# **Project Title:** Fullstack Chat Application Using MERN

**Author:** Shriya Tiwari

**Date:** 20/09/2024

## 1. Introduction

- **Overview:**
  This project is a real-time chat application built using the MERN (MongoDB, Express.js, React.js, Node.js) stack. It allows users to register, login, and engage in real-time messaging using WebSocket technology. The application provides a responsive chat interface with real-time message updates and user presence detection.
- **Objective:**
  The objective of this project is to create a scalable and responsive chat platform that integrates secure authentication, WebSocket-based real-time messaging, and a user-friendly UI.

## 2. System Architecture

**Frontend:** React.js

- Handles the UI for registration, login, chat, and user presence.
- Manages state through React hooks.
- Communicates with the backend API and WebSocket server.

**Backend:** Node.js/Express.js

- Provides RESTful API endpoints for user authentication, registration, and login using JWTs.
- Manages real-time messaging and media-sharing using WebSockets.

**Database:** MongoDB

- Stores user information and chat history.
- Ensures fast retrieval of messages and user sessions.

**Real-Time Messaging:** WebSockets

- Enables real-time communication between users.
- Updates the chat interface dynamically when a new message is sent or a user status changes.

## 3. Key Components

- **Authentication:**
  *JWT-based authentication* ensures secure user sessions. Tokens are generated on user login and stored in cookies to maintain session persistence.
- **Chat Interface:**
  Built using **Tailwind CSS** for responsive design. The UI is divided into two sections: active users on the left and the conversation view on the right. An input box is present at the bottom for sending messages and a file sharing button to share files and media.
- **Real-Time Communication:**
  *WebSockets* are used for handling real-time messaging. The server listens for messages and pushes updates to clients, ensuring instant delivery of messages and status updates.
- **Responsive UI:**
  Tailwind CSS is used for a mobile-first, flexible layout that adapts to various screen sizes. The design supports two columns: one for user lists and the other for chat messages.

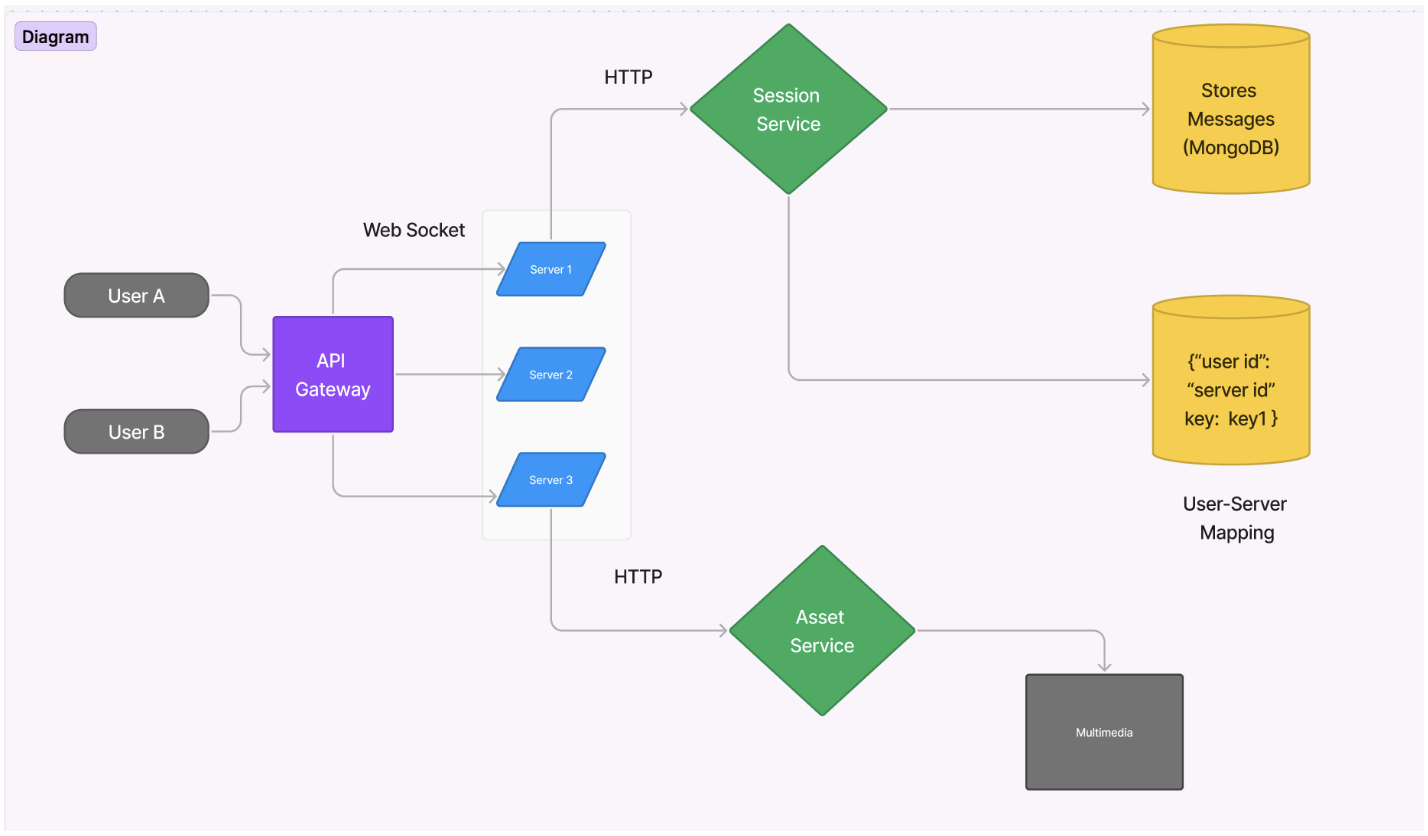
## 4. Technology Stack

- **Frontend:** React.js, Tailwind CSS
- **Backend:** Node.js, Express.js
- **Database:** MongoDB (NoSQL database)
- **Real-Time Messaging:** WebSocket (ws library)
- **Authentication:** JWT (JSON Web Tokens)
- **Version Control:** Git & GitHub

## 5. Data Flow Diagram

The following is a brief description of how data flows through the system:

1. **Client Registration/Login:**
   Users register and log in, sending credentials to the Express.js backend.
2. **Authentication:**
   The backend validates credentials and returns a JWT, which is stored in cookies for session persistence.
3. **WebSocket Connection:**
   After successful login, the client establishes a WebSocket connection with the server for real-time chat.
4. **Messaging:**
   Messages sent by a user are transmitted via WebSocket to the server, which broadcasts them to all active connections.
5. **Database Interaction:**
   MongoDB stores user information and chat history, which is fetched on demand when users join or reload the chat.



## 6. Scalability and Future Enhancements

- **Scaling:**
  The system can be scaled by deploying multiple instances of the WebSocket server using load balancers.
- **Future Enhancements:**
  - Implementing end-to-end encryption for message privacy.
  - Adding group chat and video call features.

## Setup and Run Documentation

### System Requirements

- **Operating System:** Works on Windows, macOS, Linux
- **Node.js:** Version >= 16.x
- **MongoDB:** Local instance or MongoDB Atlas
- **Browser:** Chrome, Firefox, or any modern browser

### Dependencies

- **React.js:** To build the frontend of the application.
- **Tailwind CSS:** For creating a responsive and flexible layout.
- **Express.js:** To build the backend REST API and WebSocket server.
- **MongoDB & Mongoose:** For storing user data and chat messages.
- **JWT:** To handle secure user authentication.
- **WebSocket (ws library):** For enabling real-time communication between the server and clients.

### Why These Libraries Were Used

- **React.js:** Chosen for its ability to create reusable UI components and manage state efficiently with hooks.
- **Tailwind CSS:** Provides utility classes for rapid UI development and ensures responsive designs across devices.
- **Express.js:** Offers a simple and flexible framework to handle backend requests and real-time WebSocket connections.
- **MongoDB:** A NoSQL database that allows for flexible schema designs and scalability, perfect for chat applications.
- **JWT:** Provides a secure method for handling user sessions and authentication.

- **WebSocket (ws library):** Handles bi-directional communication for real-time messaging in a lightweight manner.

## Step-by-Step Setup Guide

1. **Clone the Repository**
   bash
   ```
   git clone https://github.com/shriya-27/chatApp.git
   ```
2. **Install Dependencies** Ensure that Node.js and npm are installed on your machine. Then run the following in both `client` and `api` directories:
   For the frontend:
   bash
   ```
   cd client
   npm install
   ```
   For the backend:
   bash
   ```
   cd server
   npm install
   ```
3. **MongoDB Setup**
   - If you are using a local MongoDB instance, make sure it is running.
   - Alternatively, set up a MongoDB Atlas account and get the connection URI.
4. **Environment Variables** Create a `.env` file in the `api` directory and include the following:
   bash
   ```
   MONGO_URL=".."
   JWT_SECRET="ajkjxnjkcdkcmwemk"
   CLIENT_URL="http://localhost:5173"
   ```

**Running the Backend** In the `api` directory, start the backend server:

bash
```
node index.js
```

**Running the Frontend** In the `client` directory, start the React app:

bash

```
yarn dev
```

**Common Issues and Troubleshooting**

- **CORS Issues:**
  Ensure CORS is properly configured in the backend for the client to communicate with the server.
- **WebSocket Connection Errors:**
  Ensure that the WebSocket server is properly set up and running. Check that the client is using the correct WebSocket URL.