Project Report: Chat Application

Andre Lungu, Alex Wu, Dylan Zhou, Levy Lin, Yi Chen

Table of Contents

I. Back End
   A. Project Structure
   B. Web Server
   C. JPA Repositories
II. Front End
   A. Login/Registration
   B. Chat Window
III. Documentation
   A. User Manual
   B. Testing Manual
   C. Presentation
IV. Results

I. Back End

I.A. Project Structure Alexander Wu

The project was set up using Gradle as a multi-module project. We used Gradle to easily add dependencies to Spring Boot and Javafx without having to download Jar files and write a complicated script. The main module (chat) housed the server and various run scripts to interact with both modules such as generating Javadoc for both the app and the server. Inside this module also had the database for the server which will be touched upon in section 2A. The nested module (chatapp) was used to house the UI and any logic needed for it. With this multi-module system, we can write 2 separate sections of code in the same project; one for the front-end and one for the back-end.

I.B. Web Server Alexander Wu

We used Spring Boot to create the web server. We used this server as a place where the applications can send messages without having to have a peer-to-peer connection. With this server, we had various HTTP requests. These let the applications do various things such as create new channels for messages, sign up for new accounts, and get various user information to display to the user. We also had a websocket for users to communicate with each other. Because of these websockets, users can send messages through them and other users will automatically be notified when a message is received. With Spring Boot, we used their library JPA to store the data. This removes the majority of the boilerplate of having to set up a server and connect to it using JDBC. This also makes it easier to test as we do not have to set up a new database to test and can use an in-memory database, which we decided to use the Hibernate H2 database. Using JPA still allowed us to create and run SQL queries when necessary but mostly relied on annotations to create the schemas.

I.C. JPA Repositories Levy Lin

By leveraging Spring Boot's JPA interface which expands upon our initial Spring Boot infrastructure, we can implement a simpler and more intuitive implementation of a database. The primary goal of our application is to construct a user-login mechanism, a user-to-user communication, and a user-to-group communication. To achieve these goals we created four JPA models and JPA Repositories. A JPA model is similar to a database schema, where the model explicitly states the data types it contains and their roles. Additionally, a JPA model contains a constructor, getters, and setters, allowing for the creation of new table rows and queries or updates to the table row. A JPA Repository constructs and represents an actual table based on a JPA model. In a JPA Repository we can specify our own query functions to gather data from the table and add new table rows based on incoming HTTPS requests.

Using the JPA library, we implement the user-login mechanism with a JPA model and repository, named User. The User model contains the username and password of a specific user. While the repository queries if a user exists in the table. Furthermore, another JPA model and repository named Message, is used to implement user-to-user communication. The purpose of the model is to contain the message contexts, as well as the meta-data of the sender, receiver, and timestamp. Finally, user-to-group communication is implemented with two JPA models and

repositories. The first model and repository is named GroupChats, representing the existing or newly created groups. GroupChat acts as a wrapper of User where messages sent to a GroupChat are forwarded to all other group members. The second model and repository stores the all User and GroupChat relations.

II. Front End

II.A. Login/Register Dylan Zhou

The user should not be able to login without registering for an account first. The server and database stores all of the accounts registered. Each time a user wants to register for an account, it checks for whether there is an account that matches the same username. The username must be unique per user and there is no current way to reset a password. When the user attempts to register, the user must enter a valid username and password, that is, they cannot be empty fields or empty strings. The registering step utilizes a POST and GET request to the server which then queries to find a match for the username for the GET request and returns a boolean determining if the username is in the database or not. If the username is already taken, then there will be an error dialog that says the username has been already taken and exists. If successful, then the registration has been completed.

For the POST request, the server will create a new user object in the server that stores both the username and the password corresponding to the username. Once the user has successfully created an account, the user can now login using the same credentials used for the registering. Logging in will send a GET request to the server which returns a true or false determining whether or not the username and password exists somewhere in the database. If false, then there will be an error dialog that says the username or password was incorrect. If true, then the user should be brought to the chat page that displays all the chats.

II.B. Chat Window Andre Lungu

The chat window was created using FXML and allows a user to view their current chats and messages within each chat, create new chats, add users to chats, and send/receive messages to/from a chat. The chat window functionality was fully integrated with the web server; creating group chats, adding users to groups, and loading a user's groups and messages on app startup is done via HTTP requests to our spring boot server.

Sending and receiving messages is done via websocket connection, and sent messages are stored in our JPA repositories to be loaded upon potentially restarting or logging back in. A websocket client endpoint class was created using Jakarta EE websockets in order to connect to and communicate with the web socket server. This asynchronous connection allows for sending and receiving messages in real time, allowing for seamless messaging between users.

III. Documentation

III.A. User Manual Yi

The user manual for the application provides a comprehensive guide to help users navigate and utilize all features of the platform. It includes detailed descriptions of the application's functionalities, starting with account registration and sign-in procedures. Users will find step-by-step instructions on how to create individual and group chats, add users to group conversations, and manage their interactions within the app. Additionally, the manual offers a clear explanation of possible error messages, helping users understand and troubleshoot issues more easily. This guide is designed to ensure that both new users can efficiently use the chat application with minimal difficulty.

III.B. Testing Manual Yi

The testing manual for our chat application outlines a comprehensive test plan and strategy. A manual testing approach is used to evaluate the GUI functionality of the application. Each scenario detailed in the test plan is manually executed by the tester to ensure that the application functions as expected. This includes verifying that error messages for invalid inputs are displayed correctly and that the chat functionality is updated in real time as expected. To support this process, a comprehensive spreadsheet is provided at the end of the document that lists all the test scenarios and cases, ensuring that the testing approach is well organized. Additionally, all public functions in the source code are thoroughly documented with Javadoc comments, ensuring clarity and ease of understanding for developers.

III.C. Presentation Yi

During the Friday lecture on August 16th, our group successfully completed our in-person presentation. Each group member collaborated to create a comprehensive set of slides that detailed the tech stack used in our project and the specific contributions each of us made. We also conducted a brief live demonstration, showcasing how to use our chat application, which allowed the audience to see our work in action.

IV. Results

(Andre)

The result of this project was a chat application where multiple users can chat in real time over our server and websocket connections. The application is robust, fully functional, and well-designed, displaying new messages and handling chats with ease.