COLLEGE CODE : 9623

COLLEGE NAME : Amrita College of Engineering and Technology

DEPARTMENT : Computer Science and Engineering

STUDENT NM-ID : 3882526D227B55A46A95504A5BA2B8B3583

ROLL NO : 962323104033

DATE : 22/09/2025

Completed the project named as

Phase 3 TECHNOLOGY PROJECT

NAME : CHAT APPLICATION UI

SUBMITTED BY,
NAME : Giridhar P J
MOBILE NO : 6381055107

Project Title: Chat Application UI

Phase 3: MVP Implementation (Deadline - Week 8)

The goal of this phase is to build a fully functional, end-to-end MVP that can be used for initial testing and feedback.

1. Project Setup

This is the foundation for a scalable and maintainable project.

Frontend Repository Initialization:

Create a new frontend repository (e.g., chat-ui-frontend) on GitHub.

Initialize a new project using a framework like React, Vue, or Angular.

Configure a build tool (Vite, Webpack) for development and production builds.

Backend Repository Initialization:

Create a separate backend repository (e.g., chat-ui-api).

Initialize a new Node.js project using npm init.

Install core dependencies: express for the REST API, socket.io for real-time communication, and cors for cross-origin requests.

Monorepo (Alternative):

Consider using a monorepo setup with tools like Lerna or Turborepo. This allows you to manage both the frontend and backend from a single repository, which can simplify some aspects of development.

Environment Variables:

Set up .env files in both projects to manage environment-specific variables like API URLs, database connection strings, and secret keys. This is crucial for security and a smooth deployment process.

2. Core Features Implementation

This is the core of your MVP. Focus on building the absolute essentials based on your MVP features list from Phase 1.

User Authentication (REST API):

Registration: Implement a /api/register endpoint to create new users. This will involve hashing passwords for security.

Login: Implement a /api/login endpoint to authenticate users and generate a JSON Web Token (JWT) for secure session management.

User & Chat Management (REST API):

User Listing: Create an endpoint /api/users to fetch a list of all users. This will be used on the frontend to display who a user can chat with.

Chat History: Implement an endpoint /api/chats/:userId to retrieve the message history between the logged-in user and a specific contact.

Sending Messages: Create a /api/chats/:userId/messages endpoint to handle new messages sent from the frontend.

Real-time Messaging (Socket.io):

Socket.io Server: On the backend, set up a WebSocket server using socket.io to handle real-time events.

Connection & Disconnection: Handle user connections and disconnections to track their online status.

Message Broadcasting: When a new message is received via the API, broadcast it to the intended recipient and the sender in real time using WebSockets. This ensures both parties see the message instantly.

Frontend UI Development:

Login & Registration Pages: Build simple forms for user authentication.

User List Component: Create a component that fetches the list of users from the API and displays them.

Chat Window Component: Build the main chat interface, which includes:

A message list to display all messages.

A message input field and a "Send" button.

Integration with both the REST API (for fetching history) and Socket.io (for real-time updates).

3. Data Storage (Local State / Database)

Database:

Choose and set up a database. For a Node.js REST API, popular choices include MongoDB (for flexibility with NoSQL) or PostgreSQL (for relational data).

Create a schema or data model for Users and Messages.

User Model: Fields like id, username, email, password_hash.

Message Model: Fields like id, sender_id, recipient_id, content, timestamp.

Local State:

Use local state (e.g., React hooks like useState or a state management library like Zustand) on the frontend to manage the UI state. This includes:

Currently logged-in user information.

List of users to chat with.

The messages within the active chat window.

4. Testing Core Features

Testing is critical to ensure your MVP is stable and provides a good user experience.

Unit Tests:

Write unit tests for critical backend functions (e.g., password hashing, JWT creation).

Write unit tests for isolated frontend components (e.g., a login form component to ensure it handles input correctly).

Integration Tests:

Create integration tests to verify the full flow: a user logs in, sends a message, and the recipient receives it in real-time. This tests the interaction between the frontend, the REST API, the database, and the WebSocket server.

Manual Testing:

Perform thorough manual testing to ensure the UI is responsive, the chat bubbles appear correctly, and all features work as expected.

Use browser developer tools to check for API request/response issues and WebSocket connection errors.

5. Version Control (GitHub)

Commit Frequently: Use clear and descriptive commit messages that explain what has been changed.

Example: feat: implement user login and JWT generation

Example: fix: correct message display order in chat window

Branching Strategy:

Use a main or production branch for stable code.

Create a development branch for integrating new features.

Use feature branches for each new task (e.g., feature/user-auth, feature/realtime-messages).

Pull Requests (PRs):

Submit a pull request for every new feature or bug fix.

Require a code review from a teammate before merging into the development branch. This is a crucial step for maintaining code quality.

Issue Tracking:

Use GitHub Issues to track bugs and new features. Link your PRs to the corresponding issues to keep everything organized.