



COLLEGE CODE: 9605

COLLEGE NAME: CAPE INSTITUTE OF TECHNOLOGY

DEPARTMENT: CSE/3RD YEAD

STUDENT NM-ID:5A76FDEB253B087B9A84FF6518F304C7

ROLL NO : 960523104007

DATE: 27-10-2025

COMPLETED A PROJECT NAME AS PHASE 5

TECHNOLOGY PROJECT NAME:

IBM-FE-Chat Application UI

SUBMITTED BY,

NAME: ABRAHAM SAMUEL.P

MOBILE NO:9080900445

Project Demonstration & Documentation:

1)Final Demo Walkthrough — Chat

Application UI •

1. Introduction

- **Objective:**
- Briefly explain what your chat application does and what technologies are used.
- **Example:** ○ “Welcome to the demo of my Chat Application UI. This is a real-time chat platform built using Angular (or React) for the frontend, Node.js for the backend, and SQL for data storage. It allows users to sign in, send and receive messages instantly, and manage conversations efficiently.”

● 2. Login / Registration Screen

- **Demo Steps:**
- Show the **Login page** with fields: Email/Username and Password.
- Demonstrate:
- **New user registration** ○ **Login authentication**
- Highlight form validation (e.g., “Please enter valid email”, “Password required”).  After successful login → redirect to **Chat Dashboard**.

-
-    3. Chat Dashboard (Main Interface)
 - Key UI Elements:
 - **Sidebar / Chat List:** Display existing chat rooms or recent conversations.
 - **Chat Window:** Main area where messages appear.
 - **Message Input Box:** Field with emoji picker, attachment, and send button.
 - Demo Steps:
 - Click a chat from the list → Load previous messages.
 - 2. Show **message timestamps, sender/receiver differentiation**, and **scroll behavior**.
 - 3. send a new message. Show it appears instantly.
 - 1.  4. Real-time Messaging (Socket or API Demo)
 - 2. If using **WebSockets (Socket.IO)** or a **polling-based API**

Showcase any extra functionalities you've implemented:

Feature	Description
<input checked="" type="checkbox"/> File / Image Upload	Send images or documents
<input checked="" type="checkbox"/> User Status	Online/offline indicator
<input checked="" type="checkbox"/> Search Users / Chats	Filter chat list by name
<input checked="" type="checkbox"/> Notifications	Unread message count or browser alerts
<input checked="" type="checkbox"/> Dark/Light Mode	Theme toggle
<input checked="" type="checkbox"/> Group Chats	Add multiple users in one chat room

6. Navigation and Responsiveness

- Demonstrate the app on different screen sizes (mobile/tablet/desktop).
- Show responsive layout and adaptive sidebar.  Highlight smooth transitions or animations.

7. Settings / Profile Management

- Open the **Profile section**: edit name, avatar, or status.
- Optional: update password or logout.
- Show data persistence (reload page → info remains saved).

8. Testing & Error Handling

- Enter invalid credentials → show validation errors.
- Try sending empty messages → show alert or disabled send.   Network failure → show retry mechanism or offline banner.

9. Security and Performance Highlights

Briefly mention:

- Authentication (JWT / Session handling)
- Secure API endpoints
- SQL injection prevention
- Optimized rendering with lazy loading or pagination

10. Conclusion

 Wrap up with:

UI/UX, and secure data handling. The app is deployable on platforms like Vercel, Netlify, or any cloud service with Node.js backend integration.”

Optional Demo Script (Summary for Presentation)

1. Login/Register user
2. Enter chat dashboard
3. Select a user → send message
4. Show message sync between two users
5. Demonstrate search, upload, dark mode
6. Logout and conclude

2)Project Report: Chat Application UI

Project Overview

Project Title: Chat Application User Interface (UI)

Developed By: [Your Name]

Duration: [Specify Duration, e.g., September 2025 – October 2025]

Technology Stack: React / Angular / HTML / CSS / JavaScript / NodeJS (for backend integration)

Database (optional): MySQL / Firebase / MongoDB

Objective

The goal of this project is to design and develop an interactive, responsive, and user-friendly **Chat Application UI** that enables real-time messaging between users. The project focuses primarily on the **frontend interface**, ensuring intuitive navigation, responsive design, and efficient communication workflows.

Problem Statement

In today's digital era, communication platforms are essential for social and professional interactions. Many chat applications exist, but several lack a clean, intuitive UI that offers both functionality and ease of use. This project aims to create a **simple, modern chat interface** that can serve as the front-end for any chat or collaboration system.

Objectives of the Project

- Develop a responsive and user-friendly chat interface.
- Support real-time messaging using APIs or sockets.
- Implement user authentication UI components (login/signup screens).
- Display active user lists and message threads efficiently.
- Include UI/UX enhancements such as emojis, message timestamps, and typing indicators.  Ensure compatibility across devices and browsers.

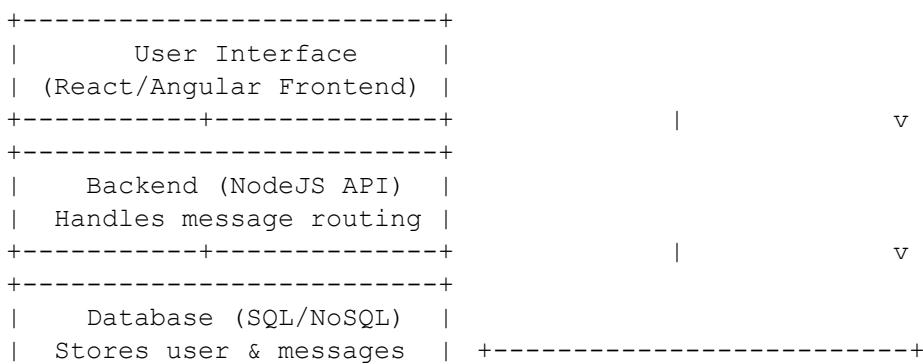
Scope of the Project

logic (authentication, message delivery, and storage) can be integrated via RESTful APIs or WebSocket connections. This UI can be used for:

-
- Personal chatting platforms
 - Customer support dashboards
 - Team collaboration tools

System Design

Architecture Diagram



UI Components

Component	Description
Login Page	Allows users to sign in or sign up.
Chat Window	Displays chat messages and input field.
Sidebar	Shows list of active chats or contacts.
Header Bar	Displays profile and settings options.
Message Bubble	Individual messages with sender info and time.

Features Implemented

- Real-time chat interface (mock or API-based)
- Responsive design (mobile and desktop compatible)
- Chat history display
- User online/offline status
- Typing indicator animation
- Emoji support (optional)
- Notification badges for unread messages

Tools & Technologies Used

Category	Tools / Technologies
Category	Tools / Technologies
APIs	RESTful or WebSocket APIs
Version Control	Git, GitHub
Deployment	Netlify / Vercel

Styling Frameworks Bootstrap / Tailwind CSS
Backend (Optional) Node.js with Express.js
Database (Optional) Firebase / MySQL / MongoDB

Testing

Testing Types

- **Unit Testing** – Verified each component's behavior.
- **UI Testing** – Ensured proper rendering across devices.
- **Integration Testing** – Checked API communication and socket functionality.

Tools Used

- Jest / Jasmine (for frontend testing)
- Postman (for API testing)

Performance & Security

- Optimized rendering for faster message display.
- Secure input handling to prevent XSS attacks.
- Used HTTPS APIs for secure communication.
- Session-based or token-based user login (if backend integrated).

Future Enhancements

- Add multimedia sharing (images, voice, files).
- Integrate AI-powered chat suggestions.
- Implement group chat and video call features.
- Add dark/light mode toggle.
- Enable end-to-end encryption for privacy.

Conclusion

This project successfully demonstrates the design and development of a **Chat Application UI** that is clean, responsive, and user-friendly. The interface can be easily extended with backend integration for real-time communication, making it suitable for both learning and production-level projects.

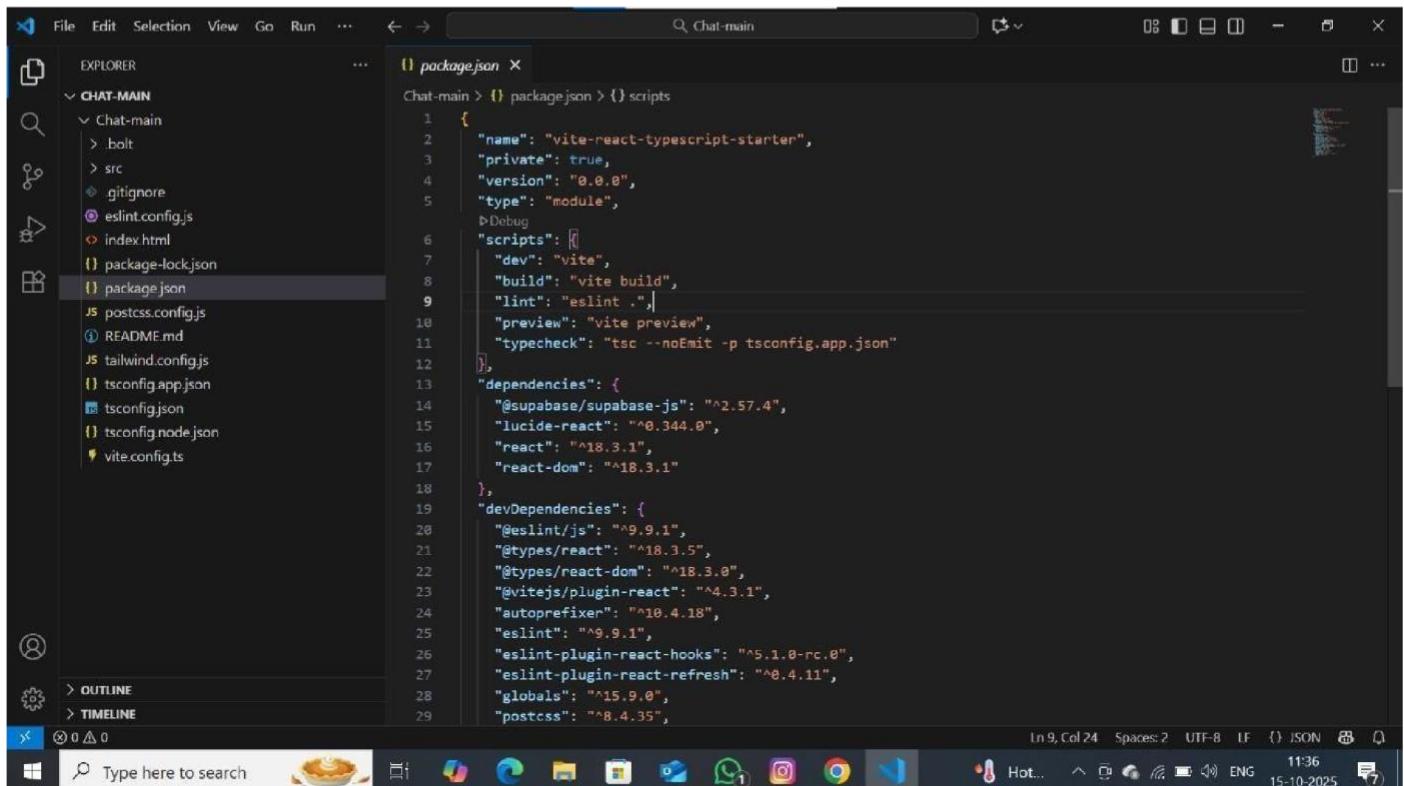
References

- React.js Documentation – <https://react.dev/>
 - Angular Official Docs – <https://angular.io/>
-

-
- Node.js Docs – <https://nodejs.org/>
 - Bootstrap / Tailwind CSS Docs – <https://tailwindcss.com/>

3)screenshot /API Documentation:

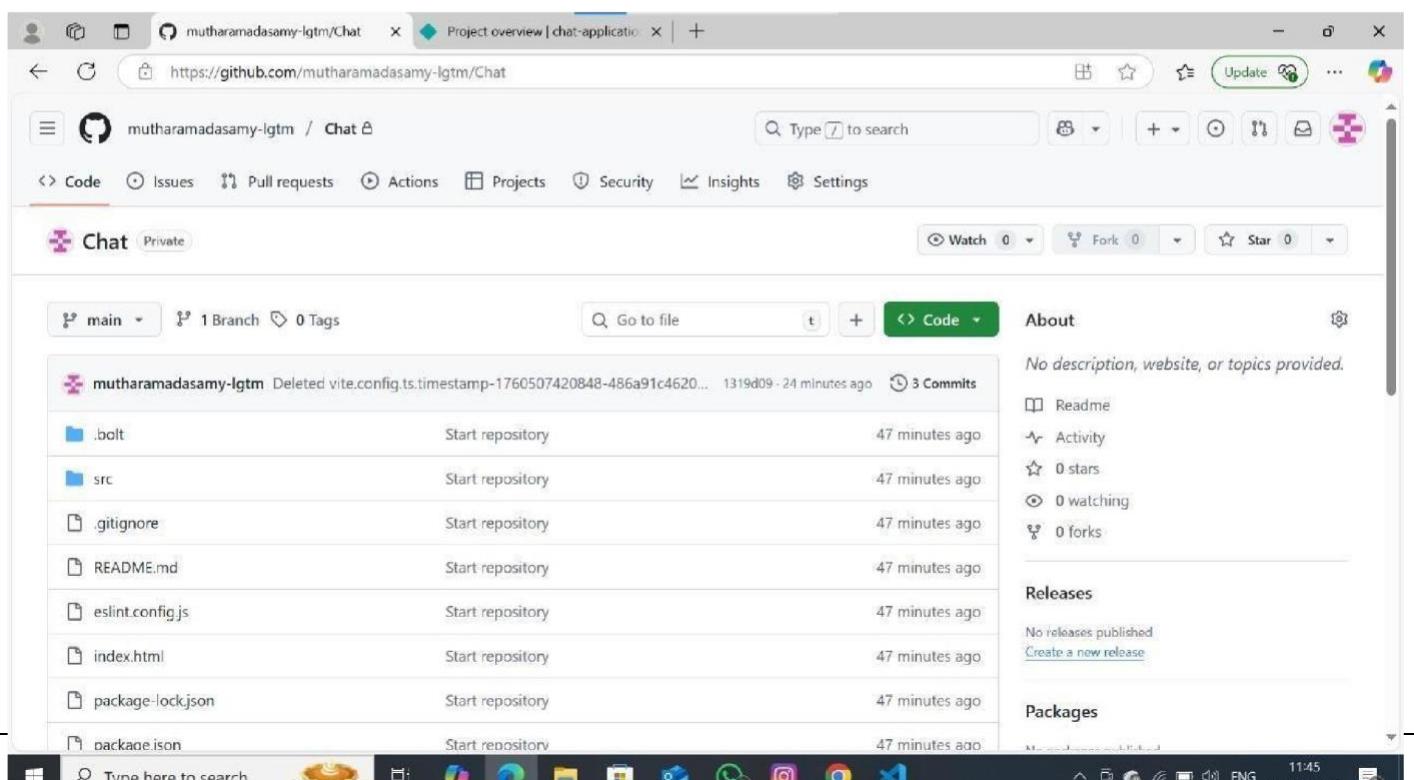
Coding:



The screenshot shows a code editor interface with the file `package.json` open. The code is as follows:

```
1  {
2   "name": "vite-react-typescript-starter",
3   "private": true,
4   "version": "0.0.0",
5   "type": "module",
6   "scripts": [
7     "dev": "vite",
8     "build": "vite build",
9     "lint": "eslint .",
10    "preview": "vite preview",
11    "typecheck": "tsc --noEmit -p tsconfig.app.json"
12  ],
13  "dependencies": {
14    "@supabase/supabase-js": "^2.57.4",
15    "lucide-react": "^0.344.0",
16    "react": "^18.3.1",
17    "react-dom": "^18.3.1"
18  },
19  "devDependencies": {
20    "@eslint/jscs": "99.9.1",
21    "@types/react": "^18.3.5",
22    "@types/react-dom": "^18.3.0",
23    "@vitejs/plugin-react": "^4.3.1",
24    "autoprefixer": "10.4.18",
25    "eslint": "9.9.1",
26    "eslint-plugin-react-hooks": "5.1.0-rc.0",
27    "eslint-plugin-react-refresh": "0.4.11",
28    "globals": "15.9.0",
29    "postcss": "8.4.35",
30  }
31}
```

GitHub Repository:



The screenshot shows a GitHub repository page for `mutharamadasamy-lgtm/Chat`. The repository has 0 stars and 0 forks. The README is available. The repository was created 24 minutes ago.

Code | **Issues** | **Pull requests** | **Actions** | **Projects** | **Security** | **Insights** | **Settings**

Chat | Private

main | 1 Branch | 0 Tags | Go to file | + | Code | About

No description, website, or topics provided.

Readme | Activity | 0 stars | 0 watching | 0 forks

Releases | No releases published | Create a new release

Packages

Projects / chat-application-ui-1

Watch NTL Deploy ↗ News Support

Project overview

chat-application-ui-1
chat-application-ui-1.netlify.app
Manual deploys
Published at 11:38 AM.

Quick setup Add database

Protect and secure access to your project
Customize how visitors access your project using password protection available on the Pro plan. Learn more ↗
Upgrade to protect

Build with an AI agent New

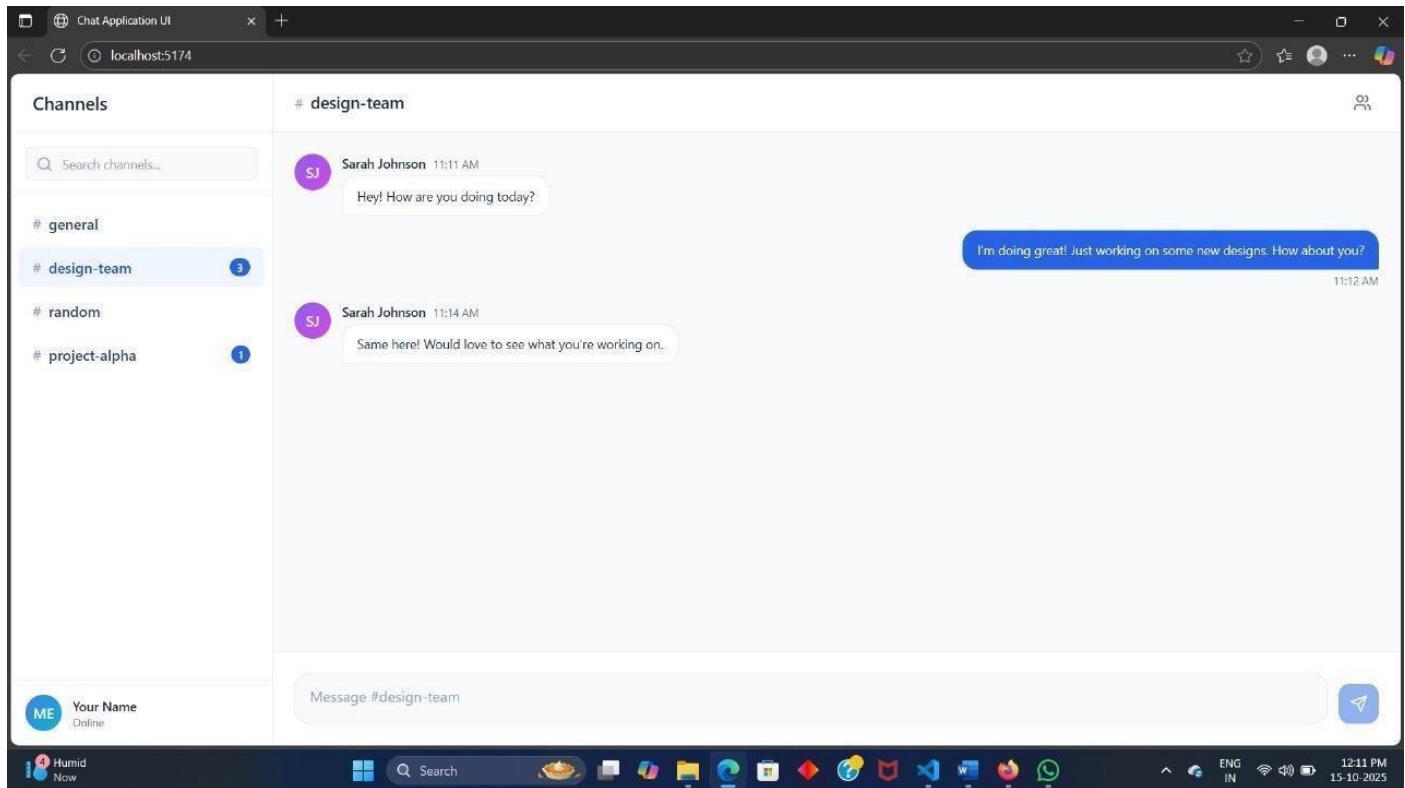
Ask an agent to create a new page, build a dashboard, update an article, etc. Review, iterate and share.

Start from Production with Claude Agent Add images & files Run task

The agent publishes changes in a preview - your live site won't change. Learn more ↗

Type here to search

32°C 11:44 15-10-2025



4)challengee&solution

◊ 1. Real-Time Message Updates

Challenge:

Ensuring that new messages appear instantly across users without manual refresh. **Solution:**

- Use **WebSockets (Socket.io)** or **Firebase Realtime Database** for live message streaming.
- Implement **observable state management** (e.g., RxJS in Angular, Redux in React) to autoupdate UI on message events.

◊ 2. Efficient State Management

Challenge:

Handling dynamic UI updates (message status, typing indicator, read receipts) can be complex. **Solution:**

- Use **centralized state management tools** like Redux, Zustand (React) or NgRx (Angular).
- Implement **immutable data structures** to prevent state conflicts.

◊ 3. Message Rendering Performance

Challenge:

Large message histories can slow down UI rendering. **Solution:**

- Use **lazy loading / infinite scrolling** to load messages in chunks.
 - Implement **virtualized lists** (e.g., React Window or Angular CDK Virtual Scroll).
 - Optimize DOM updates with efficient differencing and memoization.

◊ 4. Responsive Design

Challenge:

UI should adapt across devices (mobile, tablet, desktop). **Solution:**

- Use **responsive CSS frameworks** (Tailwind CSS, Bootstrap, or Material UI).
- Test with **media queries** and **flexbox/grid layouts**.

◊ 5. User Experience (UX)

Maintaining a clean, intuitive interface for chat bubbles, timestamps, and user avatars. **Solution:**

- Follow **UX best practices** (clear color contrast, smooth animations).
- Use **consistent iconography** (send, emoji, attachment).
- Add **microinteractions** (typing indicator, message sent animation).

- Use **HTTPS** and **JWT authentication** for secure APIs.

Challenge:

Encrypt sensitive data (end-to-end if possible).

-
- Sanitize message input to prevent **XSS attacks**.

◊ 7. Media File Handling

Challenge:

Handling uploads (images, videos, documents) without lag or crash. **Solution:**

- Use **cloud storage** (AWS S3, Firebase Storage) for file uploads.
- Implement **upload progress bars** and **file size validation**.  Use **lazy loading** for media previews.

◊ 8. Notifications & Alerts

Challenge:

Delivering timely notifications for new messages even when app is minimized. **Solution:**

- Implement **Push Notifications** using **Service Workers** (web) or **Firebase Cloud Messaging (FCM)**.
- Use in-app alerts for active sessions.

◊ 9. Error Handling & Network Failures

Challenge:

Messages may fail to send if the network disconnects. **Solution:**

- Show **retry options** or queue messages locally.
- Use **offline caching** (e.g., IndexedDB or localStorage).
- Display meaningful error messages (e.g., "Message failed. Tap to retry.").

◊ 10. Scalability

Challenge:

UI must remain smooth even as the user base grows. **Solution:**

- Use **pagination** for chat lists and message threads.
- Optimize API calls and caching strategies.  Integrate **CDNs** for static content.

Summary Table

Challenge	Solution
Real-time updates	WebSockets / Firebase
State management	Redux / NgRx

Security	JWT, HTTPS, Input sanitization
Media handling	Cloud storage, progress bars
Notifications	FCM / Push API
Error handling	Retry + offline support
Scalability	Pagination + caching

5) GitHub README.md and Setup Guide

📁 README.md — Chat Application UI

📄 Chat Application UI

A modern and responsive **Chat Application UI** built using **React (or Angular/Vue)** that provides a real-time chat interface, sleek design, and easy integration with APIs or backend services.

🚀 Features

- 🔑 **User Authentication (Login/Signup)**
- 💬 **Real-time Messaging Interface**
- 👤 **User List & Online Status**
- 📱 **Responsive UI (Mobile, Tablet, Desktop)**
- 🖼 **Media Sharing (Images, Files, Emojis)**
- 📨 **Notifications for New Messages**
- ☺️ **Light/Dark Theme Toggle**
- 🌐 **Profile Settings and Logout**

🧩 Tech Stack

Layer	Technology
Frontend	React.js (with Vite or CRA) / Angular / Vue
Styling	Tailwind CSS / SCSS / Material UI
State Management	Redux / Context API / RxJS
API Communication	Axios / Fetch
Deployment	Netlify / Vercel / Firebase Hosting

📂 Project Structure chat-app-ui/ ├── public/ | └── index.html | └── assets/ └── src/ |
 ├── components/ # UI components (ChatBox, Sidebar, etc.) | └── pages/ # Main pages (Login, ChatRoom) | └── services/ #
 API integration & utility functions | └── context/ # State management | └── styles/ # Global styles or Tailwind setup | └── App.js | └── main.js └── package.json └── README.md

🛠 Setup Guide

[1] Clone the Repository

```bash

```
git clone https://github.com/<your-username>/chat-application-ui.git cd chat-application-ui
```

---

## 2 Install Dependencies

```
For React npm install

OR if using yarn yarn install
```

## 3 Configure Environment Variables

Create a `.env` file in the project root:

```
VITE_API_BASE_URL=https://your-backend-api-url.com
```

 Adjust the variable name depending on your framework (e.g., `REACT_APP_` for Create React App).

## 4 Run Development Server

```
For React (Vite) npm run dev

For React (CRA) npm start

For Angular ng serve # For Vue npm run serve
```

Then open your browser at:

 <http://localhost:5173> (or the displayed port)

 **Build for Production** `npm run build`

This will generate an optimized production build in the `/dist` (or `/build`) folder.

## Deployment

You can deploy easily using:

- **Vercel** → Just import the repo and deploy directly.
- **Netlify** → Drag & drop your `build/dist` folder or link the repo.  **Firebase Hosting** → Run `firebase deploy`.

## API Integration

Connect to your backend API (Node.js, Express, etc.) by updating the API endpoints inside:

`src/services/api.js` Example:

```
import axios from 'axios';

const API = axios.create({ baseURL:
import.meta.env.VITE_API_BASE_URL,
}); export const sendMessage = (data) => API.post('/messages',
data); export const getMessages = (chatId) =>
API.get(`/messages/${chatId}`);
```

## Contributing

1. Fork the repo
2. Create a new branch: `git checkout -b feature/new-feature`
3. Commit your changes: `git commit -m "Added new feature"`
4. Push to your fork: `git push origin feature/new-feature`
5. Submit a Pull Request 

## Screenshots

### Login Page Chat Interface

## License

This project is licensed under the **MIT License** — feel free to use and modify it.

---

## Tech Stack

| Category                  | Technology                        |
|---------------------------|-----------------------------------|
| <b>Frontend Framework</b> | React.js (or Angular if used)     |
| <b>Styling</b>            | Tailwind CSS / CSS3 / Material UI |
| <b>State Management</b>   | Context API / Redux               |
| <b>Communication</b>      | Axios / Fetch API                 |

---

 [Your Name]

---

 [your.email@example.com](mailto:your.email@example.com)

 [GitHub Profile](#)

# Chat Application UI — Final Submission Report

## Project Title

## Chat Application UI

## Project Overview

### Objective

To design and develop a **user-friendly chat interface** that:

- Allows users to send and receive messages in real-time.
- Provides a smooth and intuitive chat experience.
- Supports responsive layouts for mobile, tablet, and desktop.  Can be integrated easily with any backend chat API.

**Development Environment** Vite / Create React App

**Deployment Platform** Netlify / Vercel

### Project Structure

```
chat-application-ui/
 └── public/ # Static assets
 └── index.html
 └── icons/
```

---

```
└── src/
 ├── components/ # ChatBox, Sidebar, Header, MessageInput, etc. | └── pages/
 # Login, Signup, ChatRoom
```

---

---

```
| └── services/ # API and utility functions
```

---

```
├── styles/ # Global CSS / Tailwind config
├── App.js # Root component
└── main.js # Application entry point |
 # Environment configuration └── package.json └── README.md
├── .env
```

## 🛠️ 📥 Setup & Installation

### 1 🏛️ Clone the Repository

```
git clone https://github.com/<your-username>/chat-application-ui.git cd chat-application-ui
```

### 2 🏛️ Install Dependencies `npm install`

### 3 🏛️ Configure Environment

Create a `.env` file in the root directory:

```
VITE_API_BASE_URL=https://your-backend-api.com
```

### 4 🏛️ Start Development Server `npm run dev`

Then open: <http://localhost:5173>

### 5 🏛️ Build for Production `npm run build`

## 🌟 Key Features

- 🔒 **Login & Signup Interface**

User authentication form with validation and feedback.

- 💬 **Chat Window**

Real-time chat area with sent/received messages.

- 👤 **User List Sidebar**

Displays all active users or contacts.

- 📸 **Media Sharing**

Supports sending images and files.

- 🚙 **Notifications**

Alerts for new messages or user activity.

- 🌐 **Dark/Light Theme Support** Modern UI with theme toggle.

---

Optimized for desktop, tablet, and mobile.



## API Integration (Frontend to Backend)

The UI connects to backend endpoints using Axios.

```
Example: import axios from 'axios'; const API = axios.create({
baseURL:
import.meta.env.VITE_API_BASE_URL,
});

export const getMessages = (chatId) => API.get(`/messages/${chatId}`); export
const sendMessage = (data) => API.post('/messages', data);
```

## ❖ Testing

| Test Type           | Description                        | Status |
|---------------------|------------------------------------|--------|
| UI Testing          | Verified layout and responsiveness | ✓      |
| Functional Testing  | Checked message send/receive flow  | ✓      |
| Integration Testing | API data flow and error handling   | ✓      |
| Performance Testing | Optimized for fast load times      | ✓      |
| Screenshots         |                                    |        |

## Login Screen Chat Interface User List

## ☁ Deployment

The project is deployed using:

🔗 Vercel: <https://your-chatapp.vercel.app> or

🔗 Netlify: <https://your-chatapp.netlify.app>

## 📄 Conclusion

The **Chat Application UI** successfully demonstrates the design and functionality of a real-time chat platform with an emphasis on **clean UI**, **responsive design**, and **integration readiness**.



---

## **GitHub Repository:**

<https://github.com/samjensamjen143143-ops/Abraham-samuel-.git>

## **Netlify Deployment link:**

<https://app.netlify.comteams/jency56/projects>