PROJECT REPORT

Abstract

This project focuses on developing a messaging service prototype that incorporates key communication features for modern chat applications. The prototype includes user registration and authentication, allowing secure user access and profile management. Users can engage in private messaging, with the ability to send and receive text messages seamlessly. The system also supports group chat functionality, enabling users to create or join multiple conversation threads with others. To ensure a real-time experience, the prototype implements instant message updates, using Web Sockets, Socket.IO or other live communication technologies to deliver messages immediately. This prototype serves as the foundation for scalable and secure messaging systems suitable for both individual and group interactions in various applications.

Introduction

In the age of digital communication, messaging services have become a cornerstone of online interaction. From personal conversations to collaborative group discussions, users demand seamless, real-time communication experiences. This project presents a messaging service prototype designed to meet these needs, featuring essential functionalities that modern users expect. The system enables secure user registration and authentication, ensuring that only verified users can participate in chats. Beyond one-on-one messaging, the prototype also supports group chats, allowing multiple users to communicate within a single thread. Real-time updates are integrated to provide a dynamic, immediate messaging experience, ensuring messages are delivered and displayed instantly.

This prototype is designed to serve as a scalable foundation for more comprehensive messaging systems, offering flexibility in its application across various domains such as social media, business communication, and collaborative platforms. By focusing on user experience, security, and real-time responsiveness, this project aims to highlight key elements that are critical to the functionality of any successful messaging service.

- **Importance of Real-time communication**
  Real-time communication (RTC) significantly enhances user engagement, allowing immediate feedback and message exchanges. According to Olsson and Väänänen (2016), real-time updates increase user satisfaction and engagement in messaging apps.
- **Technologies Enabling RTC**:
- **WebSockets**: WebSockets provide full-duplex communication channels over a single TCP connection, making it possible to send and receive messages instantly.
- **Socket.IO**: This library allows for real-time, bidirectional communication between web clients and servers, which is crucial for real-time messaging systems.

Requirement Analysis

Functional Requirements

- **User Registration and Login**
  User should be able to create accounts and log in securely.
  Real time single and group chat

Non-functional Requirement

- **Usability and User Experience:**
  The user interface should be intuitive and easy to navigate.
  Response time for displaying chats should be fast.
- **Security:**
  User authentication and authorization mechanisms should be implemented.
  Sensitive information such as password should be encrypted.
- **Performance:**
  The application should be responsive and handle concurrent user requests effectively
- **Compatibility:**
  The application should be compatible with various devices and web browsers.
- **Data Integrity:**
  Data related to user or chats should be stored securely and maintained with proper backup mechanism

System Architecture

The architecture aims to ensure scalability, maintainability, and efficient data flow within the application.

High-Level Architecture

- **Client Tier:**
  Web Browser: User access the application using a web browser on their devices.
  User Interface: The user interface is developed using React + Vite, Material -UI and JavaScript, CSS. It provides an intuitive interface to interact with the application.
- **Application Tier:**
  Web Server: Handles user requests from the web browser and serves HTML, CSS and JavaScript files.
  Application Logic: Manages user authentication provide Admin access group chat, chat between two users.
  API Layer: Provides a RESTful API for communication between the front end and back end.

- **Data Tier:**
  Database Management System:
  In this project a Non-SQL data is used (MongoDB).

That stores the user information and chats group information and other application-related data.
Cloudinary is used to store non-textual data media assets like images, videos, audio files, and documents.

Data Flow

- **User Registration and Login:**
  User submits registration details.
  Web server process the request and stores user data in the database.
  User logs in using credentials.
  The web server validates the credentials and grant access.
- **Group Creation and Chat:**
  Users can create group with their friends.
  Groups information stored into database.
  User can initiate a chat by sending a friend request to another user after accepting the friend request by other user they are updated into database in friend collection as a friend.

Technologies Used:

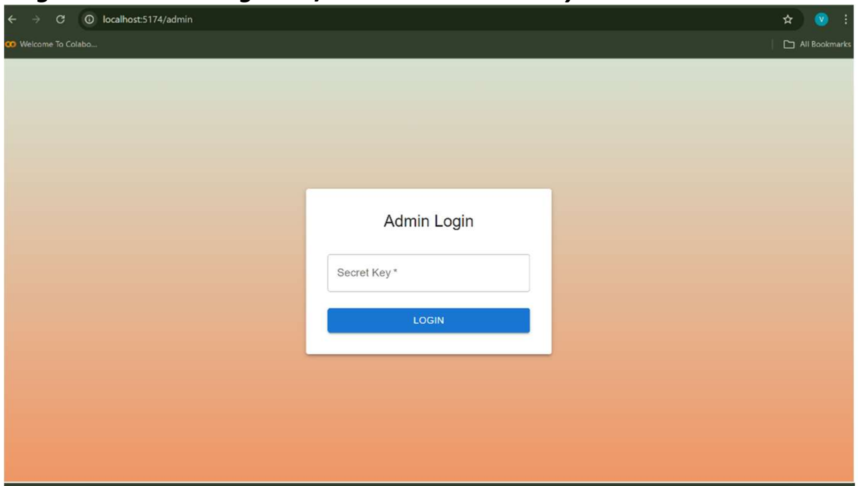
- **JavaScript**: For the core functionality, you have used JavaScript for both the frontend and backend development of the project.
- **React**: A JavaScript library for building user interfaces, especially for single-page applications, integrated with Vite for faster development.
- **Express**: A backend framework used in your Node.js server to handle routes and server-side logic.
- **MUI (Material UI)**: A React component library for designing the UI components, which follows Material Design principles.
- **MongoDB**: A NoSQL database used to store and manage application's data in a flexible, document-based structure. It supports high performance and scalability, ideal for modern web applications.
- **Cloudinary**: A cloud service used to store, manage, and deliver non-text data, such as images and videos, efficiently. It handles media uploads, optimizations, and transformations.
- **RESTful APIs**: facilitate communication between the frontend (React + Vite) and backend (Express), allowing CRUD operations over HTTP. This ensured efficient, stateless interactions between the client, server, and external services like MongoDB and Cloudinary
- **Development Tools**: Visual Studio Code, Postman.

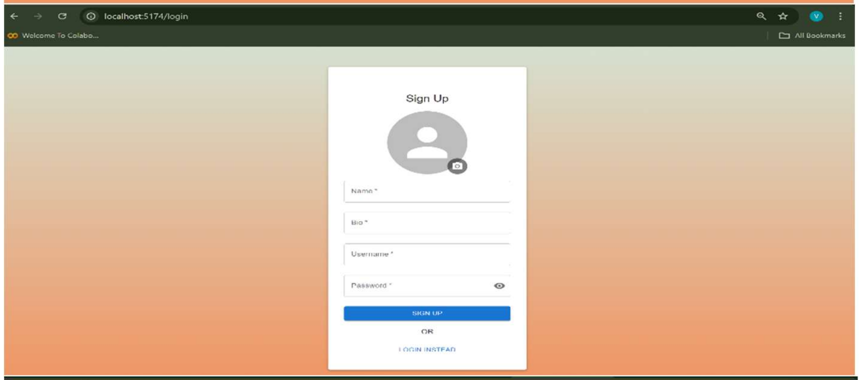
Github link :https://github.com/vishal230404/ChatHub

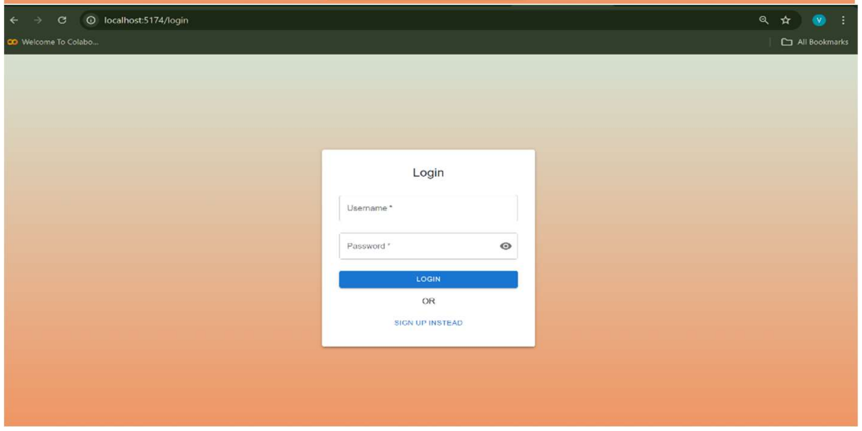
Steps To setup project:

1. fork github repo.
2. Navigate to client folder
3. In cmd run npm install
4. To start the front-end run npm start
5. Navigate to server folder
6. In cmd run npm install
7. Edit the File name .example.env to .env
8. Provide Mongodb URI (https://www.mongodb.com/cloud/atlas/register) and Cloudnary secrets (https://cloudinary.com/) to setup .env
9. To start server run npm start

To get Admin access go to "/admin" URL manually and Provide admin Secret key.


localhost:5174/admin
Welcome To Colabo...
All Bookmarks
Admin Login
Secret Key *
LOGIN


localhost:5174/login
Welcome To Colabo...
All Bookmarks
Sign Up
Name *
Bio *
Username *
Password *
SIGN UP
OR
LOGIN INSTEAD


localhost:5174/login
Welcome To Colabo...
All Bookmarks
Login
Username *
Password *
LOGIN
OR
SIGN UP INSTEAD