

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

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 Lifecycle

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
Phase 3: 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

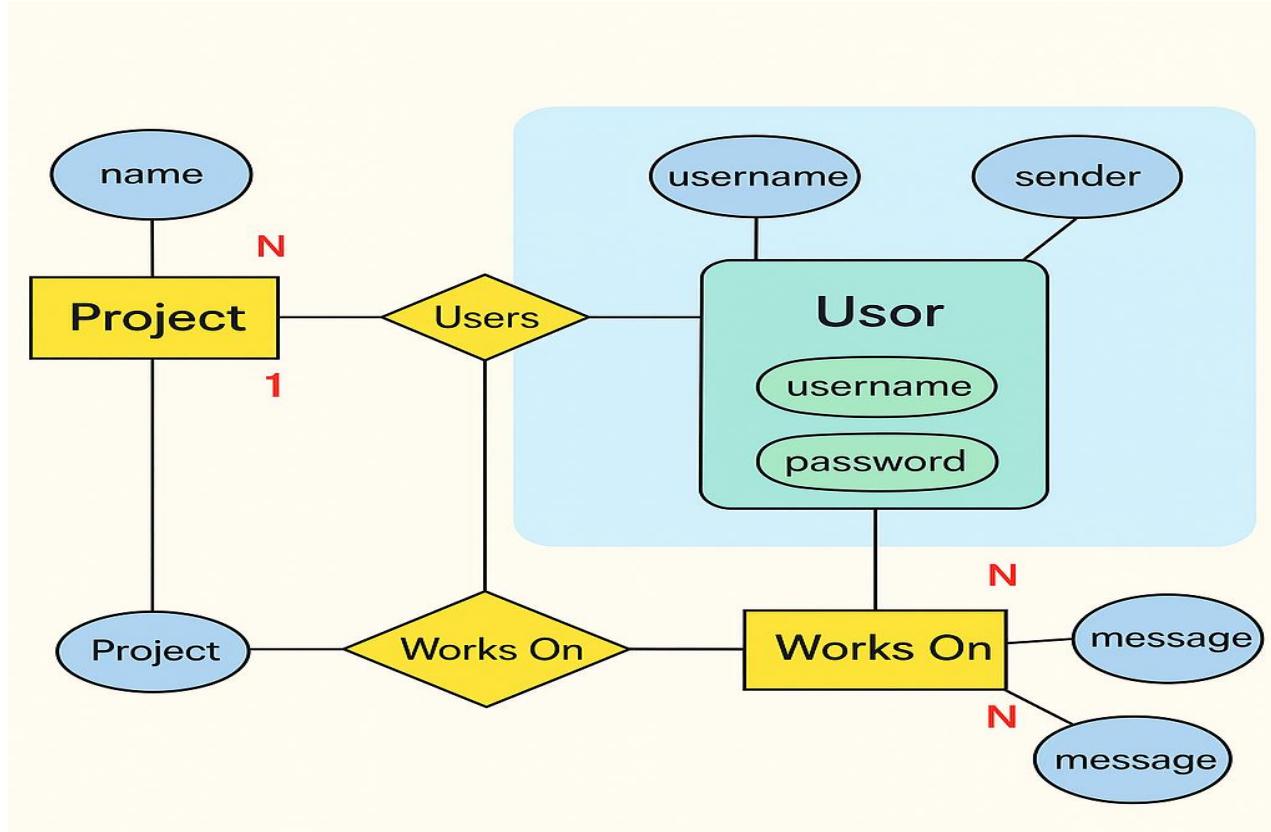
The Design Phase of the ThinkSync project focuses on planning the structure and interactions of the system's components before actual development. It ensures the system is logically organized, scalable, secure, and maintainable. The design covers the major modules, user interactions, and database schema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

1. Basic Modules and Descriptions

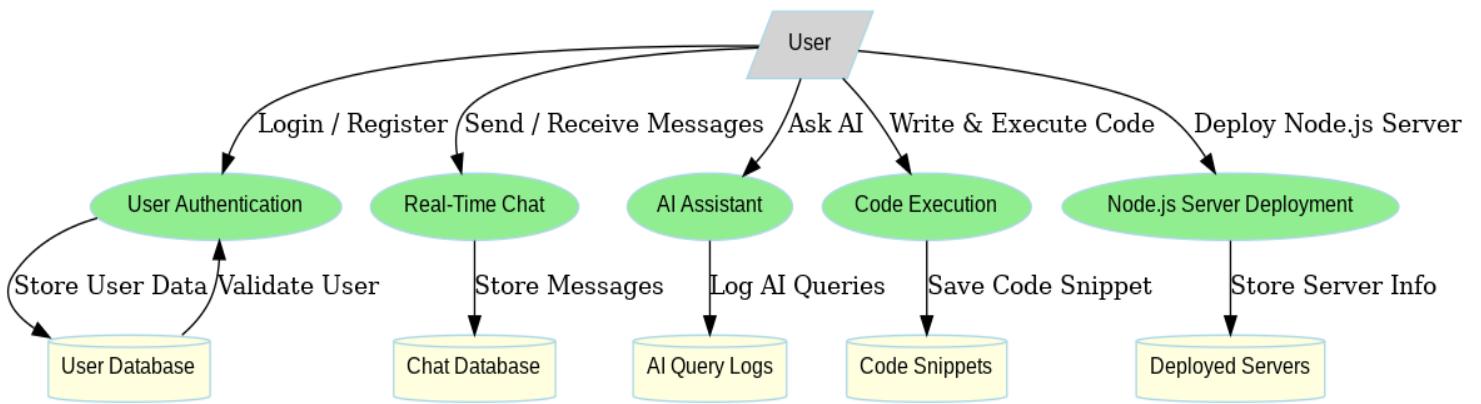
Module Name	Description
User Management	Handles user registration, login, JWT authentication, password hashing, etc.

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

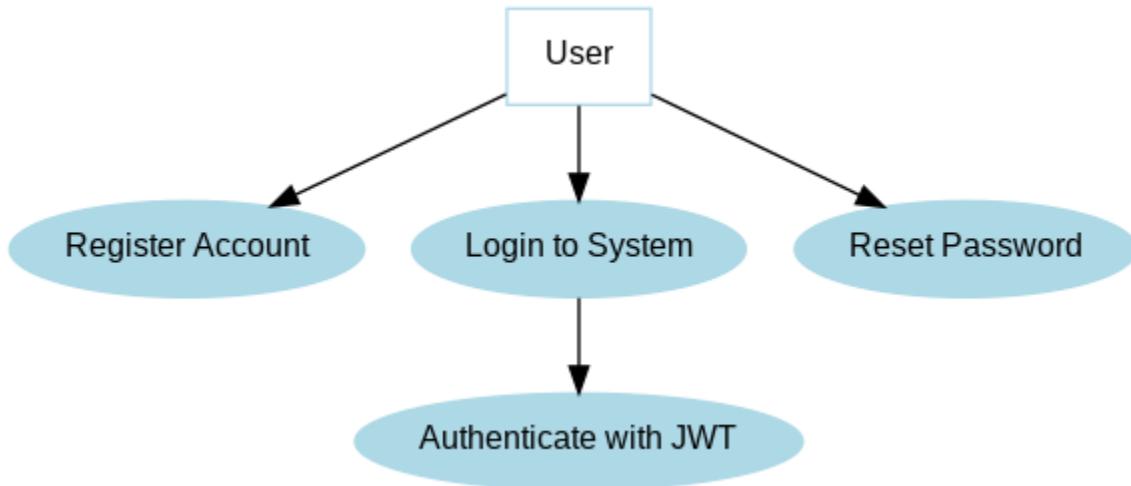


3. DFDs

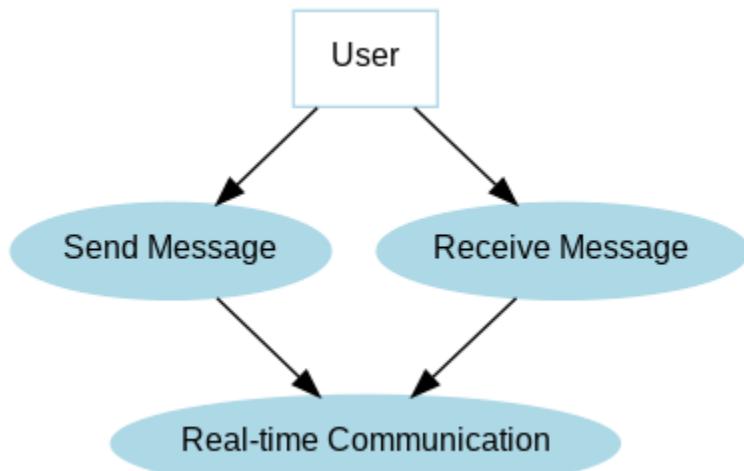


4. Use Case Diagram

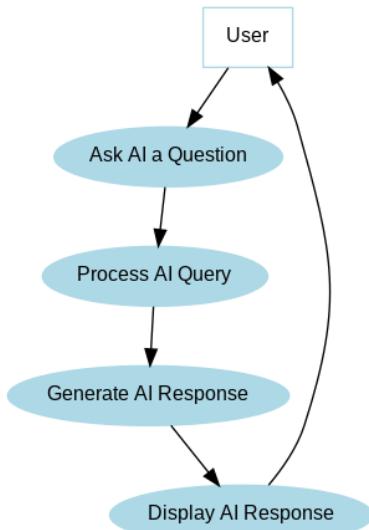
1. User Authentication Use Case Diagram



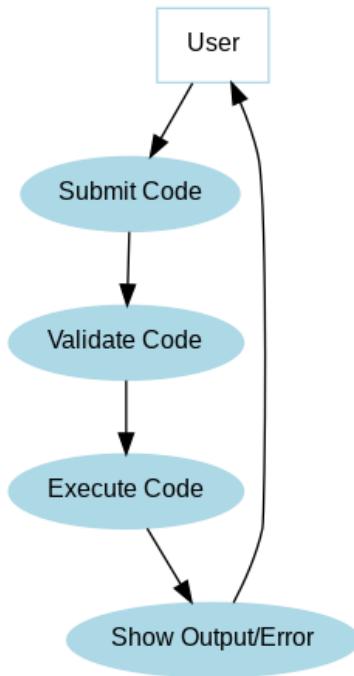
2. Chat System Use Case Diagram



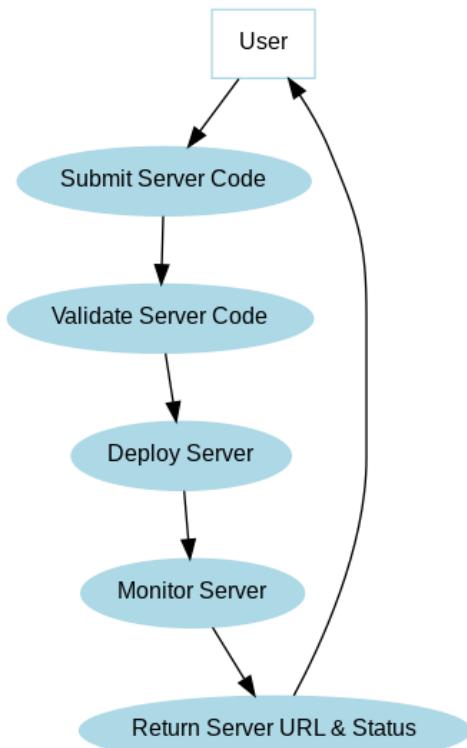
3. AI Assistant Use Case Diagram



4. Code Execution Use Case Diagram

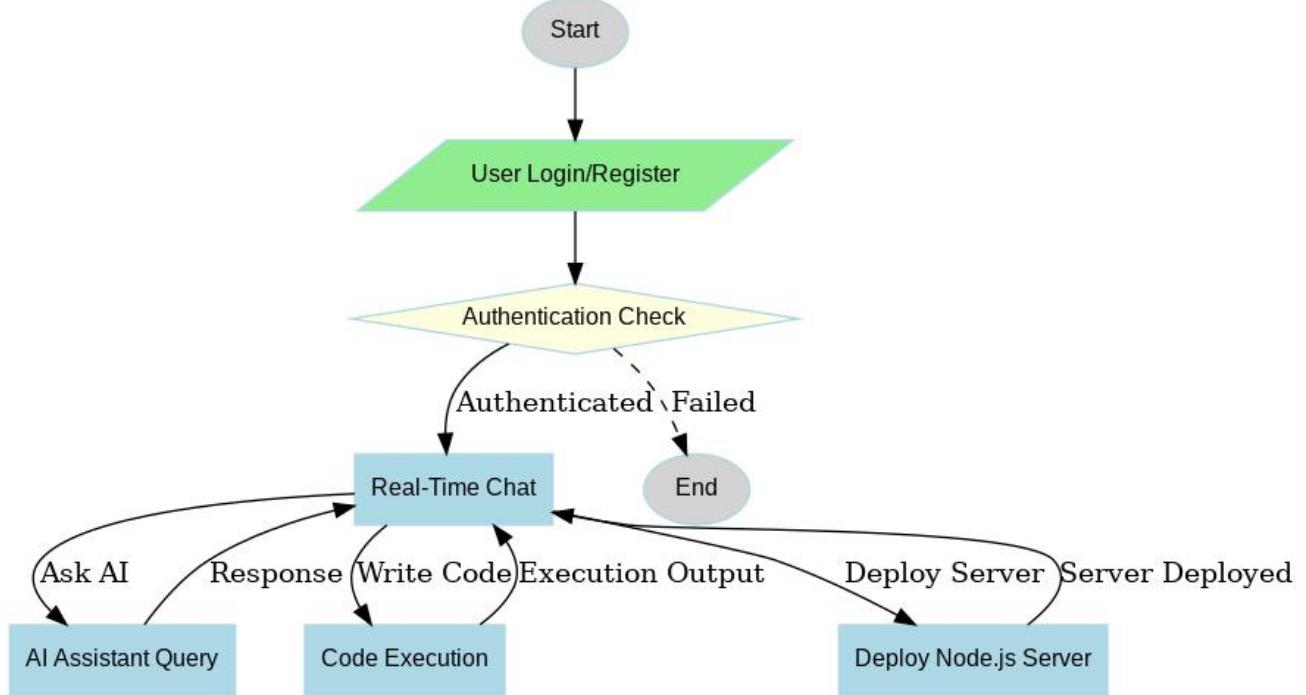


5. Server Deployment Use Case Diagram

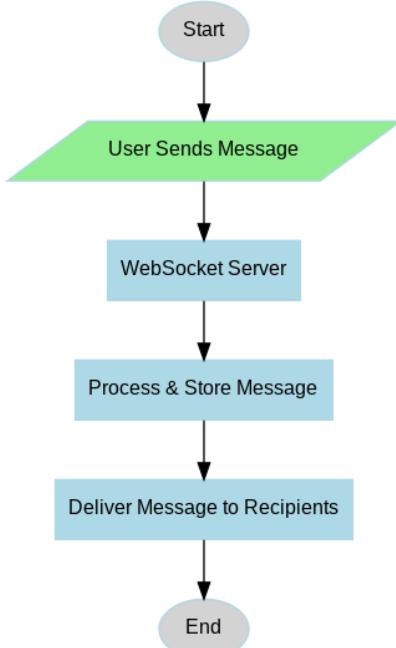


5. Flowcharts

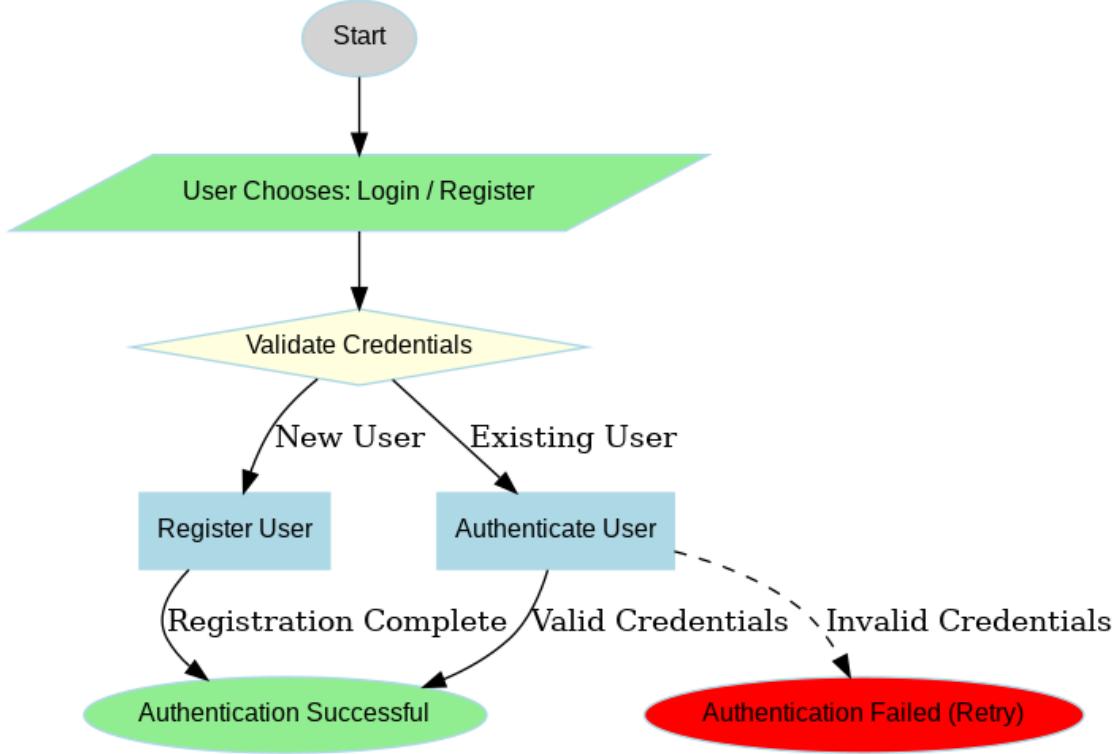
Complete system Flowchart



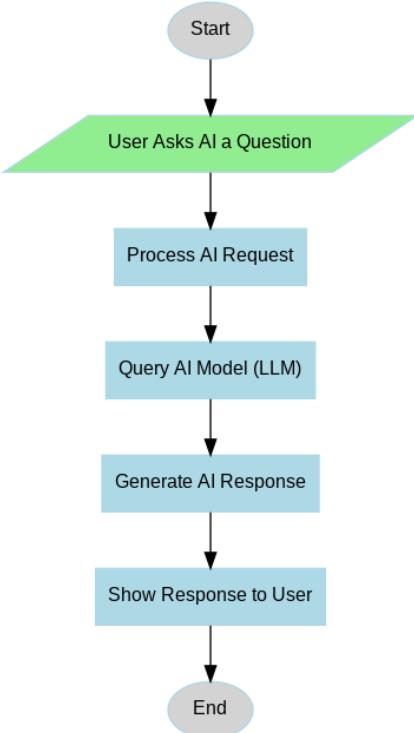
Chat Flowchart



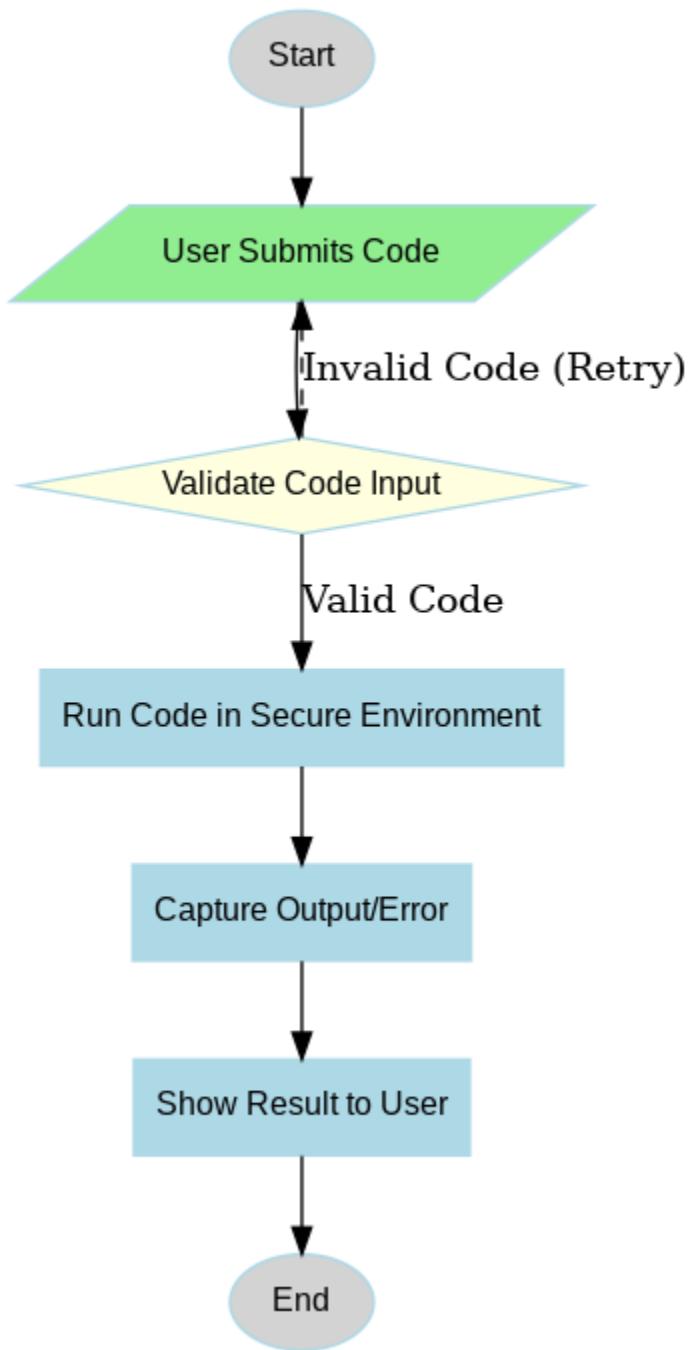
Authentication Flowchart



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)

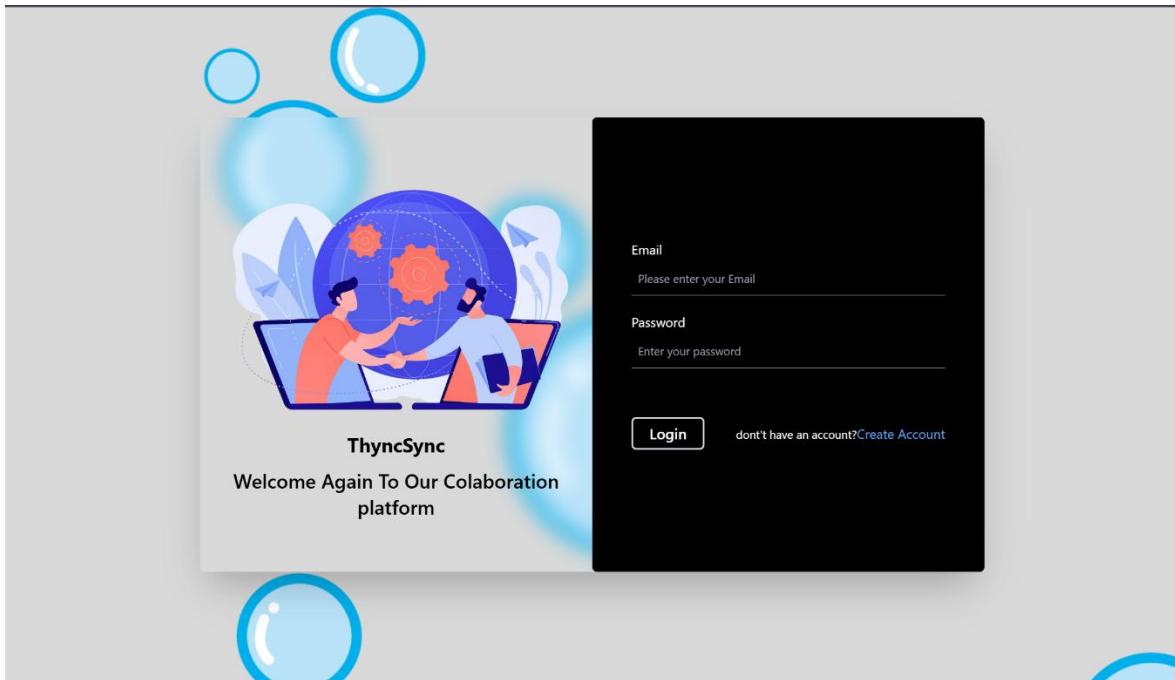
3. Messages Collection

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

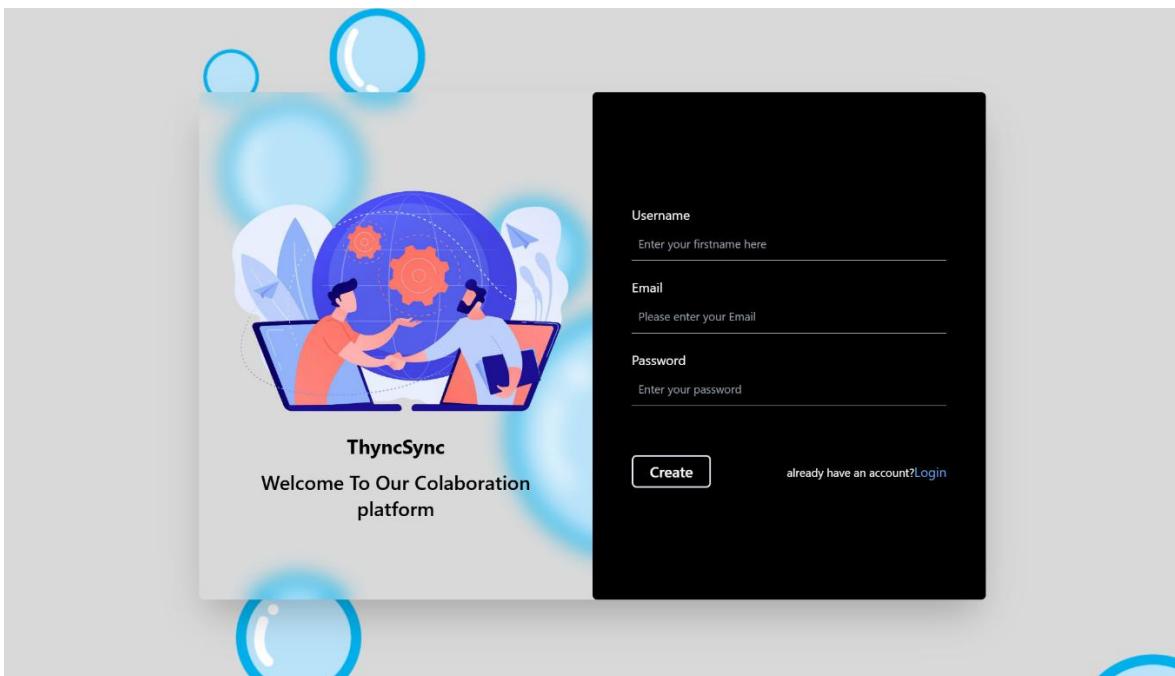
Coding Phase

Snapshots

Login



Register



Home Page

The screenshot shows the ThinkSync home page. At the top left is the logo "ThinkSync". To its right is a search bar with the placeholder "Search" and a user profile icon. Below the header is a button labeled "Create New Project". The main content area displays a grid of project cards. Each card has a title, a collaborator count, and a trash icon. The titles and collaborator counts are as follows:

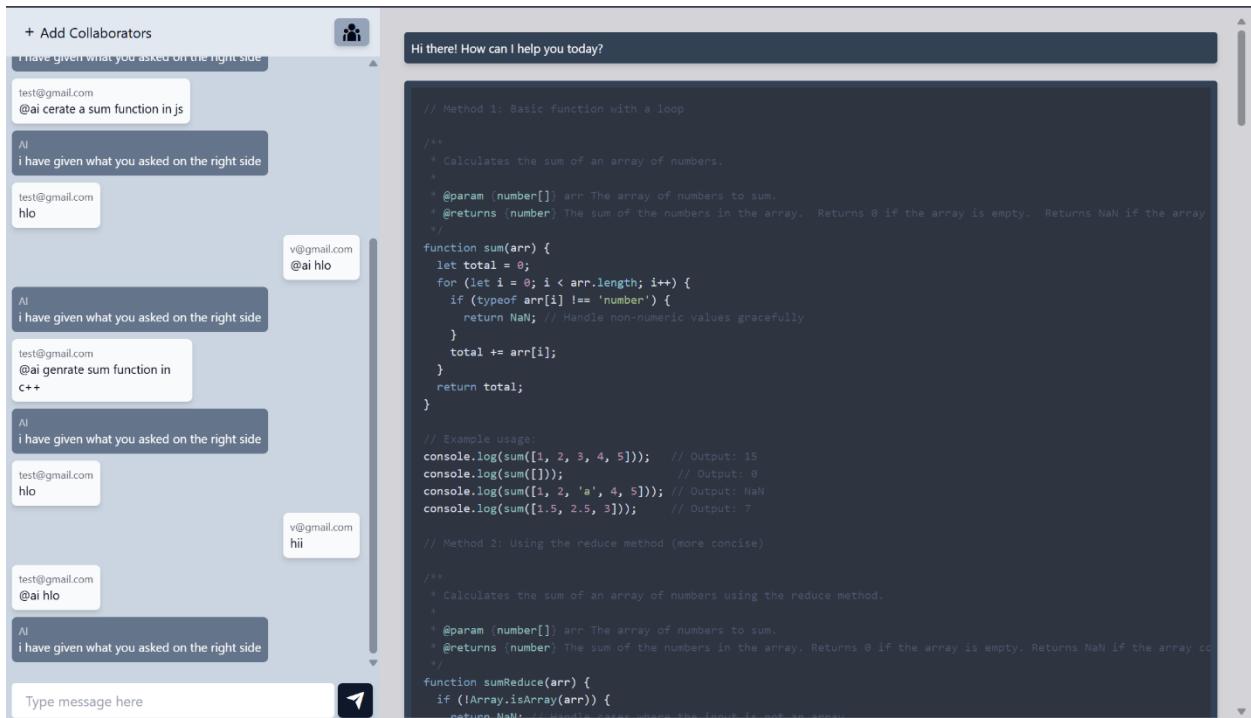
Title	Collaborators
first	3
ioioi	1
fourth	1
fifth	1
sixth	1
7th	1

At the bottom of the page is a footer bar with the copyright notice "# Copyright © 2025 - All right reserved" and social media icons for Twitter, YouTube, and Facebook.

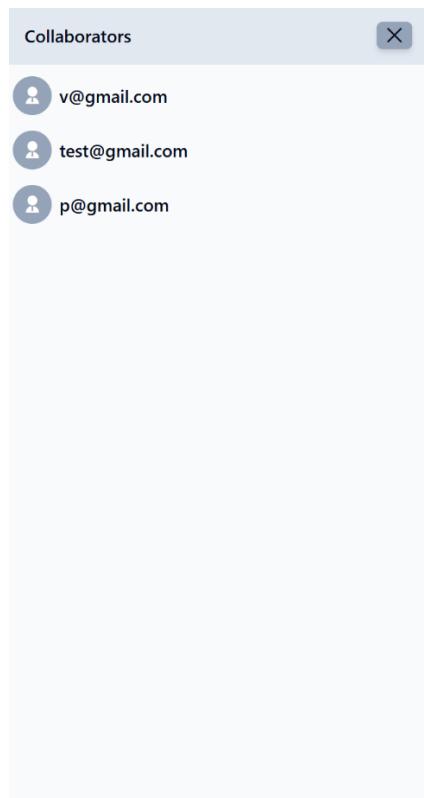
Create Project Modal

This screenshot shows the same ThinkSync home page as above, but with a modal window overlaid in the center. The modal is titled "Create New Project" and contains a single input field labeled "Project Name" with the placeholder "Enter Project Name". Below the input field is a "Create" button. The background of the page is darkened to accommodate the modal.

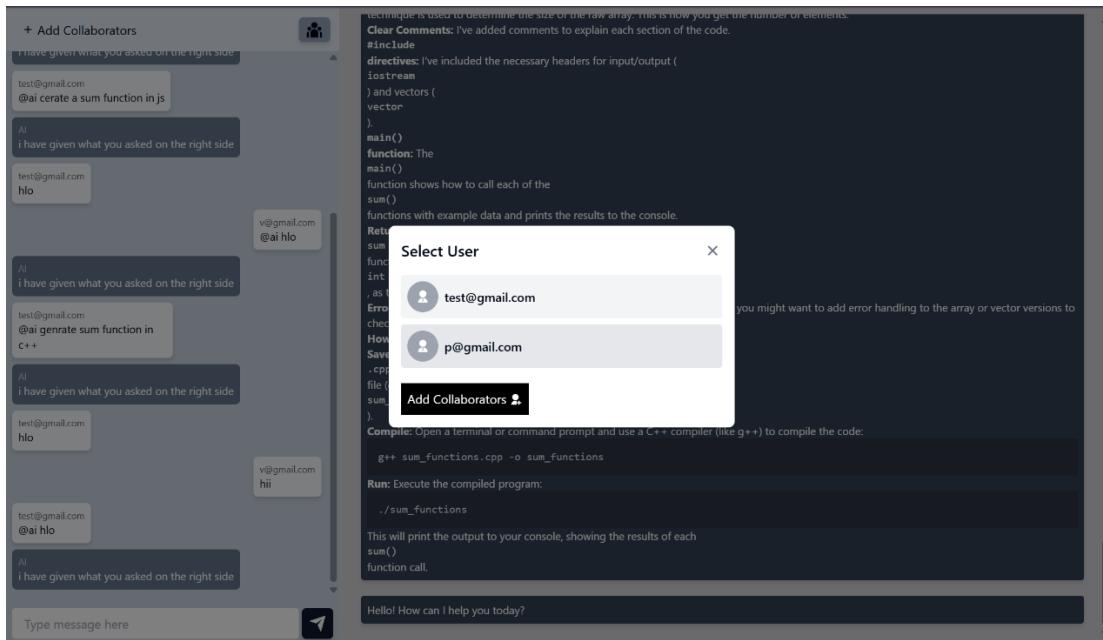
Chat Panel



Collaborators Members Panel



Add collaborators Modal



Code snippets

Backend: User Model (Schema)



Message Model (Schema)

```
Backend > models > message.model.js > messageSchema
28
29 const messageSchema = new mongoose.Schema(
30   {
31     projectId: {
32       type: mongoose.Schema.Types.ObjectId,
33       ref: "project",
34       required: true,
35     },
36     sender: {
37       _id: {
38         type: String,
39         required: true,
40       },
41       email: {
42         type: String,
43         required: true,
44       },
45     },
46     message: {
47       type: String,
48       required: true,
49     },
50     isAiResponse: {
51       type: Boolean,
52       default: false,
53     },
54   },
55 },
56 {
57   timestamps: true,
58 }
59 );
60
61 // Encrypt the message before saving
62 messageSchema.pre("save", function (next) {
63   if (this.isModified("message")) {
64     this.message = encrypt(this.message);
65   }
66   next();
67 });
68
69 // Decrypt the message when retrieving
70 messageSchema.methods.decryptMessage = function () {
71   return decrypt(this.message);
72 };
73
74 const messageModel = mongoose.model("message", messageSchema);
75
76 export default messageModel;
```

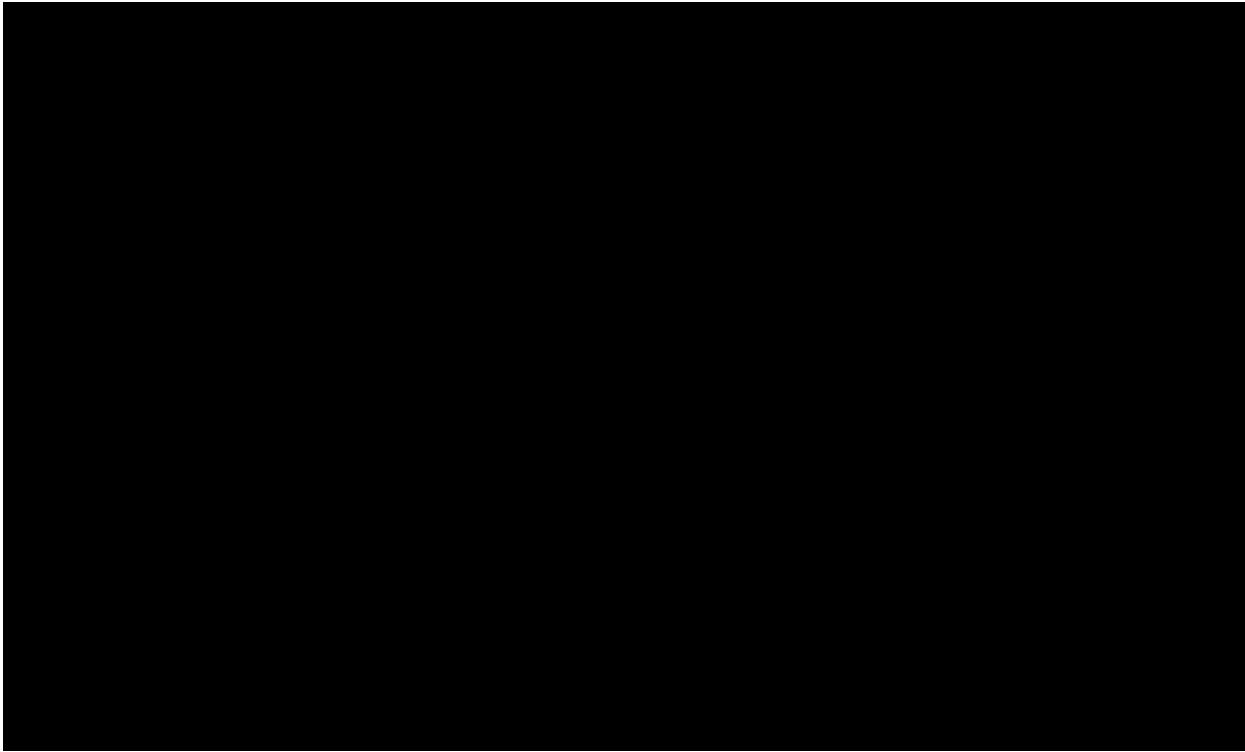
Project Model (Schemaa)

```
Backend > models > project.model.js > default
1 import mongoose from "mongoose";
2
3 const projectSchema = new mongoose.Schema({
4   name: {
5     type: String,
6     lowercase: true,
7     required: true,
8     trim: true,
9     unique: [true, 'project name must be unique'],
10  },
11
12  users:[
13    {
14      type: mongoose.Schema.Types.ObjectId,
15      ref: 'user'
16    }
17  ]
18});
19
20
21
22 const projectModel = mongoose.model('project', projectSchema)
23 export default projectModel;
```

Auth Middleware (Authentication)

```
Backend > middlewares > user.auth.middleware.js > ...
1 import jwt from "jsonwebtoken";
2 import userModel from "../models/user.model.js";
3 import redisClient from "../services/redis.service.js";
4 export const isLogin = async (req, res, next) => {
5   const token = req.cookies.token || req.headers.authorization?.split(" ")[1];
6
7   // console.log(token, "token")
8
9   if (!token) {
10     return res.status(401).json({ message: "Unauthorized: No token provided" });
11   }
12
13   const isBlacklisted = await redisClient.get(token);
14
15   if (isBlacklisted) {
16     res.cookie(token, "");
17     return res
18       .status(401)
19       .json({ message: "Unauthorized: Token is blacklisted" });
20   }
21
22   try {
23     const decoded = jwt.verify(token, process.env.JWT_SECRET);
24     const user = await userModel.findById(decoded.id);
25
26     if (!user) {
27       return res.status(401).json({ message: "Unauthorized: User not found" });
28     }
29     req.user = user;
30     return next();
31   } catch (error) {
32     console.error("Error in isLogin middleware:", error);
33     return res.status(401).json({ message: "Unauthorized", error });
34   }
35 };
36
```

Ai Service code



```
51     },
52   },
53   "package.json": {
54     "file": {
55       "contents": "
56       {
57         \"name\": \"temp_server\",
58         \"version\": \"1.0.0\",
59         \"main\": \"app.js\",
60         \"scripts\": {
61           \"start\": \"node app.js\",
62           \"test\": \"echo \\\"\\\"error: no test specified\\\"\\\" && exit 1\\\""
63         },
64         \"dependencies\": {
65           \"express\": \"^4.21.2\"
66         }
67       }
68     }
69   }
70 },
71 },
72   "buildCommand": {
73     "mainItem": "npm",
74     "commands": ["install"]
75   },
76   "startCommand": {
77     "mainItem": "npm",
78     "commands": ["start"]
79   }
80 }
81
82 user: Create an express application
83
84 </example>
85
86 <example>
87
88 user: Hello
89 response: [
90   "text": "Hello, How can I help you today?"
91 ]
92
93 </example>
94
95
96 IMPORTANT:
97 - Always return responses in valid JSON format.
98 - Include "fileTree", "buildCommand", and "startCommand" in every response.
99 - Write modular and maintainable code with comments.
100 - Do not include unnecessary explanations or examples in the response.
101 - Ensure the "fileTree" contains multiple files like "app.js", "src/routes.js", and "package.json".
```

```

95  IMPORTANT:
96  - Always return responses in valid JSON format.
97  - Include "fileTree", "buildCommand", and "startCommand" in every response.
98  - Write modular and maintainable code with comments.
99  - Do not include unnecessary explanations or examples in the response.
100 - Ensure the "fileTree" contains multiple files like "app.js", "src/routes.js", and "package.json".
101
102 User prompt: ${prompt}
103 `;
104
105 const response = await ai.models.generateContent({
106   model: "gemini-2.0-flash",
107   contents: prompt,
108   systemInstructions,
109   generationConfig: {
110     responseMimeType: "application/json",
111     temperature: 0.4,
112   },
113 });
114
115 return response.text;
116 }
117
118 async function result(prompt) {
119   return await main(prompt);
120 }
121
122 export default result;
123

```

Project Service

```

Backend > services > project.service.js > ...
1 import mongoose from "mongoose";
2 import projectModel from "../models/project.model.js";
3 import userModel from "../models/user.model.js";
4
5 export const createProject = async ({ name, userId }) => {
6   if (!name) {
7     throw new Error("Name is required");
8   }
9   if (!userId) {
10     throw new Error("User id is required");
11   }
12   try {
13     const project = await projectModel.create({
14       name,
15       users: [userId],
16     });
17
18     return project;
19   } catch (error) {
20     console.error("Error creating project:", error);
21     if (error.message.includes("project name must be unique")) {
22       throw new Error("project name must be unique");
23     }
24
25     throw new Error("Internal Server Error. Please try again later.");
26   }
27 };
28
29 export const getAllProjectByUserId = async ({ loggedInUserId }) => {
30   if (!loggedInUserId) {
31     throw new Error("User id is required");
32   }
33
34   try {
35     const allUserProjects = await projectModel.find({
36       users: loggedInUserId,
37     });
38
39     return allUserProjects;
40   } catch (error) {
41     console.error("Error fetching projects:", error);
42     throw new Error("Internal Server Error. Please try again later.");
43   }
44 };
45
46 export const addUsersToProject = async ({ projectId, users, userId }) => {
47   if (!projectId) {
48     throw new Error("projectId is required");
49   }

```

```

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("invalid 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                 { new: true }
97             }
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

```
Backend > services > user.service.js > ...
1 import userModel from "../models/user.model.js";
2
3 export const createUser = async ({ email, password, username }) => {
4   try {
5     if (!email || !password) {
6       throw new Error("Email and password are required");
7     }
8
9     const hash = await userModel.hashPassword(password);
10
11    const user = await userModel.create({
12      username,
13      email,
14      password: hash,
15    });
16
17    return user;
18  } catch (error) {
19    console.error("Error creating user:", error);
20
21    throw new Error("Internal Server Error. Please try again later.");
22  }
23}
24
25
26 export const getAllUser = async (userId) => {
27  const users = await userModel.find({ _id: { $ne: userId } });
28  return users;
29}
30
```

Ai Routes

```
Backend > routes > ai.routes.js > ...
1 import { Router } from "express";
2 import * as aiController from "../controllers/ai.controller.js";
3
4 const router = Router();
5
6 // Define the route correctly
7 router.get('/get-result', aiController.getResult);
8
9 export default router;
```

Message Routes

```
Backend > routes > message.route.js > ...
1 import express from "express";
2 import messageModel from "../models/message.model.js";
3
4 const router = express.Router();
5
6 router.get("/:projectId", async (req, res) => {
7   try {
8     const { projectId } = req.params;
9
10    const messages = await messageModel.find({ projectId }).sort({ createdAt: 1 });
11
12    // Decrypt messages before sending
13    const decryptedMessages = messages.map((msg) => ({
14      ...msg.toObject(),
15      message: msg.decryptMessage(),
16    }));
17
18    res.status(200).json(decryptedMessages);
19  } catch (error) {
20    console.error("Error fetching messages:", error);
21    res.status(500).json({ error: "Failed to fetch messages" });
22  }
23}
24);
25
26 export default router;
```

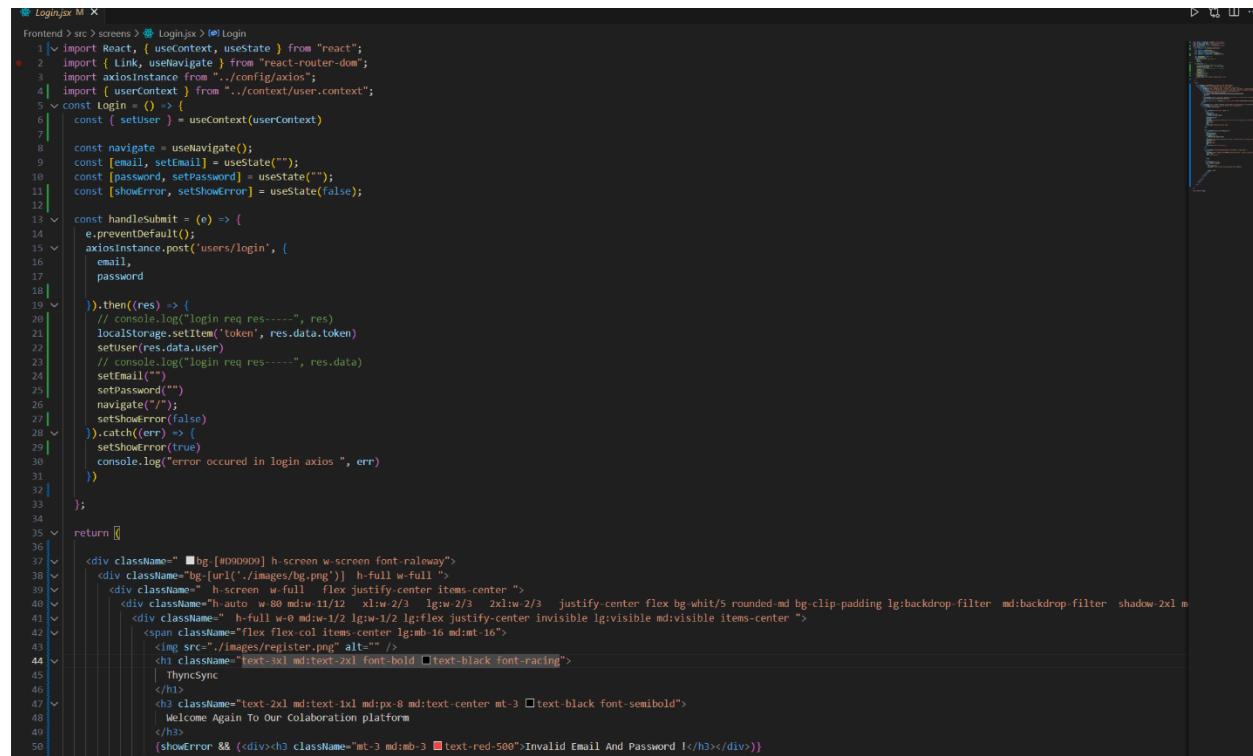
Project Routes

```
Backend > routes > project.routes.js > ...
1 import { Router } from "express";
2 import { validateProject, validateProjectUsers } from "../validators/projectValidator.js";
3 import * as projectController from "../controllers/project.controller.js";
4 import * as userAuthMiddleware from "../middlewares/user.auth.middleware.js";
5 const router = Router();
6
7 router.post('/create-project', validateProject , userAuthMiddleware.isLoggedIn, projectController.createProjectController);
8 router.get('/get-projects',userAuthMiddleware.isLoggedIn, projectController.getProjectController);
9 router.put('/add-user',validateProjectUsers, userAuthMiddleware.isLoggedIn, projectController.addUserToProjectController);
10 router.get('/get-project/:projectId', userAuthMiddleware.isLoggedIn, projectController.getProjectByIdController);
11 router.delete('/delete-project/:projectId', userAuthMiddleware.isLoggedIn, projectController.deleteProjectController);
12
13 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.isLoggedIn, userController.getProfileUserController);
11 router.get("/logout", userAuthMiddleware.isLoggedIn, userController.logoutUserController);
12 router.get("/all", userAuthMiddleware.isLoggedIn, userController.getAllUserController);
13
14 export default router;
15 |
```

Frontend : Login Page



```
Frontend > src > screens > Login.jsx > Login
1 |> import React, { useContext, useState } from "react";
  2 | import { Link, useNavigate } from "react-router-dom";
  3 | import axiosInstance from "./config/axios";
  4 | import { userContext } from "../context/user.context";
  5 < const Login = () => {
  6 |   const [setUser] = useContext(userContext);
  7 |
  8 |   const navigate = useNavigate();
  9 |   const [email, setEmail] = useState("");
10 |   const [password, setPassword] = useState("");
11 |   const [showError, setShowError] = useState(false);
12 |
13 < const handleSubmit = (e) => {
14 |   e.preventDefault();
15 <   axiosInstance.post("users/login", {
16 |     email,
17 |     password
18 |
19 <   }).then((res) => {
20 |     // console.log("login req res----", res)
21 |     localStorage.setItem("token", res.data.token)
22 |     setUser(res.data.user)
23 |     // console.log("login req res----", res.data)
24 |     setEmail("")
25 |     setPassword("")
26 |     navigate("/");
27 |     setShowError(false)
28 <   }).catch((err) => {
29 |     setShowError(true)
30 |     console.log("error occurred in login axios ", err)
31 |   })
32 |
33 > };
34
35 < return [
36 >   <div className="bg-[#000000] h-screen w-screen font-raleway">
37 >     <div className="bg-[url('../images/bg.png')] h-full w-full">
38 >       <div className="h-screen w-full flex justify-center items-center">
39 >         <div className="h-auto w-0 md:w-11/12 xl:w-2/3 lg:w-2/3 justify-center flex bg-white rounded-md bg-clip-padding lg:backdrop-filter md:backdrop-filter shadow-2xl">
40 >           <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">
41 >             
42 >             <span className="flex flex-col items-center lg:mb-16 md:mt-16">
43 >               
44 >               <h3 className="text-3xl md:text-2xl font-bold text-black font-racing">
45 >                 ThyncSync
46 >               </h3>
47 >               <h3 className="text-2xl md:text-1xl md:px-8 md:text-center mt-3 text-black font-semibold">
48 >                 Welcome Again To Our Collaboration platform
49 >               </h3>
50 >               {showError && <div><h3 className="mt-3 mb-3 text-red-500">Invalid Email And Password !</h3></div>)}
51 >           </div>
52 >         </div>
53 >       </div>
54 >     </div>
55 >   </div>
56 > ];
```

```
50      <div>
51        {showError && <div><h3 className="mt-3 mb-3 text-red-500">Invalid Email And Password !</h3></div>}
52      </div>
53    <div className="mx-2 mt-10 p-2 mb: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">
54      <div className="flex flex-col justify-center h-full items-center">
55        <form onSubmit={handleSubmit}>
56
57          <div>
58            <h3 className="mb:text-base">Email</h3>
59            <input
60              value={email}
61              onChange={(e) => {
62                setEmail(e.target.value);
63              }}
64              autoComplete="off"
65              required
66              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"
67              type="email"
68              name="email"
69              id=""
70              placeholder="Please enter your Email"
71            />
72          </div>
73
74          <div>
75            <h3 className="mb:text-base">Password</h3>
76            <input
77              value={password}
78              autoComplete="off"
79              onChange={(e) => {
80                setPassword(e.target.value);
81              }}
82              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"
83              type="password"
84              required
85              name="password"
86              id=""
87              placeholder="Enter your password"
88            />
89          </div>
90
91        <div className="flex justify-between gap-3 xl:mt-10 mt-2 items-center">
92          <button
93            className="border-2 hover:bg-[#090909] hover:text-black xl:text-lg md:text-sm font-medium rounded-md p-1 px-5"
94            type="submit"
95            name="Create Account"
96          >
97            Create Account
98          </button>
99          <Link href="#">Login</Link>
100        </div>
101      </form>
102    </div>
103  </div>
```

```
Frontend > src > screens > Login.jsx > Log in
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
```

Register Page

```

Frontend > src > screens > Register.jsx > Register
Frontend > src > screens > Register.jsx > Register
1 | import React, { useContext, useState } from 'react'
2 | import { Link, useNavigate } from 'react-router-dom'
3 | import axiosInstance from '../config/axios';
4 | import { userContext } from '../context/user.context';
5 |
6 | const Register = () => [
7 |
8 |   const { setUser, setToken } = useContext(userContext);
9 |
10 |   const navigate = useNavigate();
11 |   const [username, setUsername] = useState("");
12 |   const [email, setEmail] = useState("");
13 |   const [password, setPassword] = useState("");
14 |   const [showError, setShowError] = useState(false);
15 |   const handleSubmit = (e) => {
16 |     e.preventDefault();
17 |     axiosInstance.post('users/register', {
18 |       username,
19 |       email,
20 |       password
21 |     }).then((res) => {
22 |       localStorage.setItem("token", res.data.token);
23 |       setUser(res.data.user)
24 |       console.log(res.data.user,"first")
25 |       setShowError(false)
26 |       navigate("/")
27 |       setEmail("")
28 |       setUsername("")
29 |       setPassword("")
30 |     }).catch((err) => {
31 |       setShowError(true)
32 |       console.log("error occurred in login axios ", err)
33 |     })
34 |
35 |
36 |
37 |
38 |
39 |
40 |
41 |
42 |
43 |
44 |
45 |
46 |
47 |
48 |
49 |
50 |
51 |
52 |
53 |
54 |
55 |
56 |
57 |
58 |
59 |
60 |
61 |
62 |
63 |
64 |
65 |
66 |
67 |
68 |
69 |
70 |
71 |
72 |
73 |
74 |
75 |
76 |
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 ];

```

return (

```


<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 justify-center flex bg-white rounded-md bg-clip-padding lg:backdrop-filter md:backdrop-filter shadow-lg border border-gray-200 rounded-lg p-4">
        <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">
            


```

```
frontend > src > screens > Register.jsx > 300 Register
  04
  05   onChange={(e) => {
  06     setPassword(e.target.value);
  07   }}
  08   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"
  09   type="password"
  10   required
  11   name="password"
  12   id=""
  13   placeholder="Enter your password"
  14 >/>
  15 </div>
  16
  17 <div className="flex justify-between gap-3 xl:mt-10 mt-2 items-center">
  18   <button
  19     className="border-2 hover:bg-[#D9D9D9] hover:text-black xl:text-lg md:text-sm font-medium rounded-md p-1 px-5"
  20     type="submit"
  21     name="Create Account"
  22   >
  23     Create
  24   </button>
  25   <div className="text-sm">
  26     already have an account?
  27     <Link
  28       to="/login"
  29       className="text-blue-400 text-base hover:text-[#D9D9D9]"
  30     >
  31       Login
  32     </Link>
  33   </div>
  34 </div>
  35 </div>
  36 </form>
  37 </div>
  38 </div>
  39 </div>
  40 </div>
  41 </div>
  42 </div>
  43 </div>
  44 </div>
  45 </div>
  46 </div>
  47 </div>;
  48
  49
  50
  51
  52
  53 export default Register
```

Home Page

Frontend > src > screens > Home.jsx

```
1 import React, { useContext, useEffect, useState } from 'react'
2 import { userContext } from '../context/user.context'
3 import { useNavigate } from 'react-router-dom';
4 import axiosInstance from './config/axios';
5 import projectBg from '/images/1.jpg';
6 import {
7   Dialog,
8   DialogContent,
9   DialogHeader,
10  DialogTitle,
11  DialogTrigger,
12 } from '@/components/ui/dialog"
13 import { Input } from '@/components/ui/input';
14 import { Button } from '@/components/ui/button';
15 import ShowProject from '@/components/project/components>ShowProject';
16 import { Louder2 } from 'lucide-react';
17
18 const Home = () => {
19   const { user } = useContext(userContext);
20   const [isModalOpen, setIsModalOpen] = useState(false);
21   const [projectName, setProjectName] = useState('');
22   const [projects, setProjects] = useState([]);
23   const [refreshProjects, setRefreshProjects] = useState(false);
24   const [showError, setShowError] = useState(false);
25   const [isPending, setIsPending] = useState(false);
26
27
28
29
30
31   function createProject(e) {
32     e.preventDefault();
33     // console.log(` ${projectName}`);
34     setIsPending(true);
35     axiosInstance.post('/project/create-project', {
36       name: projectName,
37     })
38     .then((res) => {
39       // console.log(res,"ooo")
40       setIsPending(false)
41       setProjectName('');
42       setRefreshProjects((prev) => !prev);
43       setShowError(false);
44     })
45     .catch((error) => {
46       console.log(error)
47       setShowError(true);
48     })
49   }
50
51
52 }
```

```

Frontend > src > screens > Home.jsx [diff] 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.   <div className='min-h-[84vh] w-full bg-cover bg-center p-4 rounded-md' >
  69.
  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.         </DialogContent>
 102.       </Dialog>
 103.     </div>
 104.
 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 && (...
 116.     ) */}
 117.
 118. </div>
 119. </main>
 120. </>
 121.
 122. <div>
 123.   <button onClick={() => refreshProjects()}> Refresh </button>
 124. </div>
 125.
 126.
 127. <div>
 128.   <button onClick={() => handleLogout()}> Logout </button>
 129. </div>
 130.
 131.
 132. <div>
 133.   <button onClick={() => handleLogout()}> Logout </button>
 134. </div>
 135.
 136.
 137. <div>
 138.   <button onClick={() => handleLogout()}> Logout </button>
 139. </div>
 140.
 141.
 142. <div>
 143.   <button onClick={() => handleLogout()}> Logout </button>
 144. </div>

```

```

Frontend > src > screens > Home.jsx [diff] 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 && (...
  116.     ) */}
  117.
  118. </div>
  119. </main>
  120. </>
  121.
  122. <div>
  123.   <button onClick={() => refreshProjects()}> Refresh </button>
  124. </div>
  125.
  126.
  127. <div>
  128.   <button onClick={() => handleLogout()}> Logout </button>
  129. </div>
  130.
  131.
  132. <div>
  133.   <button onClick={() => handleLogout()}> Logout </button>
  134. </div>
  135.
  136.
  137. <div>
  138.   <button onClick={() => handleLogout()}> Logout </button>
  139. </div>
  140.
  141.
  142. <div>
  143.   <button onClick={() => handleLogout()}> Logout </button>
  144. </div>

```

Chat page

```
Frontend > src > screens > Project.jsx > Project
50 // const [currentfile, setCurrentFile] = useState(null)
51 // const [openfiles, setopenFiles] = useState([])
52 // const messageBox = React.createRef()
53
54 const sendMessageHandle = () => {
55   if (!user) {
56     console.error("User not logged in");
57     return;
58   }
59
60   const messageobject = {
61     message,
62     sender: { _id: user._id, email: user.email },
63   };
64
65   sendMessage('project-message', messageobject);
66
67   // appendoutgoingMessages(messageobject);
68
69   setMessages(prevMessages => [...prevMessages, { sender: user, message }]);
70
71   setMessage('');
72 }
73
74 const addcollaborators = () => {
75   axiosInstance.put(`project/add-user`, {
76     projectId: location.state.project._id,
77     users: Array.from(selectUserid)
78   }).then((res) => {
79     setIsAdduserModalOpen(false)
80     setCollaboratorsRefresh((prev) => !prev);
81   }).catch((error) => {
82     console.log(error)
83   })
84 }
85
86
87 function WriteAMessage(message) {
88   let messageobject;
89   try {
90     messageobject = JSON.parse(message);
91   } catch (error) {
92     // If parsing fails, fallback to plain text
93     messageobject = { text: message };
94   }
95   return (
96     <div className="overflow-auto bg-slate-700 m-5 text-white shadow-md rounded-sm p-2">
97       <Markdown>
98         <children>{messageobject.text}</children>
99         <options></options>
100        <overrides></overrides>
101      </Markdown>
102    </div>
103  );
104}
105
106
107
108
109
110
111 useEffect(() => {
112   const handleResize = () => {
113     setIsLargeScreen(window.innerWidth >= 768);
114   };
115
116   window.addEventListener('resize', handleResize);
117
118   return () => {
119     window.removeEventListener('resize', handleResize);
120   };
121 }, []);
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
979
979
980
981
982
983
984
985
986
987
987
988
989
989
990
991
992
993
994
995
996
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1196
1197
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1296
1297
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1496
1497
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1596
1597
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1696
1697
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1787
1788
1789
1789
1790
1791
1792
1793
1794
1795
1796
1797
1797
1798
1799
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1887
1888
1889
1889
1890
1891
1892
1893
1894
1895
1896
1897
1897
1898
1899
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1987
1988
1989
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2087
2088
2089
2089
2090
2091
2092
2093
2094
2095
2096
2096
2097
2098
2099
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2187
2188
2189
2189
2190
2191
2192
2193
2194
2195
2196
2196
2197
2198
2199
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2259
2260
2261
2262
2263
2264
2265
2266
2
```

```

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    axiosInstance.get('users/all').then((res) => {
170      // console.log(res.data.users)
171      setUsers(res.data.users)
172    }).catch((error) => {
173      console.log(error)
174    })
175
176
177    return () => {
178      socketInstance.off("project-message", receiveMessage);
179    };
180  ), [collaboratorsRefresh])
181
182
183  const handleUserClick = (id) => {
184    if (selectUserId.includes(id)) {
185      // If the user is already selected, remove them
186      setSelectUserId(selectUserId.filter(user_id => user_id !== id));
187    } else {
188      // If the user is not selected, add them
189      setSelectUserId([...selectUserId, id]);
190    }
191  }
192
193
194  function scrollBottom() {
195    if (messageBox.current) {
196      messageBox.current.scrollTop = messageBox.current.scrollHeight;
197    }
198  }
199
200
201
202  useEffect(() => {
203    if (codeEditorRef.current) {
204      codeEditorRef.current.scrollTop = codeEditorRef.current.scrollHeight;
205    }
206  }, [messages]);
207
208
209
210  // console.log(isAddUserModalOpen, "-----")
211
212
213  return (
214    <main className='h-screen w-screen flex bg-white'>
215
216      <section className='left relative flex flex-col h-screen md:w-1/3 min-w-96 w-full bg-slate-300'>
217        <header className='flex justify-between items-center p-3 w-full bg-slate-200 absolute top-0'>
218
219          <button
220            onClick={() => ( setIsAddUserModalOpen(!isAddUserModalOpen); )}
221            className='flex gap-1 hover:bg-slate-300 px-2 py-1 rounded cursor-pointer'>
222            <i className='ri-add-fill'></i> Add Collaborators</span></button>
223
224          <button className='shadow-md px-2 bg-slate-400 rounded-md cursor-pointer'
225            onClick={() => ( setAsidePanelOpen(!setAsidePanelOpen) )}>
226            <i className='ri-team-fill text-2xl'></i>
227          </button>
228        </header>
229
230
231        <div className='conversation-area mt-14 flex-grow flex flex-col relative'>
232
233          >
234
235            <div
236              ref={messageBox}
237              className='message-box flex-1 flex flex-col gap-2 p-2 overflow-y-auto max-h-[calc(100vh-120px)] scrollbar-hide scroll-smooth'>
238
239              {messages.map((msg, index) => {
240                // Check if the message is from AI
241                const isAIMessage = msg.sender._id === 'ai';
242
243                // Render AI messages only in the right section for large screens
244                if (isAIMessage && !isLargeScreen) {
245                  return <div className='message flex flex-col p-2 bg-slate-500 text-white w-fit rounded-md'>
246                    <small className='opacity-65 text-xs'>{msg.sender.email}</small>
247                    <p className='text-sm'>i have given what you asked on the right side</p></div>; // skip rendering in the left section
248                }
249
250
251                return (
252                  <div
253                    key={index}>

```

```

Frontend > src > screens > Project.jsx > [0] Project
  1
  2
  3
  4
  5
  6
  7
  8
  9
 10
 11
 12
 13
 14
 15
 16
 17
 18
 19
 20
 21
 22
 23
 24
 25
 26
 27
 28
 29
 30
 31
 32
 33
 34
 35
 36
 37
 38
 39
 40
 41
 42
 43
 44
 45
 46
 47
 48
 49
 50
 51
 52
 53
 54
 55
 56
 57
 58
 59
 60
 61
 62
 63
 64
 65
 66
 67
 68
 69
 70
 71
 72
 73
 74
 75
 76
 77
 78
 79
 80
 81
 82
 83
 84
 85
 86
 87
 88
 89
 90
 91
 92
 93
 94
 95
 96
 97
 98
 99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
  
```

App Routes

```
Frontend > src > routes > AppRoutes.jsx > default
1 import React from 'react'
2 import { Route, Routes } from "react-router-dom"
3 import Login from '../screens/Login'
4 import Register from '../screens/Register'
5 import Home from '../screens/Home'
6 import Project from '../screens/Project'
7 import UserAuth from '../auth/UserAuth'
8
9 const AppRoutes = () => {
10   return (
11     <Routes>
12       <Route path='/' element={<UserAuth><Home/></UserAuth>}/>
13       <Route path='/login' element={<Login/>}/>
14       <Route path='/register' element={<Register/>}/>
15       <Route path='/project' element={<UserAuth><Project/></UserAuth>}/>
16     </Routes>
17   )
18 }
19
20 export default AppRoutes
```

App page

```
Frontend > src > App.jsx > App
1 import { useState } from 'react'
2 import AppRoutes from './routes/AppRoutes'
3 import UserProvider from './context/user.context'
4 import ErrorBoundary from './ErrorBoundary/ErrorBoundary'
5 import Navbar from './components/headerFooter/NavBar'
6 import Footer from './components/headerFooter/Footer'
7 import { useLocation } from 'react-router-dom'
8
9 function App() {
10   const location = useLocation();
11   const hideHeaderFooter = ['/login', '/register', '/project'].includes(location.pathname);
12
13   return (
14     <>
15       <UserProvider>
16         <ErrorBoundary>
17           [&hideHeaderFooter && <Navbar />]
18           <AppRoutes />
19           {!&hideHeaderFooter && <Footer />}
20         </ErrorBoundary>
21       </UserProvider>
22     </>
23   )
24
25 export default App
26
```

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

The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (empty), 'Environments' (empty), 'Flows' (empty), and 'History' (empty). The main area shows a project named 'Ai' with two collections: 'Projects' and 'User'. Under 'Projects', there are four GET requests: 'POST create', 'get all Projects', 'get Project', and 'GET Add user to project'. Under 'User', there are five POST requests: 'Register user', 'Login user', 'Get user Profile', 'Get All users', and 'Logout user'. In the center, a specific API endpoint is selected: 'GET /ai/getting result'. The URL is set to '({{localhost:3000}}) /ai/get-result?prompt=what is the rest api'. Below the URL, under 'Params', there are two entries: 'Key' and 'prompt', both with the value 'what is the rest api'. To the right of the URL, there are buttons for 'Save', 'Share', and 'Send'. The 'Send' button is highlighted in blue. At the bottom of the interface, there's a 'Response' section with a placeholder message 'Click Send to get a response' and a small cartoon character icon.

Postman API testing tool interface showing two requests:

POST Login user

HTTP User / Login user

Method: POST | URL: {{localhost3000}} /users/login

Body (JSON):

```

1 {
2   "email": "v@gmail.com",
3   "password": "V@123456"
4 }

```

POST Logout user

HTTP User / Logout user

Method: POST | URL: {{localhost3000}} /users/logout

Headers (8):

Key	Value	Description
Authorization	Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ...	
Key	Value	Description

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.

