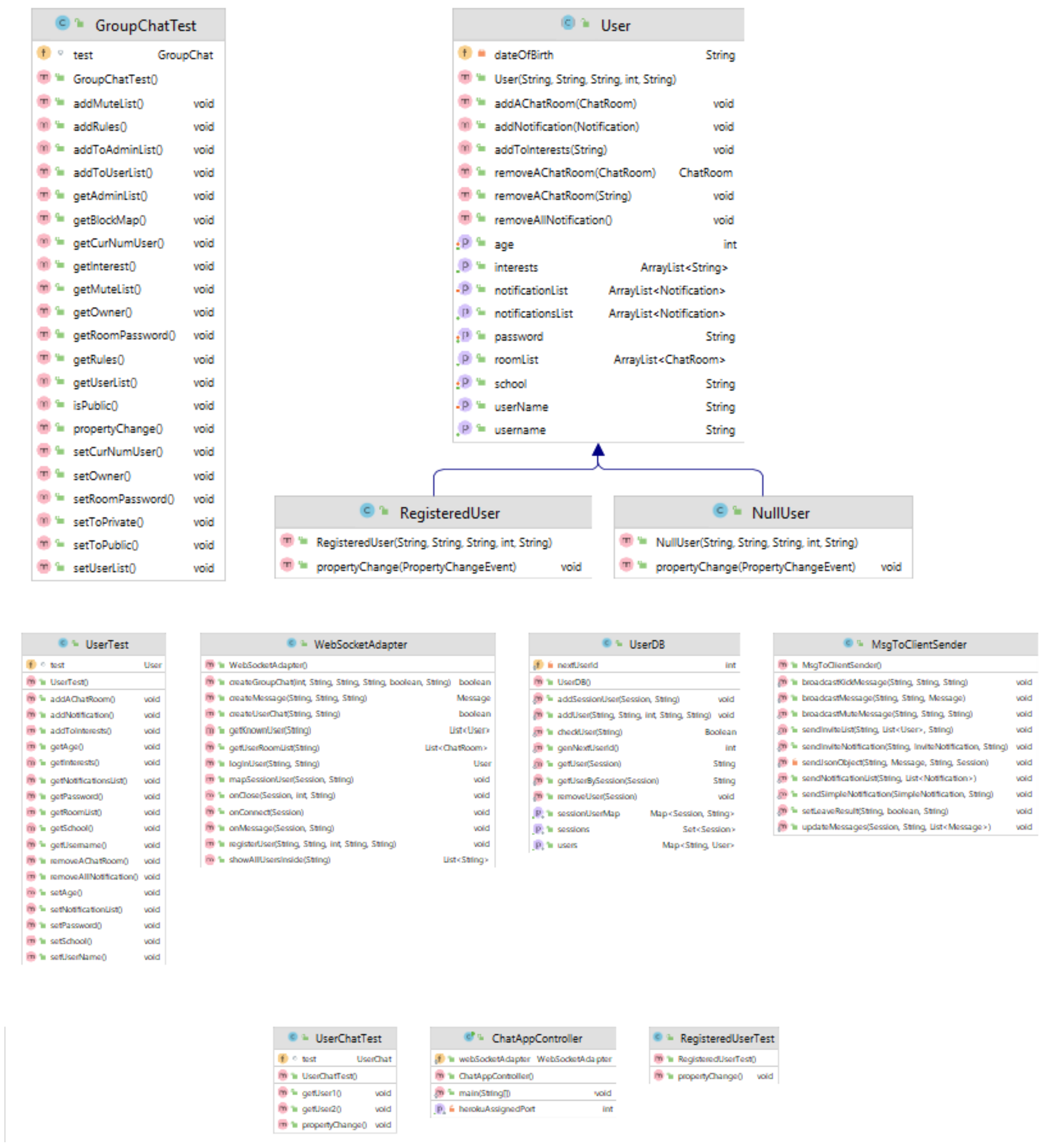
• An unblocked user should be notified that their message has been received

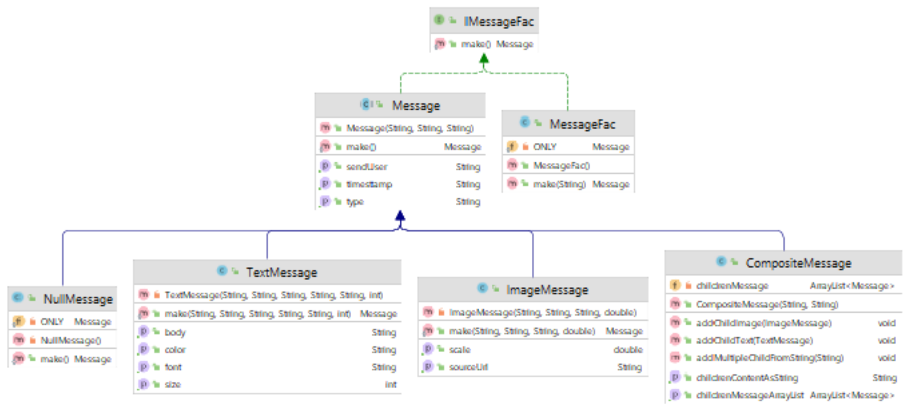
3 Project Design

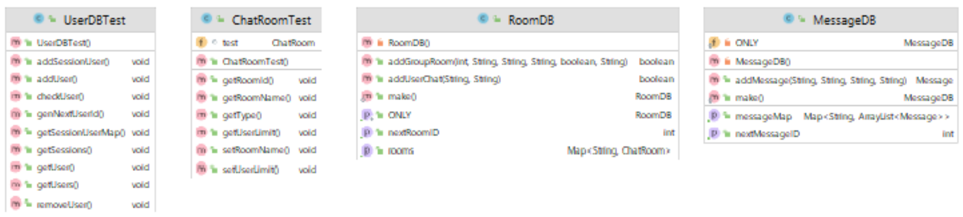
3.1 UML diagram

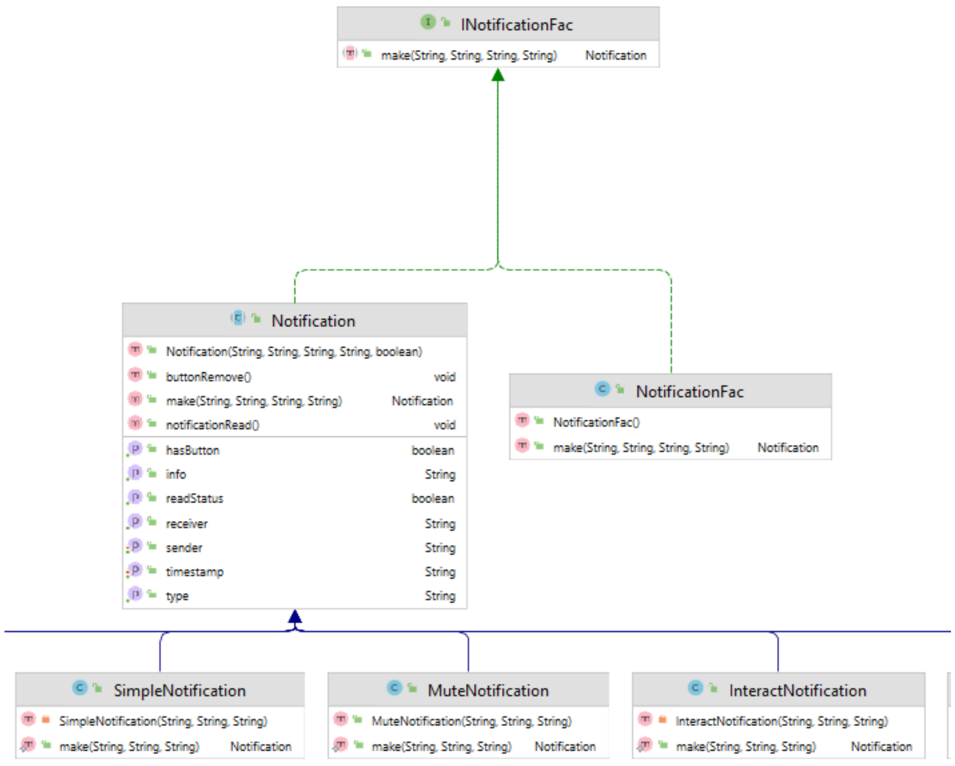



ChatRoom
ChatRoom(int, int, String, String)
roomId int
roomName String
type String
userLimit int
GroupChat
GroupChat(int, int, String, String, String, boolean, String)
addMuteList(String) void
addRules(String) void
addToAdminList(String) void
addToBlockList(String) void
addToMuteList(String) void
addToUserList(String) void
kickUser(String) void
propertyChange(PropertyChangeEvent) void
setToPrivate() void
setToPublic() void
adminList ArrayList<String>
blockMap Map<String, List<String>>
curNumUser int
interest String
muteList ArrayList<String>
owner String
public boolean
roomPassword String
rules ArrayList<String>
userList ArrayList<String>
NullRoom
NullRoom()
propertyChange(PropertyChangeEvent) void
UserChat
messages ArrayList<Message>
UserChat(int, int, int, String, String, String, String)
propertyChange(PropertyChangeEvent) void
user1 String
user2 String


GroupChatTest
test GroupChat
GroupChatTest()
addMuteList() void
addRules() void
addToAdminList() void
addToUserList() void
getAdminList() void
getBlockMap() void
getCurNumUser() void
getInterest() void
getMuteList() void
getOwner() void
getRoomPassword() void
getRules() void
getUserList() void
isPublic() void
propertyChange() void
setCurNumUser() void
setOwner() void
setRoomPassword() void
setToPrivate() void
setToPublic() void
setUserList() void
User
dateOfBirth String
User(String, String, String, int, String)
addAChatRoom(ChatRoom) void
addNotification(Notification) void
addToInterests(String) void
removeAChatRoom(ChatRoom) ChatRoom
removeAChatRoom(String) void
removeAllNotification() void
age int
interests ArrayList<String>
notificationList ArrayList<Notification>
notificationsList ArrayList<Notification>
password String
roomList ArrayList<ChatRoom>
school String
userName String
username String
RegisteredUser
RegisteredUser(String, String, String, int, String)
propertyChange(PropertyChangeEvent) void
NullUser
NullUser(String, String, String, int, String)
propertyChange(PropertyChangeEvent) void
UserTest
test User
UserTest()
addAChatRoom() void
addNotification() void
addToInterests() void
getAge() void
getInterests() void
getNotificationsList() void
getPassword() void
getRoomList() void
getSchool() void
getUsername() void
removeAChatRoom() void
removeAllNotification() void
setAge() void
setNotificationList() void
setPassword() void
setSchool() void
setUserName() void
WebSocketAdapter
WebSocketAdapter()
createGroupChat(int, String, String, String, boolean, String) boolean
createMessage(String, String, String) Message
createUserChat(String, String) boolean
getKnownUser(String) List<User>
getUserRoomList(String) List<ChatRoom>
loginUser(String, String) User
mapSessionUser(Session, String) void
onClose(Session, int, String) void
onConnect(Session) void
onMessage(Session, String) void
registerUser(String, String, int, String, String) void
showAllUsersInside(String) List<String>
UserDB
nextUserId int
UserDB()
addSessionUser(Session, String) void
addUser(String, String, int, String, String) void
checkUser(String) Boolean
genNextUserId() int
getUser(Session) String
getUserBySession(Session) String
removeUser(Session) void
sessionUserMap Map<Session, String>
sessions Set<Session>
users Map<String, User>
MsgToClientSender
MsgToClientSender()
broadcastKickMessage(String, String, String) void
broadcastMessage(String, String, Message) void
broadcastMuteMessage(String, String, String) void
sendInviteList(String, List<User>, String) void
sendInviteNotification(String, InviteNotification, String) void
sendJsonObject(String, Message, String, Session) void
sendNotificationList(String, List<Notification>) void
sendSimpleNotification(SimpleNotification, String) void
setLeaveResult(String, boolean, String) void
updateMessages(Session, String, List<Message>) void
UserChatTest
test UserChat
UserChatTest()
getUser1() void
getUser2() void
propertyChange() void
ChatAppController
webSocketAdapter WebSocketAdapter
ChatAppController()
main(String[]) void
herokuAssignedPort int
RegisteredUserTest
RegisteredUserTest()
propertyChange() void


IMessageFac
make() Message
Message
Message(String, String, String)
make() Message
sendUser String
timestamp String
type String
MessageFac
ONLY Message
MessageFac()
make(String) Message
NullMessage
ONLY Message
NullMessage()
make() Message
TextMessage
TextMessage(String, String, String, String, String, int)
make(String, String, String, String, String, int) Message
body String
color String
font String
size int
ImageMessage
ImageMessage(String, String, String, double)
make(String, String, String, double) Message
scale double
sourceUrl String
CompositeMessage
childrenMessage ArrayList<Message>
CompositeMessage(String, String)
addChildImage(ImageMessage) void
addChildText(TextMessage) void
addMultipleChildFromString(String) void
childrenContentAsString String
childrenMessageArrayList ArrayList<Message>


UserDBTest
UserDBTest()
addSessionUser() void
addUser() void
checkUser() void
genNextUserId() void
getSessionUserMap() void
getSessions() void
getUser() void
getUsers() void
removeUser() void
ChatRoomTest
test ChatRoom
ChatRoomTest()
getRoomId() void
getRoomName() void
getType() void
getUserLimit() void
setRoomName() void
setUserLimit() void
RoomDB
RoomDB()
addGroupRoom(int, String, String, String, boolean, String) boolean
addUserChat(String, String) boolean
make() RoomDB
ONLY RoomDB
nextRoomID int
rooms Map<String, ChatRoom>
MessageDB
ONLY MessageDB
MessageDB()
addMessage(String, String, String, String) Message
make() MessageDB
messageMap Map<String, ArrayList<Message>>
nextMessageID int


INotificationFac
make(String, String, String, String) Notification
Notification
Notification(String, String, String, String, boolean)
buttonRemove() void
make(String, String, String, String) Notification
notificationRead() void
hasButton boolean
info String
readStatus boolean
receiver String
sender String
timestamp String
type String
NotificationFac
NotificationFac()
make(String, String, String, String) Notification
SimpleNotification
SimpleNotification(String, String, String)
make(String, String, String) Notification
MuteNotification
MuteNotification(String, String, String)
make(String, String, String) Notification
InteractNotification
InteractNotification(String, String, String)
make(String, String, String) Notification

3.2 interface/abstract class

3.2.1 WebsocketAdapter

Class description

Create a web socket for the server.

Method Details

Method	Description
public void onConnect(Session session)	Open user's session.
public void onClose(Session session, int statusCode, String reason)	Close the user's session.
public void onMessage(Session session, String message)	Send a message in a session based on different types of actions.
public void mapSessionUser(Session session, String username)	Map websocket session with current username when logging in.
Public void registerUser(String username, String school, int age, String interestsTemp, String password)	Add a user to UserDB.
public User logInUser(String username, String password)	To validate a user is registered and the password match.
public List<ChatRoom> getUserRoomList(String userName)	Get room list of a user.
public List<User> getKnownUser(String userName)	Get users in same rooms.
public boolean createGroupChat(int userLimit, String roomName, String interest, String ownerUsername,	Create a room, add this room to it's owner's roomList

boolean isPublic, String roomPassword)	
public boolean createUserChat(String userName, String friendName)	Create a user chat, add this room to it's owner's roomList
public List<String> showAllUsersInside(String roomname)	Return the list of user/admin/owner of the chosen room
public Message createMessage(String sender, String room, String body)	Adapter's create message function.

3.2.2 Package chatroom

Class description

class	Description
ChatRoom	An abstract class to represent all chat room related objects.
GroupChat	A concrete class to represent multiple users chat room.
NullRoom	A concrete class to represent non-meaningful chat room, prevent null pointer exception.
UserChat	A concrete class to represent user to user chat room.

3.2.3 Package message

Class description

class	Description

CompositeMessage	A concrete class to represent multiple text and images mixed message.
ImageMessage	A concrete class to represent one image type message.
Message	An abstract class to represent all messages object.
MessageFac	A factory class to make messages.
NullMessage	A concrete class to represent one non-meaningful message, prevent null pointer exception.
TextMessage	A concrete class to represent one text type message.

Interface description

Interface	Description
IMessageFac	Interface for Message Factory.

3.2.4 Package notification

Class description

class	Description
AcceptNotification	A concrete class to represent all those notification which type is accept.
ApplyNotification	A concrete class to represent all those notification which type is apply.
InteractNotification	A concrete class to represent all those notifications which can have interaction buttons.
InviteNotification	A concrete class to represent all those notification which type is invite.

KickNotification	A concrete class to represent all those notification which type is kick.
MuteNotification	A concrete class to represent all those notification which type is mute.
Notification	An abstract class to represent all notifications with invariant attributes across all types of notifications.
NotificationFac	A factory class to make different types of notification.
NullNotification	A concrete class to represent all those non-meaningful notifications, prevent null pointer exception
RejectNotification	A concrete class to represent all those notification which type is reject.
SimpleNotification	A concrete class to represent all those notifications without any interactions.

Interface description

interface	Description
INotificationFac	Interface for Notification Factory.

3.2.5 Package user

Class description

class	Description
NullUser	NullUser class is to implement User abstract class to prevent null pointer exception.
RegisteredUser	RegisteredUser class is a concrete class to implement User abstract class to any registered user, we should use this class to store data and make functions.

User	User abstract class to store User's related attributes.

3.2.6 Other

Class description

class	Description
MessageDB	MessageDB is used to store messages
MsgToClientSender	Send messages to the client.
RoomDB	RoomDB is used to store rooms
UserDB	UserDB is used to store users

3.3 Server API

endpoint	verb	payload	response	description
/login	post	Username, password	User object	Check if username is in UserDB and then check if the password is correct.
/register	post	Username, school,age, interests, password	true	Check if the username is in UserDB, if in, then return false.
/userInfo	post	username	User object	Find a specific user
/roomInfo	post	roomName		Find a specific room
/join/getRooms	post	username	A List<ChatRoom> object	Get all rooms a user not join yet.

/join/group	post	Username, roomName	ArrayList<ChatRoom>	Check and add a validate user to a group.
/join/notification/accept	post	Sender, receiver, roomName	A string	Put the applicant into the room member list.
/join/notification/reject	post	Sender, receiver, roomName	A string	Send a rejection notification to the applicant.
/logout	get	username	true	To let a user log out and remove the session related info.
/room/update	get	username	List<ChatRoom>	Update room list per 0.1 second .
/room/members	get	Username,roomname	List<String>	Get the member list of a given roomId.
/create/groupchat	post	roomName, maxUser, isPublic, password, interest, username	boolean	Create a new group chat.
/chat/getUsers	post	username	List<User>	Get users except the current user.
/create/userchat	post	Username, chatName,room	boolean	Create a new user chat.

3.4 Client request

getSendMsgRequest

getLoginRequest

getInviteRequest

getMuteRequest

getKickRequest

getBlockRequest

getLeaveRequest

getUpdateMsg

getLogoutRequest

getDeleteRequest

getRecallUpdateRequest

getInviteUsersRequest

3.4 Design Pattern

3.4.1 Template Design Pattern

To design API that meet all the use cases and complete implement, we analyzed variant and invariant operations for the project. Interfaces and abstract class set up invariant skeleton of the algorithm, and concrete subclasses implement specific variant details.

3.4.2 MVC Design Pattern

Our group used controller to instantiate the adapter and communicate with the model and view.

3.4.3 Factory design pattern

To develop a robust and scalable application, the controller and DispatchAdapter should not care how an object is created. In this project, our team used the factory to handle details. The MessageFac is a factory class to make messages. The NotificationFac is a factory to make different types of notifications. We also designed the RoomStrategyFac to make different types of room related strategies, as well as the RoomCmdFac to make different types of commands.

3.4.4 Composite design pattern

It's normal that users send a message that contains both text and image(emoji). To meet this requirement, we applied the composite design pattern in this application. The CompositeMessage class has an ArrayList of Messages. In this case, users may request a composite with specific children and send composite messages.

3.4.5 Null object design pattern

In the process of registering, joining a room, and chatting, it is inevitable to have some non-meaningful cases. To represent non-meaningful instances, we designed the NullRoom, NullMessage, NullNotification, and NullUser class. Meanwhile, it can also prevent null pointer exceptions.

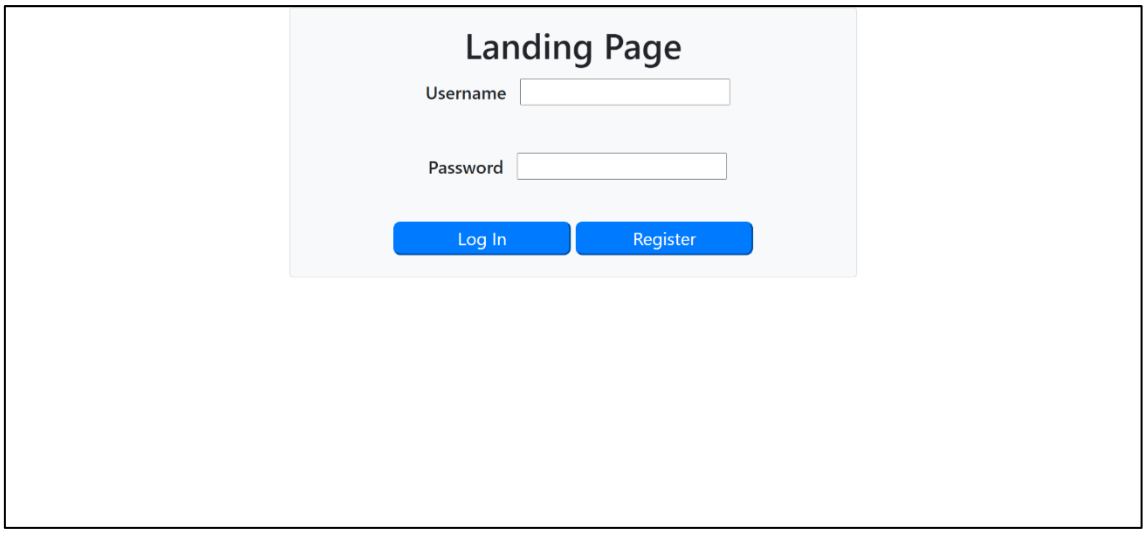
3.4.6 Singleton Design Pattern

Some class contains invariant attributes and information. We only need one instance of the class to save space, instead of allocating space for objects again and again. Therefore, we used the singleton design pattern for concrete classes NullNotification, MessageFac, and NullMessage.

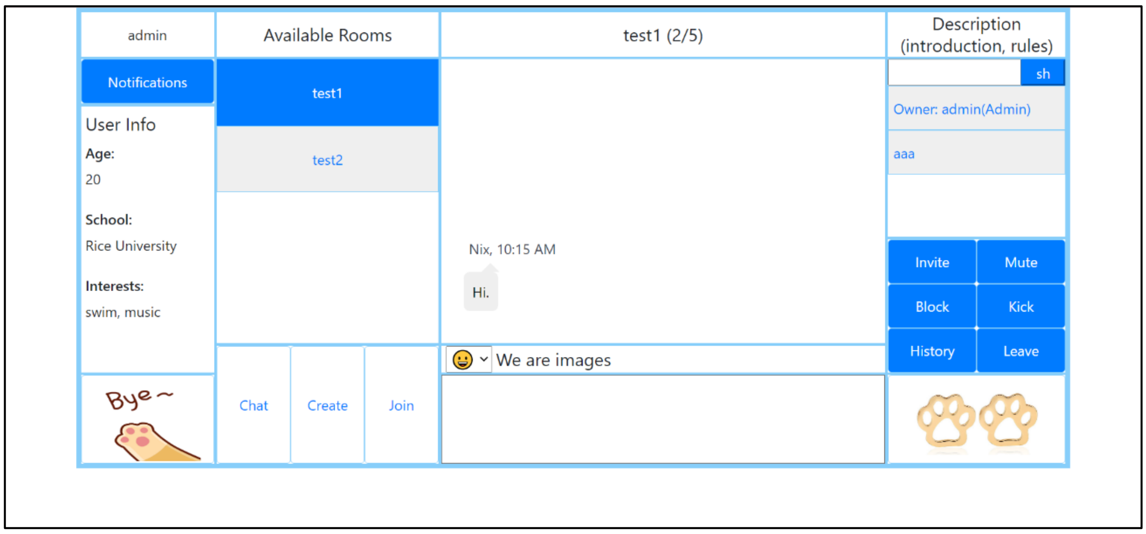
4 result

4.1 Heroku app login page

https://chatapp-final-team-dog.herokuapp.com/



4.2 Main page


admin
Available Rooms
test1 (2/5)
Description (introduction, rules)
Notifications
test1
test2
sh
Owner: admin(Admin)
aaa
User Info
Age:
20
School:
Rice University
Interests:
swim, music
Nix, 10:15 AM
Hi.
Invite
Mute
Block
Kick
History
Leave
We are images
Bye~
Chat
Create
Join