Messaging Service Prototype

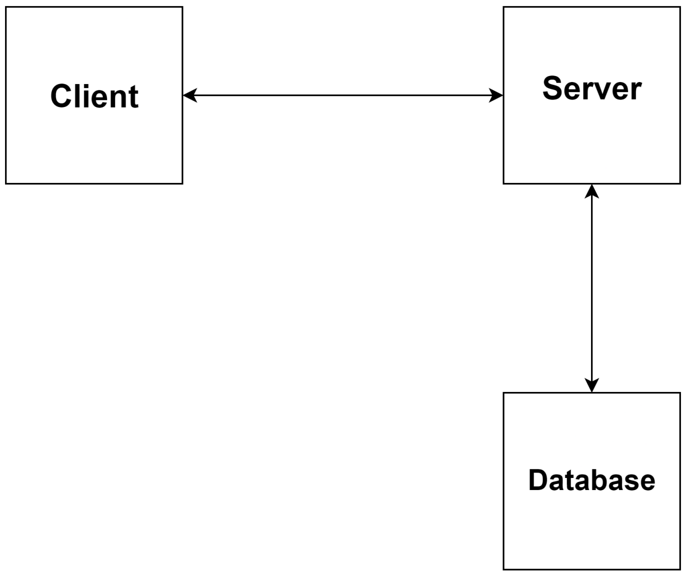
Overview

The Messaging Service Prototype is a real-time communication platform that allows users to register, log in, send and receive messages, create and join chat groups, and maintain chat history. The application uses WebSocket for real-time communication and MySQL as the database for storing user information and chat data.

System Architecture

The architecture of the messaging service consists of the following key components:

- Client-Side: The frontend application built using Next.js provides a user-friendly interface for interacting with the messaging service.
- Server-Side: A WebSocket server that handles client connections, user authentication, message routing, and database interactions.
- Database: MySQL database to store user credentials, chat history, groups, and group messages.



Components

Client-Side

- Next.js: Framework for building the frontend application, enabling server-side rendering and API routes.
- CSS: Used to style the components, providing a clean and intuitive user interface. CSS modules or styled-components can be employed for modular and reusable styling across components.

Server-Side

- Node.js: JavaScript runtime for building the WebSocket server.
- ws: WebSocket library for handling real-time communication.
- mysql2: Promise-based MySQL client for interacting with the MySQL database.

Database

- MySQL: Relational database management system for storing user data and chat history.

Libraries

Server-Side Dependencies

ws:

- Purpose: For handling WebSocket connections and real-time messaging.
- Installation: npm install ws

mysql2:

- Purpose: For interacting with the MySQL database, supporting promises for better async handling.
- Installation: npm install mysql2

Client-Side Dependencies

Next.js:

- Purpose: Framework for building server-rendered React applications.
- Installation: npm install next react react-dom

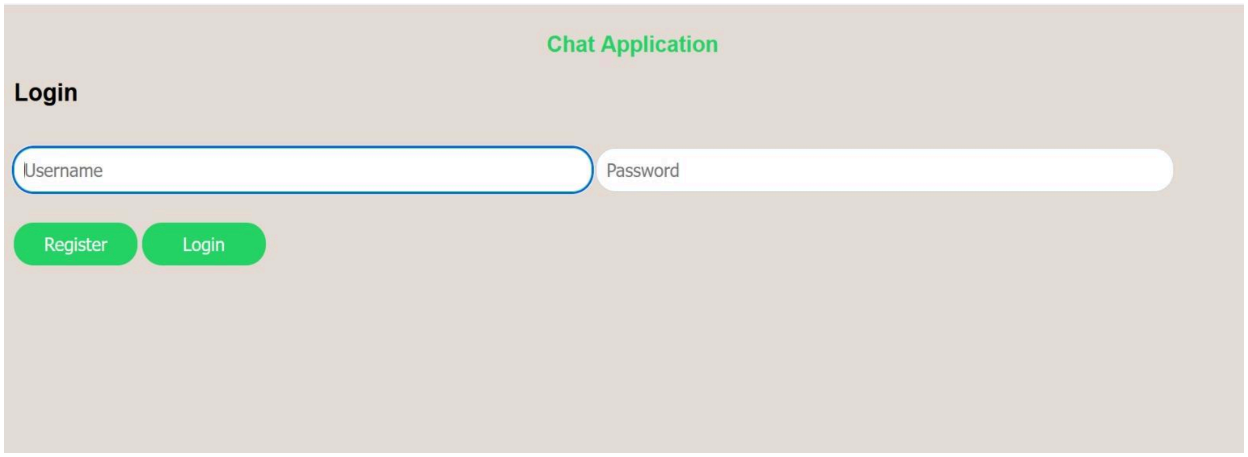
Database Schema

The database schema for the messaging service includes the following tables:

- users: Stores user information (username and password).
- chat_history: Stores direct messages between users.
- chat_groups: Stores group information (group names).
- group_members: Tracks which users belong to which groups.
- group_chat_history: Stores messages sent within groups.

User Interface

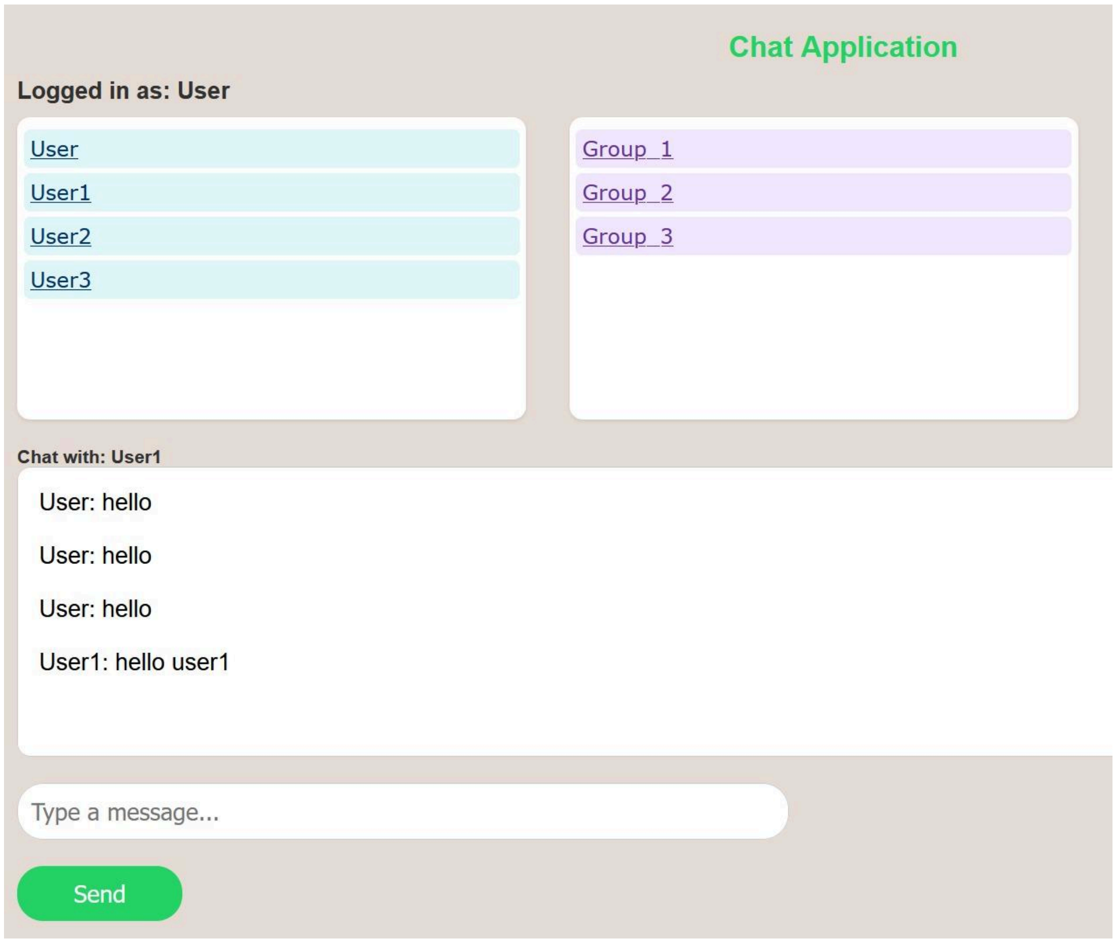
Register &Login



Dashboard

Registered Users

The left side panel shows all the registered users username and allows user to select users to chat



User: hello

User: hello

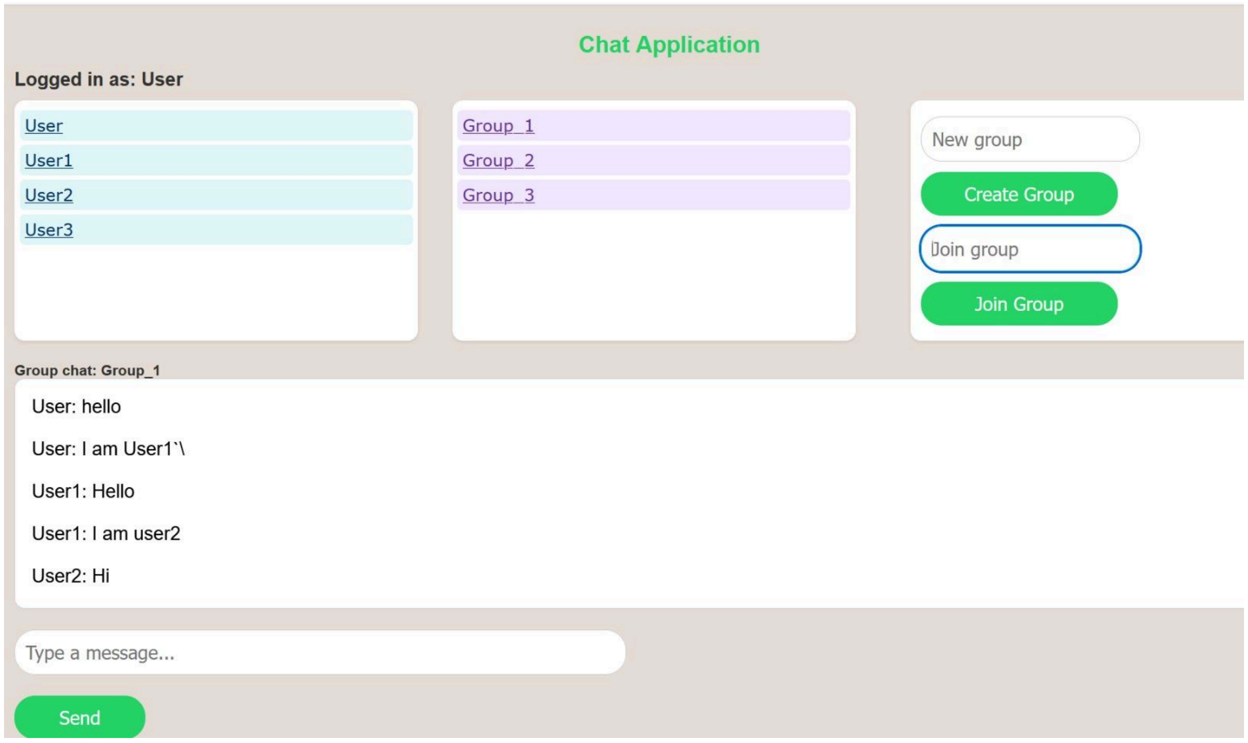
User: hello

User1: hello user1

Groups space

Lists all the groups that are present and by clicking on the group they can switch between groups.

- Create group: User can create a group by selecting create Group button
- Join Group: User can join a group by entering the group names



User: hello

User: I am User1`\

User1: Hello

User1: I am user2

User2: Hi

Chat Space

Displays the chat between the user to other user or group selected from the listing panel of group or users

Setup and Installation

Prerequisites

- Node.js (v14 or later)
- MySQL server
- Git

Dependencies

- Install Node.js Packages:
  Navigate to the project directory in your terminal.
  Run the following command to install dependencies:
  npm install

- Clone the Repository:
  Clone the repository to your local machine:
  git clone link

Database Setup

Create Database:

- Open your MySQL command-line interface or a MySQL client (MySQL Workbench).
- Execute the SQL schema provided to create the necessary tables in the messaging_service database.

Modify the Database Credentials:

Open server.js and update the hardcoded MySQL username and password.

server.js:

```
// Create a connection to the MySQL database
const db = await mysql.createConnection({
  host: 'localhost',
  user: 'username', // Replace with your MySQL username
  password: 'password', // Replace with your MySQL password
  database: 'messaging_service'
});
```

Run the Server

Run the command in the terminal

- node server.js

Access the Client

- Open client/index.html in your browser.

Conclusion

The Messaging Service Prototype provides a functional platform for users to communicate through direct messages and group chats. The system is designed to be scalable and extensible, allowing for future enhancements and additional features.