Development of a Real-Time Fullstack Chat Web-Application

Chheng Bunheang
Bachelor of Computer Science
Cambodia Academy of Digital Technology (CADT)
Phnom Penh, Cambodia
Email: bunheang.chheng@student.cadt.edu.kh

Kheang Kimang
Supervisor
Cambodia Academy of Digital Technology (CADT)
Phnom Penh, Cambodia
Email: kimang.kheang@cadt.edu.kh

Abstract—**This proposal outlines the development of a real-world fullstack web-based chat application using modern JavaScript technologies such as React, Express.js, Node.js, and MySQL. The system supports real-time messaging, user authentication, media uploads, and persistent message storage using Sequelize ORM. This project focuses on utilizing a relational database (MySQL) to ensure better alignment with structured data and scalable relationships.**

Index Terms—**Chat App, Fullstack Web Development, MySQL, Sequelize, Express.js, React.js, Node.js, Real-Time Messaging, REST API.**

I. Introduction

Messaging applications are essential in modern digital communication. This project proposes the implementation of a fullstack chat application leveraging React for the frontend, Node.js with Express for the backend, and MySQL for the relational database layer. It aims to provide a smooth user experience with features such as authentication, real-time chat functionality, and media support.

II. Motivation

While many chat applications exist today, most are built using NoSQL databases, which may not suit all data structures—especially where data normalization and relationships matter. This project is motivated by the need to build a real-time messaging platform using a relational database (MySQL), showcasing how structured data management can scale effectively in a fullstack real-time environment. The project also serves as a hands-on exploration of modern web technologies, contributing to both academic learning and practical application in web development.

III. Objectives

- Develop a responsive frontend using React and Tailwind CSS.
- Implement backend services using Express and Node.js.
- Use MySQL as the relational database with Sequelize ORM.
- Enable real-time communication with support for WebSockets.
- Ensure secure user authentication and session management using JWT.
- Integrate Cloudinary for media file storage.

IV. Main Features

The application is designed with the following key features:

- **User Authentication:** Secure login and registration system using JWT-based tokens.
- **Real-time Messaging:** Real-time communication between users using Socket.IO.
- **Persistent Chat History:** All messages are stored in a MySQL database for future retrieval.
- **User Profiles:** Users can update their personal information and profile pictures.
- **Image Messaging:** Ability to send images, which are uploaded and served via Cloudinary.
- **Responsive UI:** Built with Tailwind CSS for a fast, mobile-friendly interface.

V. Technology Stack

The fullstack chat application is developed using the following technologies:

- **Frontend:** React.js, Vite, Zustand, Tailwind CSS
- **Backend:** Node.js, Express.js, Socket.IO
- **Database:** MySQL using Sequelize ORM
- **Authentication:** JSON Web Tokens (JWT)
- **Image Storage:** Cloudinary API for media hosting
- **Development Tools:** Git, GitHub, Postman

VI. System Architecture

The application follows a standard MVC (Model-View-Controller) architecture. The backend exposes RESTful API endpoints for user authentication, message handling, and profile updates. Middleware ensures authenticated access, while Sequelize handles model relations and data persistence.

A. Frontend

The frontend is built using React and Vite, styled with Tailwind CSS. It communicates with the backend using HTTP requests and handles JWT-based cookie sessions.

B. Backend

The backend uses Express.js to manage routes and controllers. Sequelize connects to a MySQL database where user data and messages are stored. Cloudinary is used for image uploads.

C. Sequence Diagram

The following diagram illustrates the system's flow from user authentication to real-time communication.

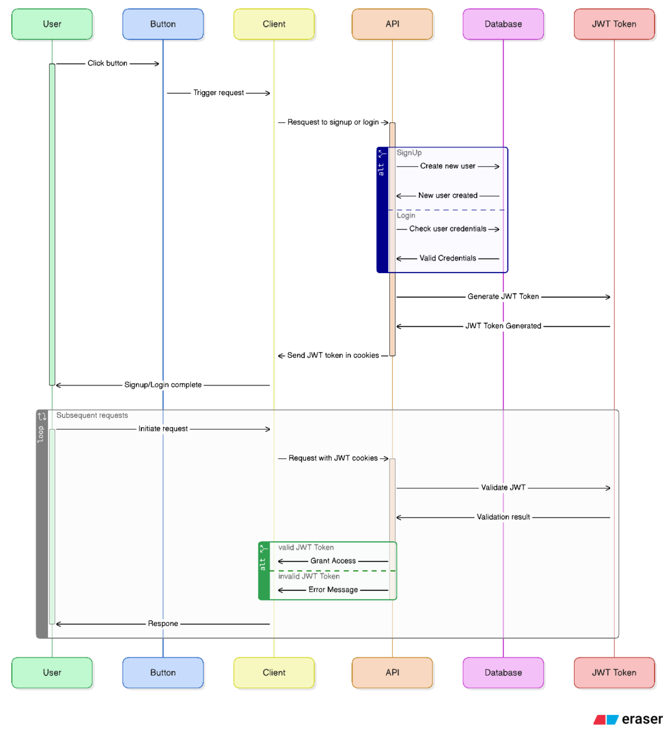

Fig. 1. Sequence Diagram of the Chat Authentication and Messaging Flow

VII. Database Design

The database uses relational models:

- **Users**: Stores user credentials and profile data.
- **Messages**: Stores text and image messages with references to sender and receiver.

Foreign keys and associations are managed through Sequelize's ORM capabilities.

VIII. Implementation Plan

1) Set up project structure and environment.
2) Configure Sequelize and define database models.
3) Develop authentication system using JWT.
4) Build message APIs and connect frontend.
5) Implement profile update and Cloudinary image upload.
6) Test and refine the application.

IX. Future Plans

While the current project implements a functional MVP (Minimum Viable Product), there are several potential extensions:

- Group chat support with admin/moderator roles
- Message deletion and editing
- Online/offline presence indicators
- Push notifications for new messages
- End-to-end encryption for sensitive communications
- Mobile version with React Native
- Dark mode toggle and theme customization

X. Conclusion

The proposed chat application demonstrates practical implementation of fullstack technologies and relational databases in real-time communication. By building on and improving an existing resource, this project provides a robust foundation for scalable chat applications and further feature integration such as Socket.IO and group messaging.

References

[1] Eraser.io – Diagram and documentation tool for developers. https://eraser.io

[2] GeeksforGeeks, "Introduction to MySQL". https://www.geeksforgeeks.org/mysql-introduction/

[3] GeeksforGeeks, "Introduction to Express.js". https://www.geeksforgeeks.org/express-js/

[4] GeeksforGeeks, "Node.js Introduction". https://www.geeksforgeeks.org/introduction-to-node-js/

[5] GeeksforGeeks, "ReactJS Introduction". https://www.geeksforgeeks.org/reactjs-introduction/