UNIVERSITY OF SCIENCE - VNUHCM
FACULTY OF INFORMATION TECHNOLOGY

FINAL PROJECT REPORT

Java Chat System

COURSE: JAVA PROGRAMMING

Group 8 - CSC13102_21KTPM4

Student	Nguyen Tuan Kiet	**ID**	21127089
	Pham Anh Nhu Ngoc		21127119
	Pham Tran Tuan Tu		21127717
Lecturer	Truong Phuoc Loc Ho Tuan Thanh Nguyen Thi Minh Tuyen		

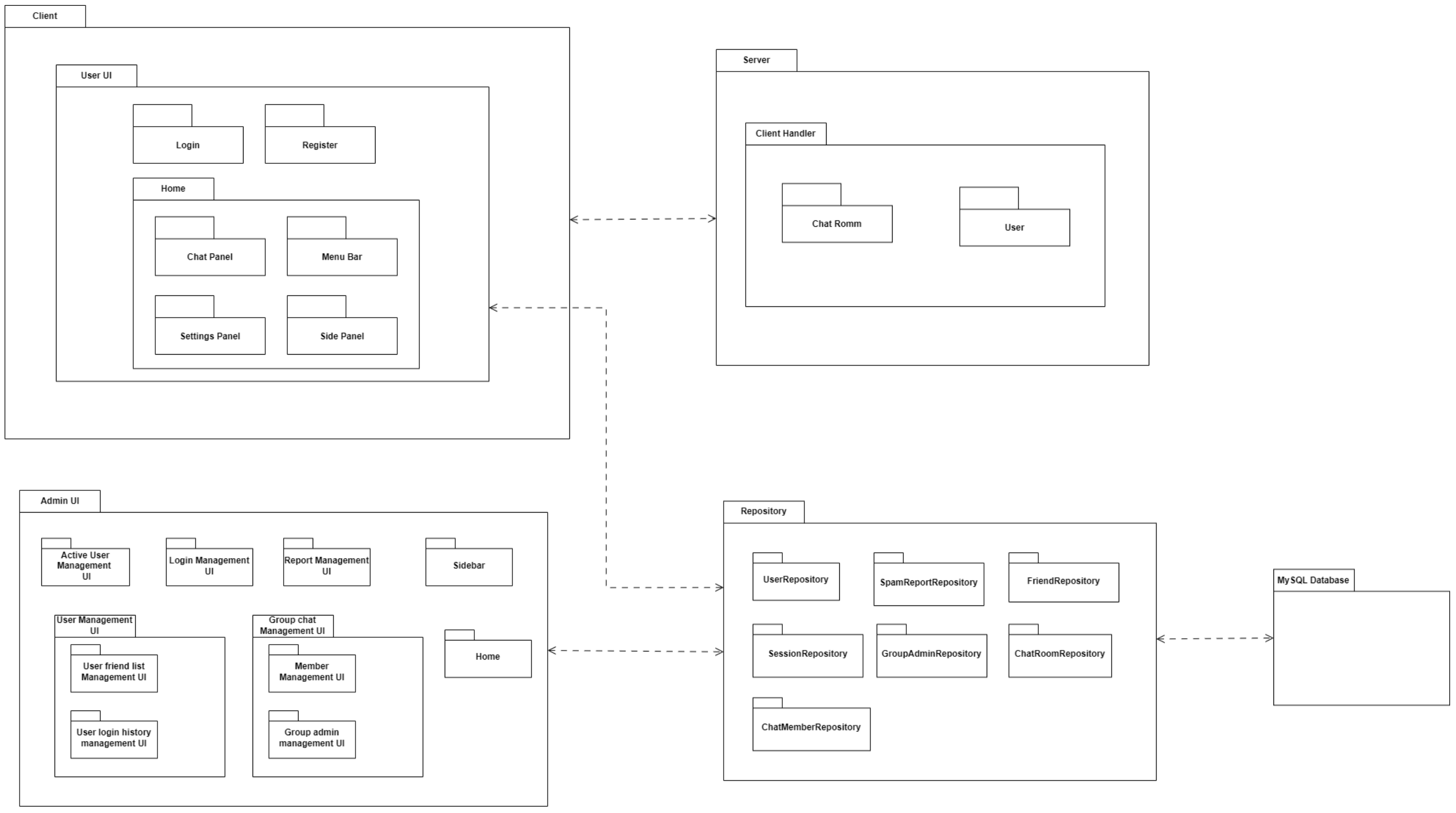
HCM City, January 6th, 2024

Contents

I. ROLES AND RESPONSIBILITIES

Name	ID	Role	Email

II. SYSTEM DESIGN



Picture link: chat-system.drawio.png

1. Client-Server architecture

To implement the ability to chat and many other features, we use client-server architecture with the help of java.net.

On the server side, we store a list of current active chat rooms (an active chat room is defined as a chat room with at least 1 online chat member, and a chat room can be both a group chat or a private chat). When a user logs in, he will send a message from the client side indicating he is online, the server will then put that user in the "Lobby" room which contains every user that is online currently. From that point forward, the server will either broadcast chat messages or act accordingly with the commands sent from users from the client side.

The server and clients communicate using messages with this format:
command*<splitter>****data_1****<splitter>****data_2****.....*

- *<splitter>*: a string used to separate and extract the command and data. We use "*<21127089>*" as our splitter.
- ***data_ith:*** data related to the messages. For example, this could be a chat room ID, username, message content,...
- ***command:*** There is a predefined set of commands that a client can use to achieve real-time functionality and *efficiently* update the UI with real-time data. These commands include
    - "*login*" or "*logout*": to notify other users that this user is online/offline in a real-time manner to efficiently update the UI accordingly.
    - "*joinRoom*": to join a chat room.
    - "*message*": to send messages to a chatroom in real-time.
    - "*block*" or "*unblock*": to implement the block/unlock feature in a real-time manner.
    - "*addFriend*" or "*unfriend*": to add friend/unfriend in real-time.
    - "*createGroup*", "*editGroupName*", "*addMember*", "*updateMemberList*", and "*removedFromGroup*": to implement group chat related functionalities in real time.
- An example of a formatted message to remove a member from a group chat:

"removedFromGroup<21127089>chatRoomId<21127089>username"

2. Repository design pattern

a. **Purpose**: The Repository pattern is used to create an abstraction layer between the data access layer and the business logic layer of an application. It's essentially a middleman that handles all data access.

b. **Application Workflow:** In our scenario, the Repository pattern is seamlessly integrated into the application's architecture. Here, the business logic resides within the UI components, which are tasked with processing user actions. These UI components not only apply relevant business rules but also seamlessly interact with the repositories. The repositories, in turn, handle the retrieval or storage of data, establishing a cohesive workflow within the application.

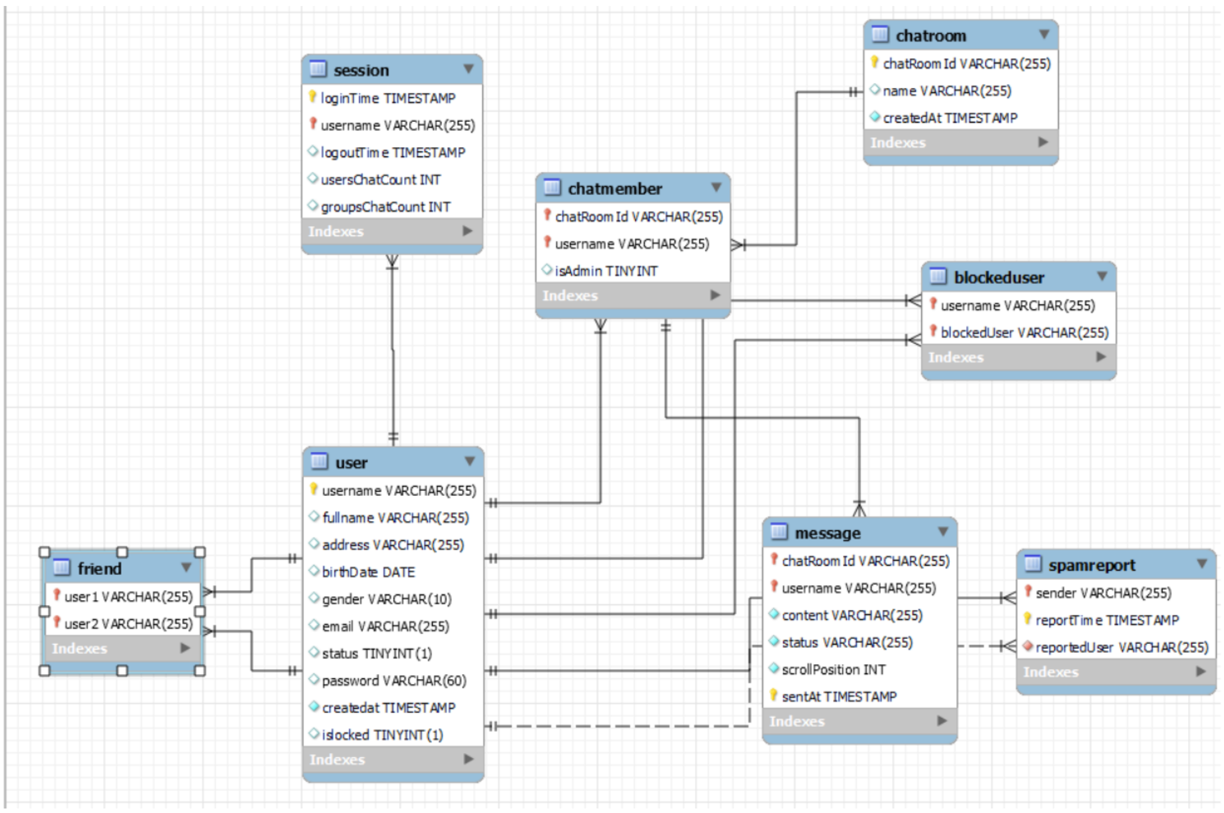
c. **Repositories**:

- UserRepository: Manages user data interactions in the application. Facilitates username and email checks, user authentication, creation, retrieval, and search functionalities. Handles online status, password updates, and account lock status checks.
- SpamReportRepository: Manages database interactions related to spam reports within the application. Provides methods for retrieving all spam reports, obtaining reports within a date range, inserting new reports, and removing specific reports.
- SessionRepository: Manages database interactions related to user sessions within the application. Provides methods for obtaining statistics on app opens, chat interactions, and group interactions. Retrieves user activities within a specified date range, sessions for a particular user or all users, and sessions for a specific year. Also includes methods to start a new user session.
- GroupAdminRepository: Manages database interactions related to group administrators within the application. Provides a method to retrieve a list of users who are administrators for a specific chat room.
- FriendRepository: Manages database interactions related to friends within the application. Provides functions for adding friends, retrieving friend lists with online status, checking friendship status, and unfriending. Additionally, it creates a private chat room upon adding a friend.
- ChatRoomRepository: Manages database interactions related to chat rooms. Provide functions for creating a private or a group chat, updating the name of a chatroom, retrieving all chatrooms that have this user as a member,..
- ChatMemberRepository: Manages database interactions related to chat members in a group chat. Provide functions for adding/removing a new member to the group, assigning admins, retrieving chat members,...

3. Database

Our Java application employs a MySQL database, and a comprehensive description of it can be found in Section III.

III. DATABASE DESIGN



1. Entities

USER			
#	Fields	Constraints	Descriptions
1	username	- Primary key - varchar(255)	User account name
2	fullName	- varchar(255)	Full name of the user
3	address	- varchar(255)	Address of the user
4	birthdate	- date	User's date of birth
5	gender	- varchar(10) - check(gender = ‘male’ or gender = ‘female’)	Gender of the user
6	email	- varchar(255)	Email address of the user

7	status	- tinyint	User's activity status
8	password	- varchar(60)	User account password
9	createdAt	- timestamp	Date when the account was created
10	isLocked	- tinyint	Check if the user account is locked or not

CHATROOM			
#	Fields	Constraints	Descriptions
1	chatRoomId	- Primary key - varchar(255) - not null	Chat room ID
2	name	- varchar(255) - default null	Chat room name, empty string indicating a private chat
3	createdAt	- timestamp - not null	Date when the chat room was created

CHATMEMBER			
#	Fields	Constraints	Descriptions
1	chatRoomId	- Primary key - Foreign key references chatroom(chatRoomId) - varchar(255) - not null	Chat room ID
2	username	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username of the member participating in this chat room
3	isAdmin	- tinyint - default ‘0’	Check if the member is an admin of the group or not.

FRIEND			
#	Fields	Constraints	Descriptions
1	user1	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username 1
2	user2	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username 2

BLOCKEDUSER			
#	Fields	Constraints	Descriptions
1	username	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username of the user who instigated block action
2	blockedUser	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username of the user who was blocked

SESSION			
#	Fields	Constraints	Descriptions
1	loginTime	- Primary key - timestamp - not null	Login time

2	username	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username associated with this session
4	logoutTime	- timestamp	Logout time
5	usersChatCount	- int - default ‘0’	Number of user accounts messaged by the user in the session
6	groupsChatCount	- int - default ‘0’	Number of chat groups user messaged during the session

MESSAGE			
#	Fields	Constraints	Descriptions
1	chatRoomId	- Primary key - Foreign key references chatroom(chatRoomId) - varchar(255) - not null	Group chat ID
2	username	- Primary key - Foreign key references user(username) - varchar(255) - not null	Username associated with the message
3	content	- varchar(255) - not null	Message content
4	status	- varchar(255) - not null	Status has one of the following values: sent/not sent/seen
5	scrollPosition	- int - not null	Determine the scroll position of the message in a chatroom
6	sentAt	- Primary key - timestamp - not null	Time when the user sent the message

SPAMREPORT			
#	Fields	Constraints	Descriptions
1	sender	- Primary key - Foreign key references user(username) - varchar(255) - not null	Name of user submitting the report
2	reportedTime	- Primary key - timestamp - not null	Time when the user created the report
3	reportedUser	- Foreign key references user(username) - varchar(255) - not null	User account name being reported

2. Relationships

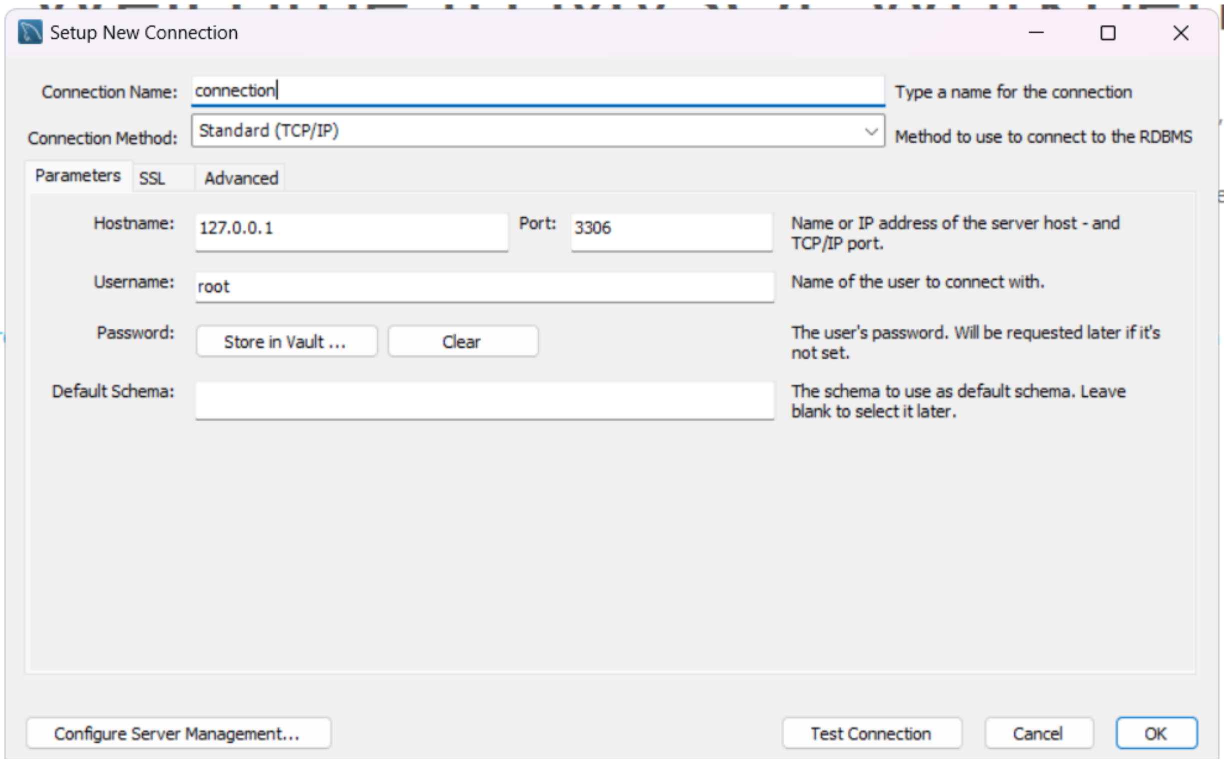
- **User - ChatMember:** A one-to-many relationship exists between users and chat members. A user can be a member of multiple chat rooms, but each chat member can only be associated with one user.
- **User - Friend:** A one-to-many relationship exists between users and friends. A user can have multiple friendships, but each friendship is associated with one user.
- **User - BlockedUser:** A one-to-many relationship exists between users and blocked users. A user can block multiple users, but each user - blocked user pair can only be associated with one user.
- **User - Session:** A one-to-many relationship exists between users and sessions. A user can have multiple sessions, but each session can only be associated with one user.
- **User - SpamReport:** A one-to-many relationship exists between users and spam reports. A user can make multiple spam reports, but each report can only be created by one user.
- **ChatMember - ChatRoom:** A may-to-one relationship exists between char rooms and chat members. A chat member can only be a part of one chat room, but a chat room can have multiple chat members

- **ChatMember - Message:** A one-to-many relationship exists between chat members and messages. A chat member can send multiple messages, but each message can only be created by one chat member.

III. INSTRUCTIONS FOR USE

1. Set up

a. Set up connection


Setup New Connection

Connection Name: connection — Type a name for the connection

Connection Method: Standard (TCP/IP) — Method to use to connect to the RDBMS

Hostname: 127.0.0.1 Port: 3306 — Name or IP address of the server host - and TCP/IP port.

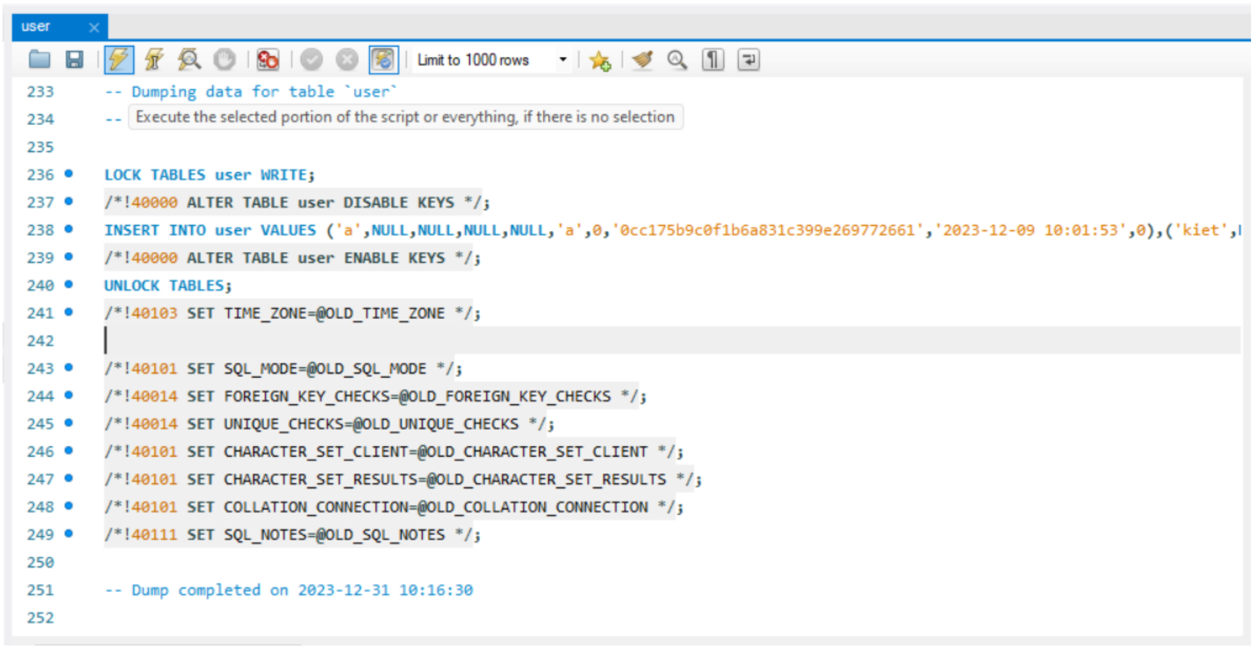
Username: root — Name of the user to connect with.

Password: — The user's password. Will be requested later if it's not set.

Default Schema: — The schema to use as default schema. Leave blank to select it later.

b. Run the database script

Open database script *db.sql* in MySQL Workbench and click on the lightning bolt icon to execute the script


user
Limit to 1000 rows
Execute the selected portion of the script or everything, if there is no selection

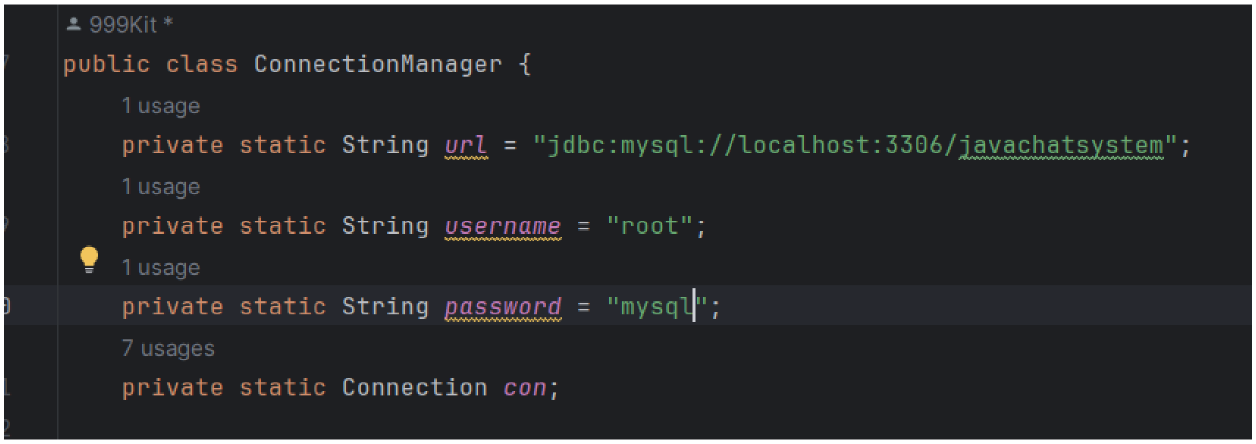
```sql
-- Dumping data for table `user`
--

LOCK TABLES user WRITE;
/*!40000 ALTER TABLE user DISABLE KEYS */;
INSERT INTO user VALUES ('a',NULL,NULL,NULL,NULL,'a',0,'0cc175b9c0f1b6a831c399e269772661','2023-12-09 10:01:53',0),('kiet',
/*!40000 ALTER TABLE user ENABLE KEYS */;
UNLOCK TABLES;
/*!40103 SET TIME_ZONE=@OLD_TIME_ZONE */;

/*!40101 SET SQL_MODE=@OLD_SQL_MODE */;
/*!40014 SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS */;
/*!40014 SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
/*!40101 SET CHARACTER_SET_RESULTS=@OLD_CHARACTER_SET_RESULTS */;
/*!40101 SET COLLATION_CONNECTION=@OLD_COLLATION_CONNECTION */;
/*!40111 SET SQL_NOTES=@OLD_SQL_NOTES */;

-- Dump completed on 2023-12-31 10:16:30
```

c. Set up the connection in code



```java
public class ConnectionManager {
    private static String url = "jdbc:mysql://localhost:3306/javachatsystem";
    private static String username = "root";
    private static String password = "mysql";
    private static Connection con;
```

Go to ConnectionManager.java. Change username and password according to your database connection.

d. Build the project

In terminal, run: .\mvnw *clean*. Then, run: .\mvnw *package* (This command automatically runs the server)

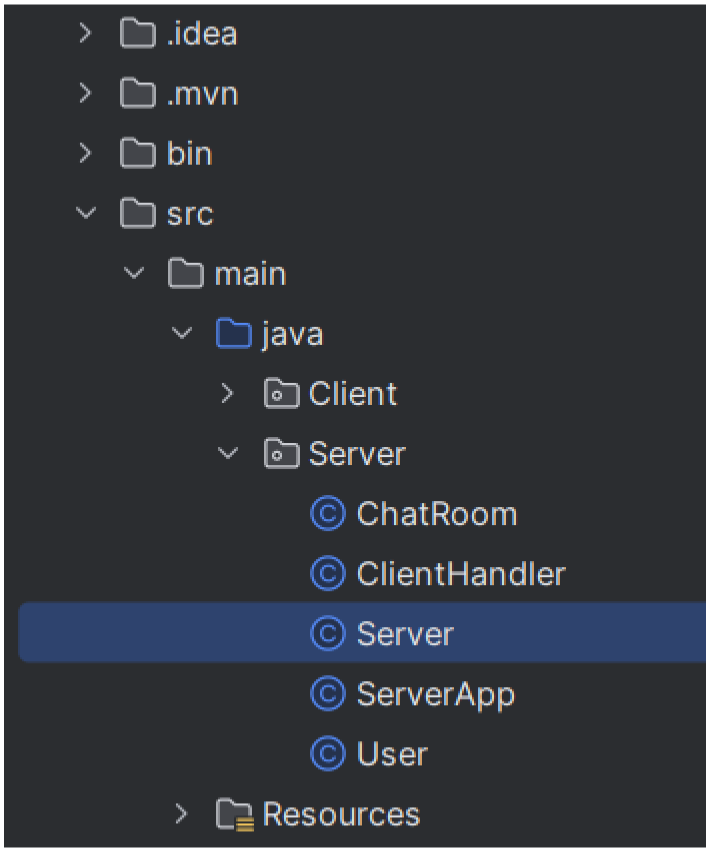
2. Run User

2.1. Run using Maven:

- (Only run this if the server is not already running) Run the server: .\mvnw exec:java
- Run user: .\mvnw exec:java@client-user

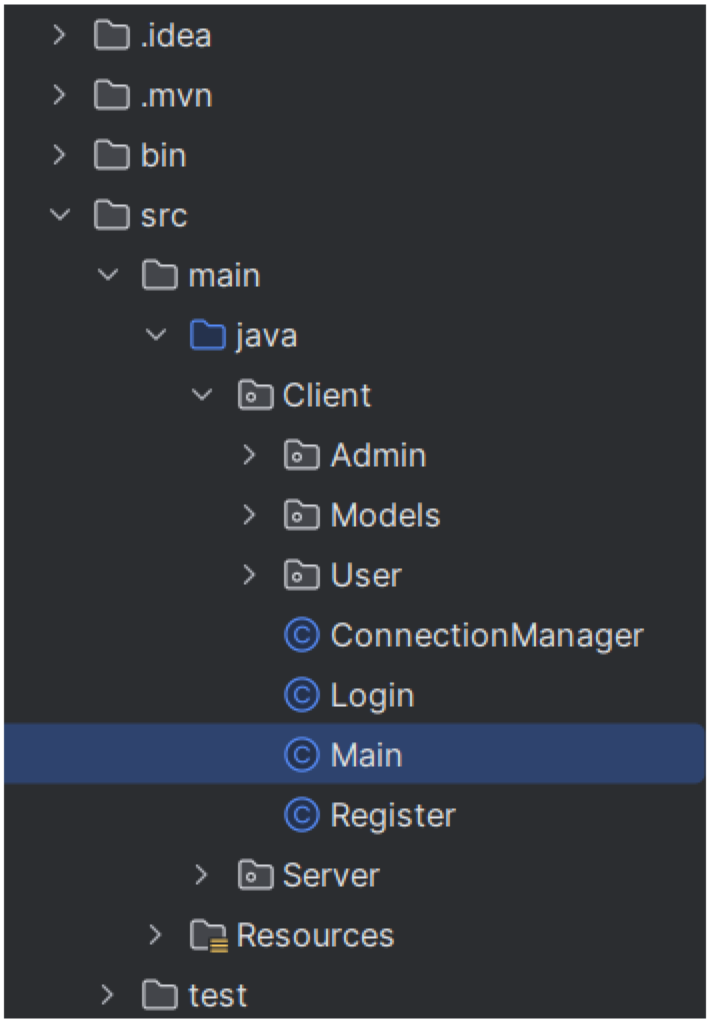
2.2. Run using file:

- Step 1: Run the server by running this file “Server”:



```
> .idea
> .mvn
> bin
v src
  v main
    v java
      > Client
      v Server
          ChatRoom
          ClientHandler
          Server
          ServerApp
          User
    > Resources
```

- Step 2: Run the app by running this file “Main”:



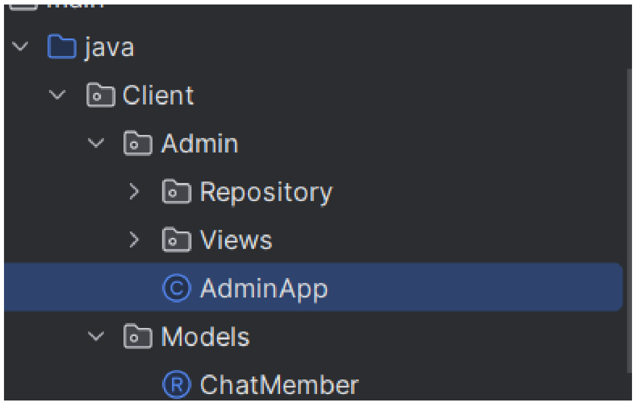
3. Run Admin

3.1. Run using Maven:

- Run admin: .\mvnw exec:java@client-admin

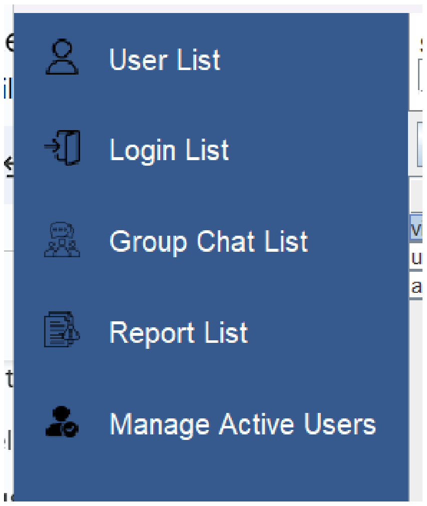
3.2. Run using file:

Run AdminApp.java file:



Refresh data for Admin

To refresh any screen simply click on its according icon on the sidebar:



For example, to refresh the User List screen, simply press the User List icon on the sidebar.