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:

Rajwinder Kaur

Submitted By:

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

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

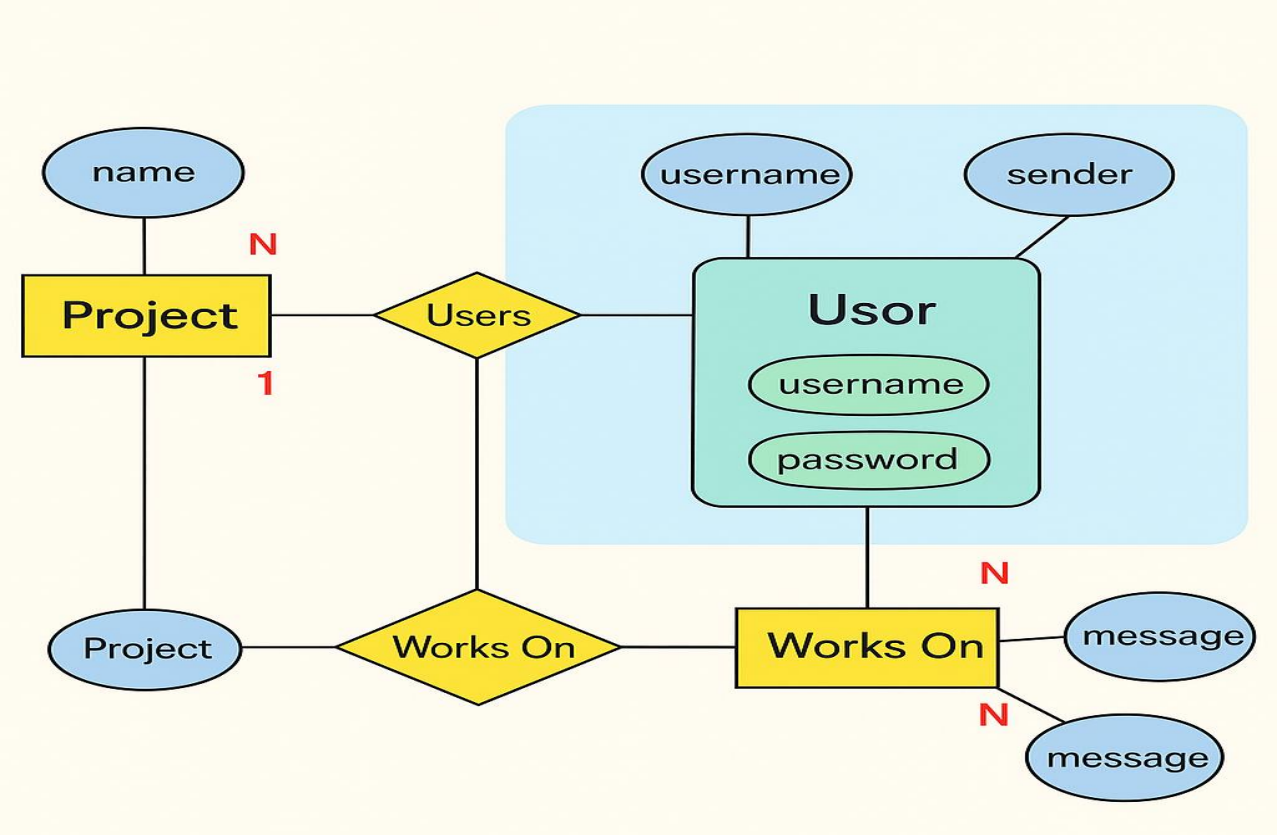
The Design Phase of the ThinkSync project focuses on planning the structure and interactions of the system's components before actual development. It ensures the system is logically organized, scalable, secure, and maintainable. The design covers the major modules, user interactions, and database schema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

1. Basic Modules and Descriptions

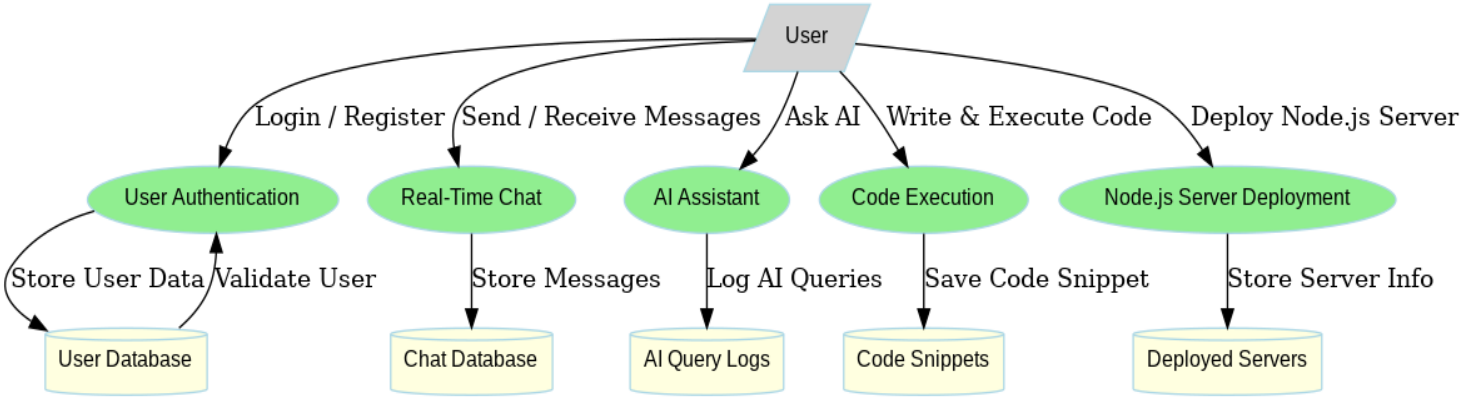
Module Name	Description
User Management	Handles user registration, login, JWT authentication, password hashing, etc.

Module Name	Description
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

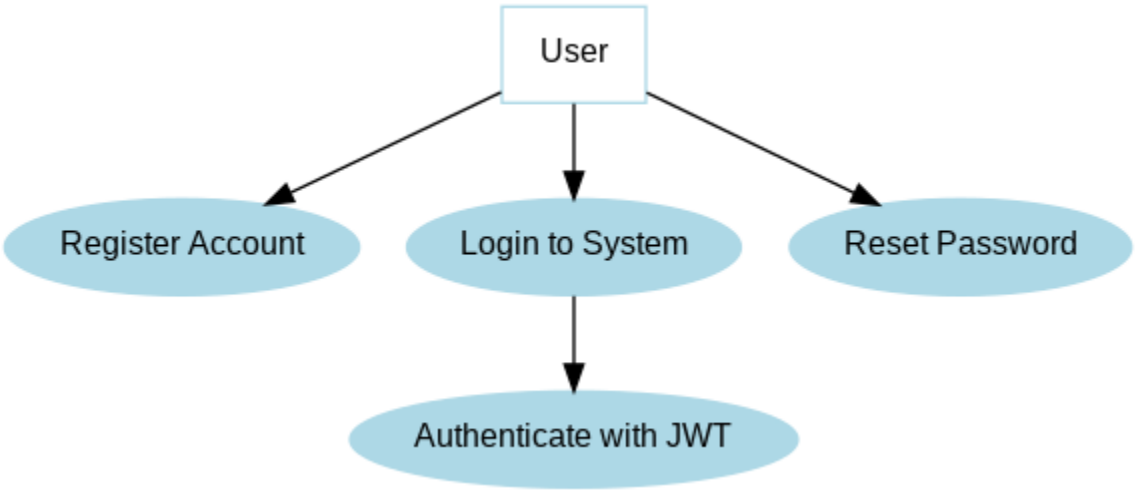


3. DFDs

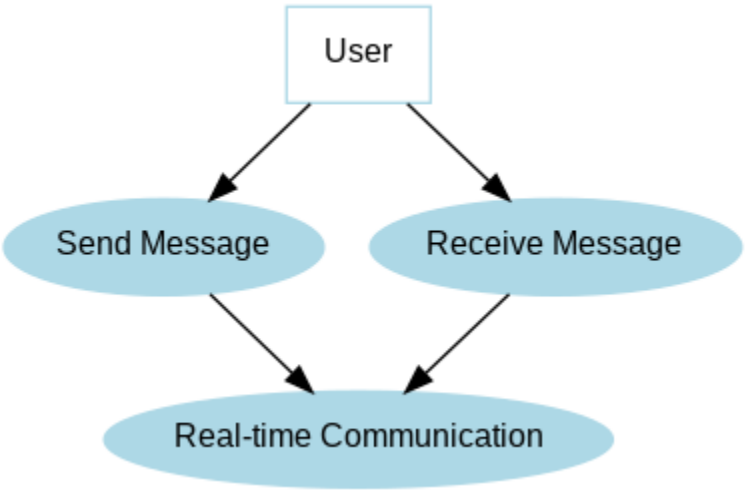


4. Use Case Diagram

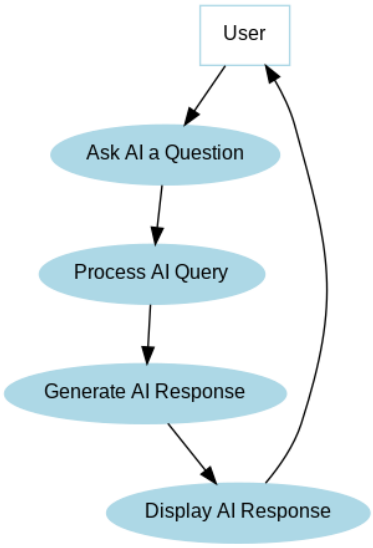
1. User Authentication Use Case Diagram



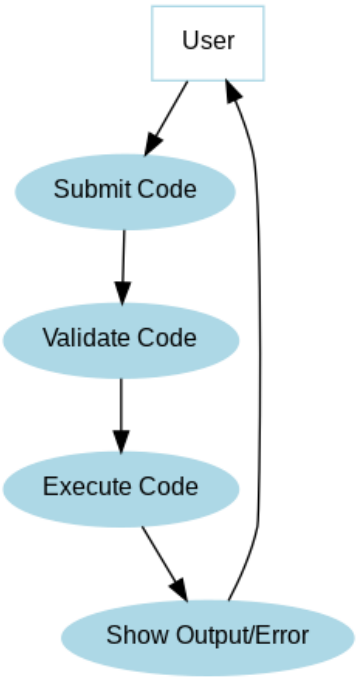
2. Chat System Use Case Diagram



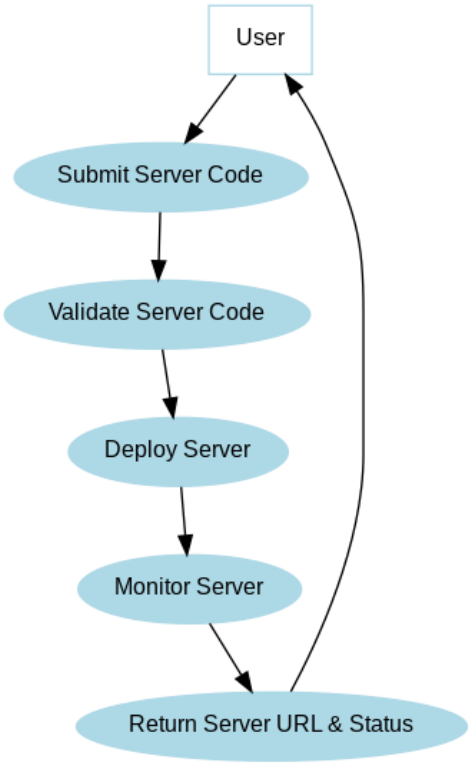
3. AI Assistant Use Case Diagram



4. Code Execution Use Case Diagram

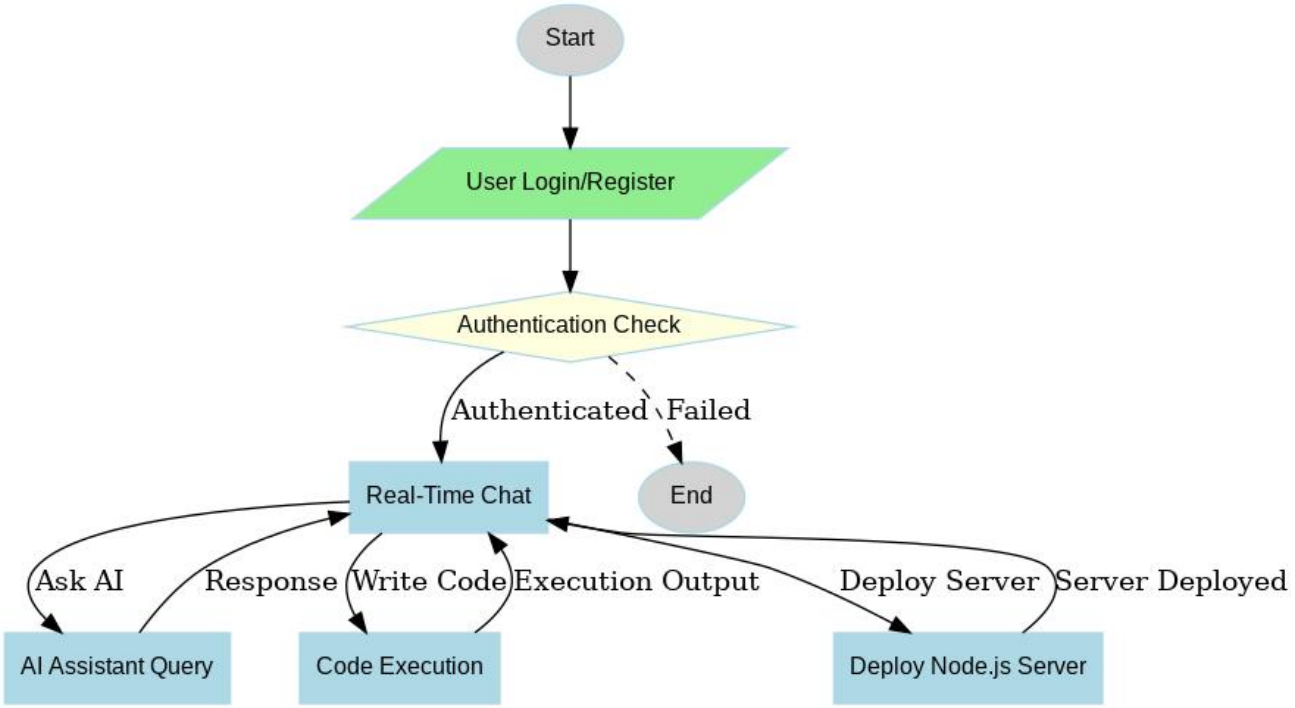


5. Server Deployment Use Case Diagram

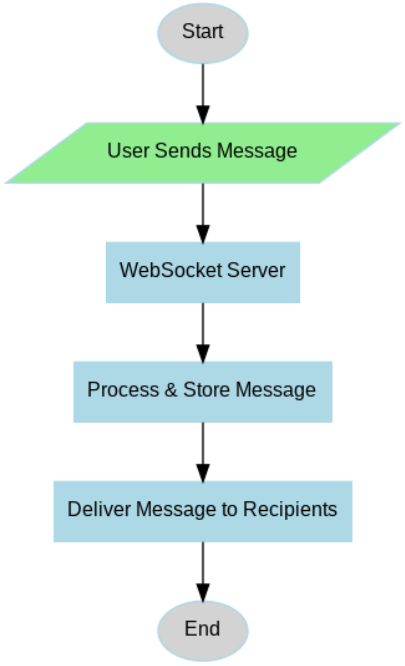


5. Flowcharts

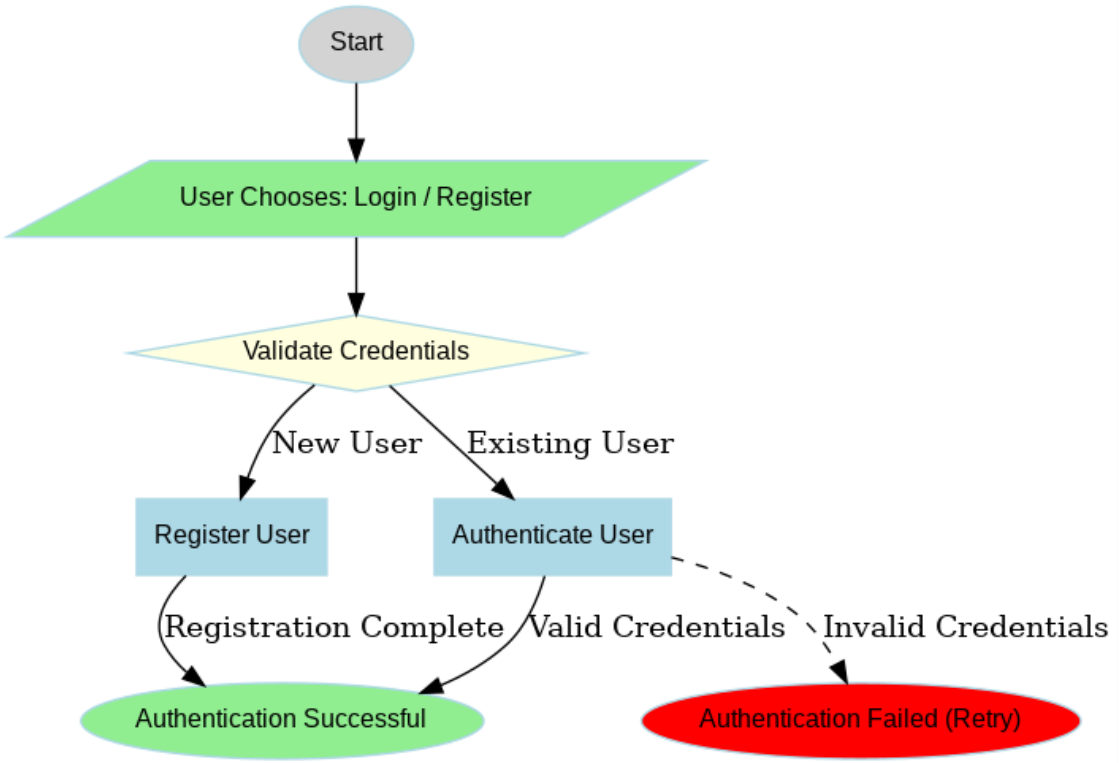
Complete system Flowchart



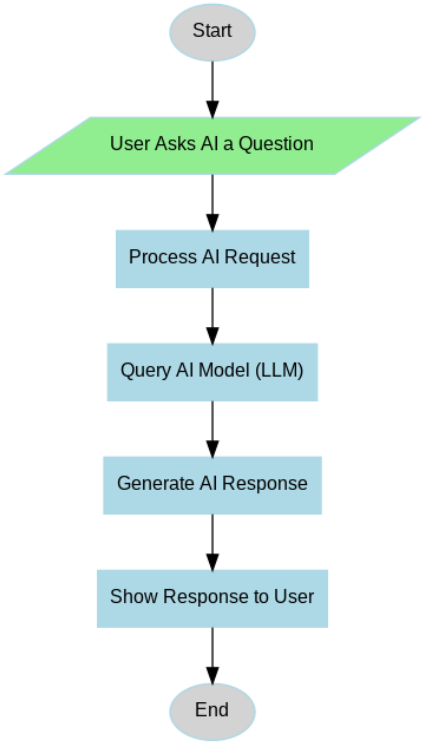
Chat Flowchart



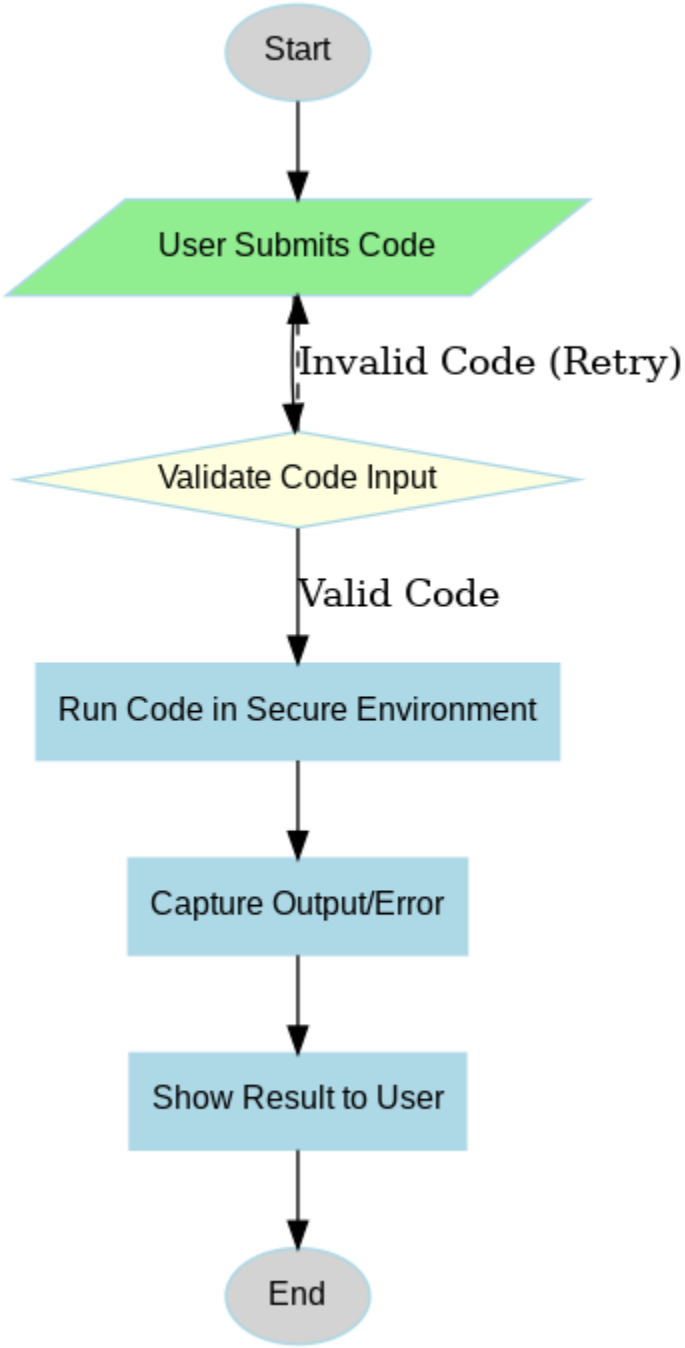
Authentication Flowchart



Ai Interaction flowchart



Code Execution Flowchart



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)

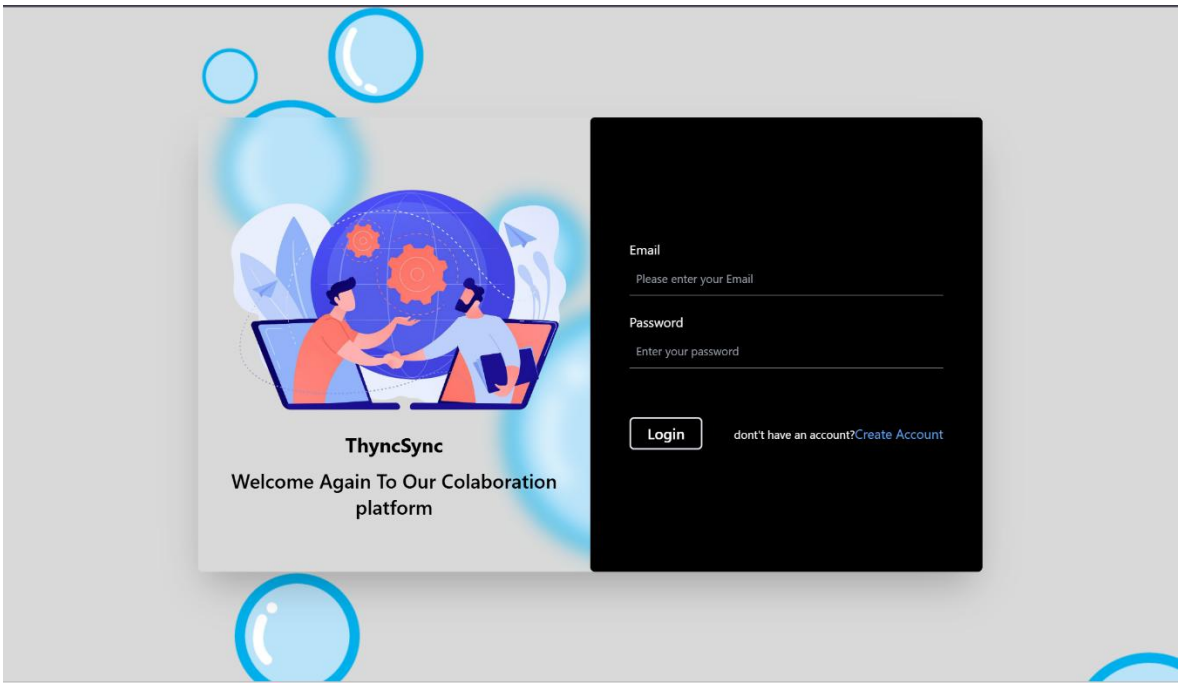
3. Messages Collection

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

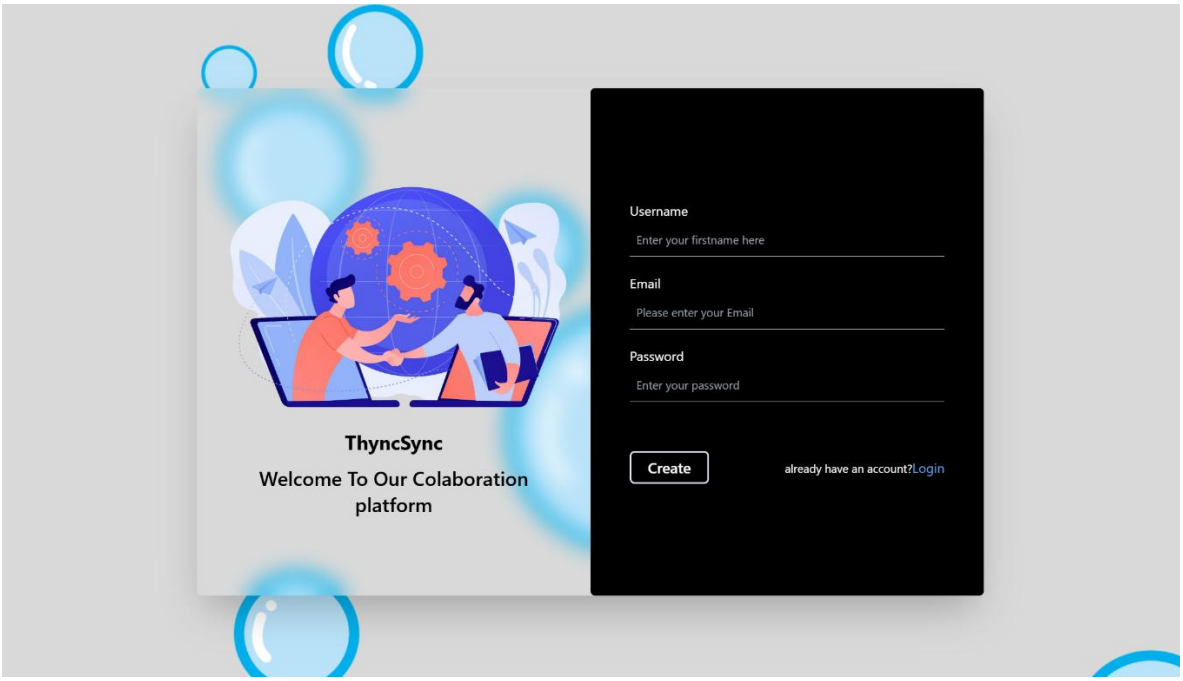
Coding Phase

Snapshots

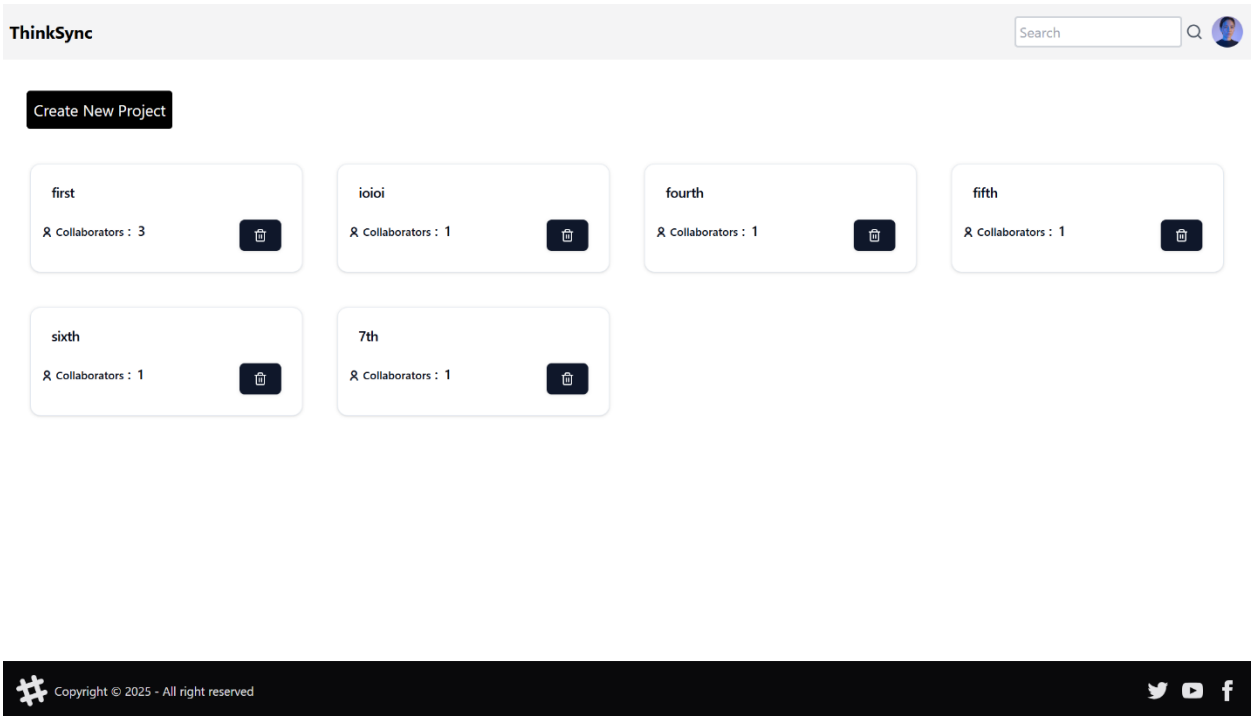
Login



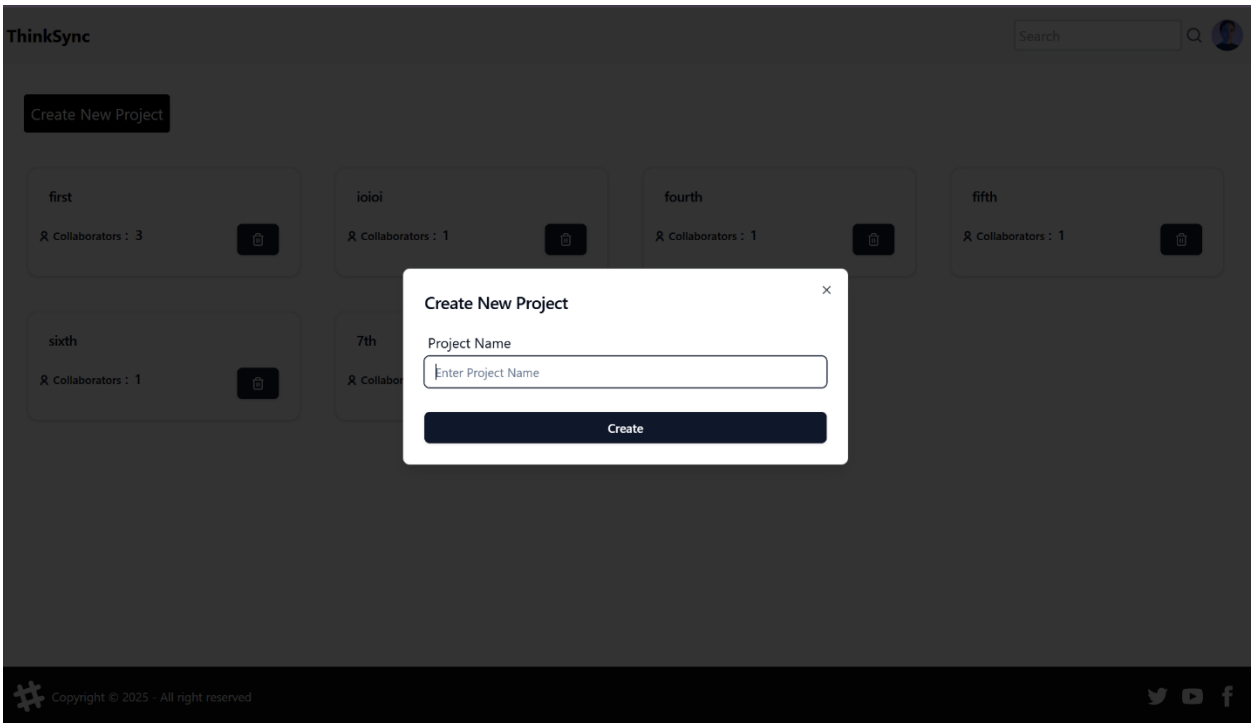
Register



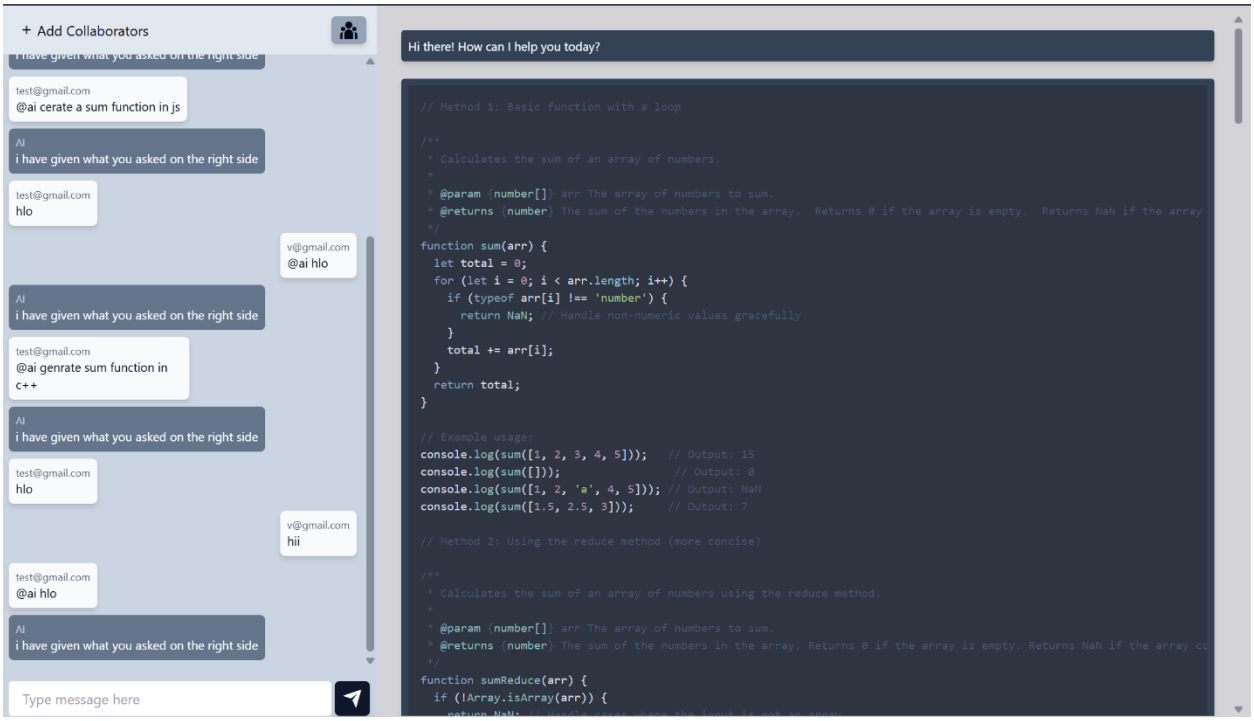
Home Page



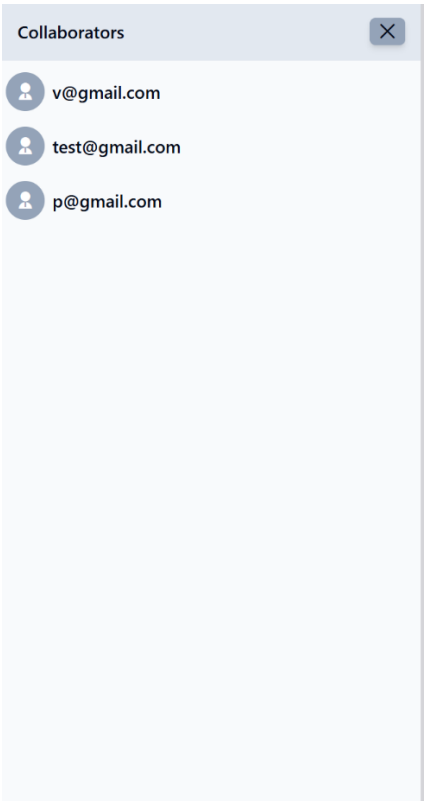
Create Project Modal



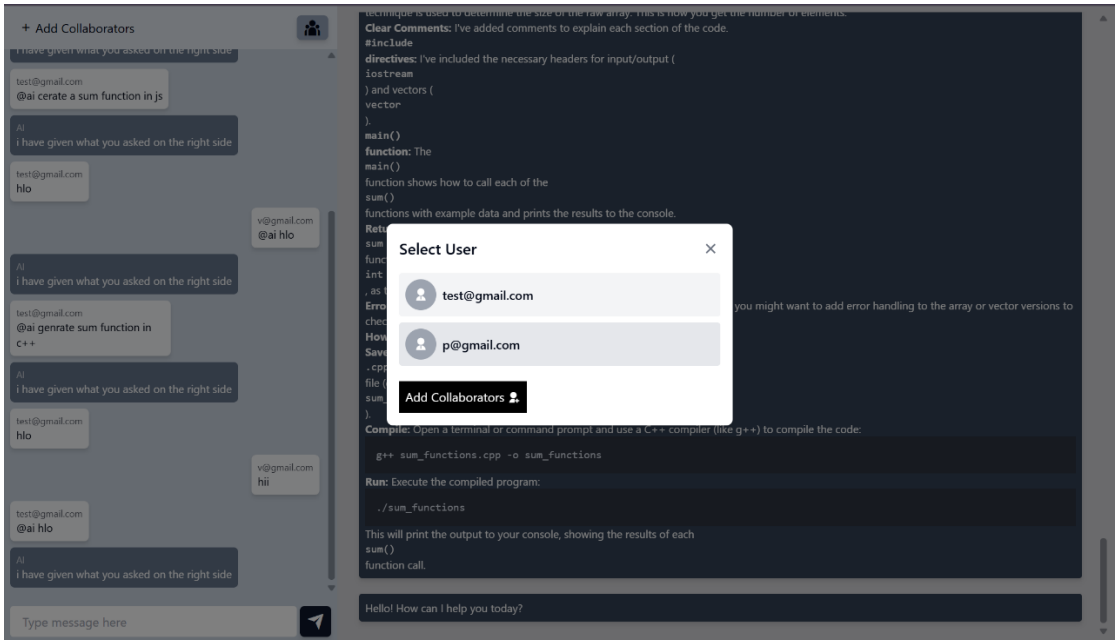
Chat Pannel



Collaborators Members Panel

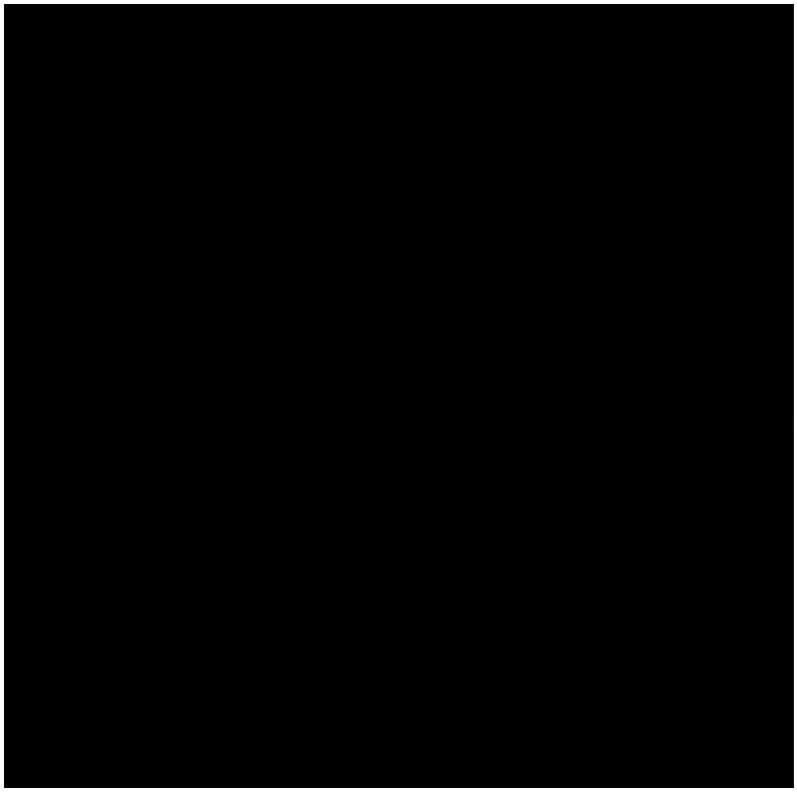


Add collaborators Modal

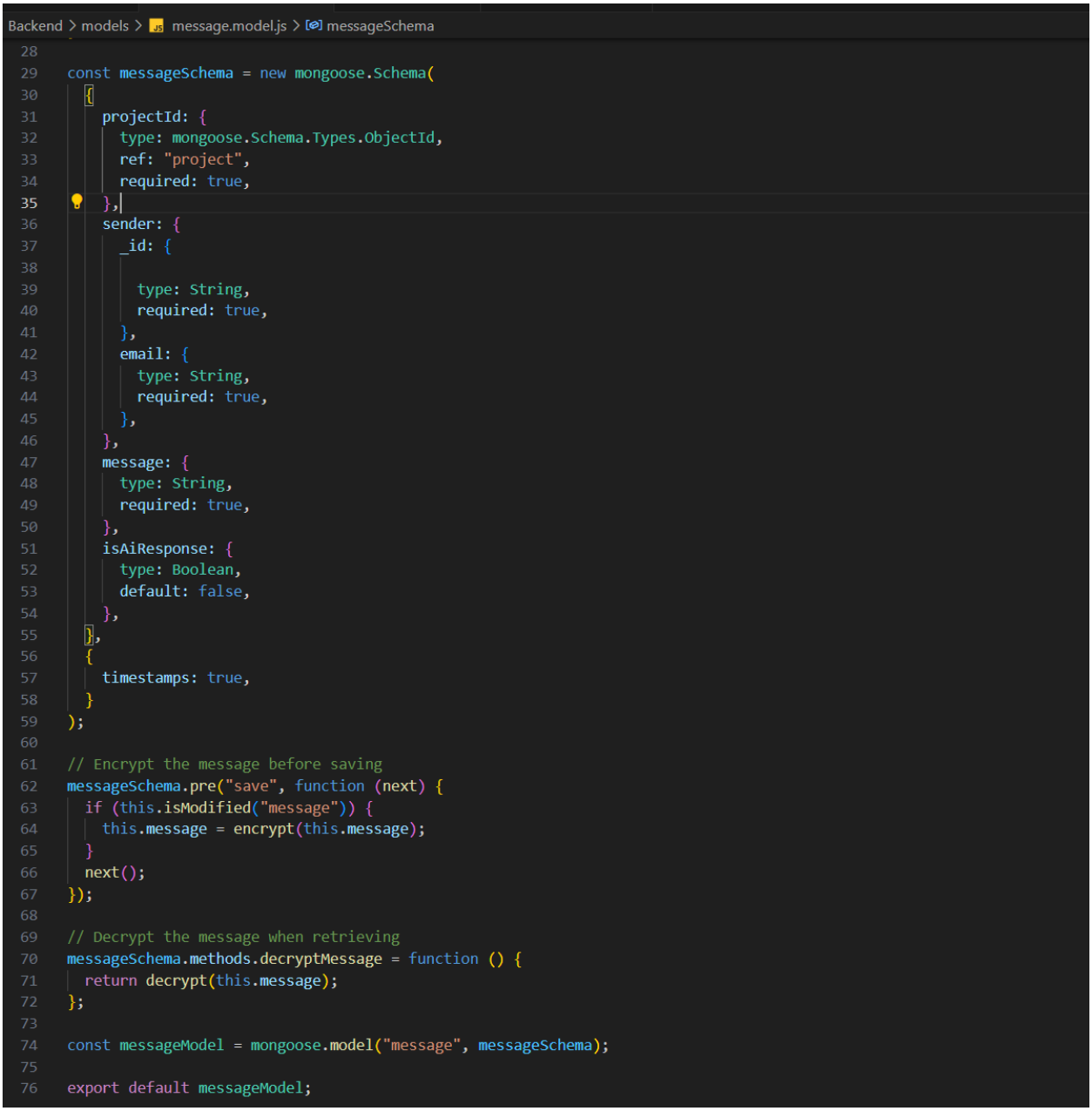


Code snippets

Backend: User Model (Schema)



Message Model (Schema)



Project Model (Schemaa)

Backend > models > project.model.js > default

```
import mongoose from "mongoose";

const projectSchema = new mongoose.Schema({
    name: {
        type: String,
        lowercase: true,
        required: true,
        trim: true,
        unique: [true,'project name must be unique'],
    },

    users:[
        {
            type: mongoose.Schema.Types.ObjectId,
            ref:'user'
        }
    ]

});


const projectModel = mongoose.model('project', projectSchema)
export default projectModel;
```

Auth Middleware (Authentication)

Backend > middlewares > user.auth.middleware.js > ...

```
import jwt from "jsonwebtoken";
import userModel from "../models/user.model.js";
import redisClient from "../services/redis.service.js";
export const isLogin = async (req, res, next) => {
  const token = req.cookies.token || req.headers.authorization?.split(" ")[1];

  // console.log(token, "token")

  if (!token) {
    return res.status(401).json({ message: "Unauthorized: No token provided" });
  }

  const isBlacklisted = await redisClient.get(token);

  if (isBlacklisted) {
    res.cookie(token, "");
    return res
      .status(401)
      .json({ message: "Unauthorized: Token is blacklisted" });
  }

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const user = await userModel.findById(decoded.id);

    if (!user) {
      return res.status(401).json({ message: "Unauthorized: User not found" });
    }
    req.user = user;
    return next();
  } catch (error) {
    console.error("Error in isLogin middleware:", error);
    return res.status(401).json({ message: "Unauthorized", error });
  }
};
```

Ai Service code

```
        }
      },
      "package.json": {
        "file": {
          "contents": "
          {
            \"name\": \"temp-server\",
            \"version\": \"1.0.0\",
            \"main\": \"app.js\",
            \"scripts\": {
              \"start\": \"node app.js\",
              \"test\": \"echo \\\"Error: no test specified\\\" && exit 1\"
            },
            \"dependencies\": {
              \"express\": \"^4.21.2\"
            }
          }
          "
        }
      }
    },
    "buildCommand": {
      "mainItem": "npm",
      "commands": ["install"]
    },
    "startCommand": {
      "mainItem": "npm",
      "commands": ["start"]
    }
  }

  user: Create an express application

  </example>

  <example>

  user: Hello
  response: {
    "text": "Hello, How can I help you today?"
  }

  </example>

  IMPORTANT:
  - Always return responses in valid JSON format.
  - Include "fileTree", "buildCommand", and "startCommand" in every response.
  - Write modular and maintainable code with comments.
  - Do not include unnecessary explanations or examples in the response.
  - Ensure the "fileTree" contains multiple files like "app.js", "src/routes.js", and "package.json".
```

```
    IMPORTANT:
    - Always return responses in valid JSON format.
    - Include "fileTree", "buildCommand", and "startCommand" in every response.
    - Write modular and maintainable code with comments.
    - Do not include unnecessary explanations or examples in the response.
    - Ensure the "fileTree" contains multiple files like "app.js", "src/routes.js", and "package.json".

    User prompt: ${prompt}
    `;

    const response = await ai.models.generateContent({
      model: "gemini-2.0-flash",
      contents: prompt,
      systemInstructions,
      generationConfig: {
        responseMimeType: "application/json",
        temperature: 0.4,
      },
    });

    return response.text;
  }

  async function result(prompt) {
    return await main(prompt);
  }

  export default result;
```

Project Service

Backend > services > project.service.js > ...

```
import mongoose from "mongoose";
import projectModel from "../models/project.model.js";
import userModel from "../models/user.model.js";

export const createProject = async ({ name, userId }) => {
  if (!name) {
    throw new Error("Name is required");
  }
  if (!userId) {
    throw new Error("User id is required");
  }
  try {
    const project = await projectModel.create({
      name,
      users: [userId],
    });

    return project;
  } catch (error) {
    console.error("Error creating project:", error);
    if (error.message.includes("project name must be unique")) {
      throw new Error("project name must be unique");
    }

    throw new Error("Internal Server Error. Please try again later.");
  }
};

export const getAllProjectByUserId = async ({ loggedInuserId }) => {
  if (!loggedInuserId) {
    throw new Error("User id is required");
  }

  try {
    const allUserProjects = await projectModel.find({
      users: loggedInuserId,
    });

    return allUserProjects;
  } catch (error) {
    console.error("Error fetching projects:", error);
    throw new Error("Internal Server Error. Please try again later.");
  }
};

export const addUsersToProject = async ({ projectId, users, userId }) => {
  if (!projectId) {
    throw new Error("projectId is required");
  }
```

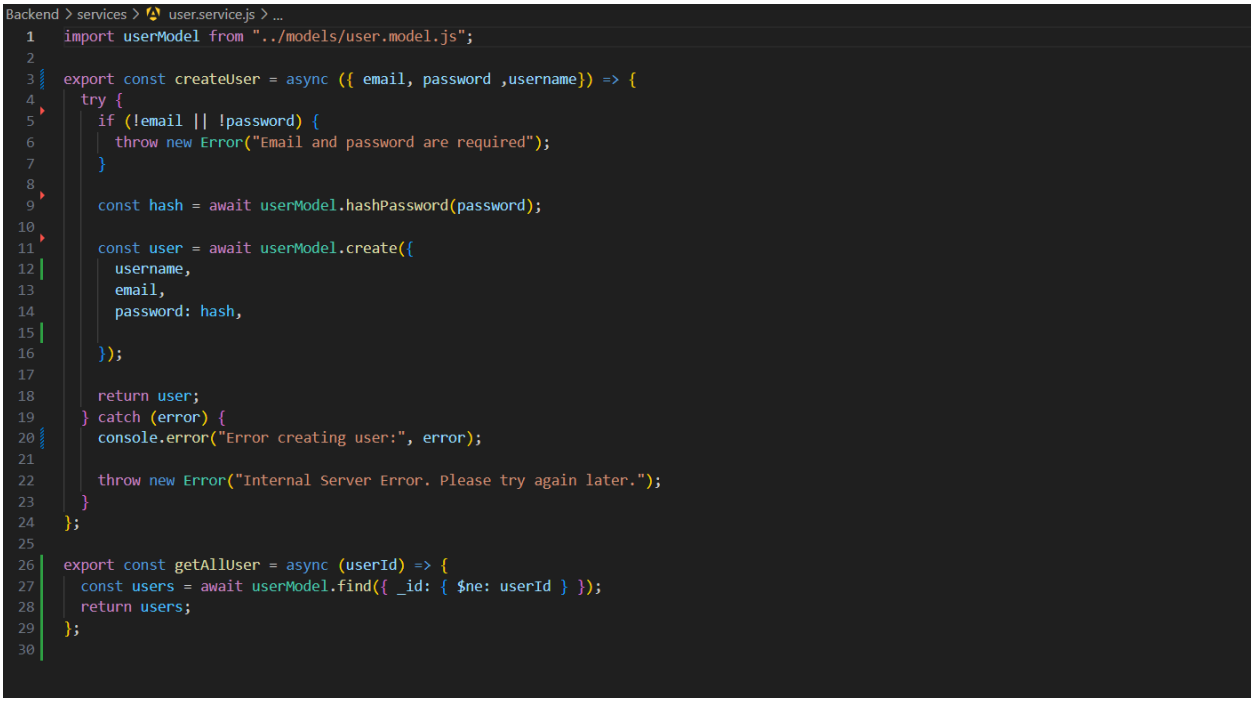
Backend > services > project.service.js > ...

```
50
51  if (!userId) {
52    throw new Error("userId is required");
53  }
54
55  if (
56    !mongoose.Types.ObjectId.isValid(projectId) ||
57    !mongoose.Types.ObjectId.isValid(userId)
58  ) {
59    throw new Error("invailid projectId or userid");
60  }
61
62  if (!users) {
63    throw new Error("Users are required");
64  }
65  if (
66    !Array.isArray(users) ||
67    users.some((userId) => {
68      if (!mongoose.Types.ObjectId.isValid(userId)) {
69        throw new Error("Invalid userid");
70      }
71      return false;
72    })
73  ) {
74    throw new Error("Invalid users array");
75  }
76  try {
77    const project = await projectModel.findOne({
78      _id: projectId,
79      users: userId,
80    });
81
82    if (!project) {
83      throw new Error("user not belong to the this poroject");
84    }
85
86    const updatedProject = await projectModel.findOneAndUpdate(
87      {
88        _id: projectId,
89      },
90      {
91        $addToSet: {
92          users: {
93            $each: users,
94          },
95        },
96      },
97      { new: true }
98    );
99
100   return updatedProject;
```

Backend > services > project.service.js > ...

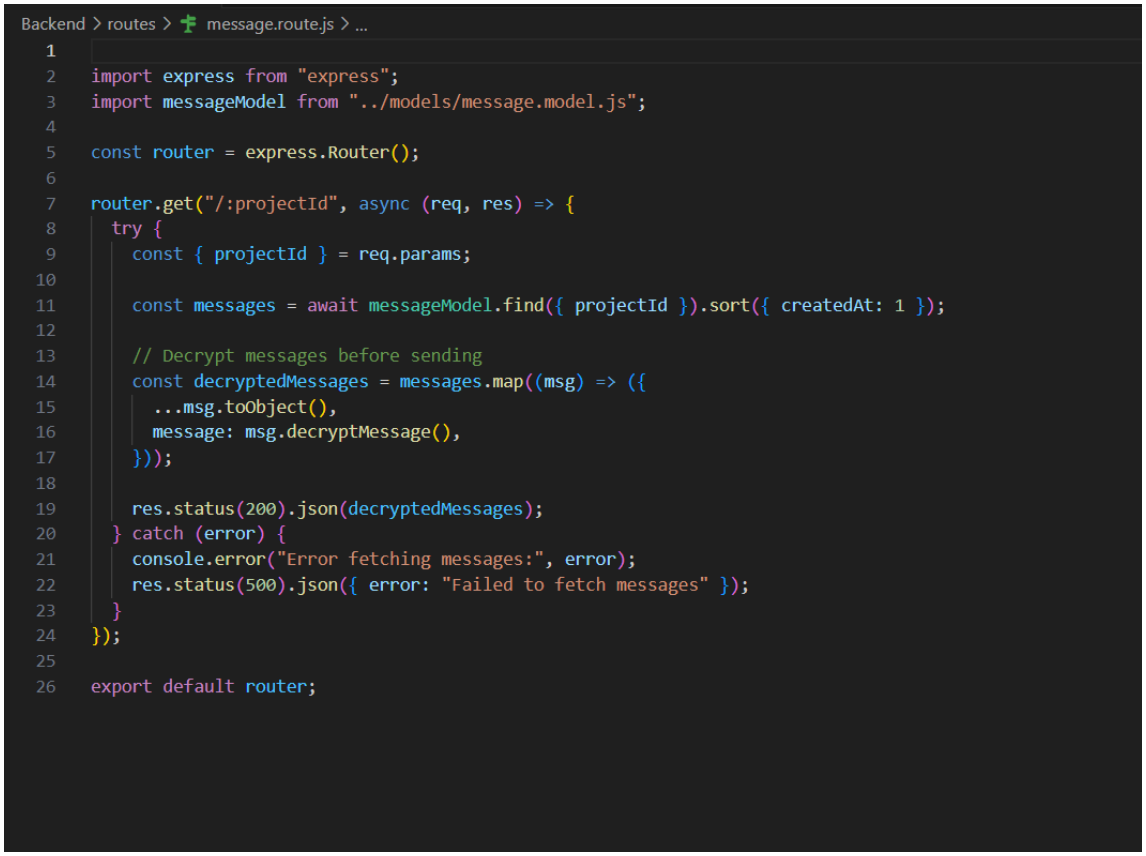
```
119       })
120       .populate("users");
121
122     return project;
123   };
124
125
126   export const deleteProject = async ({ projectId, userId }) => {
127     if (!projectId) {
128       throw new Error("Project ID is required");
129     }
130
131     if (!mongoose.Types.ObjectId.isValid(projectId)) {
132       throw new Error("Invalid project ID");
133     }
134
135     try {
136       const project = await projectModel.findOne({ _id: projectId });
137
138       if (!project) {
139         throw new Error("Project not found");
140       }
141
142       // Check if the user is authorized to delete the project
143       if (!project.users.includes(userId)) {
144         throw new Error("Unauthorized");
145       }
146
147       await projectModel.deleteOne({ _id: projectId });
148     } catch (error) {
149       console.error("Error in deleteProject:", error);
150       throw new Error("Internal Server Error. Please try again later.");
151     }
152   };
```

User Service



Ai Routes

Message Routes



```
Backend > routes > message.route.js > ...

import express from "express";
import messageModel from "../models/message.model.js";

const router = express.Router();

router.get("/:projectId", async (req, res) => {
  try {
    const { projectId } = req.params;

    const messages = await messageModel.find({ projectId }).sort({ createdAt: 1 });

    // Decrypt messages before sending
    const decryptedMessages = messages.map((msg) => ({
      ...msg.toObject(),
      message: msg.decryptMessage(),
    }));

    res.status(200).json(decryptedMessages);
  } catch (error) {
    console.error("Error fetching messages:", error);
    res.status(500).json({ error: "Failed to fetch messages" });
  }
});

export default router;
```

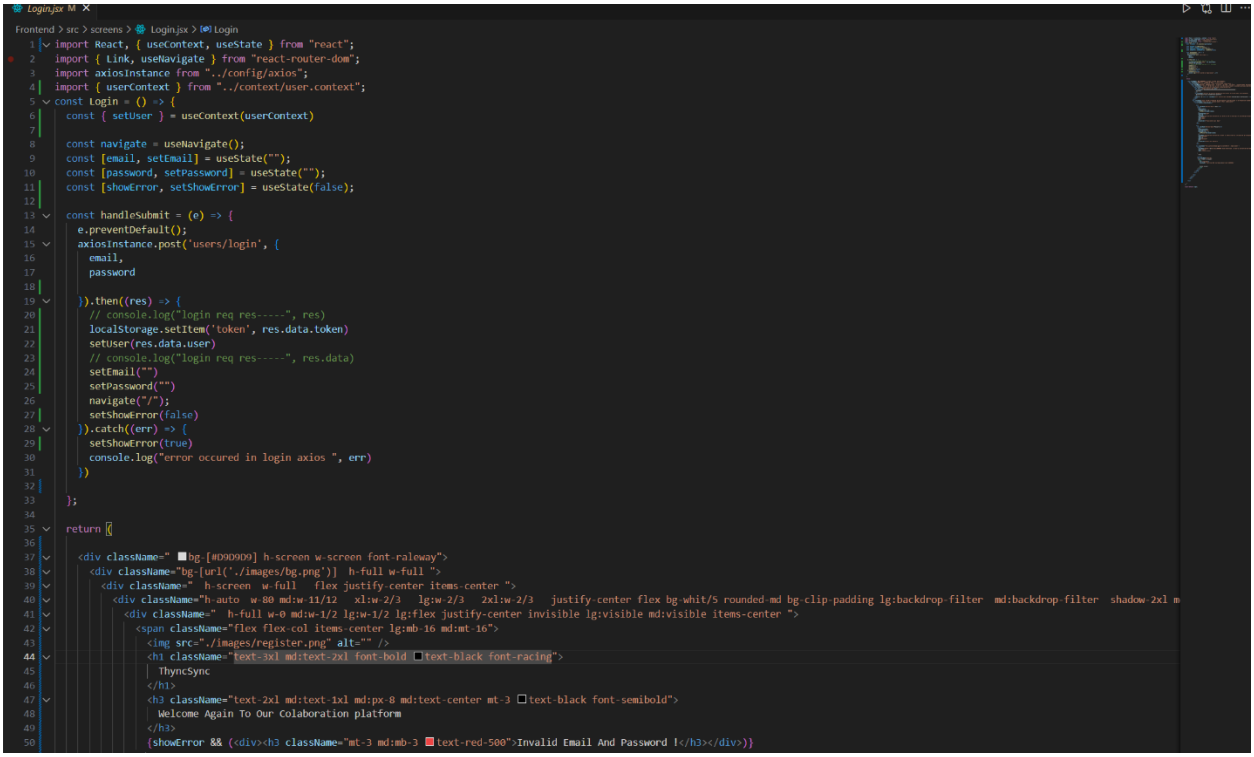
Project Routes

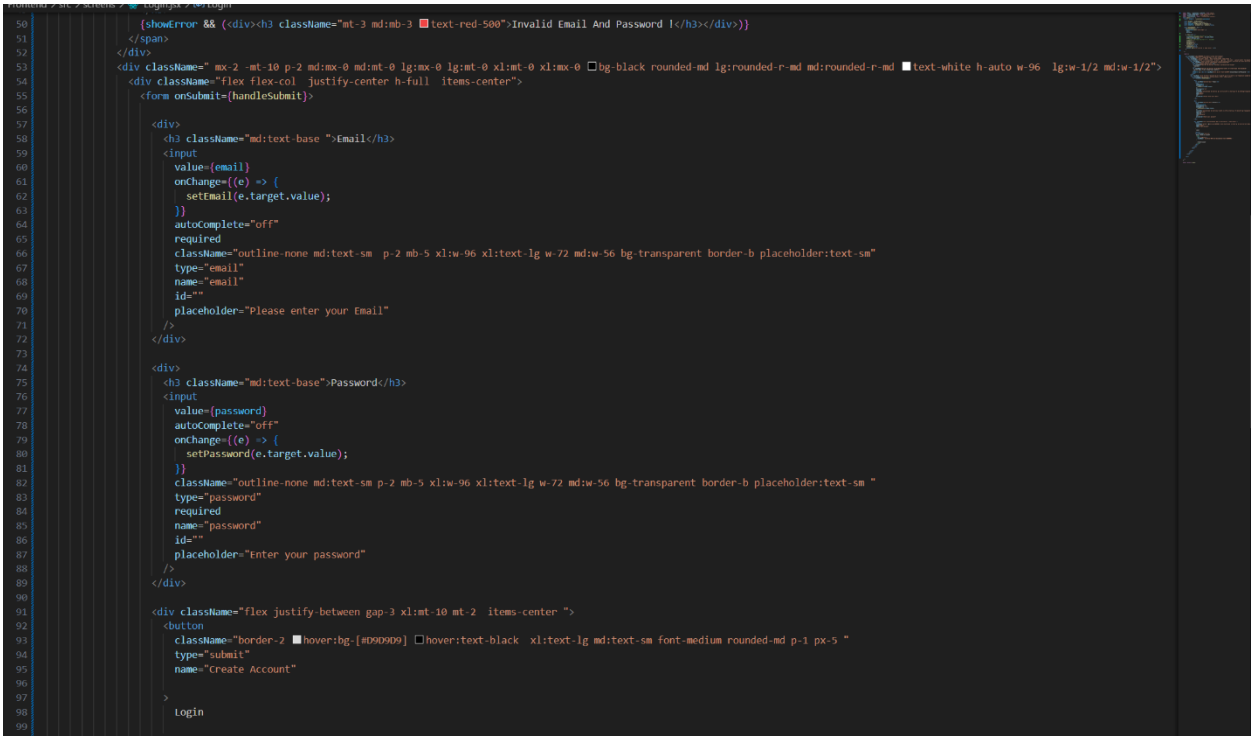
```
Backend > routes > project.routes.js > ...

import { Router } from "express";
import { validateProject, validateProjectUsers } from "../validators/projectValidator.js";
import * as projectController from "../controllers/project.controller.js";
import * as userAuthMiddleware from "../middlewares/user.auth.middleware.js";
const router = Router();

router.post('/create-project',validateProject , userAuthMiddleware.isLogin, projectController.createProjectController);
router.get('/get-projects',userAuthMiddleware.isLogin, projectController.getProjectController);
router.put('/add-user',validateProjectUsers, userAuthMiddleware.isLogin, projectController.addUserToProjectController);
router.get('/get-project/:projectId', userAuthMiddleware.isLogin, projectController.getProjectByIdController);
router.delete('/delete-project/:projectId', userAuthMiddleware.isLogin, projectController.deleteProjectController);

export default router;
```

User Routes

```
Backend > routes > user.routes.js > ...
1  import { Router } from "express";
2  import * as userController from "../controllers/user.controller.js";
3  import { validateUser, loginValidator } from "../validators/userValidator.js";
4  import * as userAuthMiddleware from "../middlewares/user.auth.middleware.js";
5
6  const router = Router();
7
8  router.post("/register", validateUser, userController.createUserController);
9  router.post("/login", loginValidator , userController.loginUserController);
10 router.get("/profile", userAuthMiddleware.isLogin,  userController.getProfileUserController);
11 router.get("/logout", userAuthMiddleware.isLogin,  userController.logoutUserController);
12 router.get("/all", userAuthMiddleware.isLogin,  userController.getAllUserController);
13
14 export default router;
15
```

Frontend : Login Page



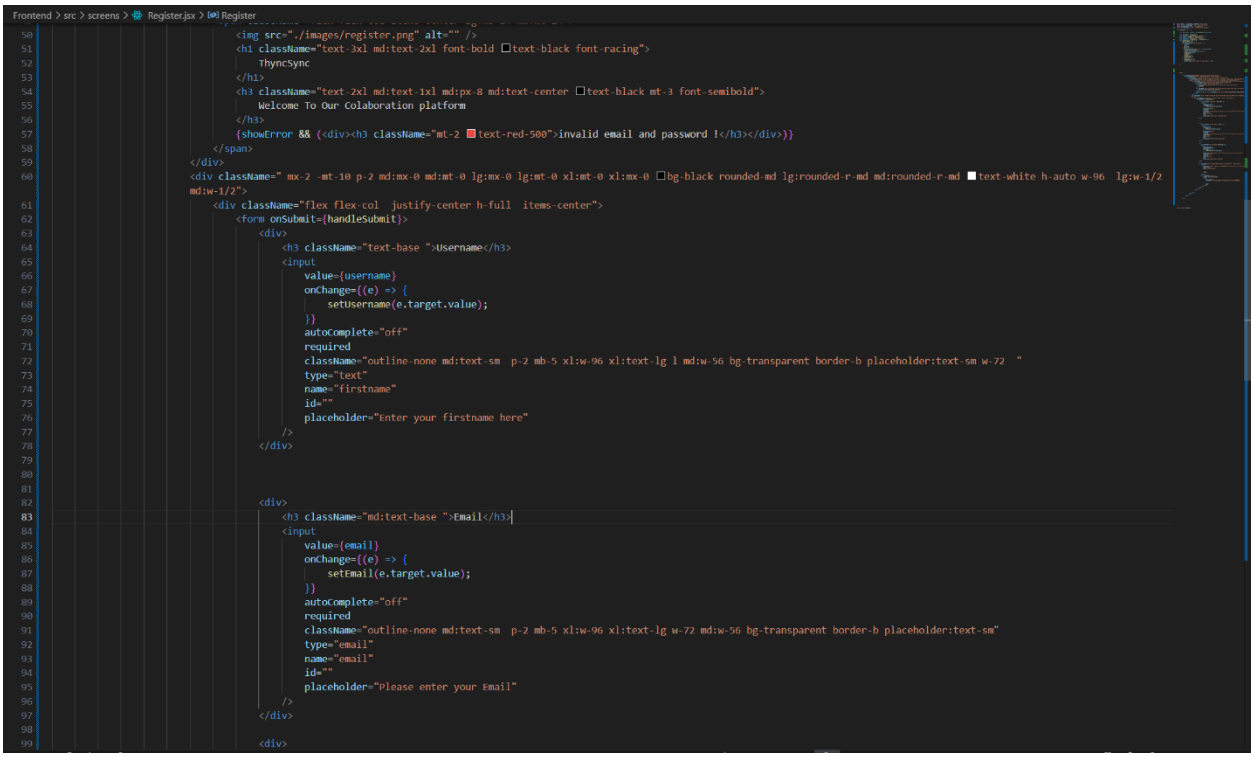


```
                {showError && (<div><h3 className="mt-3 md:mb-3 text-red-500">Invalid Email And Password !</h3></div>)}
              </span>
            </div>
            <div className=" mx-2 -mt-10 p-2 md:mx-0 md:mt-0 lg:mx-0 lg:mt-0 xl:mt-0 xl:mx-0 bg-black rounded-md lg:rounded-r-md md:rounded-r-md text-white h-auto w-96  lg:w-1/2 md:w-1/2">
              <div className="flex flex-col  justify-center h-full  items-center">
                <form onSubmit={handleSubmit}>

                  <div>
                    <h3 className="md:text-base ">Email</h3>
                    <input
                      value={email}
                      onChange={(e) => {
                        setEmail(e.target.value);
                      }}
                      autoComplete="off"
                      required
                      className="outline-none md:text-sm  p-2 mb-5 xl:w-96 xl:text-lg w-72 md:w-56 bg-transparent border-b placeholder:text-sm"
                      type="email"
                      name="email"
                      id=""
                      placeholder="Please enter your Email"
                    />
                  </div>

                  <div>
                    <h3 className="md:text-base">Password</h3>
                    <input
                      value={password}
                      autoComplete="off"
                      onChange={(e) => {
                        setPassword(e.target.value);
                      }}
                      className="outline-none md:text-sm p-2 mb-5 xl:w-96 xl:text-lg w-72 md:w-56 bg-transparent border-b placeholder:text-sm "
                      type="password"
                      required
                      name="password"
                      id=""
                      placeholder="Enter your password"
                    />
                  </div>

                  <div className="flex justify-between gap-3 xl:mt-10 mt-2  items-center ">
                    <button
                      className="border-2 hover:bg-[#D9D9D9] hover:text-black  xl:text-lg md:text-sm font-medium rounded-md p-1 px-5 "
                      type="submit"
                      name="create Account"

                    >
                      Login
```

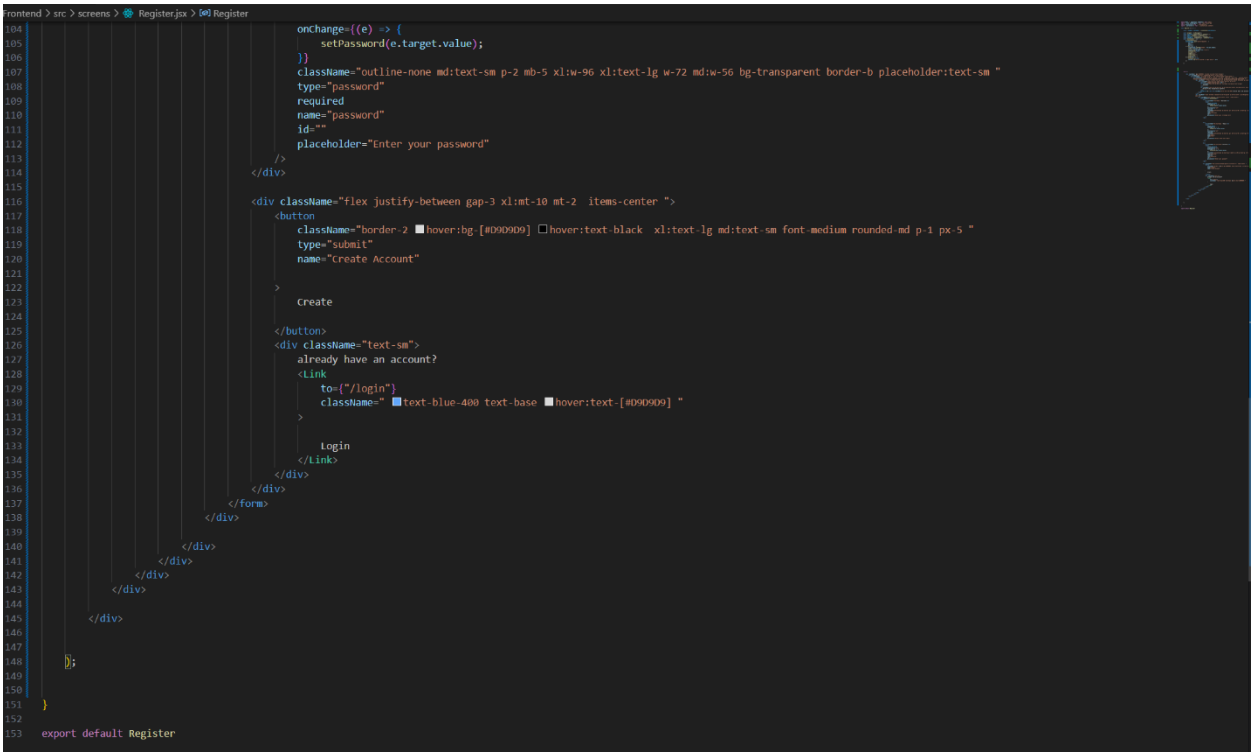
```
Frontend > src > screens > Login.jsx > Login
                    />
                  </div>

                  <div className="flex justify-between gap-3 xl:mt-10 mt-2  items-center ">
                    <button
                      className="border-2 hover:bg-[#D9D9D9] hover:text-black  xl:text-lg md:text-sm font-medium rounded-md p-1 px-5 "
                      type="submit"
                      name="Create Account"

                    >
                      Login

                    </button>
                    <div className="text-sm">
                      dont't have an account?
                      <Link
                        to={"/register"}
                        className=" text-blue-400 text-base hover:text-[#D9D9D9] "
                      >

                        Create Account
                      </Link>
                    </div>
                  </div>
                </form>
              </div>

            </div>
          </div>
        </div>
      </div>

    </div>

  );
};

export default Login;
```

Register Page

```jsx
import React, { useContext, useState } from 'react'
import { Link, useNavigate } from 'react-router-dom'
import axiosInstance from '../config/axios';
import { userContext } from '../context/user.context';

const Register = () => {

    const { setUser, setToken } = useContext(userContext);

    const navigate = useNavigate();
    const [username, setUsername] = useState("");
    const [email, setEmail] = useState("");
    const [password, setPassword] = useState("");
    const [showError, setShowError] = useState(false);
    const handleSubmit = (e) => {
        e.preventDefault();
        axiosInstance.post('users/register', {
            username,
            email,
            password
        }).then((res) => {
            localStorage.setItem("token", res.data.token);
            setUser(res.data.user)
            console.log(res.data.user,"first")
            setShowError(false)
            navigate("/");
            setEmail("")
            setUsername("")
            setPassword("")
        }).catch((err) => {
            setShowError(true)
            console.log("error occured in login axios ", err)
        })

    };



    return (

        <div className=" bg-[#D9D9D9] h-screen w-screen font-raleway">
            <div className="bg-[url('./images/bg.png')]  h-full w-full  ">
                <div className="  h-screen  w-full   flex justify-center items-center ">
                    <div className="h-auto  w-80 md:w-11/12   xl:w-2/3   lg:w-2/3   2xl:w-2/3   justify-center flex bg-whit/5 rounded-md bg-clip-padding lg:backdrop-filter  md:backdrop-filter  sh
                        <div className="  h-full w-0 md:w-1/2 lg:w-1/2 lg:flex justify-center invisible lg:visible md:visible items-center ">
                            <span className="flex flex-col items-center lg:mb-24 md:mt-24">
                                <img src="./images/register.png" alt="" />
```

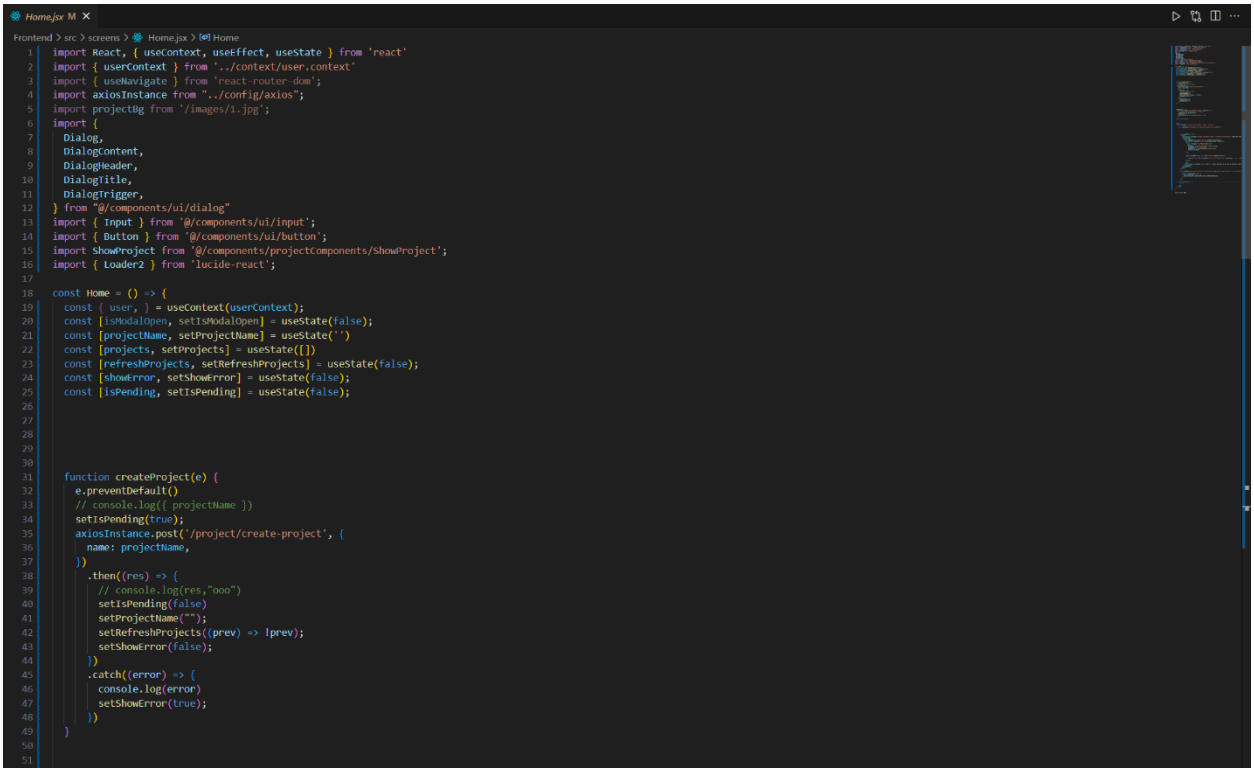


```jsx
                                <h1 className="text-3xl md:text-2xl font-bold text-black font-racing">
                                    ThyncSync
                                </h1>
                                <h3 className="text-2xl md:text-1xl md:px-8 md:text-center text-black mt-3 font-semibold">
                                    Welcome To Our Colaboration platform
                                </h3>
                                {showError && (<div><h3 className="mt-2 text-red-500">invalid email and password !</h3></div>)}
                            </span>
                        </div>
                        <div className=" mx-2 -mt-10 p-2 md:mx-0 md:mt-0 lg:mx-0 lg:mt-0 xl:mt-0 xl:mx-0 bg-black rounded-md lg:rounded-r-md md:rounded-r-md text-white h-auto w-96  lg:w-1/2
                        md:w-1/2">
                            <div className="flex flex-col  justify-center h-full  items-center">
                                <form onSubmit={handleSubmit}>
                                    <div>
                                        <h3 className="text-base ">Username</h3>
                                        <input
                                            value={username}
                                            onChange={(e) => {
                                                setUsername(e.target.value);
                                            }}
                                            autoComplete="off"
                                            required
                                            className="outline-none md:text-sm  p-2 mb-5 xl:w-96 xl:text-lg l md:w-56 bg-transparent border-b placeholder:text-sm w-72  "
                                            type="text"
                                            name="firstname"
                                            id=""
                                            placeholder="Enter your firstname here"
                                        />
                                    </div>



                                    <div>
                                        <h3 className="md:text-base ">Email</h3>
                                        <input
                                            value={email}
                                            onChange={(e) => {
                                                setEmail(e.target.value);
                                            }}
                                            autoComplete="off"
                                            required
                                            className="outline-none md:text-sm  p-2 mb-5 xl:w-96 xl:text-lg w-72 md:w-56 bg-transparent border-b placeholder:text-sm"
                                            type="email"
                                            name="email"
                                            id=""
                                            placeholder="Please enter your Email"
                                        />
                                    </div>

                                    <div>
```

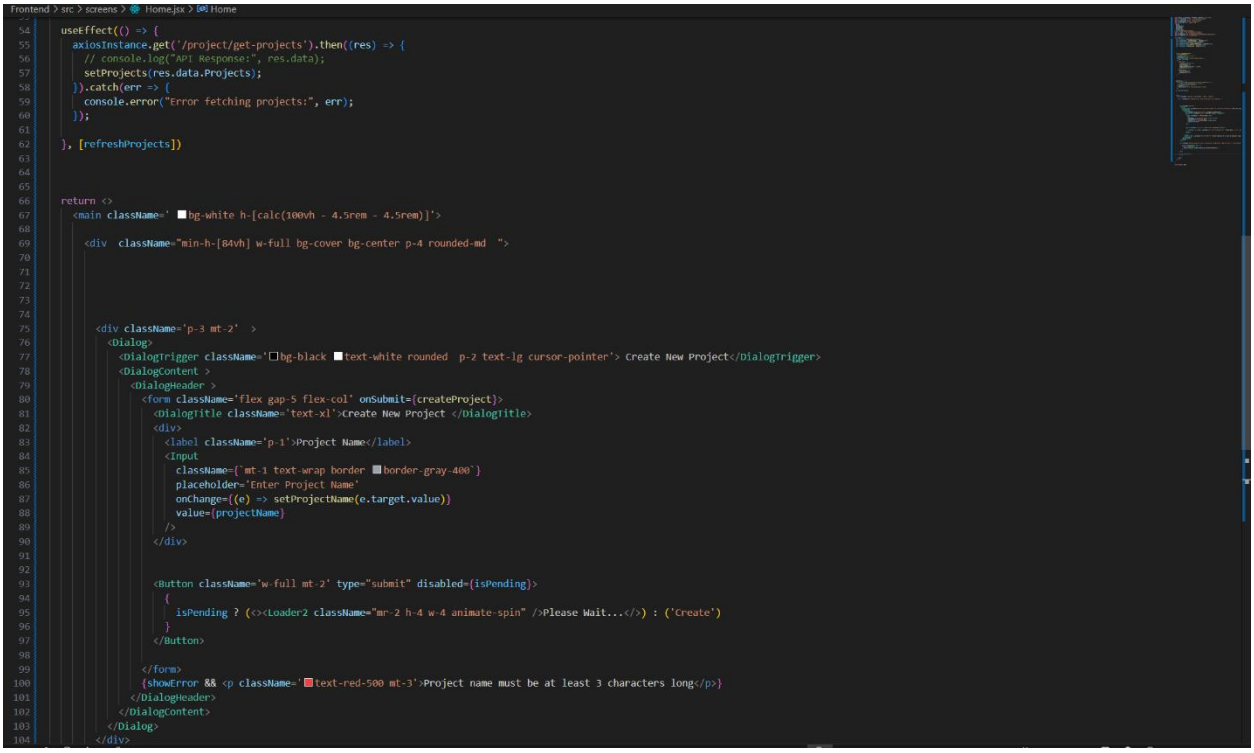


```
Frontend > src > screens > Register.jsx > Register
                                        onChange={(e) => {
                                            setPassword(e.target.value);
                                        }}
                                        className="outline-none md:text-sm p-2 mb-5 xl:w-96 xl:text-lg w-72 md:w-56 bg-transparent border-b placeholder:text-sm "
                                        type="password"
                                        required
                                        name="password"
                                        id=""
                                        placeholder="Enter your password"
                                    />
                                </div>

                                <div className="flex justify-between gap-3 xl:mt-10 mt-2  items-center ">
                                    <button
                                        className="border-2 hover:bg-[#D9D9D9] hover:text-black  xl:text-lg md:text-sm font-medium rounded-md p-1 px-5 "
                                        type="submit"
                                        name="Create Account"

                                    >
                                        Create

                                    </button>
                                    <div className="text-sm">
                                        already have an account?
                                        <Link
                                            to={"/login"}
                                            className=" text-blue-400 text-base hover:text-[#D9D9D9] "
                                        >

                                            Login
                                        </Link>
                                    </div>
                                </div>
                            </form>
                        </div>

                    </div>
                </div>
            </div>
        </div>

    );

}
export default Register
```

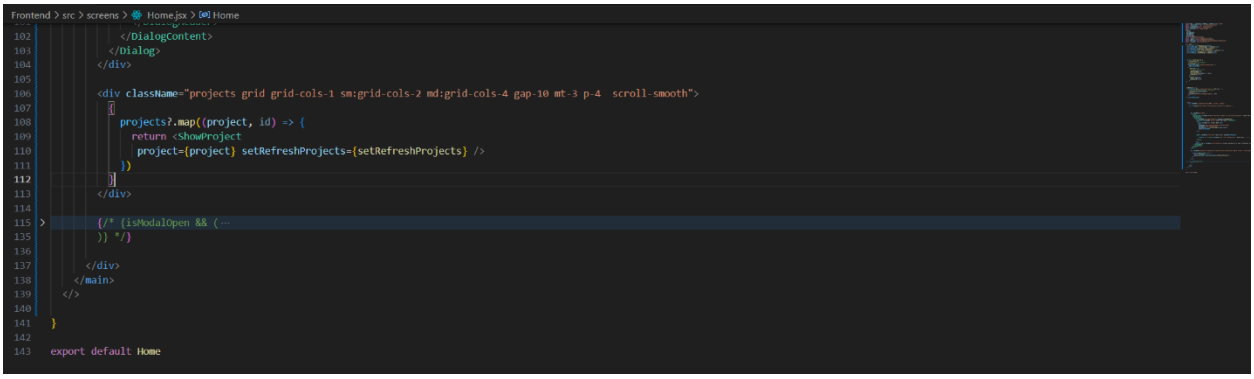
Home Page



```
Home.jsx M
Frontend > src > screens > Home.jsx > Home
import React, { useContext, useEffect, useState } from 'react'
import { userContext } from '../context/user.context'
import { useNavigate } from 'react-router-dom';
import axiosInstance from "../config/axios";
import projectBg from '/images/1.jpg';
import {
  Dialog,
  DialogContent,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
} from "@/components/ui/dialog"
import { Input } from '@/components/ui/input';
import { Button } from '@/components/ui/button';
import ShowProject from '@/components/projectComponents/ShowProject';
import { Loader2 } from 'lucide-react';

const Home = () => {
  const { user, } = useContext(userContext);
  const [isModalOpen, setIsModalOpen] = useState(false);
  const [projectName, setProjectName] = useState('')
  const [projects, setProjects] = useState([])
  const [refreshProjects, setRefreshProjects] = useState(false);
  const [showError, setShowError] = useState(false);
  const [isPending, setIsPending] = useState(false);



  function createProject(e) {
    e.preventDefault()
    // console.log({ projectName })
    setIsPending(true);
    axiosInstance.post('/project/create-project', {
      name: projectName,
    })
      .then((res) => {
        // console.log(res,"ooo")
        setIsPending(false)
        setProjectName("");
        setRefreshProjects((prev) => !prev);
        setShowError(false);
      })
      .catch((error) => {
        console.log(error)
        setShowError(true);
      })
  }
```

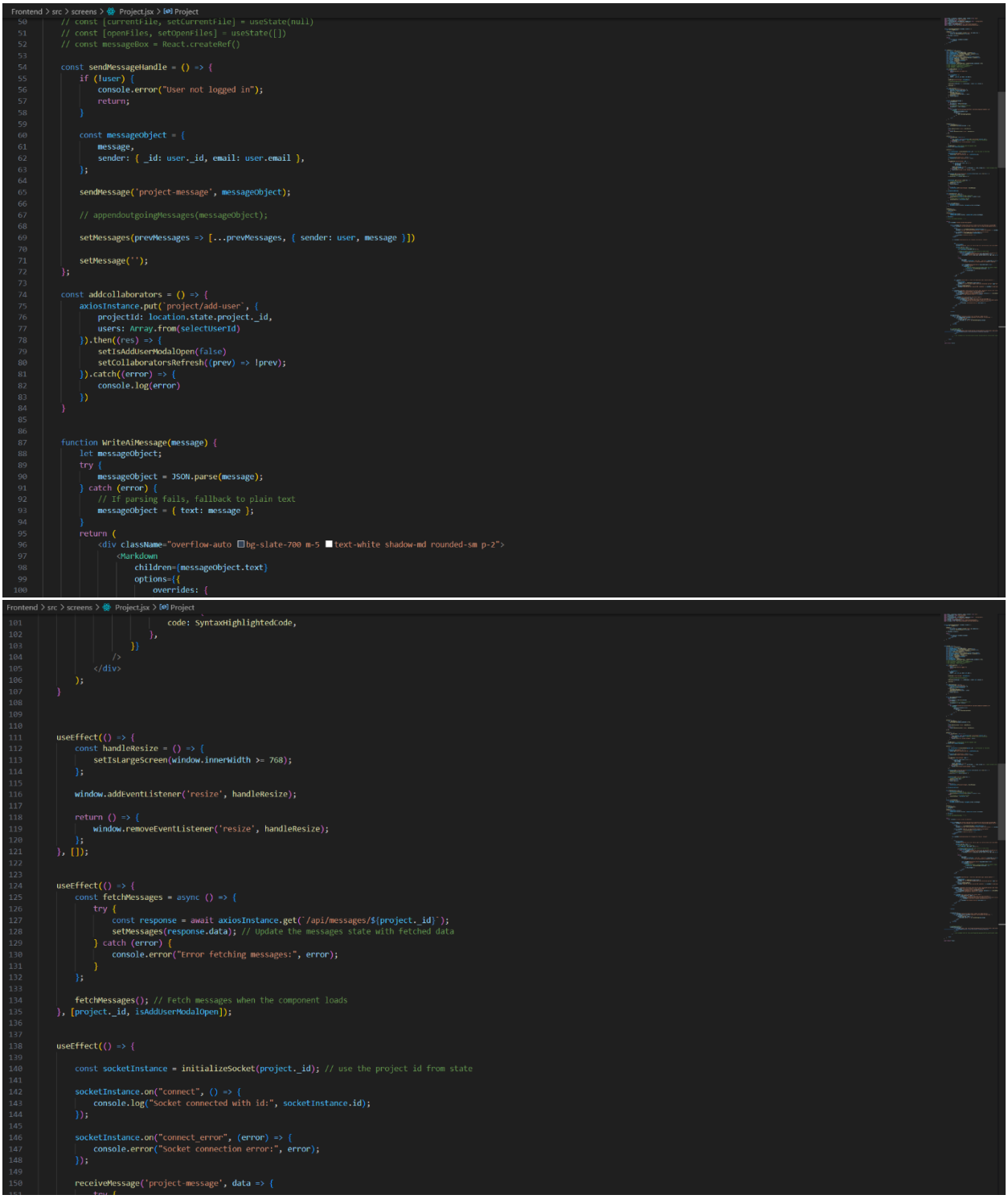


```
Frontend > src > screens > Home.jsx > Home
54    useEffect(() => {
55      axiosInstance.get('/project/get-projects').then((res) => {
56        // console.log("API Response:", res.data);
57        setProjects(res.data.Projects);
58      }).catch(err => {
59        console.error("Error fetching projects:", err);
60      });
61
62    }, [refreshProjects])
63
64
65
66    return <>
67      <main className=' bg-white h-[calc(100vh - 4.5rem - 4.5rem)]'>
68
69        <div  className="min-h-[84vh] w-full bg-cover bg-center p-4 rounded-md  ">
70
71
72
73
74
75          <div className='p-3 mt-2'  >
76            <Dialog>
77              <DialogTrigger className='bg-black text-white rounded  p-2 text-lg cursor-pointer'> Create New Project</DialogTrigger>
78              <DialogContent >
79                <DialogHeader >
80                  <form className='flex gap-5 flex-col' onSubmit={createProject}>
81                    <DialogTitle className='text-xl'>Create New Project </DialogTitle>
82                    <div>
83                      <label className='p-1'>Project Name</label>
84                      <Input
85                        className={`mt-1 text-wrap border  border-gray-400`}
86                        placeholder="Enter Project Name"
87                        onChange={(e) => setProjectName(e.target.value)}
88                        value={projectName}
89                      />
90                    </div>
91
92
93                    <Button className='w-full mt-2' type="submit" disabled={isPending}>
94                      {
95                        isPending ? (<><Loader2 className="mr-2 h-4 w-4 animate-spin" />Please Wait...</>) : ('Create')
96                      }
97                    </Button>
98
99                  </form>
100                 {showError && <p className='text-red-500 mt-3'>Project name must be at least 3 characters long</p>}
101               </DialogHeader>
102             </DialogContent>
103           </Dialog>
104         </div>
```



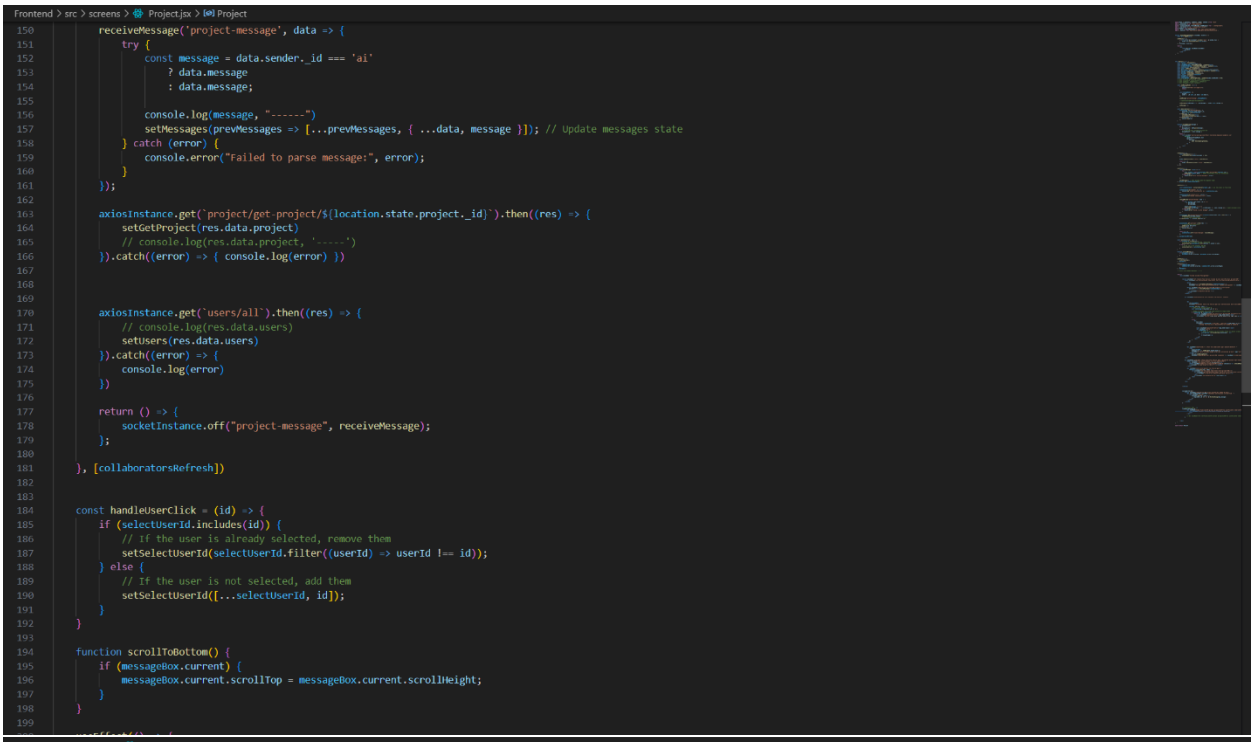
```
Frontend > src > screens > Home.jsx > Home
102             </DialogContent>
103           </Dialog>
104         </div>
105
106         <div className="projects grid grid-cols-1 sm:grid-cols-2 md:grid-cols-4 gap-10 mt-3 p-4  scroll-smooth">
107           {
108             projects?.map((project, id) => {
109               return <ShowProject
110                 project={project} setRefreshProjects={setRefreshProjects} />
111             })
112           }
113         </div>
114
115         {/* {isModalOpen && (…
135         )) */}
136
137       </div>
138     </main>
139   </>
140
141 }
142
143 export default Home
```

Chat page



```
Frontend > src > screens > Project.jsx > Project
    // const [currentFile, setCurrentFile] = useState(null)
    // const [openFiles, setOpenFiles] = useState([])
    // const messageBox = React.createRef()

    const sendMessageHandle = () => {
        if (!user) {
            console.error("User not logged in");
            return;
        }

        const messageObject = {
            message,
            sender: { _id: user._id, email: user.email },
        };

        sendMessage('project-message', messageObject);

        // appendoutgoingMessages(messageObject);

        setMessages(prevMessages => [...prevMessages, { sender: user, message }])

        setMessage('');
    };

    const addcollaborators = () => {
        axiosInstance.put(`project/add-user`, {
            projectId: location.state.project._id,
            users: Array.from(selectUserId)
        }).then((res) => {
            setisAddUserModalOpen(false)
            setCollaboratorsRefresh((prev) => !prev);
        }).catch((error) => {
            console.log(error)
        })
    }


    function WriteAiMessage(message) {
        let messageObject;
        try {
            messageObject = JSON.parse(message);
        } catch (error) {
            // If parsing fails, fallback to plain text
            messageObject = { text: message };
        }
        return (
            <div className="overflow-auto bg-slate-700 m-5 text-white shadow-md rounded-sm p-2">
                <Markdown
                    children={messageObject.text}
                    options={{
                        overrides: {
```

```
Frontend > src > screens > Project.jsx > Project
                            code: SyntaxHighlightedCode,
                        },
                    }}
                />
            </div>
        );
    }



    useEffect(() => {
        const handleResize = () => {
            setIsLargeScreen(window.innerWidth >= 768);
        };

        window.addEventListener('resize', handleResize);

        return () => {
            window.removeEventListener('resize', handleResize);
        };
    }, []);


    useEffect(() => {
        const fetchMessages = async () => {
            try {
                const response = await axiosInstance.get(`/api/messages/${project._id}`);
                setMessages(response.data); // Update the messages state with fetched data
            } catch (error) {
                console.error("Error fetching messages:", error);
            }
        };

        fetchMessages(); // Fetch messages when the component loads
    }, [project._id, isAddUserModalOpen]);


    useEffect(() => {

        const socketInstance = initializeSocket(project._id); // use the project id from state

        socketInstance.on("connect", () => {
            console.log("Socket connected with id:", socketInstance.id);
        });

        socketInstance.on("connect_error", (error) => {
            console.error("Socket connection error:", error);
        });

        receiveMessage('project-message', data => {
            try {
```



```
Frontend > src > screens > Project.jsx > Project
150         receiveMessage('project-message', data => {
151             try {
152                 const message = data.sender._id === 'ai'
153                     ? data.message
154                     : data.message;
155 
156                 console.log(message, "------")
157                 setMessages(prevMessages => [...prevMessages, { ...data, message }]); // Update messages state
158             } catch (error) {
159                 console.error("Failed to parse message:", error);
160             }
161         });
162 
163         axiosInstance.get(`project/get-project/${location.state.project._id}`).then((res) => {
164             setGetProject(res.data.project)
165             // console.log(res.data.project, '-----')
166         }).catch((error) => { console.log(error) })
167 
168 
169 
170         axiosInstance.get(`users/all`).then((res) => {
171             // console.log(res.data.users)
172             setUsers(res.data.users)
173         }).catch((error) => {
174             console.log(error)
175         })
176 
177         return () => {
178             socketInstance.off("project-message", receiveMessage);
179         };
180 
181     }, [collaboratorsRefresh])
182 
183 
184     const handleUserClick = (id) => {
185         if (selectUserId.includes(id)) {
186             // If the user is already selected, remove them
187             setSelectUserId(selectUserId.filter((userId) => userId !== id));
188         } else {
189             // If the user is not selected, add them
190             setSelectUserId([...selectUserId, id]);
191         }
192     }
193 
194     function scrollToBottom() {
195         if (messageBox.current) {
196             messageBox.current.scrollTop = messageBox.current.scrollHeight;
197         }
198     }
199 
```

```
Frontend > src > screens > Project.jsx > Project
203 
204     useEffect(() => {
205         if (codeEditorRef.current) {
206             codeEditorRef.current.scrollTop = codeEditorRef.current.scrollHeight;
207         }
208     }, [messages]);
209 
210     // console.log(isAddUserModalOpen, "----")
211 
212 
213     return (
214         <main className='h-screen w-screen flex bg-white'>
215 
216 
217             <section className="left relative flex flex-col h-screen md:w-1/3 min-w-96 w-full bg-slate-300">
218                 <header className='flex justify-between items-center p-3 w-full bg-slate-200 absolute top-0 '>
219 
220                     <button
221                         onClick={() => { setIsAddUserModalOpen(!isAddUserModalOpen); }}
222                         className='flex gap-1 hover:bg-slate-300 px-2 py-1 rounded cursor-pointer '><i className="ri-add-fill "></i> <span>Add Collaborators</span></button>
223 
224                     <button className='shadow-md px-2 bg-slate-400 rounded-md cursor-pointer'
225                         onClick={() => { setIsSidePanelOpen(!isSidePanelOpen) }}
226                     >
227                         <i className="ri-team-fill text-2xl "></i>
228                     </button>
229                 </header>
230 
231 
232                 <div className="conversation-area mt-14 flex-grow flex flex-col  relative"
233 
234                 >
235 
236                     <div
237                         ref={messageBox}
238                         className="message-box flex-1 flex flex-col gap-2 p-2 overflow-y-auto max-h-[calc(100vh-120px)] scrollbar-hide scroll-smooth"
239                     >
240                         {messages.map((msg, index) => {
241                             // Check if the message is from AI
242                             const isAiMessage = msg.sender._id === 'ai';
243 
244                             // Render AI messages only in the right section for large screens
245                             if (isAiMessage && isLargeScreen) {
246                                 return <div className='message flex flex-col p-2 bg-slate-500 text-white w-fit rounded-md'>
247                                     <small className="opacity-65 text-xs">{msg.sender.email}</small>
248                                     <p className='text-sm'>i have given what you asked on the right side</p></div>; // Skip rendering in the left section
249                             }
250 
251                             return (
252                                 <div
253                                     key={index}
```

```
Frontend > src > screens > Project.jsx > Project
                                key={index}
                                className={`${isAiMessage ? 'max-w-80' : 'max-w-52'} ${msg.sender._id === user._id.toString() && 'ml-auto'
                                    } message flex flex-col p-2 bg-slate-50 w-fit rounded-md shadow-md`}
                            >
                                <small className="opacity-65 text-xs">{msg.sender.email}</small>
                                <div className="text-sm ">
                                    {isAiMessage ? (
                                        // Render AI message in a user-friendly format for smaller screens
                                        !isLargeScreen ? WriteAiMessage(msg.message) : null
                                    ) : (
                                        <p>{msg.message}</p>
                                    )}
                                </div>
                            </div>
                        );
                    })}
                </div>



                <div className="input-field p-2  w-full flex items-center gap-1 absolute bottom-0  ">
                    <input
                        value={message}
                        onChange={(e) => { setMessage(e.target.value) }}
                        className='p-2 px-4 flex-grow rounded border-none outline-none bg-white' type="text" placeholder='Type message here ' />
                    <button
                        onClick={sendMessageHandle}
                        className='shadow-md px-2 py-1 bg-slate-900 rounded-md '><i className="ri-send-plane-fill text-2xl  text-white"></i> </button>
                </div>
            </div>

            <div className={`side-panel w-full h-full flex flex-col gap-2  bg-slate-50 absolute top-0 transition-all duration-500 ease-in-out  ${isSidePanelOpen ? "-translate-x-0" : "-trans
                <header className='flex justify-between p-4 w-full bg-slate-200'>
                    <h1 className='font-semibold text-lg'>Collaborators</h1>
                    <button className='shadow-md px-2 bg-slate-400 rounded-md' onClick={() => { setIsidePanelOpen(prev => !prev) }}>
                    <i className="ri-close-large-fill text-xl"></i></button>
                </header>

                <div className="users cursor-pointer flex flex-col gap-2">
                    {getProject.users.map((user, index) => (
                        <div className="user flex gap-2 items-center hover:bg-slate-200 p-1">
                            <div className='bg-slate-400 aspect-square flex w-fit h-fit items-center justify-center p-5  rounded-full' >
                                <i className="ri-user-2-fill absolute text-white text-xl"></i>
                            </div>
                            <h1 className='font-semibold text-lg'>{user.email}</h1>
                        </div>
                    ))}
                </div>

            </div>
```

```
Frontend > src > screens > Project.jsx > Project
                    </div>

                </div>



            </section>



            {isLargeScreen && (
                <section className="right bg-zinc-300  h-full w-[70vw] p-2 hidden md:flex">
                    <div ref={codeEditorRef} className="code-editor overflow-auto scrollbar-hide  ">
                        <div className='text-sm'>
                            {messages.map((msg, index) => (
                                msg.sender._id === 'ai' && WriteAiMessage(msg.message)
                            ))}
                        </div>
                    </div>
                </section>
            )}



            {/* colaborator Modal  */}
            {isAddUserModalOpen && (
                    <div className="modaal fixed inset-0 bg-black bg-opacity-50 flex justify-center items-center z-50">
                        <div className="bg-white  w-11/12 max-w-md p-4 rounded-lg shadow-lg">…
                        </div>
                    </div>

                    // <div className='h-10 w-20 fixed inset-0 bg-black bg-opacity-50 flex justify-center items-center z-50 bg-red-200'>hlo</div>
                )}


        </main>
    )
}

export default Project
```

App Routes

```jsx
import React from 'react'
import { Route, Routes } from "react-router-dom"
import Login from '../screens/Login'
import Register from '../screens/Register'
import Home from '../screens/Home'
import Project from '../screens/Project'
import UserAuth from '../auth/UserAuth'

const AppRoutes = () => {
    return (
        <Routes>
            <Route path='/' element={<UserAuth><Home/></UserAuth>}/>
            <Route path='/login' element={<Login/>}/>
            <Route path='/register' element={<Register/>}/>
            <Route path='/project' element={<UserAuth><Project/></UserAuth>}/>
        </Routes>
    )
}

export default AppRoutes
```

App page

```jsx
import { useState } from 'react'
import AppRoutes from './routes/AppRoutes'
import UserProvider from './context/user.context'
import ErrorBoundary from './ErrorBoundary/ErrorBoundary'
import Navbar from './components/headerFooter/NavBar'
import Footer from './components/headerFooter/Footer'
import { useLocation } from 'react-router-dom'
function App() {
  const location = useLocation();
  const hideHeaderFooter = ['/login', '/register','/project'].includes(location.pathname);

  return (
    <>
      <UserProvider>
        <ErrorBoundary>
        {!hideHeaderFooter && <Navbar />}
          <AppRoutes />
          {!hideHeaderFooter && <Footer />}
        </ErrorBoundary>
      </UserProvider>
    </>
  )
}

export default App

```

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)

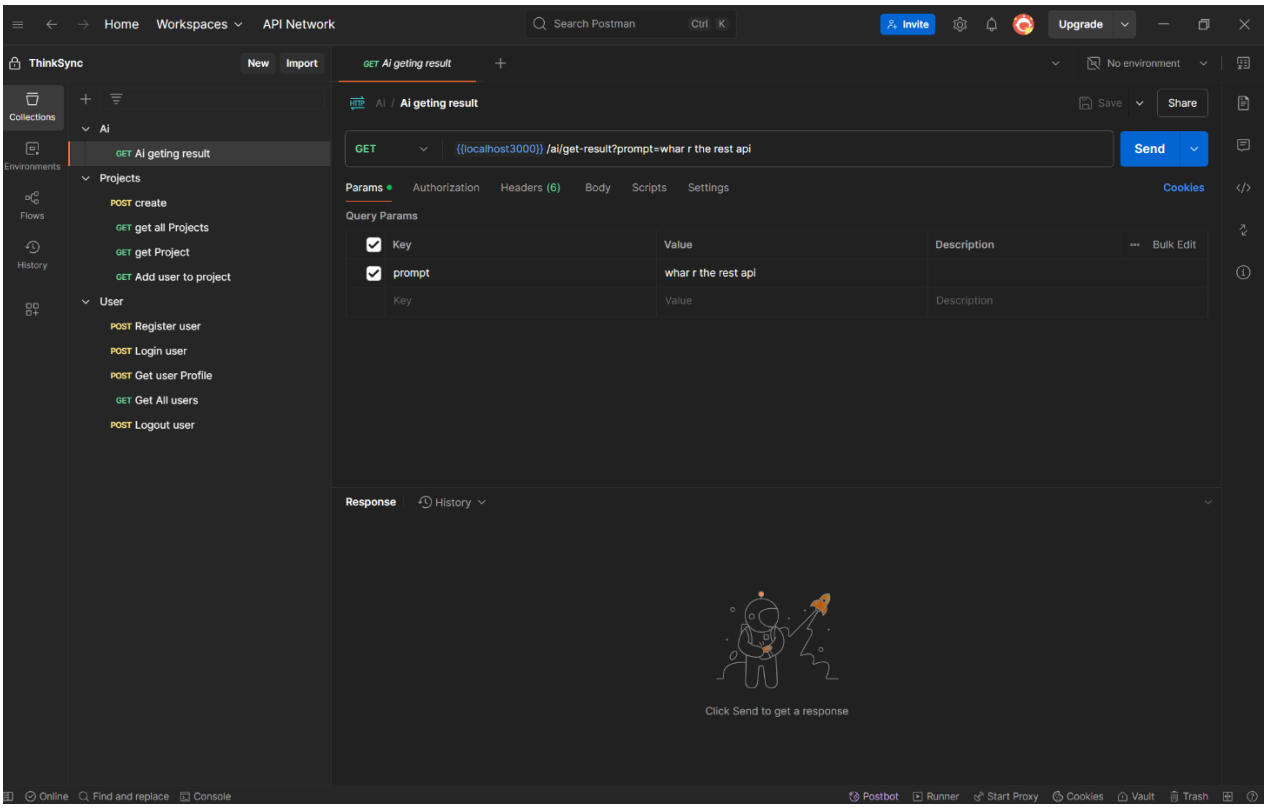
- A closed group of users tested the application for usability and performance.
- Based on feedback:
    - AI responses were improved.
    - Chat layout was optimized for better readability.
    - Added error messages and loading indicators for better UX.

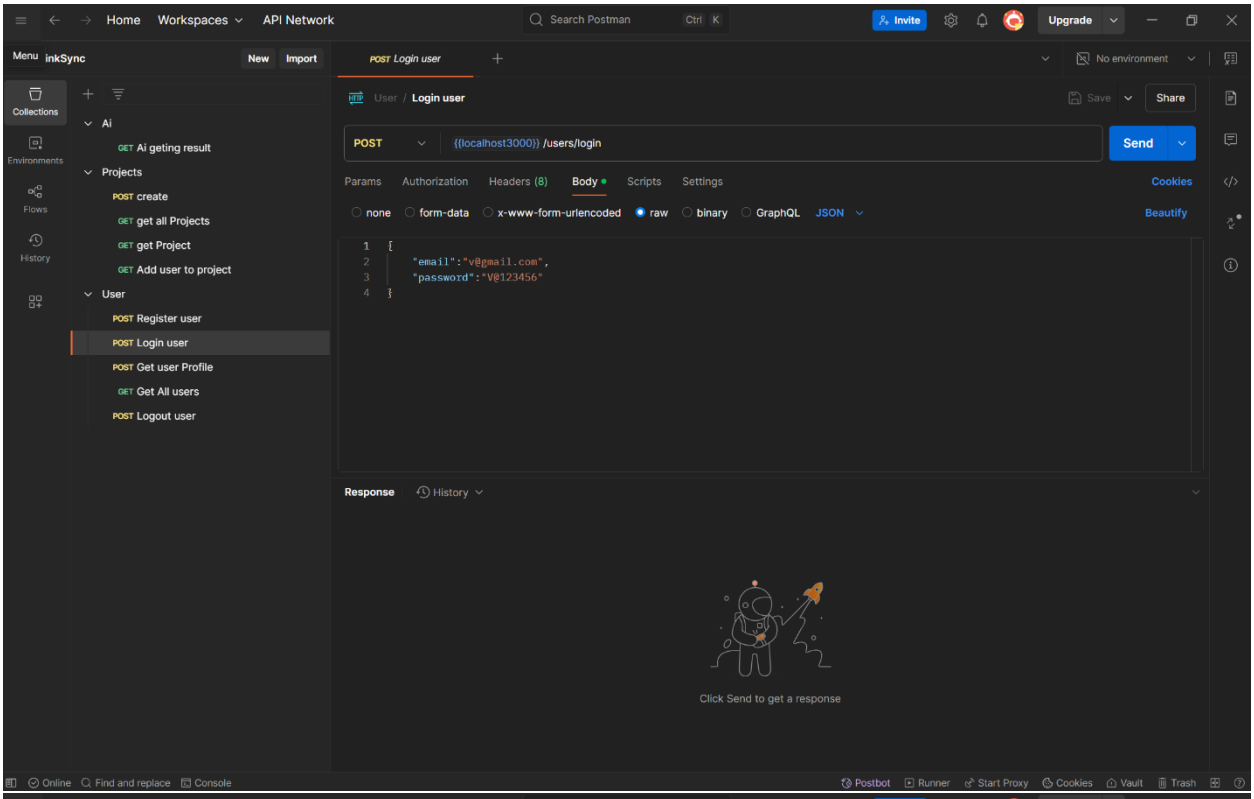
Testing Tools Used

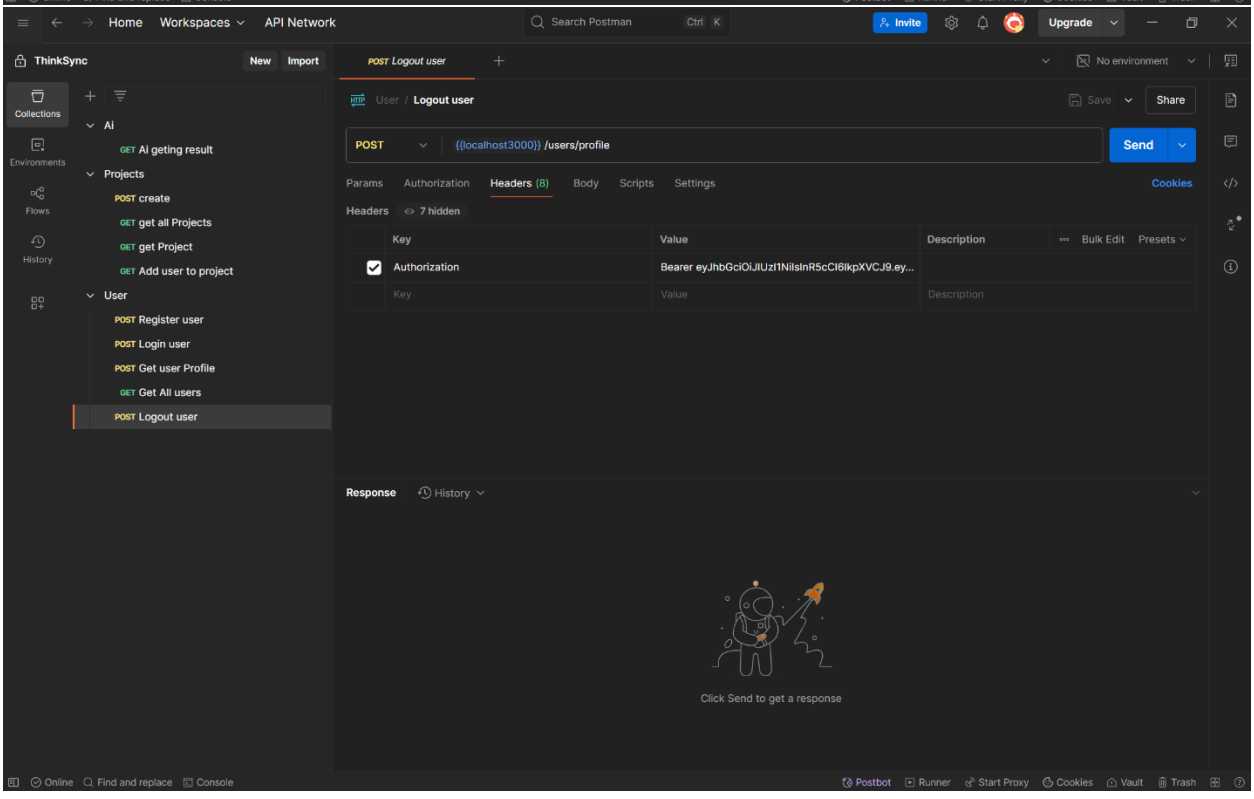
Tool	Purpose
Jest	Unit testing
Postman	API endpoint testing
Supertest	HTTP integration testing
Socket.IO Client	Real-time communication testing
Browser DevTools	Frontend debugging and testing

Snapshot of Postman

Api Testing




Home
Workspaces
API Network
Search Postman
Invite
Upgrade
inkSync
New
Import
POST Login user
No environment
Collections
Environments
Flows
History
Ai
GET Ai geting result
Projects
POST create
GET get all Projects
GET get Project
GET Add user to project
User
POST Register user
POST Login user
POST Get user Profile
GET Get All users
POST Logout user
User / Login user
Save
Share
POST
{{localhost3000}} /users/login
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
{
"email":"v@gmail.com",
"password":"V@123456"
}
Response
History
Click Send to get a response
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash


Home
Workspaces
API Network
Search Postman
Invite
Upgrade
ThinkSync
New
Import
POST Logout user
No environment
Collections
Environments
Flows
History
Ai
GET AI geting result
Projects
POST create
GET get all Projects
GET get Project
GET Add user to project
User
POST Register user
POST Login user
POST Get user Profile
GET Get All users
POST Logout user
User / Logout user
Save
Share
POST
{{localhost3000}} /users/profile
Send
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
Headers
7 hidden
Key
Value
Description
Bulk Edit
Presets
Authorization
Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.ey...
Key
Value
Description
Response
History
Click Send to get a response
Online
Find and replace
Console
Postbot
Runner
Start Proxy
Cookies
Vault
Trash

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.