**THINKSYNC:** A COLLABORATIVE GROUP CHAT PLATFORM WITH INTEGRATED AI

**Project Report**

Submitted to the Faculty of Engineering and Technology

For the partial fulfilment of the requirements of

**Master of Computer Applications**

**Supervised By: Submitted By:**

Rajwinder Kaur Vishal (28212301618)

Class: MCA (TYP) 4th Sem



**Master of Computer Applications**

**Department of Computer Science**

**Guru Nanak Dev University Amritsar-143005 India May, 2025**

**DECLARATION**

The work embodied in this project entitled **ThinkSync** submitted to the Department of Computer Science, Guru Nanak Dev University, Amritsar, for the award of degree of Master of Computer Applications has been done by me. The project report is entirely based on my own work and not submitted elsewhere for the award of any other degree. All ideas and references have been duly acknowledged.

Name\_\_\_\_\_\_\_\_\_\_

Date\_\_\_\_\_\_\_\_\_\_\_

Signature of Candidate

**CERTIFICATE**

This is to certify that this project entitled, **ThinkSync**, submitted to the Department of Computer Science, Guru Nanak Dev University, Amritsar, for the degree of Master of Computer Applications was carried out by Mr.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_, Roll No. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_is an authentic work carried out by him at \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ under my guidance. The matter embodied in this project work has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

Supervisor Name

Dated:\_\_\_\_\_\_\_\_\_

**ABSTRACT**

ThinkSync is an innovative, AI-powered group chat web application designed to enhance real-time collaboration among users, particularly developers and technical teams. Built using the MERN stack (MongoDB, Express.js, React.js, and Node.js), the platform allows users to chat, collaborate on code, interact with an integrated AI assistant, and even create and run Node.js servers within the same interface.

The application aims to eliminate the need for switching between multiple tools by combining essential developer functionalities into a single, intuitive workspace. ThinkSync enables users to not only communicate seamlessly through real-time messaging but also generate code snippets, receive AI-powered suggestions or explanations, and test or deploy backend servers directly inside the chat environment.

Key features include secure user authentication, project-based group chats, an AI assistant capable of handling technical queries, and a code execution sandbox for real-time testing. The system’s architecture is designed to be modular and scalable, supporting future enhancements such as version control, language support expansion, and cloud deployment.

ThinkSync represents a modern approach to collaborative development, empowering teams to work more efficiently and creatively in a unified digital space. This project demonstrates the integration of AI, web technologies, and backend execution into a cohesive solution that addresses the growing demand for intelligent and interactive team collaboration tools.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Sr No.** | **Topic** | **Page No.** |
| 1 | Introduction To the Project | 5 |
| 2 | Problem Definition and Possible Solution | 7 |
| 3 | Project Objectives | 8 |
| 4 | Software Development Lifecycle | 9 |
| 5 | Limitations of Project | 47 |

**Introduction to Project and Features**

ThinkSync is a cutting-edge group chat platform designed for seamless collaboration, enhanced by AI-powered assistance and live code execution. This platform bridges the gap between communication, coding, and deployment by integrating real-time messaging with AI-driven support and the ability to create and run Node.js servers directly within the chat. ThinkSync aims to streamline workflows for developers, teams, and communities, eliminating the need to switch between multiple tools for communication, coding, and server deployment.

By incorporating advanced AI features, users can generate code, ask questions, receive explanations, and deploy solutions on the fly. ThinkSync not only fosters creativity and innovation but also makes development more accessible and collaborative.

**Key Features of ThinkSync**

1. **Real-Time Group Chat**
   * Instant messaging with WebSocket technology for real-time communication.
   * Multiple chat rooms for topic-based discussions.
   * Message history storage for easy reference.
2. **Integrated AI Assistant**
   * AI-powered bot for answering questions, generating code snippets, and providing explanations.
   * Supports multiple programming languages.
   * Uses advanced NLP to understand and respond to user queries accurately.
3. **Code Sharing and Execution**
   * Built-in editor to write and execute code snippets within the chat.
   * Real-time code output for immediate feedback.
   * Supports popular languages, focusing on JavaScript and Node.js.
4. **Node.js Server Creation and Deployment**
   * Users can create and deploy Node.js servers directly from the chat.
   * Backend support for spinning up temporary containerized servers.
   * Secure and isolated environments for each server instance.
5. **User Authentication and Role Management**
   * Secure login with OAuth or similar protocols.
   * Role-based access control (Admin, Moderator, Member).
   * Ensures secure and controlled access to advanced features.
6. **Scalable and Secure Architecture**
   * Cloud-based hosting for high availability and scalability.
   * End-to-end encryption for secure messaging.
   * Sandboxed code execution to prevent security vulnerabilities.

**Problem Definition and Possible Solutions**

**Problem Statement**

In a rapidly advancing digital world, real-time communication and collaboration are essential for teamwork and project development. Existing chat applications often lack advanced features such as integrated AI support for instant assistance or the ability to directly generate and execute code in a collaborative environment. This results in inefficient workflows, as users have to switch between multiple tools for communication, coding, and AI-driven problem-solving.

**Solution**

This project aims to develop a **Group Chat Web Application with Integrated AI** that allows users to:

1. Collaborate in real-time through group chats.
2. Leverage AI for answering questions, generating code snippets, and providing on-the-fly assistance.
3. Dynamically create, deploy, and execute Node.js servers directly within the chat interface.
4. End-to-end encryption for chat messages to ensure privacy and strict input validation and output sanitization for code execution to prevent security vulnerabilities.

By integrating communication, AI-driven assistance, and live coding features in a single platform, this application will improve productivity and enhance collaborative problem-solving for developers and other user groups.

**Benefits of the Proposed Solution:**

1. **Efficiency**: Users can collaborate, code, and deploy in one place, reducing the need for context switching.
2. **Accessibility**: The platform is accessible to both technical and non-technical users, as AI assistance can help bridge knowledge gaps.
3. **Scalability**: The architecture ensures the platform can handle a growing number of users and projects without performance degradation.
4. **Innovation**: This solution fosters creativity and innovation by enabling users to brainstorm ideas, generate code, and instantly deploy solutions.

**PROJECT OBJECTIVES**

The primary goal of the ThinkSync project is to develop a powerful, real-time group chat application that integrates AI capabilities, supports collaborative coding, and allows users to deploy Node.js servers directly within the chat environment. The objectives are:

1. Facilitate Real-Time Communication:  
   To enable seamless and instant messaging between users using WebSocket-based chat functionality.
2. Integrate AI Assistant:  
   To embed an AI assistant in the chat that can answer questions, provide suggestions, and assist users in problem-solving.
3. Support Collaborative Coding:  
   To allow users to write, edit, and execute code snippets within the chat interface for real-time coding discussions and learning.
4. Enable Server Deployment:  
   To provide a feature where users can generate basic Node.js servers, execute them in a sandboxed environment, and view outputs—all within the app.
5. Ensure Secure User Authentication:  
   To implement secure login, registration, and role-based access to safeguard user data and communication.
6. Create a Modular and Scalable Architecture:  
   To build the system using the MERN stack in a modular format that supports future expansion and additional features.
7. Enhance Developer Collaboration:  
   To provide a platform where developers can brainstorm, code, and test ideas together more efficiently with AI-powered support.

**Software Development Lifcycle**

**Introduction**

The Software Development Life Cycle (SDLC) is a structured process followed to develop high-quality software systematically. It outlines various stages involved in software development, ensuring a logical flow from initial planning to final deployment. For the ThinkSync project, SDLC helps manage complexity by dividing the development process into clearly defined phases, ensuring that each part of the system is analyzed, designed, developed, tested, and deployed efficiently.

The main SDLC phases applied in this project include:

* Requirement Gathering and Analysis
* Feasibility Study
* Design
* Coding
* Implementation and Testing

**Requirement Gathering and Analysis**

This initial phase involves understanding the expectations of the end-users and defining the system's functional and non-functional requirements. For ThinkSync, requirement gathering was conducted through brainstorming, reviewing similar platforms (e.g., Slack, Discord, ChatGPT), and outlining the needs of technical teams who wish to collaborate in real time with AI support and live coding environments.

The analysis identified what features are necessary, what technology stack will be used, and how users will interact with the system.

**1.Requirement Specification**

**Functional Requirements:**

* User registration and login functionality.
* Real-time group chat using WebSockets (Socket.IO).
* AI Assistant that can answer questions, explain code, and generate code snippets.
* Code editor for collaborative coding within chat.
* Node.js server creation and execution environment.
* Project-based chat rooms (multiple users per project).
* Message status indicators (sent, delivered, seen).
* Typing indicators and notification system.

**Non-Functional Requirements:**

* Scalability to support multiple users and projects simultaneously.
* Secure user data handling and authentication using JWT.
* Fast performance and low-latency communication.
* Cross-browser compatibility and responsive UI.

**2. Solution Strategy**

To meet the specified requirements, a **modular and layered architecture** has been adopted using the **MERN stack**:

* **Frontend** (React.js): Provides a responsive, real-time user interface.
* **Backend** (Node.js + Express.js): Handles API requests, user authentication, and server communication.
* **Database** (MongoDB): Stores user information, messages, project data, and server code.
* **WebSockets (Socket.IO)**: Enables real-time bi-directional communication between users.
* **AI Integration (OpenAI API or similar)**: Powers the chatbot assistant embedded in chat.

The project follows **component-based development** and is built in modules to ensure maintainability and ease of feature expansion.

**3. Software & Hardware Requirements**

**Software Requirements:**

* Frontend: React.js, Tailwind CSS, Redux (or React Query), Socket.IO client
* Backend: Node.js, Express.js, Socket.IO server, JWT, bcrypt, dotenv
* Database: MongoDB (Cloud via MongoDB Atlas or Local)
* AI API: OpenAI or custom LLM API
* Code Editor: Monaco Editor or Ace Editor
* Version Control: Git, GitHub
* Development Tools: VS Code, Postman, Node Package Manager (npm/yarn)

**Hardware Requirements:**

Minimum for Development:

* Processor: Intel i5 or equivalent
* RAM: 8 GB
* Storage: 100 GB HDD/SSD
* OS: Windows 10 / Linux / macOS
* Internet: Required for AI API and deployment testing

For Deployment (Cloud Server):

* RAM: 4–8 GB
* Storage: 50–100 GB SSD
* Node.js, MongoDB, and NGINX/PM2 for production

**Feasibility Study**

The feasibility study is essential to determine whether the proposed ThinkSync project is practical, achievable, and beneficial. It examines various aspects to assess the success potential of the project before full-scale development begins.

**1. Technical Feasibility**

The project is technically feasible as it uses proven and widely adopted technologies such as the MERN stack (MongoDB, Express.js, React.js, Node.js), Socket.IO for real-time communication, and OpenAI’s API for AI integration. The required libraries, tools, and frameworks are open-source and compatible with modern development environments.

**2. Economic Feasibility**

ThinkSync requires minimal financial investment since it is being developed using free and open-source tools. AI usage may incur API costs (e.g., OpenAI), but during development, free-tier usage is sufficient. Hosting can be done on platforms like Vercel (frontend) and Render or Railway (backend), which offer generous free tiers for student and personal projects.

**3. Operational Feasibility**

The system is designed to be user-friendly, interactive, and intuitive. The chat-based collaboration model is already familiar to most users, which ensures smooth adoption. AI integration adds value and enhances user experience without requiring specialized knowledge.

**4. Legal Feasibility**

No sensitive or regulated data is handled. Basic compliance with standard security practices (data encryption, secure authentication, etc.) ensures the system is legally sound for educational and personal use.

**5. Time Feasibility**

The project is planned to be completed within a reasonable time frame of **2–3 weeks**, allowing sufficient time for design, development, testing, and documentation.

**1. Planning and Scheduling**

The ThinkSync project is structured into multiple phases with clearly defined goals and time estimates to ensure timely and organized development. A Gantt chart (not included here) can visually represent the timeline.

**Timeline**

* **Week 1-3:** Requirement gathering, research, and feasibility study.
* **Week 4-6:** UI/UX design, database design, and ER diagram creation.
* **Week 7-14:** Frontend and backend development, integrating authentication and chat system.
* **Week 15-18:** AI integration, real-time communication setup, and Node.js execution.
* **Week 19-20:** System testing, bug fixing, and performance optimization.

|  |  |  |
| --- | --- | --- |
| **Phase** | **Task Description** | **Duration** |
| **Phase 1:** Planning & Requirement Analysis | Identifying system requirements, use cases, and feasibility study. | 3 weeks |
| **Phase 2:** Design | Creating wireframes, database design, ER diagrams, and architecture. | 3 weeks |
| **Phase3:**Frontend Development | Developing UI components, authentication, and chat interface. | 4 weeks |
| **Phase 4:** Backend Development | Setting up Node.js, Express, database models, and API endpoints. | 4 weeks |
| **Phase 5:** AI & Code Execution Integration | Implementing AI chat, code execution, and AI-assisted responses. | 3 weeks |
| **Phase 6:** Server Deployment Feature | Enabling Node.js server creation and real-time execution. | 2 weeks |
| **Phase 7:** Testing & Debugging | Unit testing, integration testing, bug fixes. | 2 weeks |

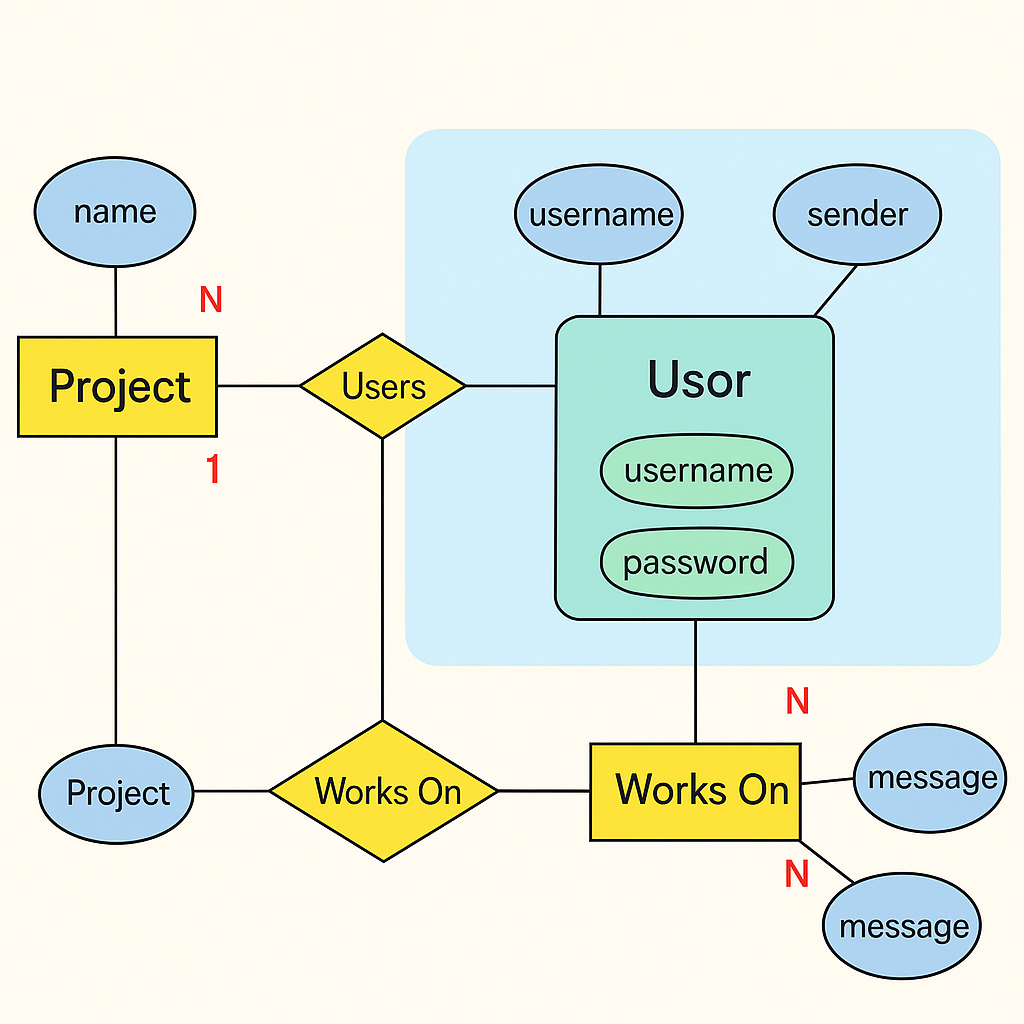
**Design Phase**

The Design Phase of the ThinkSync project focuses on planning the structure and interactions of the system's components before actual development. It ensures the system is logically organized, scalable, secure, and maintainable. The design covers the major modules, user interactions, and database schema necessary to support all functionalities.

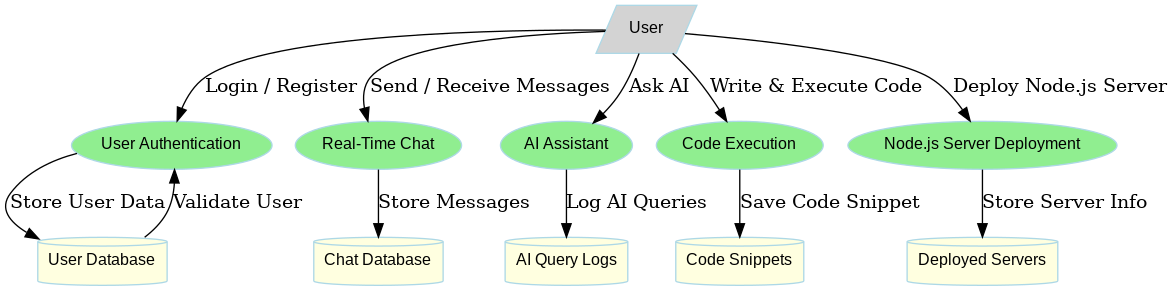
**1. Basic Modules and Descriptions**

| Module Name | Description |
| --- | --- |
| User Management | Handles user registration, login, JWT authentication, password hashing, etc. |
| Project/Group Chat | Enables users to create/join chat groups (projects) for real-time communication. |
| Messaging System | Facilitates real-time chat using Socket.IO, with message statuses and typing indicators. |
| AI Assistant Module | Provides chatbot-like functionality using OpenAI API to assist with queries, code, and explanations. |
| Code Editor Module | Enables users to write and view code collaboratively in the chat (using Monaco or Ace Editor). |
| Node.js Server Execution | Allows users to create basic Node.js servers and test them in a sandboxed environment. |
| Notification Module | Sends real-time notifications about new messages, code updates, or replies. |

**2. ER Diagram**

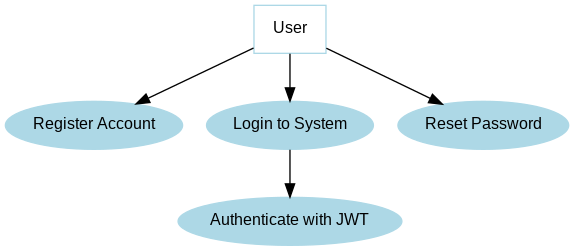
****

**3. DFDs**



**4. Use Case Diagram**

1. User Authentication Use Case Diagram

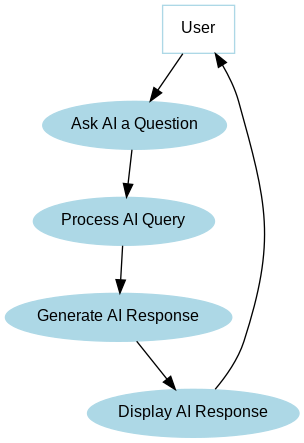


2. Chat System Use Case Diagram

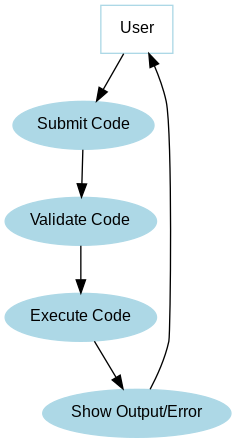
**A diagram of a user

AI-generated content may be incorrect.**

3. AI Assistant Use Case Diagram

****

4. Code Execution Use Case Diagram

****

5. Server Deployment Use Case Diagram

**A diagram of a server

AI-generated content may be incorrect.**

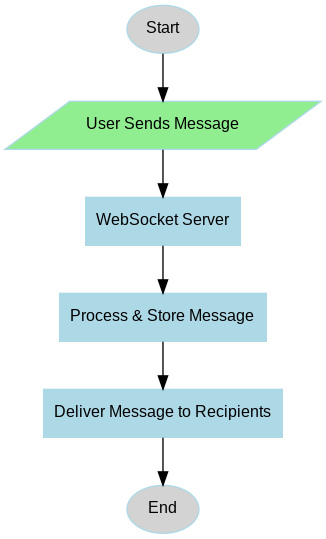
**5. Flowcharts**

Complete system Flowchart

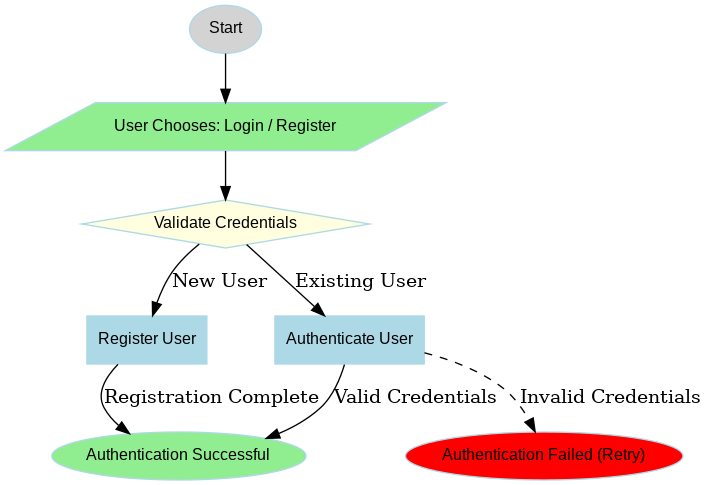
**A diagram of a software process

AI-generated content may be incorrect.**

**Chat Flowchart**



Authentication Flowchart



Ai Interaction flowchart

A diagram of a process flow

AI-generated content may be incorrect.

Code Execution Flowchart

A diagram of a code

AI-generated content may be incorrect.

**Schema Design (Tables)**

**1. Users Collection**

| Field Name | Data Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Primary Key – Auto-generated unique user ID |
| username | String | Unique username (lowercased and trimmed) |
| email | String | Unique user email (used for login) |
| password | String | Encrypted password (not selected by default) |

**2. Projects Collection**

| Field Name | Data Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Primary Key – Unique identifier for each project group |
| name | String | Unique name of the project (lowercase and trimmed) |
| users | Array of ObjectIds | References to users participating in this project (foreign keys to Users) |

**3. Messages Collection**

| Field Name | Data Type | Description |
| --- | --- | --- |
| \_id | ObjectId | Primary Key – Unique ID for each message |
| projectId | ObjectId | Foreign Key – References the associated project |
| sender.\_id | String | Sender’s user ID (stringified, not object ID here) |
| sender.email | String | Email of the sender |
| message | String (Encrypted) | Encrypted message text |
| isAiResponse | Boolean | True if the message was sent by AI |
| createdAt | Date | Auto-generated timestamp for creation (via timestamps: true) |
| updatedAt | Date | Auto-updated timestamp on modification |

**Coding Phase**

**Snapshots**

Login

A screenshot of a computer

AI-generated content may be incorrect.

Register

A screenshot of a computer

AI-generated content may be incorrect.

Home Page

A screenshot of a computer

AI-generated content may be incorrect.

Create Project Modal

A screenshot of a computer

AI-generated content may be incorrect.

Chat Pannel

A screenshot of a computer

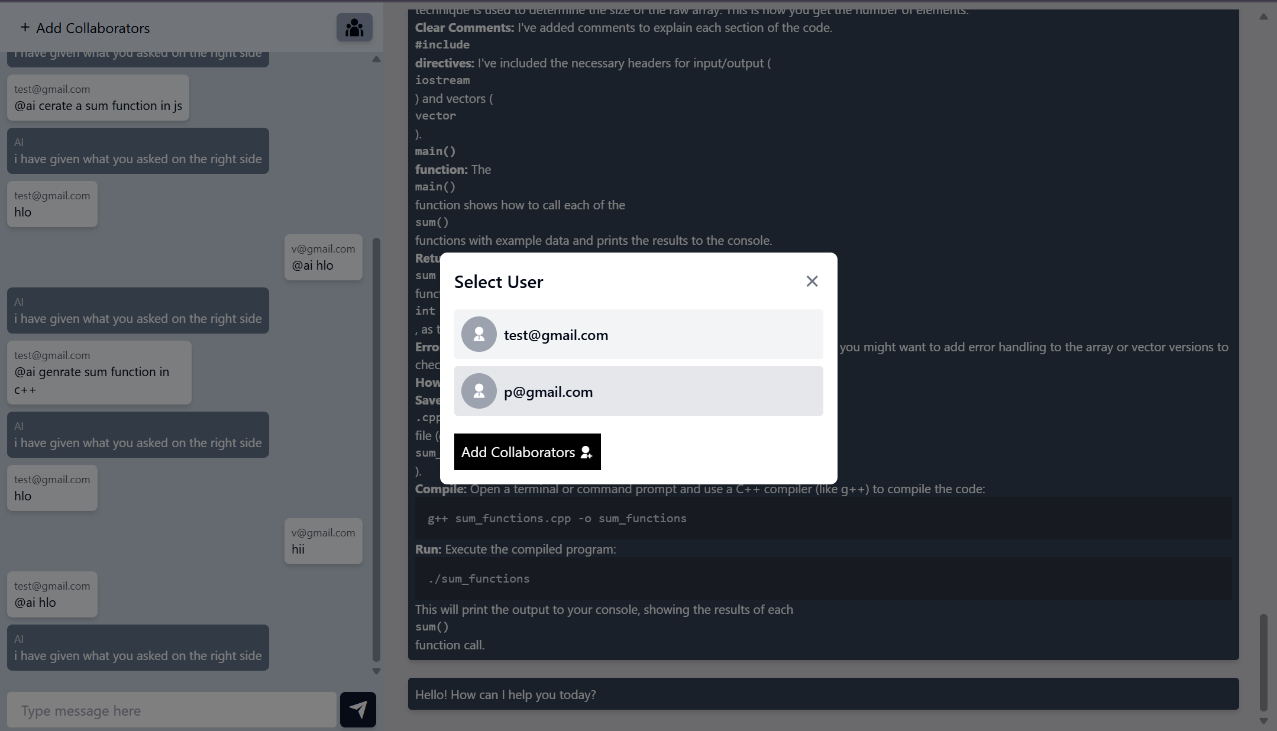
AI-generated content may be incorrect.

Collaborators Members Panel

A screenshot of a computer

AI-generated content may be incorrect.

Add collaborators Modal



**Code snippets**

**Backend**: User Model (Schema)

A screen shot of a computer

AI-generated content may be incorrect.

Message Model (Schema)

A screen shot of a computer program

AI-generated content may be incorrect.

Project Model (Schemaa)

A screen shot of a computer program

AI-generated content may be incorrect.

Auth Middleware (Authentication)

A screen shot of a computer program

AI-generated content may be incorrect.

Ai Service code

A screenshot of a computer program

AI-generated content may be incorrect.

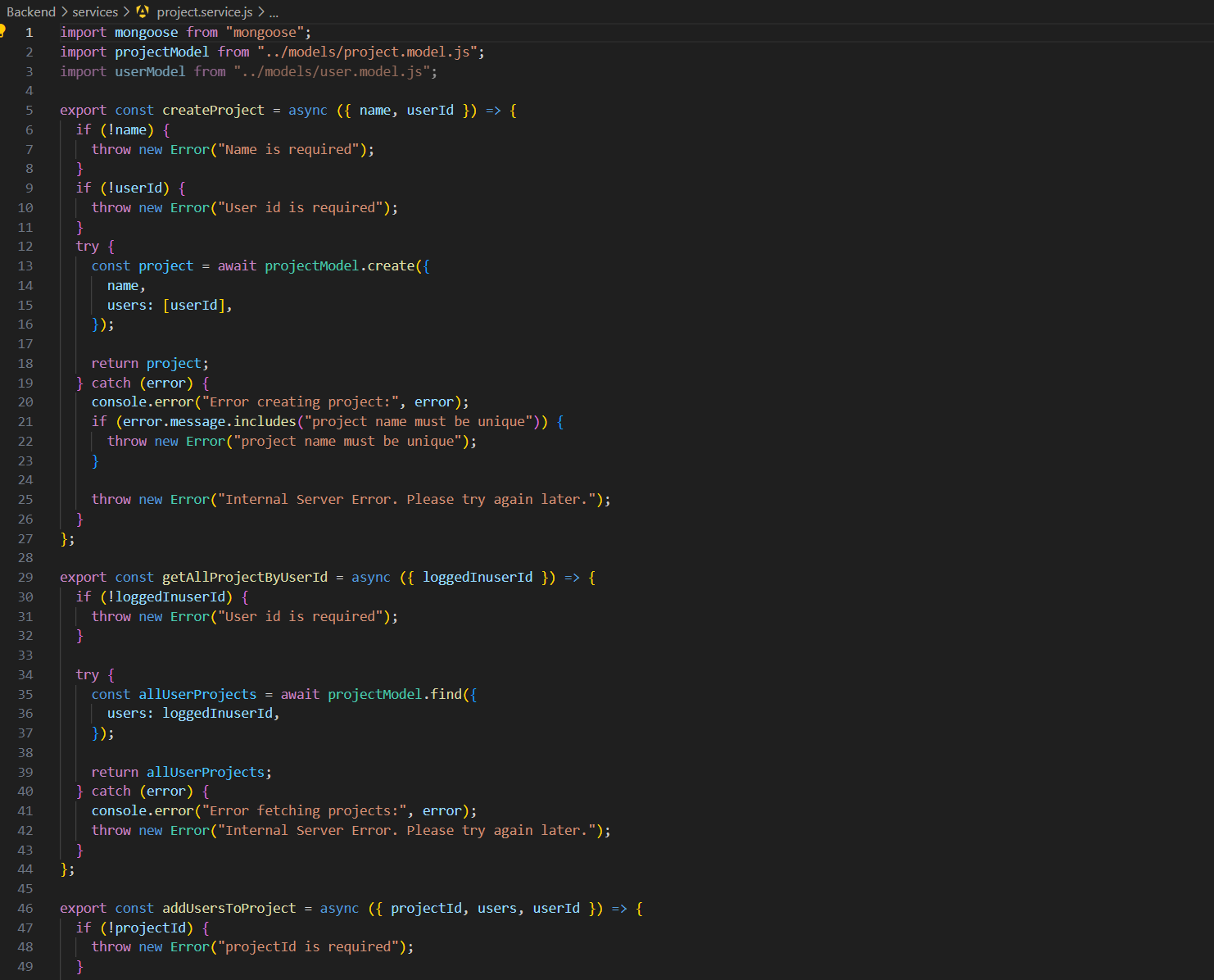
A black screen with orange and white lines

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

Project Service



A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect.

User Service

A screen shot of a computer program

AI-generated content may be incorrect.

Ai Routes

A screen shot of a computer code

AI-generated content may be incorrect.

Message Routes

A screen shot of a computer program

AI-generated content may be incorrect.

Project Routes

A black screen with colorful text

AI-generated content may be incorrect.

User Routes

A screen shot of a computer code

AI-generated content may be incorrect.

**Frontend** : Login Page

A screenshot of a computer program

AI-generated content may be incorrect.

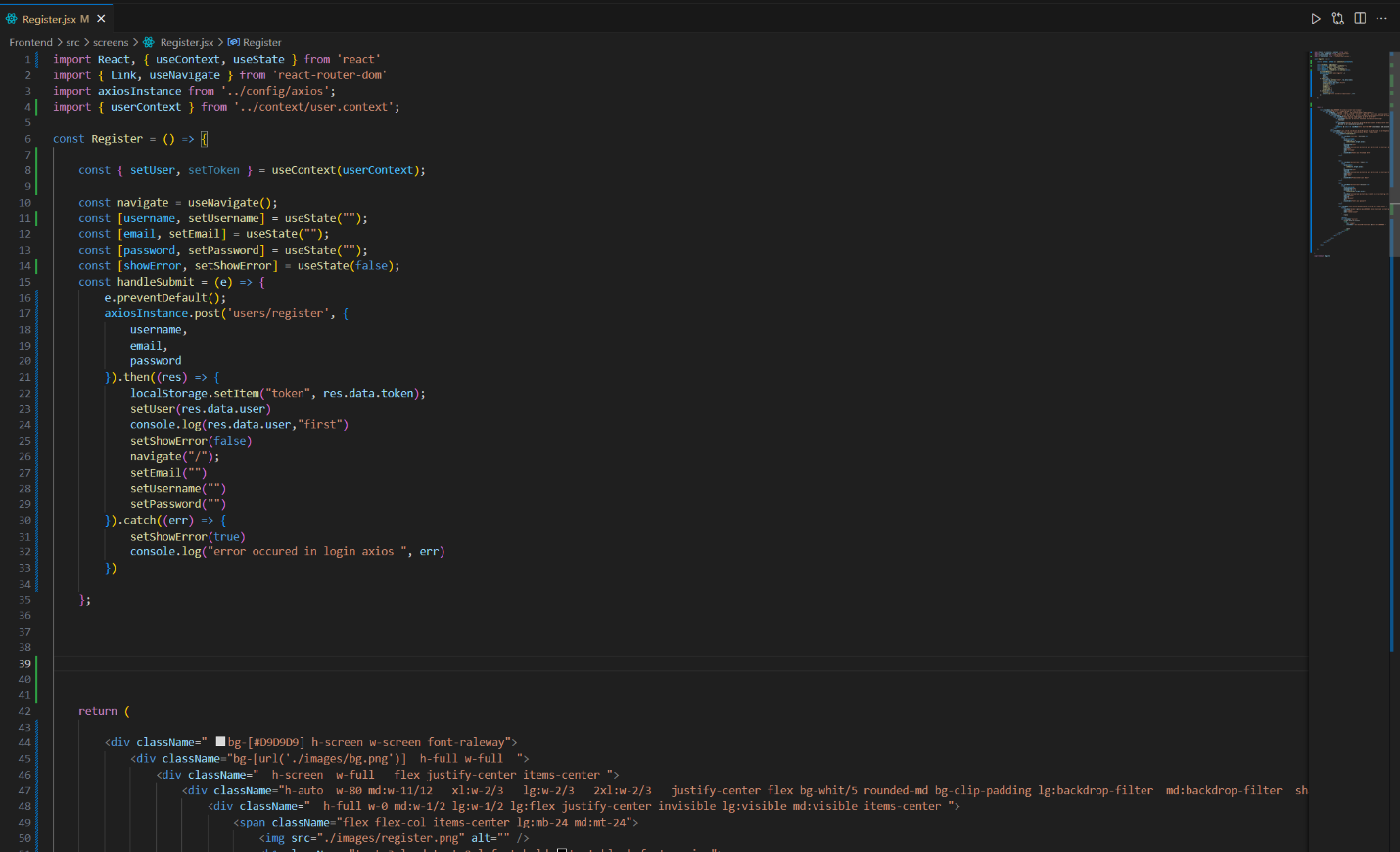
A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Register Page



A screen shot of a computer program

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

Home Page

A screen shot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

Chat page

A screenshot of a computer program

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.A screen shot of a computer

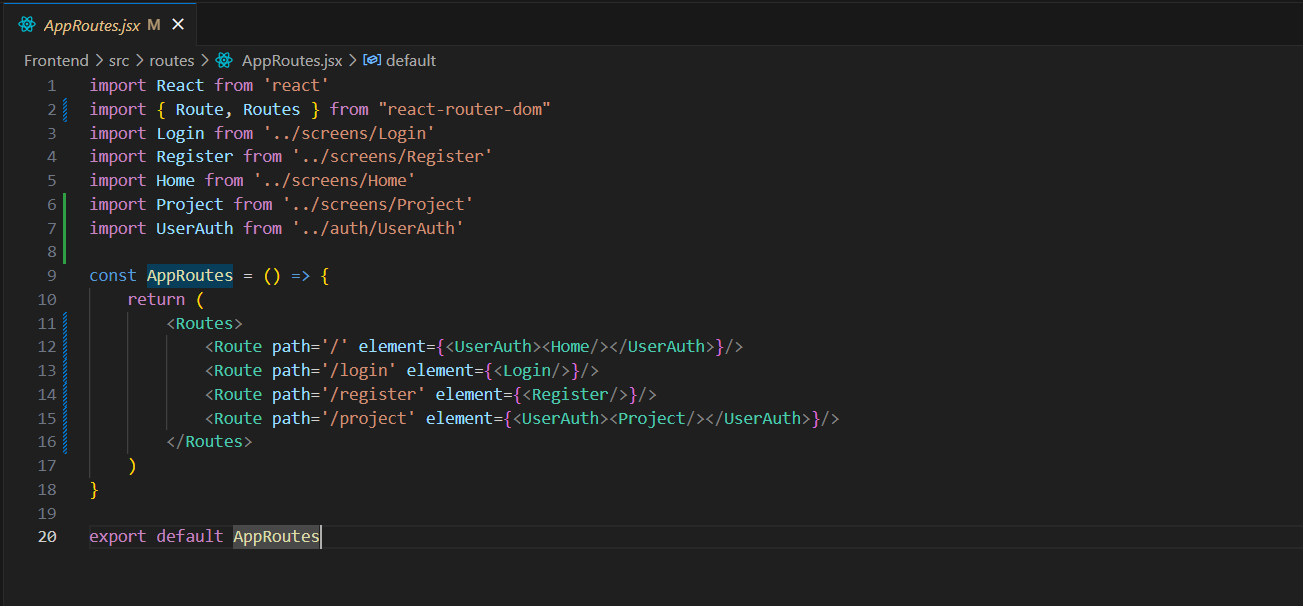
AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.A computer screen shot of text

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

App Routes



App page

A screenshot of a computer

AI-generated content may be incorrect.

**Implementation and Testing**

**Implementation Phase**

The implementation phase involves the actual construction of the ThinkSync application using the MERN (MongoDB, Express.js, React.js, Node.js) stack. Each module was implemented and integrated incrementally, followed by continuous testing. Key stages of implementation included:

**1. Frontend (React.js)**

* **Component-based structure** was used to ensure scalability and reusability.
* Implemented secure **user authentication** using JWT and login/signup forms.
* Integrated **Socket.IO** for real-time chat and typing indicators.
* Added **code editor** using Monaco Editor for collaborative coding.
* Included **AI chat assistant** using OpenAI API for smart responses.
* Responsive UI/UX designed using **Tailwind CSS** for modern aesthetics.

**2. Backend (Node.js with Express )**

* Built secure APIs using **Express.js** and modular structure
* Created endpoints for **user authentication**, **message management**, and **project/group creation**.
* Used **MongoDB with Mongoose** for data persistence and schema modeling.
* Used **AES encryption** for message storage security.
* Integrated **OpenAI API** for generating intelligent AI responses to chat queries.
* Employed **Socket.IO** for WebSocket-based real-time communication.

**3. Database (MongoDB)**

* Managed using **Mongoose ODM**.
* Schemas designed for Users, Projects (groups), and Messages.
* Encrypted sensitive data like messages.
* Used populate() method for referencing documents between collections (User ↔ Project).

**Testing Phase**

Testing was done throughout the development process to ensure the stability, performance, and correctness of the application.

**1. Unit Testing**

* Conducted using **Jest** and **Supertest** for backend services.
* Functions like password hashing, JWT token generation, message encryption, and decryption were unit tested.

**2. Integration Testing**

* Validated how different modules (authentication, chat, AI response) worked together.
* Backend API integration with frontend tested for correct data flow and response codes.

**3. Real-time Testing**

* Tested **Socket.IO** channels for:
  + Sending and receiving messages instantly.
  + Typing indicators and read receipts.
  + AI response integration in real time.

**4. User Acceptance Testing (UAT)**

* A closed group of users tested the application for usability and performance.
* Based on feedback:
  + AI responses were improved.
  + Chat layout was optimized for better readability.
  + Added error messages and loading indicators for better UX.

**Testing Tools Used**

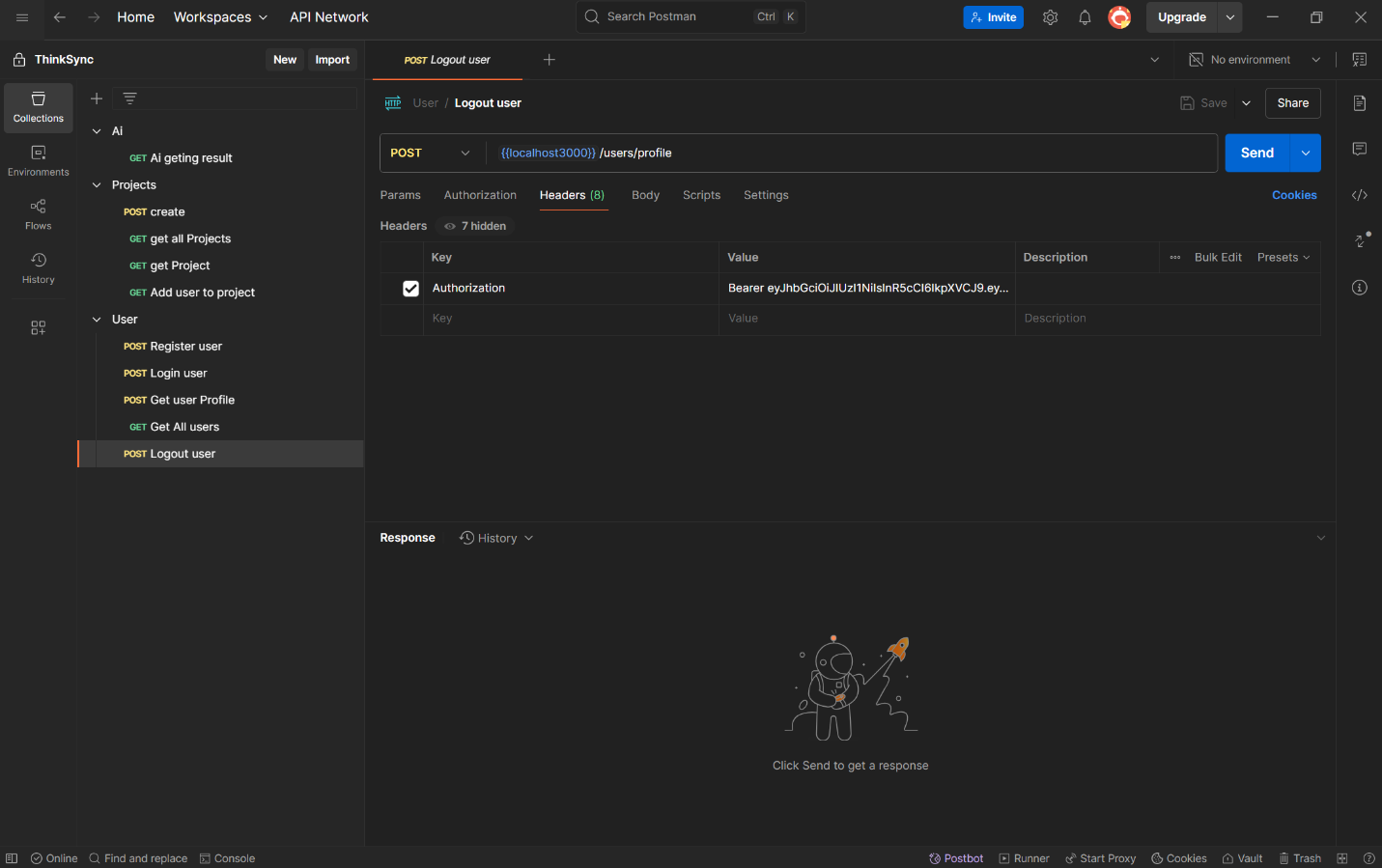
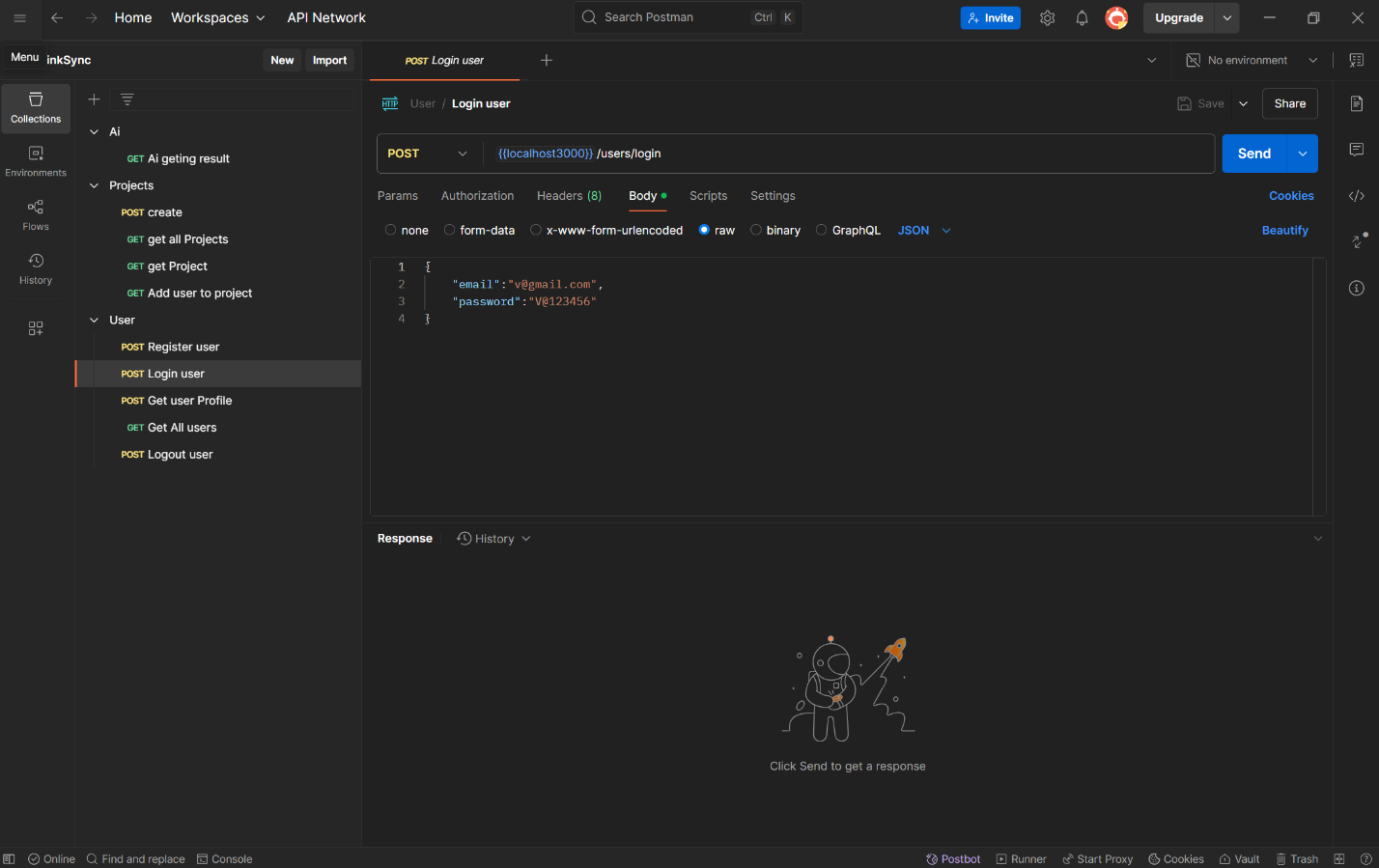
| **Tool** | **Purpose** |
| --- | --- |
| Jest | Unit testing |
| Postman | API endpoint testing |
| Supertest | HTTP integration testing |
| Socket.IO Client | Real-time communication testing |
| Browser DevTools | Frontend debugging and testing |

**Snapshot of Postman**

Api Testing

A screenshot of a computer

AI-generated content may be incorrect.



**Limitation of Project**

While **ThinkSync** provides a robust platform for collaborative chatting and AI-assisted development, the following limitations were identified during development and testing:

**1. AI Dependency**

* The AI responses rely on third-party APIs (e.g., OpenAI).
* If the API service is unavailable or rate-limited, the AI features may not function properly.

**2. Limited Offline Support**

* ThinkSync requires a constant internet connection.
* Users cannot send or receive messages or access AI features without being online.

**3. Performance with Large-Scale Users**

* The current system is designed for small to mid-sized teams.
* Performance may degrade with thousands of concurrent users or high-volume message traffic without proper scaling (e.g., load balancing, sharding).

**4. No Role-Based Access Control (RBAC)**

* Currently, all users in a project have the same privileges.
* There is no implementation of admin/moderator roles for managing project settings or content.

**5. Limited File Sharing**

* The current version does not support real-time file uploads or attachments.
* Users are limited to text-based communication and code sharing.

**6. Basic Encryption**

* Although messages are encrypted using AES, end-to-end encryption (E2EE) is not fully implemented.
* Data security can be further enhanced with advanced encryption strategies.

**7. Browser Compatibility**

* Some advanced features like the embedded code editor may not work smoothly across all browsers, especially older versions.

**8. AI Accuracy and Context**

* AI responses are sometimes generic and may not fully understand the project-specific context or collaborative history.

**Future Enhancements:**

To make ThinkSync more powerful and production-ready, the following enhancements are proposed:

**1. Implement Role-Based Access Control (RBAC)**

* Introduce Admin, Moderator, and Viewer roles.
* Admins can manage users, settings, and permissions for each project.

**2. Develop Mobile Applications**

* Build Android and iOS apps using React Native or Flutter.
* Enable push notifications and mobile-first UI enhancements.

**3. Integrate File and Media Sharing**

* Allow users to upload and preview documents, images, and videos within chat.
* Add cloud storage options like AWS S3 or Firebase.

**4. End-to-End Encryption**

* Secure messages so only the sender and receiver can decrypt them.
* Even server-side processes cannot read the message content.

**5. AI Memory & Personalization**

* Enhance AI capabilities by maintaining per-user or per-project memory.
* Train custom AI agents with user or project-specific data for more accurate results.

**6. Advanced Notification System**

* Real-time browser and mobile notifications.
* Optional email alerts for project activity, deadlines, or AI-suggested tasks.

**7. Git-like Code Collaboration**

* Allow collaborative code editing with syntax highlighting, version control, and execution logs.
* Add integration with GitHub or GitLab for live updates.

**8. Scalability Optimization**

* Deploy on scalable infrastructure using Docker, Kubernetes, or serverless platforms.
* Add caching (e.g., Redis), message queuing (e.g., RabbitMQ), and monitoring tools.

**9. UI/UX Improvements**

* Enhance chat interface with reactions, threads, pinned messages, and mentions.
* Provide dark/light mode themes.

**10. Automated Testing Suite**

* Introduce CI/CD pipelines with Jest, Cypress, and other testing frameworks.
* Ensure 90%+ code coverage for robustness and security.