

**MENTOR HUB WITH PROJECT TRACKER AND  
PLANNER**

**A PROJECT REPORT**

Submitted by

**SENTHILKUMAR T**  
(Reg. No: 24MCR099)

**SHREEVAISHNAVI B**  
(Reg. No: 24MCR100)

**SUGUMAR M**  
(Reg. No: 24MCR109)

*in partial fulfilment of the requirements  
for the award of the degree  
of*

**MASTER OF COMPUTER APPLICATIONS  
DEPARTMENT OF COMPUTER APPLICATIONS**



**KONGU ENGINEERING COLLEGE**

(Autonomous)

**PERUNDURAI, ERODE – 638 060**

**DECEMBER 2024**

**DEPARTMENT OF COMPUTER APPLICATIONS**

**KONGU ENGINEERING COLLEGE**

**(Autonomous)**

**PERUNDURAI ERODE-638 060**

**December 2024**

**BONAFIDE CERTIFICATE**

This is to certify that the project report entitled “**MENTOR HUB WITH PROJECT TRACKER AND PLANNER**” is the bonafied record of Project work done by **SENTHILKUMAR T** (Reg.No:24MCR099), **SHREEVAISHNAVI B** (Reg.No:24MCR100), **SUGUMAR M** (Reg.No:24MCR109) in partial fulfilment for the award of the Degree of Master of Computer Applications of Anna University, Chennai during the year 2024-2025.

**SUPERVISOR**

**HEAD OF THE DEPARTMENT**

**(Signature with seal)**

**Date:**

Submitted for the end semester viva voce examination held on \_\_\_\_\_

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## **DECLARATION**

We affirm that the project entitled “**MENTOR HUB WITH PROJECT TRACKER AND PLANNER**” being submitted in partial fulfilment of the requirements for the award of Master of Computer Applications is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidates.

**Date:**

(Signature of the candidate)

**SENTHILKUMAR T**

**(Reg:No:24MCR099)**

**SHREEVAISHNAVI B**

**(Reg:No:24MCR100)**

**SUGUMAR M**

**(Reg:No:24MCR109)**

We certify that the declaration made by the above candidates is true to the best of my knowledge.

**Date:**

**Name & Signature of the supervisor**

**(Mrs.T.KALPANA)**

## **ABSTRACT**

The project as Digital Platform for Agro Center focuses on developing a professional web application for a Agricultural needs. As demand for digital experiences in the retail sector continues to grow, businesses must keep up with changing customer expectations. The main objective of this project is to improve the customer experience of the Agricultural needs using a web application. A thorough exploration of web technologies, design concepts and e-commerce capabilities will result in an online platform that showcases stores and a wide range of products and provides an easy and engaging shopping experience.

This web application streamlines the purchasing process, ensuring security and efficiency. During online payments, customers receive a One-Time Password (OTP) via SMS, providing an additional layer of security to safeguard transactions. This feature not only protects sensitive information but also builds trust between the platform and its users.

By facilitating effective communication between the frontend and backend, API integration will improve features like order processing and user authentication. The application will provide a dynamic and responsive user experience because it was developed with the help of modern web technologies like React, HTML, CSS, JavaScript, and JSX. Furthermore, NPM will simplify dependency management throughout the development process, and JSON will be used for data transmission, guaranteeing smooth interoperability across several components. By offering a safe, effective, and entertaining online marketplace that satisfies the demands of both customers and companies in the agricultural industry, the Digital Platform for Agro Center ultimately seeks to revolutionize the retail landscape of agriculture.

## ACKNOWLEDGEMENT

We express our sincere thanks to our beloved Correspondent **Thiru.A.K.ILANGO B.Com., M.B.A., LLB.**, and other philanthropic trust members of Kongu Vellalar Institute of Technology Trust for having provided with necessary resources to complete this project. We are always grateful to our beloved visionary Principal **Dr.V.BALUSAMY B.E.(Hons), M.Tech., Ph.D.**, and thank him for his motivation and moral support.

We express our deep sense of gratitude and profound thanks to **Dr.A.TAMILARASI M.sc.,M.Phil.,Ph.D.,M.Tech.**, Head of the Department and Associate Professor in Computer Application for her invaluable commitment and guidance for this project.

We also like to express our gratitude and sincere thanks to our project coordinators **Mrs.S.HEMALATHA MCA.,(Sr.G)** Assistant Professor, Department of Computer Applications, Kongu Engineering College who have motivated us in all aspects for completing the project in scheduled time.

We would like to express our gratitude and sincere thanks to our project guide **Mrs.T.KALPANA MCA.**, Assistant Professor, Department of Computer Applications, Kongu Engineering College for giving his valuable guidance and suggestions which helped us in the successful completion of the project.

We owe a great deal of gratitude to our parents for helping us to overwhelm in all proceedings. We bow our heart and head with heartfelt thanks to all those who thought us their warm services to succeed and achieve our work.

# TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No.
	ABSTRACT	iv
	ACKNOWLEDGEMENT	v
	LIST OF FIGURES	ix
	LIST OF ABBREVIATIONS	xi
1.	INTRODUCTION	1
	1.1 ABOUT THE PROJECT	1
	1.2 EXISTING SYSTEM	1
	1.3 DRAWBACKS OF EXISTING SYSTEM	1
	1.4 PROPOSED SYSTEM	2
	1.5 ADVANTAGES OF PROPOSED SYSTEM	2
2.	SYSTEM ANALYSIS	3
	2.1 IDENTIFICATION OF NEED	3
	2.2 FEASIBILITY STUDY	3
	2.2.1 TECHNICAL FEASIBILITY	4
	2.2.2 OPERATIONAL FEASIBILITY	4
	2.2.3 ECONOMIC FEASIBILITY	4

	2.3 SOFTWARE REQUIREMENT SPECIFICATION	5
	2.3.1 SOFTWARE REQUIREMENT	5
	2.3.2 HARDWARE REQUIREMENT	5
	2.4 SOFTWARE DESCRIPTION	5
<b>3.</b>	<b>SYSTEM DESIGN</b>	<b>8</b>
	3.1 MODULE DESCRIPTION	8
	3.1.1 DASHBOARD MODULE	8
	3.1.2 PRODECT MODULE	8
	3.1.3 MEMBER MODULE	9
	3.1.4 PROJECT MODULE	9
	3.1.5 AUTHENTICATION MODULE	10
	3.2 SYSTEM FLOW DIAGRAM	11
	3.3 USE CASE DIAGRAM	14
	3.4 DATABASE DESIGN	16
	3.5 DATA FLOW DIAGRAM	17
	3.6 INPUT DESIGN	20
	3.6 OUTPUT DESIGN	21
<b>4.</b>	<b>IMPLEMENTATION</b>	<b>22</b>
	4.1 SYSTEM IMPLEMENTATION	22
	4.2 CODING	22
	4.3 TESTING	23
	4.4 INSTALLATION	23
	4.5 DOCUMENTATION	24

<b>5.</b>	<b>TESTING AND RESULTS</b>	<b>25</b>
	5.1 TESTING	25
	5.1.1 UNIT TESTING	25
	5.1.2 INTEGRATION TESTING	27
<b>6.</b>	<b>CONCLUSION AND FUTURE ENHANCEMENT</b>	<b>30</b>
	6.1 CONCLUSION	30
	6.2 FUTURE ENHANCEMENT	30
	<b>APPENDICES</b>	<b>31</b>
	<b>A.SAMPLE CODING</b>	<b>31</b>
	<b>B.SCREENSHORT</b>	<b>44</b>
	<b>REFERENCES</b>	<b>49</b>



## LIST OF FIGURES

FIGURE No.	TITLE	PAGE No.
3.2	User System Flow Diagram	11
3.3	Use Case Diagram	14
3.4	DATABASE DESIGN	16
3.5	DATA FLOW DIAGRAM	17
B.1	User Sign Up Page	44
B.2	User Login Page	45
B.3	Home Page	45
B.4	Dashboard Page	46
B.5	User Profile Page	47
B.6	Member Page	47
B.7	Project Page	48
B.8	View Project Page	48

## LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
MERN MongoDB,	Express.js, React.js, Node.js
HTML	Hypertext Mark up Language
CSS	Cascading Style Sheets
JS	JavaScript
JSON JavaScript Object	JavaScript Object
API	Application Programming Interface
DOM	Document Object Model
NPM	Node Package Manager
JWT	JSON Web Token
UI	User Interface
CRUD	Create, Read, Update, Delete
DB	Database
UX	User Experience

# **CHAPTER 1**

## **INTRODUCTION**

### **1.1 ABOUT THE PROJECT**

Mentor Hub with Project Tracker & Planner is an innovative web application designed to enhance real-time collaboration and simplify project management for teams. Built on the MERN stack—comprising MongoDB, Express.js, React.js, and Node.js—it integrates modern web technologies to deliver a seamless user experience. This platform serves as a centralized hub where developers, educators, and remote teams can collaborate on coding projects, manage tasks, and track progress efficiently. Its design focuses on fostering productivity, teamwork, and streamlined project execution.

The platform provides a robust project management dashboard tailored for team-based workspaces. Each team can manage multiple projects with features that include task creation, assignment, and progress tracking.

### **1.2 EXISTING SYSTEM**

The platform includes a feature-rich project management dashboard designed for team-based workflows, enabling task creation, assignment, and progress tracking in real time. These features collectively enhance teamwork, productivity, and streamlined execution, making Mentor Hub a versatile solution for developers, educators, and remote teams. Its current system effectively supports dynamic collaboration, precise task management, and centralized project oversight.

### **1.3 DRAWBACKS OF EXISTING SYSTEM**

- Limited Integration with Version Control
- Basic Security Features
- No Analytics Dashboard
- Absence of Notification System

## **1.4 PROPOSED SYSTEM**

The proposed system for Mentor Hub with Project Tracker & Planner is a MERN-stack-based platform designed to facilitate real-time collaboration, project management, and team communication. It features secure user authentication using JWT and role-based access control, dynamic workspaces with live coding powered by Socket.io, and a project management dashboard for task delegation and progress tracking.

Integration with GitHub/GitLab, a notification system, and team analytics further enhance usability. The platform prioritizes scalability, security, and a seamless user experience, making it ideal for developers, educators, and remote teams to collaborate efficiently and achieve project goals.

## **1.5 ADVANTAGES OF THE PROPOSED SYSTEM**

- Real-Time Collaboration
- Streamlined Project Management
- Secure and Scalable Authentication
- Enhanced Productivity
- Integration with Development Tools
- Versatility for Diverse Use Cases
- Modern and Responsive Design
- Future-Proof Scalability
- Improved Communication

## CHAPTER 2

### SYSTEM ANALYSIS

#### 2.1 IDENTIFICATION OF NEED

The need for Mentor Hub with Project Tracker & Planner arises from the challenges faced by developers, educators, and remote teams in managing real-time collaboration and project workflows effectively. Existing tools often lack integration, requiring teams to juggle multiple platforms for coding, task management, and communication, leading to inefficiencies and fragmented processes. current systems fall short in providing secure environments and features tailored for dynamic, distributed collaboration.

Mentor Hub addresses these gaps by offering a centralized platform that integrates real-time coding, robust project management, and secure user authentication, all designed to enhance productivity, streamline teamwork, and support diverse use cases like software development, education, and remote work.

#### 2.2 FEASIBILITY STUDY

The feasibility study for **Mentor Hub with Project Tracker & Planner** demonstrates that the platform is technically achievable, operationally viable, economically promising, and legally compliant. Leveraging the MERN stack, the platform ensures scalability, reliability, and efficient development, while Socket.io enables real-time collaboration and JWT provides secure authentication. The target audience, including developers, educators, and remote teams, indicates strong market demand and ease of adoption due to intuitive design and essential features.

### **2.2.1 Technical Feasibility**

which is well-suited for building scalable and interactive web applications. Real-time collaboration is enabled through Socket.io, ensuring smooth data synchronization and minimal latency during multi-user sessions. Authentication is managed securely with JSON Web Tokens (JWT), and planned enhancements like two-factor authentication (2FA) and role-based access control (RBAC) bolster security. The platform's architecture supports future scalability, allowing the integration of features such as GitHub integration and analytics dashboards. The chosen technologies are mature and supported by a wide developer community, ensuring technical viability.

### **2.2.2 Operational Feasibility**

The platform targets developers, educators, and remote teams, all of whom have a growing need for centralized tools to manage projects and collaborate in real-time. Its intuitive interface minimizes the learning curve, facilitating adoption even for users with minimal technical expertise. Operational support is feasible with a dedicated team familiar with the MERN stack, ensuring regular updates, maintenance, and responsiveness to user feedback. The platform's functionality aligns well with market demand, addressing gaps in existing solutions by integrating coding, task management, and collaboration into one tool.

### **2.2.3 Economic Feasibility**

Development costs include hiring developers, acquiring cloud infrastructure, and operational expenses, all of which are within a typical SaaS startup budget. Revenue opportunities include subscription plans for individuals and teams, tiered pricing for premium features, and institutional licenses for educational and corporate clients. With the rise in demand for remote collaboration tools, the platform is well-positioned to capture a significant market share. The potential for recurring revenue streams ensures a high return on investment (ROI) over time, making the project economically viable.

## **2.3 SOFTWARE REQUIREMENT SPECIFICATION**

### **2.3.1 Software Requirements**

Operating System	: Windows 11 and above
Environment	: Chrome, Visual Studio Code
Frontend	: HTML, CSS, React.JS
Scripting language	: JavaScript
Backend	: JWT
Database	: Mongo DB

### **2.3.2 Hardware Requirements**

Processor	: Intel Core i5
RAM	: 4 GB RAM
Hard disk	: 500 GB
Keyboard	: Standard 102 keys
Mouse	: Optical Mouse

## **2.4 SOFTWARE DESCRIPTION**

**Front End : HTML , CSS , REACT JS**

## HTML

HTML, or Hypertext Markup Language, is the foundational structure of the World Wide Web, enabling the creation of documents that browsers can display. Each element is defined by tags that enclose content and can include attributes to specify behavior or appearance, such as the "src" and "alt" attributes in the `<img>` tag. HTML documents are organized hierarchically, starting with a root `<html>` element that contains `<head>` and `<body>` sections, where the `<head>` includes metadata and the `<body>` holds visible content. HTML5, the latest version, introduced new semantic elements and enhanced support for multimedia and interactive features, improving modern web development practices.

## CSS

CSS (Cascading Style Sheets) is a vital tool for shaping the visual presentation and layout of web pages, serving as the styling language that dictates the appearance of HTML elements and enhances user experience. It enables the creation of responsive and adaptive designs through features like media queries, ensuring optimal display across various devices. CSS allows for extensive customization of colors, fonts, spacing, and other stylistic attributes, fostering a coherent design language.

## REACT.JS

React.js is an open-source JavaScript library developed by Facebook for building user interfaces in web applications. It utilizes a component-based architecture, allowing developers to create reusable and maintainable UI components. React employs a virtual DOM to optimize UI updates, enhancing performance by reducing direct interactions with the actual DOM. The library supports a declarative programming style, enabling developers to define the desired UI state easily.

With unidirectional data flow, React simplifies state management, and its ecosystem includes tools like React Router for routing and Redux for state management. Additionally, React supports server-side rendering (SSR) and static site generation (SSG), which improve performance and SEO.



## Back End

### JWT

**JWT (JSON Web Token)** and **Socket.IO** play crucial roles in ensuring secure, real-time collaboration among students, mentors, and faculty. **JWT** serves as the backbone of the authentication system, allowing users to securely log in and access personalized features. When a user logs in, they receive a token that grants access to protected routes, such as project creation, viewing assignments, or managing their workspace. The token is validated with each request, ensuring that only authorized users can perform specific actions. This stateless authentication model provides scalability and security, which is essential for an environment with many users and sensitive project data. On the other hand, **Socket.IO** enables real-time communication and collaboration between users. This allows team members to instantly update each other on project progress, share insights, or discuss issues without needing to refresh their screens. Whether it's an urgent code review, a team meeting, or task delegation, Socket.IO facilitates seamless interaction.

## DATABASE

### MongoDB

It serves as the primary database, providing a flexible and scalable solution for storing user data, project details, and real-time collaboration information. MongoDB is a NoSQL database that uses a document-based model, which allows for a dynamic schema and the storage of complex, hierarchical data structures. This is particularly useful for managing various user roles (students, mentors, faculty), project details, and interactions within workspaces. Each user's information, including authentication details (stored in encrypted form), is stored in user-specific collections, allowing for fast retrieval and updates. For project management, MongoDB stores tasks, deadlines, progress updates, and communication logs in separate collections. This structure allows project teams to track the status of individual tasks and make changes in real time.

## **CHAPTER 3**

### **SYSTEM DESIGN**

#### **3.1 MODULE DESCRIPTION**

The Mentor Hub with Workspace module is a core component of the Mentor Hub platform, designed to foster real-time collaboration and streamline project management for coding teams. This module provides a dynamic, team-based workspace where users can interact, share ideas, code simultaneously, and manage tasks related to their projects. The workspace serves as a centralized location for team activities, ensuring seamless collaboration, efficient task management, and real-time updates.

##### **3.1.1 Dashboard Module**

The Dashboard module serves as the central hub for users to access an overview of their activities, projects, tasks, and discussions. It provides a quick glance of key data such as upcoming deadlines, ongoing projects, and recent updates, enhancing the user's ability to stay on top of their responsibilities. For the backend, the dashboard fetches data from various modules like Projects, To-Do Lists, and Discussions, ensuring that the data is dynamic and up-to-date. JWT authentication secures the module, ensuring that only authorized users can access the dashboard. On the frontend, users are presented with an intuitive interface, featuring statistics, progress bars, and summary views of key activities. These elements make it easy for mentors and students to track their progress, communicate with teammates, and stay organized. The Dashboard also offers visualizations, such as graphs or pie charts, to highlight project milestones and task completion rates. Ultimately, the Dashboard module provides a convenient and efficient way for users to monitor their workflows and ensure productivity within the platform.

### **3.1.2 Profile Module**

The Profile module is where users can manage and update their personal details, such as their name, bio, profile picture, and contact information. This module allows users to create a personalized space within the platform, making it easier for mentors and mentees to connect and collaborate. The backend ensures that all user data is securely stored and accessible only by the authenticated user, with JWT being used to verify identity and permissions. The module also supports features such as uploading a profile picture and managing notifications. Additionally, users can track their past contributions, including projects and tasks they've been involved with, providing a sense of accomplishment and history. On the frontend, users are presented with a clean and easy-to-navigate profile page, allowing them to update their details as needed. The Profile module plays a crucial role in building relationships on the platform, as it allows members to introduce themselves, highlight their skills, and share their achievements. It ensures that mentors and students can always keep their profiles current and accurate, fostering effective communication and collaboration.

### **3.1.3 Member Module**

The Member module is designed to manage the users within the platform, including students, mentors, and admins. It facilitates the registration, updating, and deletion of user accounts. The backend handles user roles and permissions, ensuring that each user has appropriate access to the platform's features based on their role (e.g., mentor, student). JWT authentication is used to ensure secure access and authorization, preventing unauthorized users from performing restricted actions. Admins can also perform CRUD (Create, Read, Update, Delete) operations on member data, such as assigning or changing roles. On the frontend, the Member module allows administrators to easily search for, view, and manage users. It can display member profiles, and roles, and provide options to invite or remove users from the platform.

### 3.1.4 Project Module

The Project module is a core part of the platform, enabling users to create, manage, and collaborate on projects. The backend ensures that each project can be associated with relevant members, allowing for seamless collaboration between mentors and students. Using JWT, the system ensures that only authorized users can create or modify projects. The project data, such as deadlines, milestones, and statuses, are stored and tracked, helping members monitor their progress. This module allows projects to be organized into tasks, which can be assigned to specific team members. On the frontend, the Project module provides a user-friendly interface for creating new projects, assigning roles, and managing project timelines. Features like project descriptions, file attachments, and progress tracking make it easy for all team members to stay aligned on the project's goals and deadlines. This module also supports the real-time updating of project data, ensuring that all users have access to the latest information. By managing and organizing projects effectively, the Project module enhances collaboration, communication, and accountability within the platform.

### 3.1.5 Authentication Module

Critical component that ensures secure access to the platform's features by verifying the identity of users. It leverages MongoDB for storing user credentials and JWT (JSON Web Tokens) for authentication. The user registration process involves securely capturing user details such as name, email, and password. The password is hashed using a secure hashing algorithm like bcrypt before being stored in the MongoDB database. Once the data is saved, a JWT token is generated, which serves as a credential for future authentication requests. When a user logs in, the system verifies the provided credentials against the stored data in MongoDB. The entered password is compared with the stored hashed password, and if valid, the server generates a new JWT token. This token is sent back to the frontend, where it is stored securely, typically in localStorage or cookies, to persist the user's session across multiple requests. The token contains encoded user information, such as their role (mentor, student), which helps manage access control to various sections of the platform. This module ensures that only authenticated and authorized users can access sensitive data and features, keeping the platform secure and protecting user privacy while offering a seamless user experience.

### 3.2 SYSTEM FLOW DIAGRAM

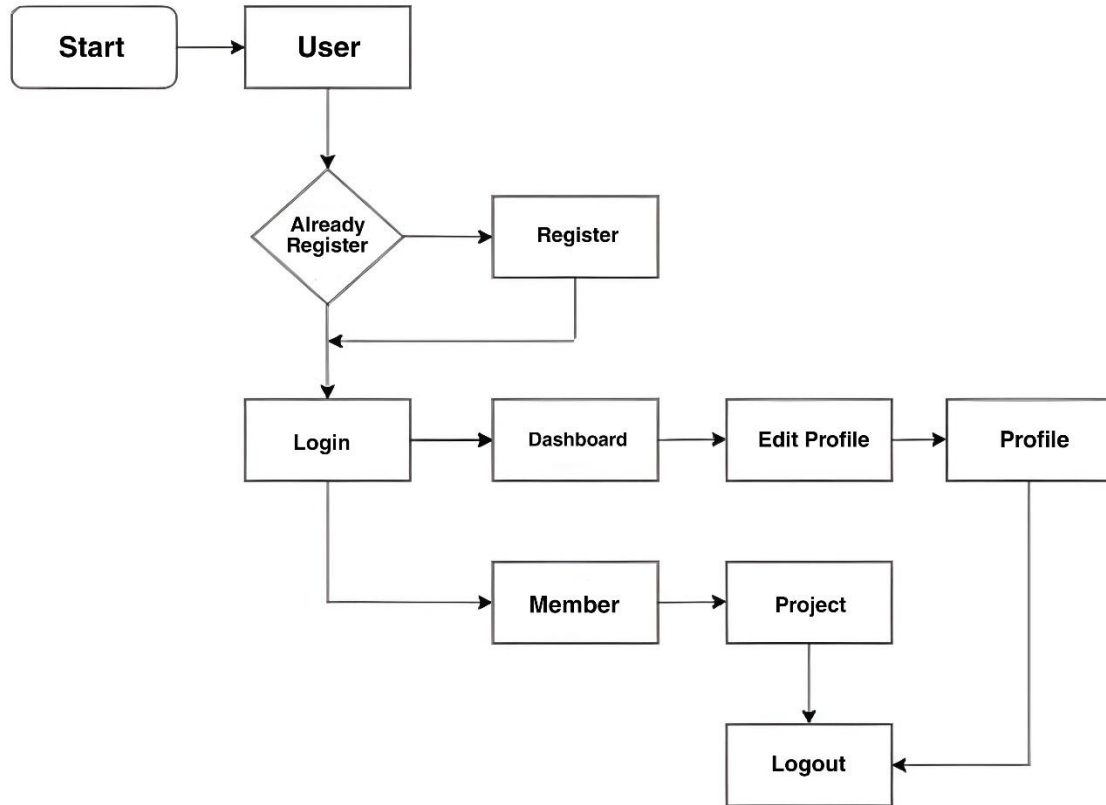


Figure 3.1 User System Flow Diagram

#### Description for the provided flow diagram:

**Shapes:** The flowchart uses rectangles for processes (e.g., Start, User, Register, Login, Dashboard, Edit Profile, Profile, Member, Project, Logout), and a diamond for a decision (Already Registered?). Arrows indicate the flow of steps, showing the sequence of actions and decisions within the system. This visually represents a user's interaction with a system, likely a web application or similar.

#### DESCRIPTION:

1. **Start:** The process begins, signifying the user's initial interaction with the system, perhaps by navigating to a website or opening an application.
2. **User:** This represents the user interacting with the system. It's the starting point for user-initiated actions.
3. **Already Registered?:** This is a decision point. The system checks if the user has an existing account. This check might involve looking for stored cookies, session data, or prompting the user to enter their credentials.

- **Yes (User is Registered):** The flow proceeds directly to the Login process.
  - **No (User is Not Registered):** The flow directs the user to the Register process.
4. **Register:** This is the user registration process. It likely involves:
- The user filling out a registration form with information like name, email, password, etc.
  - Validation of the entered data (e.g., checking for valid email format, password strength).
  - Creation of a new user account in the system's database.
  - Potentially sending a confirmation email to the user. After successful registration, the flow proceeds to the Login process.
5. **Login:** This is the user login process. It typically involves:
- The user entering their registered credentials (e.g., username/email and password).
  - The system verifying these credentials against the stored user data.
  - Upon successful verification, creating a user session and granting access to the system's features. If login fails (e.g., incorrect credentials), the user would likely be prompted to re-enter their information or offered a password recovery option (not shown in the diagram).
6. **Dashboard:** After successful login, the user is redirected to the dashboard. This is a central hub providing access to various features and information. It might display:
- Personalized information for the user.
  - Navigation links to other sections of the system.
  - Recent activity or notifications.
7. **Edit Profile:** From the dashboard, the user can choose to edit their profile information.
8. **Profile:** This section allows the user to view and modify their personal details. This may include:
- Updating contact information.
  - Changing password.
  - Managing preferences. From the profile view, the user can return to the dashboard or choose to Logout.
9. **Member:** This section suggests access to member-specific features or content. This could involve:
- Viewing a list of members.
  - Accessing member-only forums or groups.

- Managing memberships or subscriptions. It's unclear from the diagram how the user gets here (from the dashboard or login page directly), which could be clarified in a more detailed diagram.

10. **Project:** This section provides access to project-related information or functionalities. This may include:

- Viewing project details.
- Collaborating on projects.
- Managing project tasks. The diagram suggests that this is accessed through the "Member" section.

11. **Logout:** This process terminates the user's session and logs them out of the system. This can be accessed from multiple points in the flow (Profile and Project), allowing the user to log out at any time after logging in.

### 3.3 USE CASE DIAGRAM

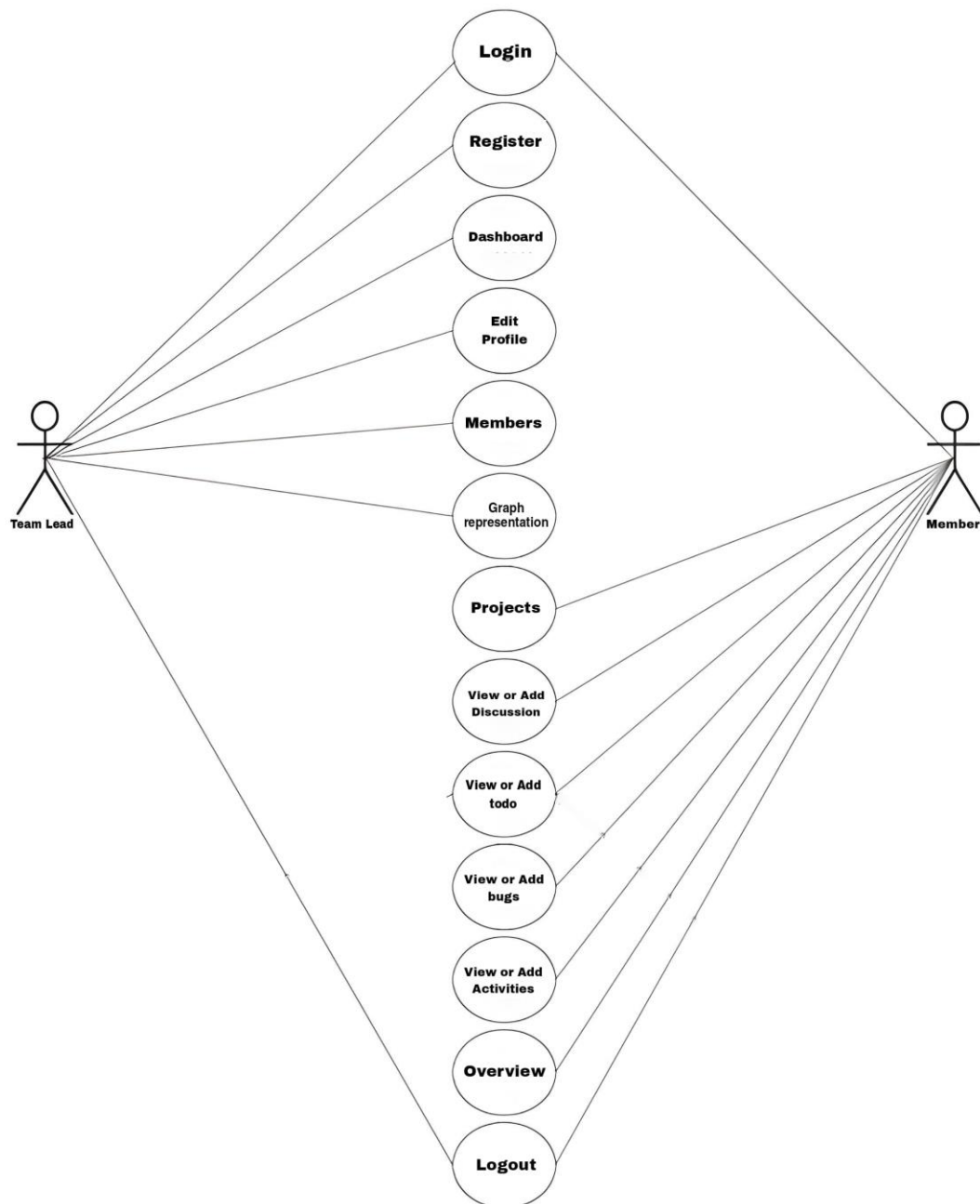


Figure 3.3 Project Tracker Use Case Diagram



**DESCRIPTION:****Use Cases:**

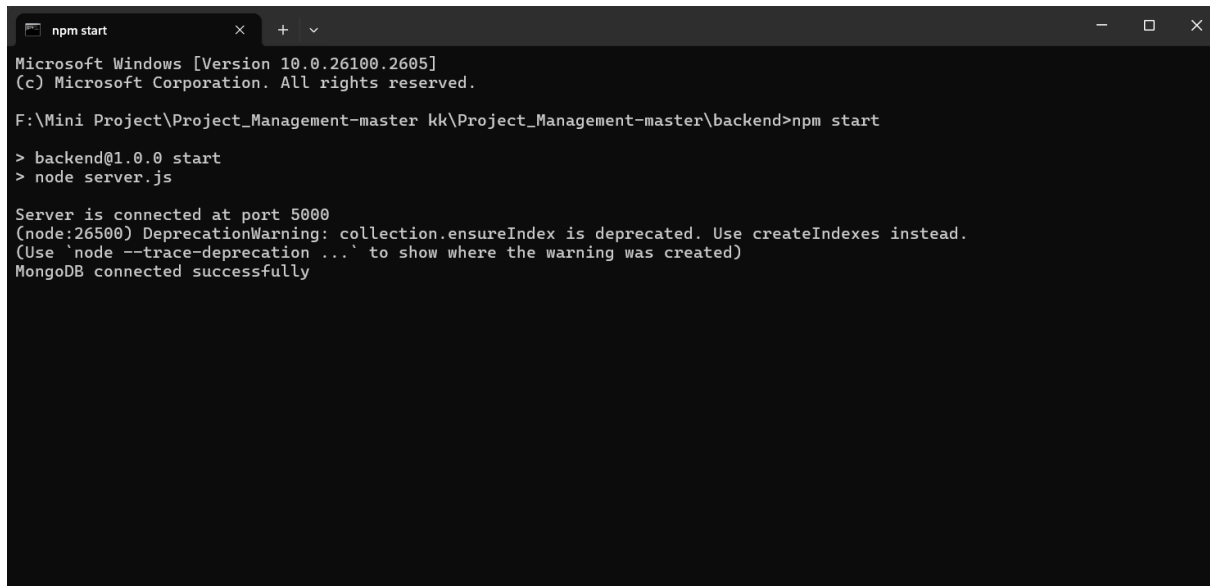
1. Login : For both Admin and User.
2. Register : For Users to create an account.
3. Create Project : For Admin to add new projects to the system.
4. Manage Projects : Admin oversees existing projects.
5. Manage Tasks : Admin and authorized Users manage tasks within projects.
6. Manage Users : Admin manages user accounts.
7. View Users : Admin can view user profiles.
8. View Projects : Users browse projects.
9. View Project Details : Users view details of a specific project.
10. Create Task : Users create new tasks within assigned projects.
11. Update Task Status : Users update the status of their assigned tasks.
12. Update Task Details : Users modify details of their tasks.
13. View Profile : Users view their personal details.
14. Logout : Ends the session for all actors.

**Relationships:**

1. Lines connect the actors to the use cases they interact with.
2. Admin has more system management-related use cases.
3. Users are mainly focused on project and task management activities.

### 3.4 DATABASE DESIGN

The backend of this project utilizes Node.js and Express.js to create a robust and efficient server-side environment. This server is configured to listen on port 5000, establishing a communication channel for client-side applications to interact with the project's data and logic. A crucial component of the backend infrastructure is the integration with MongoDB, a NoSQL database that provides flexible and scalable data storage. The successful connection to the MongoDB database is confirmed by the console message "MongoDB connected successfully," indicating that the server can effectively interact with the database to perform operations such as data retrieval, storage, and manipulation.



```
npm start
Microsoft Windows [Version 10.0.26100.2605]
(c) Microsoft Corporation. All rights reserved.

F:\Mini Project\Project_Management-master kk\Project_Management-master\backend>npm start

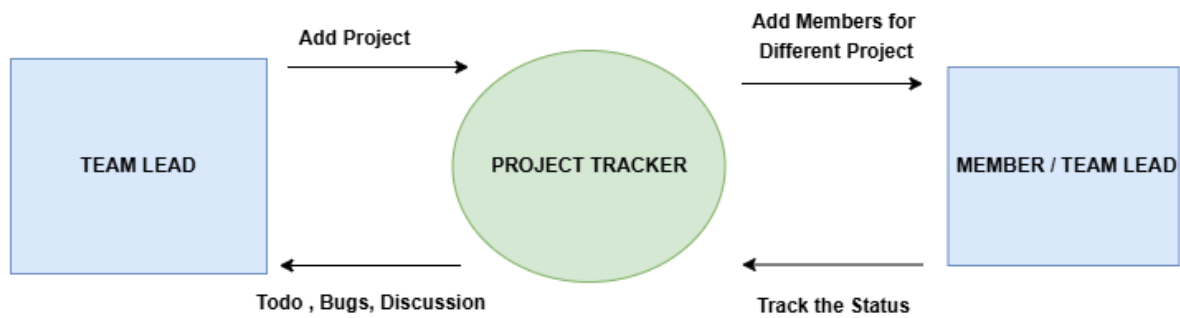
> backend@1.0.0 start
> node server.js

Server is connected at port 5000
(node:26500) DeprecationWarning: collection.ensureIndex is deprecated. Use createIndexes instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
MongoDB connected successfully
```

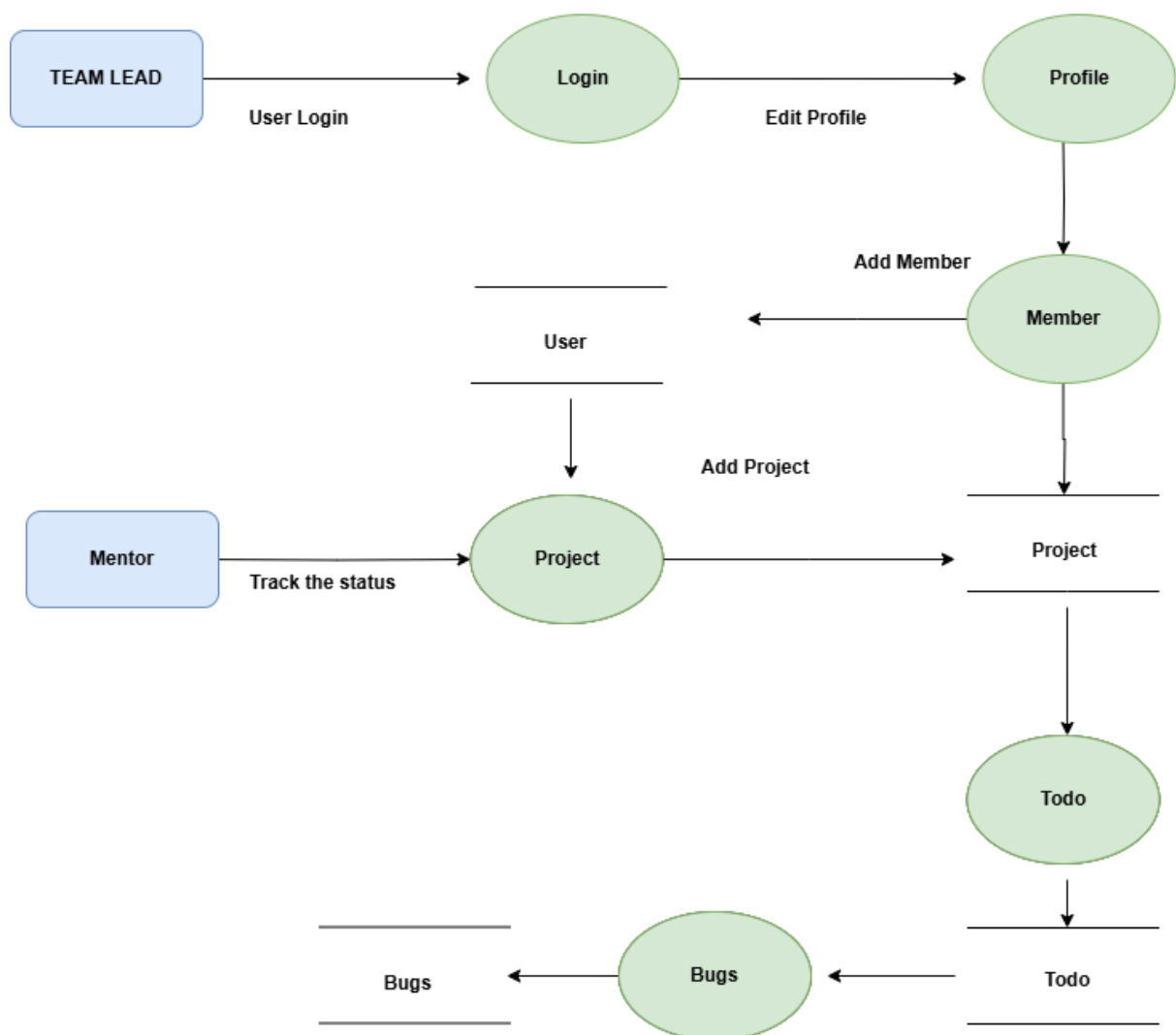
Figure 3.4 Mongo Database Structure

### 3.5 DATAFLOW DIAGRAM

#### Level 0



#### Level 1



This diagram represents a Data Flow Diagram (DFD) for a project tracking system, showing the flow of data between different entities and processes. It's presented in two levels: Level 0 (Context Diagram) and Level 1 (Detailed Diagram).

#### Level 0: Context Diagram

This diagram provides a high-level overview of the system and its external entities.

- External Entities
  - TEAM LEAD: Provides input to the system by adding projects and members.
  - MEMBER/TEAM LEAD: Receives output from the system by tracking the status of projects.
- Process
  - PROJECT TRACKER: Represents the entire project tracking system as a single process.
- Data Flows
  - Add Project: Data flows from the TEAM LEAD to the PROJECT TRACKER, representing the addition of new projects.
  - Add Members for Different Project: Data flows from the TEAM LEAD to the PROJECT TRACKER, representing the addition of members to projects.
  - Todo, Bugs, Discussion: Data flows within the PROJECT TRACKER, representing internal data storage and processing related to tasks, bugs, and discussions.
  - Track the Status: Data flows from the PROJECT TRACKER to the MEMBER/TEAM LEAD, representing the provision of project status updates.

#### Level 1: Detailed Diagram

This diagram decomposes the PROJECT TRACKER process from Level 0 into more detailed sub-processes and data stores.

- External Entities
  - TEAM LEAD: Same as in Level 0.

- Mentor: Tracks the status of projects, similar to MEMBER/TEAM LEAD in Level 0, but shown as a separate entity here.
- Processes
  - Login: Handles user authentication.
  - Profile: Manages user profile information.
  - Member: Manages project membership.
  - Project: Manages project information.
  - Todo: Manages tasks within projects.
  - Bugs: Manages bug reports.
- Data Stores
  - The horizontal lines represent data stores (databases or files) that hold information. We can infer the following data stores:
    - User data (accessed by Login, Profile, and Member processes)
    - Project data (accessed by Project, Member, and Todo processes)
    - Todo data (accessed by Todo and Bugs processes)
    - Bugs data (accessed by Bugs process)
- Data Flows
  - User Login: Data flows from the TEAM LEAD to the Login process.
  - Edit Profile: Data flows between the Login process and the Profile process.
  - Add Member: Data flows between the Profile process and the Member process.
  - User: Data flows from the Member process to the Project process.
  - Add Project: Data flows from the Mentor to the Project process.
  - Project: Data flows between the Project process and the Todo process.
  - Todo: Data flows between the Todo process and the Bugs process.
  - Bugs: Data flows within the Bugs process.

- Track the status: Data flows from the Project process to the Mentor.

### Key Observations

- The Level 1 DFD shows a more detailed breakdown of the system's functionality, including user authentication, profile management, project management, task management, and bug tracking.
- The diagram clearly shows the flow of data between different processes and data stores.
- The use of data stores helps to visualize where data is stored within the system.
- The diagram could be further improved by labeling the data flows with more descriptive names (e.g., instead of just "User," a more descriptive label like "User Project Assignment" could be used).

**Summary:** The DFD provides a clear representation of the data flow within the project tracking system. The two levels offer different levels of detail, allowing for both a high-level understanding and a more granular view of the system's components and their interactions.

## 3.6 INPUT DESIGN

The Input Design for your Mentor Hub project is focused on creating an intuitive and user-friendly interface to allow seamless interaction between users and the system. The Registration and Login Forms enable users to create an account or log in to the platform by providing essential credentials like username, email, and password. These forms include text fields, masked password entries for security, and validation checks, such as ensuring the email is correctly formatted and passwords meet strength requirements. Feedback messages, such as error alerts or success notifications, guide users through the process. For the Profile Module, users can input their personal details, including name, bio, and contact information. They can also upload a profile picture via an image upload field. In the Project Module, users can create and manage projects by entering a project title, description, timeline, and assigning team members. Dropdown menus allow easy assignment of roles within the project. The Output Design ensures that users can easily access and interpret the data presented.

### 3.7 OUTPUT DESIGN

The **Output Design** of your Mentor Hub project focuses on presenting information in a user-friendly, organized, and visually appealing manner, ensuring that users can easily access and understand the data. The **Dashboard** acts as the central hub, displaying an overview of ongoing projects, upcoming tasks, and key notifications. It includes visually distinct widgets with progress bars, color-coded indicators, and recent activity logs to help users stay updated. The **Profile Module** showcases user details, including their bio and profile picture, in a clean, editable format. It allows users to quickly view and update their personal information. The **Project Module** presents a list of active projects with key details such as title, description, timeline, and task assignments. Projects are displayed in easily clickable cards or sections that expand for more information, including member roles and project milestones. The **Discussion Module** offers a chat-style interface for project-related conversations, with real-time message updates, thread organization, and participant details. Lastly, the **To-Do List Module** allows users to manage tasks, with checkboxes for marking completion, editable deadlines, and priority levels. The output design is centered around clarity and accessibility, ensuring users can efficiently navigate the system and manage their tasks and projects.

## CHAPTER 4

### IMPLEMENTATION

#### 4.1 SYSTEM IMPLEMENTATION

The **System Implementation** of the Mentor Hub project involves translating the design into a working application using the MERN stack (MongoDB, Express.js, React.js, Node.js). The backend implementation starts with setting up an Express.js server that handles HTTP requests, manages routes, and processes business logic. MongoDB is used as the database to store user data, project details, tasks, and discussion threads. Data is organized into collections like users, projects, and tasks, ensuring efficient data retrieval and management. JWT (JSON Web Tokens) is integrated for secure authentication and authorization, enabling users to log in and access protected resources. The frontend is developed using React.js, which ensures a dynamic, single-page application (SPA) with a smooth user experience. Components are structured modularly for maintainability, with each module (Dashboard, Profile, Project, Discussion, To-Do List) developed as a separate React component. To manage state efficiently, React's state management is used, and Redux can be employed for larger-scale state handling. Real-time features like messaging and project updates are implemented using Socket.io, enabling instant communication and updates across users. System implementation also includes integrating third-party libraries for validation, error handling, and UI components to enhance functionality and the user interface's overall aesthetics.

#### 4.2 CODING

**Coding** for the Mentor Hub project is structured across both frontend and backend components. The **backend** is built using Node.js and Express.js. The first step involves setting up a server with Express.js that listens for incoming HTTP requests, processes the requests, and communicates with MongoDB to fetch or save data. MongoDB is used to store essential data like user credentials, project information, tasks, and discussion threads. The database is connected using Mongoose, which provides an easy-to-use interface for interacting with MongoDB. JWT (JSON Web Tokens) is used for user authentication. When a user logs in, their credentials are validated, and a JWT is generated, providing secure access to protected routes. The **frontend** is developed using React.js, ensuring that the user interface is dynamic, responsive, and interactive. The React components are designed to handle various functionalities, including user profiles, task management, and project tracking. State management is implemented using React's `useState` or



more advanced solutions like Redux for centralized state management. Each module, such as the Dashboard, Project Module, and Discussion Module, is built separately for maintainability and scalability. The UI is designed to be user-friendly, with clear navigation and an intuitive layout. Backend APIs are connected to the frontend using Axios for data fetching.

## 4.3 TESTING

**Testing** is an essential phase of the Mentor Hub project to ensure that the system functions correctly, performs optimally, and meets user expectations. Testing is divided into **unit testing**, **integration testing**, and **end-to-end testing**. **Unit testing** focuses on testing individual functions and components to ensure they work as expected in isolation. For the frontend, tools like Jest and React Testing Library are used to test React components and ensure they render correctly, handle state changes properly, and respond to user interactions. For the backend, Mocha and Chai are used for testing API routes and ensuring that they return the correct responses, handle edge cases, and interact correctly with the database. **Integration testing** ensures that different modules and components work together as expected. This includes testing the interaction between the frontend and backend, ensuring data flows correctly through the system and that APIs are returning the expected data. **End-to-end testing** simulates user behavior from start to finish, verifying that the entire application works as a cohesive unit. Tools like Cypress or Selenium are used for this purpose. Additionally, **user acceptance testing (UAT)** is performed to ensure the system meets the end-users' needs and is intuitive to use. Bugs or issues discovered during testing are logged, addressed, and re-tested to maintain high-quality standards.

## 4.4 INSTALLATION

The **Installation** process for the Mentor Hub project involves setting up the necessary software, dependencies, and tools for both the development and production environments. To begin, **Node.js** must be installed, as it serves as the runtime environment for the backend. The installation process begins with cloning the repository from GitHub or downloading the project files. Once the repository is set up, the necessary **dependencies** are installed by running `npm install` in the project directory. This will install the required packages, such as Express.js for the server, Mongoose for MongoDB interaction, and other dependencies like JWT for authentication and Axios for API calls. Additionally, the **MongoDB** database needs to be set up either locally or using a cloud service like MongoDB Atlas. The backend can be configured to connect to the MongoDB database using environment variables stored in a `.env` file for security.

reasons. On the frontend, React.js dependencies are managed via npm install. After installing all dependencies, the application can be run locally by starting the server with npm start. For production, the project can be deployed to cloud platforms like Heroku or AWS. Clear **deployment instructions** are included in the documentation to guide users through setting up the project on various hosting platforms. The installation guide also includes steps for troubleshooting common issues.

## 4.5 DOCUMENTATION

**Documentation** plays a crucial role in making the Mentor Hub project accessible and easy to understand for both developers and users. The project documentation is structured into several sections, ensuring that both technical and non-technical stakeholders can easily navigate and understand the system. The **setup guide** outlines step-by-step instructions for installing and configuring the project in a local or production environment. It includes installation prerequisites such as Node.js, MongoDB, and any required dependencies, followed by clear commands for setting up the project. The **API documentation** provides a comprehensive reference for all the backend API endpoints, including their purpose, required parameters, request methods (GET, POST, PUT, DELETE), and example responses. It also includes details on how to authenticate using JWT tokens and how users can access protected routes. The **frontend documentation** explains the structure of React components and their state management, providing an overview of how the UI is designed and the interaction between components.

## CHAPTER 5

### TESTING AND RESULTS

#### 5.1 TESTING

**Testing** is a vital part of the Mentor Hub project to ensure the system's functionality, performance, and reliability. The testing process involves **unit testing**, **integration testing**, and **end-to-end testing** to guarantee that all parts of the application work as expected. Unit testing focuses on individual components or functions within the system to verify their behavior in isolation. It ensures that the logic of a specific part of the code works correctly before it interacts with other components. Unit tests are written to test functions in isolation, such as checking if a user can successfully authenticate or if a task is correctly assigned within the system. The purpose of integration testing is to ensure that different parts of the system work together as expected, such as the frontend communicating correctly with the backend or the database integration working seamlessly. End-to-end testing simulates real user interactions with the application, ensuring that the entire system, from the frontend interface to the backend database, functions as a unified whole. Testing tools like Jest, Mocha, and Chai are utilized for backend testing, while React Testing Library and Cypress are used for frontend testing. All tests are executed continuously to ensure the system remains stable and free of errors, helping identify and fix issues early.

##### 5.1.1 Unit Testing

**Unit Testing** is crucial for ensuring that individual components and functions in the Mentor Hub project work as expected. In unit testing, each unit (usually a function or component) is tested in isolation to verify its behavior and correctness. The goal is to catch errors early in the development process before components are integrated into the larger system. In the Mentor Hub project, unit tests are written for both the frontend and backend components. On the **backend**, unit tests are focused on API routes, authentication mechanisms, and database operations. For example, functions that handle user registration or task assignment are tested for edge cases like invalid input or missing parameters. Testing libraries like Mocha and Chai are used to test these backend functions, ensuring that they return the correct responses, handle errors appropriately, and interact with the database without issues. On the **frontend**, unit tests focus on React components, ensuring that the components render properly, handle user input as expected, and interact with state

correctly. For instance, testing components like the Project Dashboard or Profile Update form ensures that changes made in the UI are reflected in the application state

### Test Case 1

**Module** : User Authentication  
**Test Type** : Unit Testing  
**Input** : Username and Password  
**Expected Output** : Authenticate user

### Sample Test

**Output** : User successfully authenticated

**Analysis** : The test ensures that the system can authenticate users based on the provided username and password. If authentication fails for valid credentials or successful authentication occurs for invalid credentials, it indicates a problem with the user authentication module.

### Test Case 2

**Module** : Checkout & Product  
**Test Type** : Unit Testing  
**Input** : Product ID, Quantity  
**Expected Output** : Add product to cart

### Sample Test

**Output** : Product successfully added to the cart

**Analysis** : This test checks whether the system can add a product to the user's cart with the specified quantity. If the product addition fails or incorrect products are added, it suggests issues with the cart management functionality.

## 5.3 Integration Testing

Integration testing is a critical phase in the software testing lifecycle where multiple software components are combined and tested as a group, following unit testing, which verifies the functionality of individual modules. The primary objective is to ensure that the components work together seamlessly, particularly in areas such as data sharing, error handling, and concurrent processing. Various levels of integration are assessed, including pairwise integration (testing interactions between two components), group integration (testing three or more components together), and system integration (testing the entire system as a whole). This phase involves progressively complex scenarios, necessitating careful planning regarding the order of tests and the selection of test cases, as the sequence can significantly impact outcomes. Integration testing helps identify issues that may not be apparent during

unit testing, such as incompatibilities between modules, data flow problems, and errors in communication. Thorough documentation and traceability are crucial for effective integration testing, ensuring that all test cases are accounted for and facilitating communication among team members. By addressing interactions between different modules, integration testing plays a vital role in ensuring that the final software product meets user requirements and functions as intended, ultimately contributing to the delivery of a reliable, robust, and high-quality software application while reducing the risk of defects in the production environment.

### Test Case 1

**Module:** User Registration and Login

**Test Type:** Validation of registration and login functionality

**Input:** Name, Email, Password (for registration); Email, Password (for login)

**Output:** Redirect to the dashboard after successful login

**Sample Test Output:** User receives a JWT token and is redirected to the main dashboard page.

**Analysis:** This test ensures that the user registration process works smoothly. The system should properly validate the email format and password strength during registration. Upon login, the credentials are compared against the stored data, and the user is authenticated successfully. If any mismatch occurs (e.g., incorrect password or email), an appropriate error message should be displayed. This test helps ensure that security measures like JWT authentication are working as expected, and users can access their dashboard securely after successful login.

## Test Case 2

**Module:** Project Creation

**Test Type:** Validation of project creation functionality

**Input:** Title, Description, User ID (for creating a project)

**Output:** Successful project creation and display on project list

**Sample Test Output:** The new project is visible in the project list, and its details (Title, Description) are correct.

**Analysis:** This test checks if a user can create a new project. The system should validate that the title and description fields are filled out correctly, and the project should be added to the database upon submission. The response should include the project details, and a GET request should retrieve the newly created project from the project list. If any required fields are missing or invalid, the system should display an error message. This test ensures the integrity of the project creation process and proper interaction with the database.

## Test Case 3

**Module:** Task Assignment

**Test Type:** Validation of task assignment functionality

**Input:** Task Title, Assigned User ID, Due Date (for task creation)

**Output:** Successful task creation and assignment

**Sample Test Output:** The new task appears in the project's task list, assigned to the correct user.

**Analysis:** This test ensures that tasks can be properly assigned to users within a project. The system should validate that all fields (task title, assigned user, and due date) are filled in and correctly formatted. Once the task is created, it should be stored in the database and associated with the correct project and user. The task should also appear in the task list for the relevant project. If any data is missing or invalid (e.g., a user ID that doesn't exist), the system should reject the task creation and return an error message.

## Test Case 4

**Module:** Real-Time Discussion

**Test Type:** Validation of real-time messaging functionality

**Input:** Message input by a user in the discussion section

**Output:** Real-time message delivery to all users connected to the discussion

**Sample Test Output:** The message is received instantly by all users in the discussion without refreshing the page.

**Analysis:** This test validates the real-time communication feature using WebSockets or Socket.io. When a user sends a message in the discussion section, it should be broadcast to all users connected to the same project in real-time, without requiring a page reload. The test ensures that the communication channel is functioning correctly and that all users can see the latest messages immediately. This test also checks for message consistency and the system's ability to handle multiple users concurrently. If the message is not displayed in real-time, it may indicate a problem with the WebSocket connection or the backend logic.

## CHAPTER 6

### CONCLUSION AND FUTURE ENHANCEMENT

#### 6.1 CONCLUSION

The Mentor Hub project successfully streamlines project management, team collaboration, and task tracking in a cohesive platform. With features like secure user authentication, real-time discussion, and project tracking, it offers a comprehensive solution for effective team coordination. Rigorous testing ensures reliability, with the platform performing as expected across different modules. The system's design focuses on usability and functionality, making it a useful tool for academic and professional environments. As a collaborative tool, Mentor Hub has the potential to significantly improve communication and workflow, making project management more efficient and organized.

#### 6.2 FUTURE ENHANCEMENT

- **Augmented Reality (AR) Support:** Incorporate AR technology to provide interactive project visualizations, allowing users to engage with projects and tasks in a more immersive way.
- **Voice Command Integration:** Enable voice-controlled navigation and task management for hands-free use, enhancing accessibility and user experience.
- **Blockchain for Task Tracking:** Use blockchain to securely track task assignments and project milestones, ensuring transparency and verifiability of progress.
- **Collaborative Code Editor:** Implement a collaborative code editor within the platform, allowing users to work on code in real-time with version control.
- **AI-driven Mentor Matching:** Develop an AI algorithm to match users with mentors based on skills, project experience, and learning goals, improving mentor-mentee connections.



## APPENDICES

### A. SAMPLE CODING

#### AUTHENTICATION SCREEN

```
import React, { useState, useEffect } from 'react';

import { Link, useHistory } from 'react-router-dom';

import { PropTypes } from 'prop-types';

import { connect } from 'react-redux';

import { useForm } from "../../hooks/form-hook";

import { useHttpClient } from "../../hooks/http-hook";

import { login, register, loadUser } from "../../actions/auth-action";

import { initSwitchLayout } from "../initSwitchLayout";

import Input from "../../components/shared/FormElements/Input";

import {
  VALIDATOR_EMAIL,
  VALIDATOR_MINLENGTH,
  VALIDATOR_REQUIRE
} from "../../utils/validators";

import '../../assets/stylesheets/authScreen.css';

import Swal from "sweetalert2";

// import M from "materialize-css";

const AuthScreen = ({ login, register, isAuthenticated, loadUser, user, token }) => {
  const [ isLoginMode, setIsLoginMode ] = useState(true);
```

```

const [ isMobile, setIsMobile ] = useState(false);

const { sendRequest } = useHttpClient();

const history = useHistory();

useEffect(() => {

  if(user) {

    history.push('/dashboard');

  }

  if (/Mobi/.test(navigator.userAgent)) setIsMobile(true);

  // eslint-disable-next-line

}, [user])

const [ formState, inputHandler, setFormData ] = useForm(

  {

    email: {

      value: "",

      isValid: false,

    },

    password: {

      value: "",

      isValid: false

    }

  },

  false

```

```
);
```

```
useEffect(() => {
  initSwitchLayout();
}, [])
```

```
const switchModeHandler = async () => {
  if(!isLoginMode) {
    //Switching from sign up to sign in mode
    await setFormData(
      {
        ...formState.inputs,
        name: undefined,
        confirmPassword: undefined,
        username: undefined
      },
      formState.inputs.email.isValid && formState.inputs.password.isValid
    );
  } else {
    //Switching from sign in to sing up mode
    await setFormData(
      {
        ...formState.inputs,
        name: {
```

```

        value: "",
        isValid: false
    },
    username: {
        value: "",
        isValid: false,
    },
    confirmPassword: {
        value: "",
        isValid: false
    }
},
false
);
}

setIsLoginMode(prevMode => !prevMode);
};

const authSubmitHandler = async event => {
    event.preventDefault();

    if(isLoginMode) {
        try {
            await login(formState.inputs.email.value, formState.inputs.password.value,
sendRequest);

            await loadUser(sendRequest);

```

```

    } catch (error) {

        console.error(error);

    }

} else {

    try {

        if(formState.inputs.password.value !== formState.inputs.confirmPassword.value) {

            //TODO: HAVE TO FIX

            // M.toast({html: 'Confirm password and Confirm new password have to be same',
classes: 'red'}));

            Swal.fire({

                title: 'Error!',

                text: 'Confirm password and Confirm new password have to be same',

                icon: 'error',

            });

        } else {

            try {

                await register(formState.inputs.name.value, formState.inputs.username.value,

                    formState.inputs.email.value, formState.inputs.password.value, sendRequest);

                await loadUser(sendRequest);

            } catch (error) {

                console.error(error);

            }

        }

    } catch (err) {

        console.error(err);

```

```
    }  
  }  
};  
  
const emailInput = <Input  
  element="input"  
  elementTitle="email"  
  type="email"  
  placeholder="Email"  
  validators={[VALIDATOR_EMAIL()]}  
  errorText="Please enter a valid email address."  
  onInput={inputHandler}  
>;
```

```
const passwordInput = <Input  
  element="input"  
  placeholder="Password"  
  elementTitle="password"  
  type="password"  
  validators={[VALIDATOR_MINLENGTH(6)]}  
  errorText="Please enter at least 6 character."  
  onInput={inputHandler}  
>;
```

```

return (
  <div className="main signin__signup">
    <div className="row">
      <br/><br/>
      {!isMobile && (
        <div className="col s12 m12 l12">
          <div className="container" id="container">
            {!isLoginMode && (
              <div className="form-container sign-up-container">
                <form onSubmit={authSubmitHandler}>
                  <h4>Create Account</h4>
                  <Input
                    element="input"
                    elementTitle="name"
                    type="text"
                    placeholder="Name"
                    validators={[VALIDATOR_REQUIRE()]}
                    errorText="Please enter your full name."
                    onInput={inputHandler}
                  />

```

## DASHBOARD SCREEN

```

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

```

```

import { prepareTodoAndBugForPreview } from "../actions/project-action";

import { getAllProjects } from "../actions/projects-action";

import { loadUser } from "../actions/auth-action";

import { useHttpClient } from "../hooks/http-hook";

import ChartItem from "../components/ChartItem";

const DashboardScreen = ({ projects, auth, getAllProjects, prepareTodoAndBugForPreview })
=> {

  const { sendRequest } = useHttpClient();

  const { user,

    chartData, //for showing chart of finished todos and fixed bugs of a member

    todoBugCountSummary, //Count of how many todos are completed or incomplete and bug
fixed or not fixed yet.

    activitySummary, // All completed or incomplete todos and fixed or not fixed bugs of a
member

  } = auth;

  const summaryCardDetails = [

    { title: 'Remaining Todo', count: todoBugCountSummary?.todoNotDone },

    { title: 'Remaining Bug', count: todoBugCountSummary?.notFixedBug },

    { title: 'Finished Todo', count: todoBugCountSummary?.todoDone },

    { title: 'Fixed Bug', count: todoBugCountSummary?.fixedBug },

  ];

```



```

useEffect(() => {

  if(user) getAllProjects(sendRequest);

  // eslint-disable-next-line

}, [user]);

useEffect(() => {

  if(projects.length > 0 && user) prepareTodoAndBugForPreview(user.username, projects);

  // eslint-disable-next-line

}, [projects, user]);

return (

  <section className="w-full bg-default text-white-light min-h-screen flex flex-col lg:gap-
28 md:gap-14 gap-10 lg:p-8 md:p-6 p-4">

    <div className="mx-auto max-w-screen-xl">

      <dl className="grid lg:grid-cols-4 md:grid-cols-4 grid-cols-2 gap-4">

        {summaryCardDetails?.map((item, index) => (

          <div key={index} className="flex flex-col gap-2 rounded-lg bg-[#1f2937]
lg:px-4 md:px-3 px-2 lg:py-8 md:py-6 py-4 text-center">

            <dt className="order-last lg:text-lg md:text-lg text-md font-medium text-
white-light">

              {item.title}

            </dt>

            <dd className="lg:text-4xl md:text-3xl text-3xl font-extrabold text-orange-
500">

              {item.count}

```

```

        </dd>

      </div>

    )})

  </dl>

</div>

```

## MEMBER SCREEN

```

import React, { useEffect } from 'react';

import { connect } from 'react-redux';

import { getAllUser } from "../actions/auth-action";

import { useHttpClient } from "../hooks/http-hook";

import Member from "../components/Member";

const MembersScreen = ({ auth: { users }, getAllUser }) => {

  const { sendRequest } = useHttpClient();

  useEffect(() => {

    getAllUser(sendRequest);

  }, []);

  return (

    <div className="grid lg:grid-cols-3 md:grid-cols-2 grid-cols-1 lg:gap-8 md:gap-6 gap-4">

      {users && (

        users?.map(user => (

          <Member key={user?.username} user={user}/>

```

```

        ))
    })
</div>

);

};

const mapStateToProps = state => ({
    auth: state.auth
})

export default connect(mapStateToProps, { getAllUser })(MembersScreen);

```

## PROJECT SCREEN

```

import React, { useEffect } from 'react';

import { connect } from "react-redux";

import { useParams } from 'react-router-dom';

import ProjectSummaryRow from "../components/project/projectSummary/ProjectSummary";

import Overview from "../components/project/overview/Overview";

import Discussion from "../components/project/discussion/Discussions";

import Activities from "../components/project/activities/Activities";

import EditProjectDetails from "../components/project/edit-project-details/EditProjectDetails";

import Todos from "../components/project/todos/Todos";

import Bugs from "../components/project/bugs/Bugs";

import { useHttpClient } from "../hooks/http-hook";

import {

```

```

    getProjectById, getNotAssignedMember, prepareWorkDonePreview,
    getIsMemberAndCreatorOfProject

  } from "../actions/project-action";

const ProjectScreen = ({ project, getProjectById, selectedItem, getNotAssignedMember,
  prepareWorkDonePreview,

    getIsMemberAndCreatorOfProject

  }) => {

  const { sendRequest } = useHttpClient();

  const projectId = useParams().projectId;

  useEffect(() => {

    getProjectById(projectId, sendRequest);

    // eslint-disable-next-line

  }, []);

  useEffect(() => {

    if(!project) {

      getIsMemberAndCreatorOfProject(projectId, sendRequest);

      getNotAssignedMember(projectId, sendRequest);

      prepareWorkDonePreview(projectId, sendRequest);

    }

    // eslint-disable-next-line

  }, [project])

  return (

```

```

<div className="flex flex-col lg:gap-8 md:gap-6 gap-4">

  <◇>

    <ProjectSummaryRow projectId={projectId} selectedItem={selectedItem} />

    {selectedItem === 'overview' && <Overview />}

    {selectedItem === 'activities' && <Activities />}

    {selectedItem === 'discussion' && <Discussion />}

    {selectedItem === 'todolist' && <Todos />}

    {selectedItem === 'bugs' && <Bugs />}

    {selectedItem === 'edit-project' && <EditProjectDetails />}

  </◇>

</div>

);

};

const mapStateToProps = state => ({

  auth: state.auth,

  project: state.project.project

});

export default connect(mapStateToProps,

  { getProjectById, getNotAssignedMember, prepareWorkDonePreview,

  getIsMemberAndCreatorOfProject})

(ProjectScreen);

```

## B.SCREENSHOTS

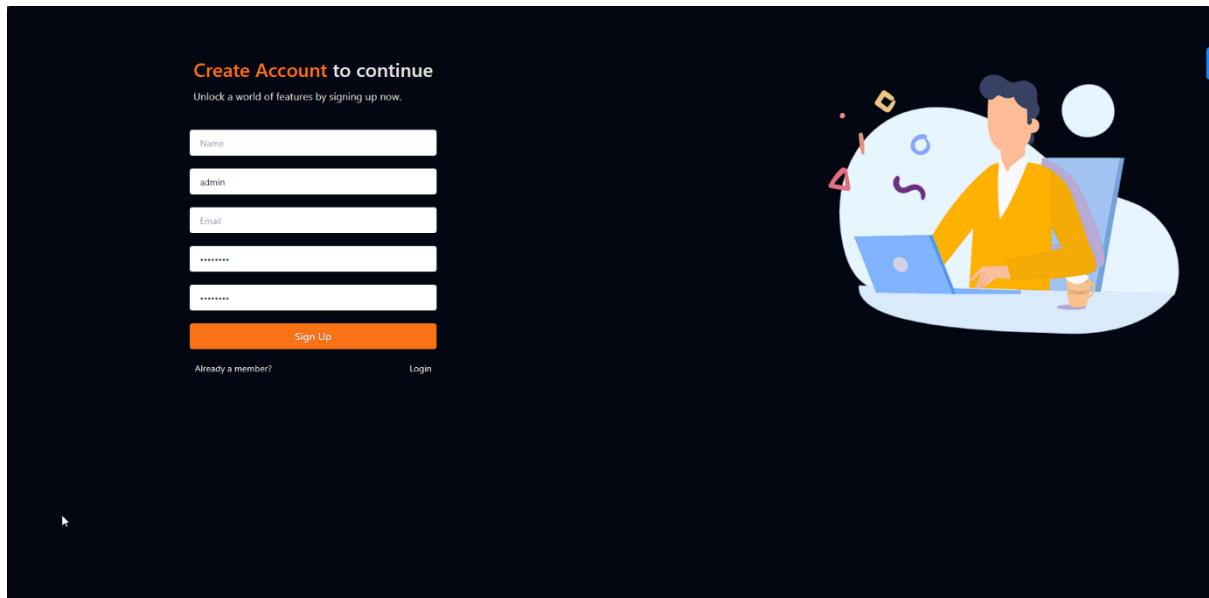


Figure B.1 User Sign Up Page (Member)

This figure depicts a user sign up form. It includes input fields for user details such as first name, last name, email, password, mobile number. There is a "Sign Up" button and an option to navigate to the login page for existing users.

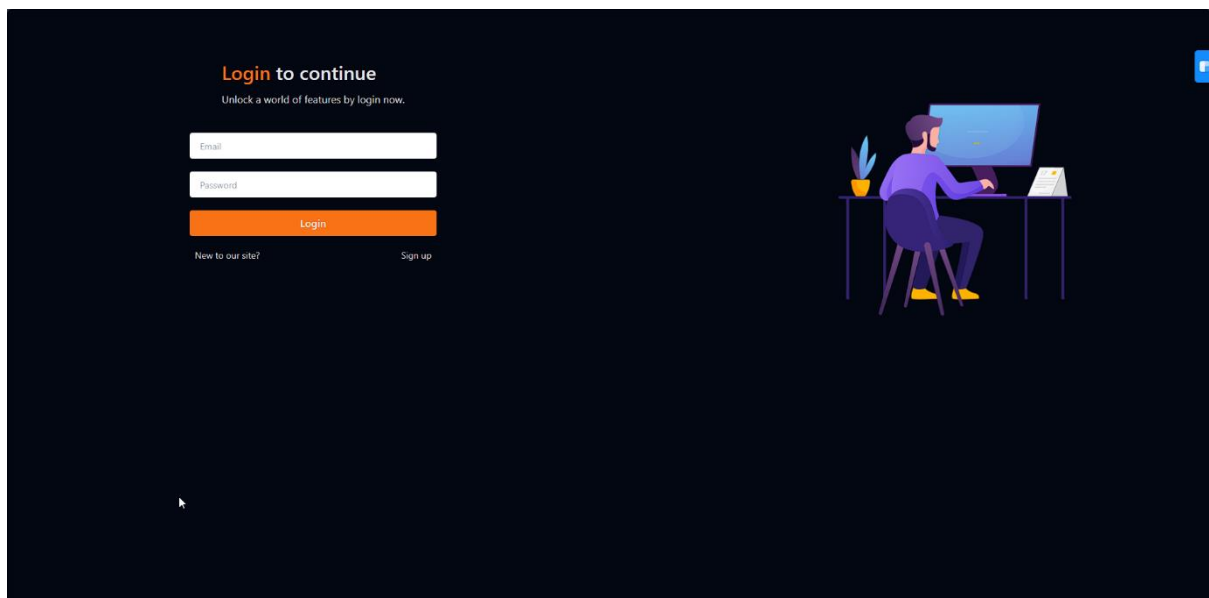


Figure B.2 User Login Page (Member)

User Login Page This figure showcases a login form with fields for "Username or Email" and "Password." It includes a "Forgot password?" link for recovery and a prominent "Sign In" button. Users without an account are directed to a "Register" link for account creation.

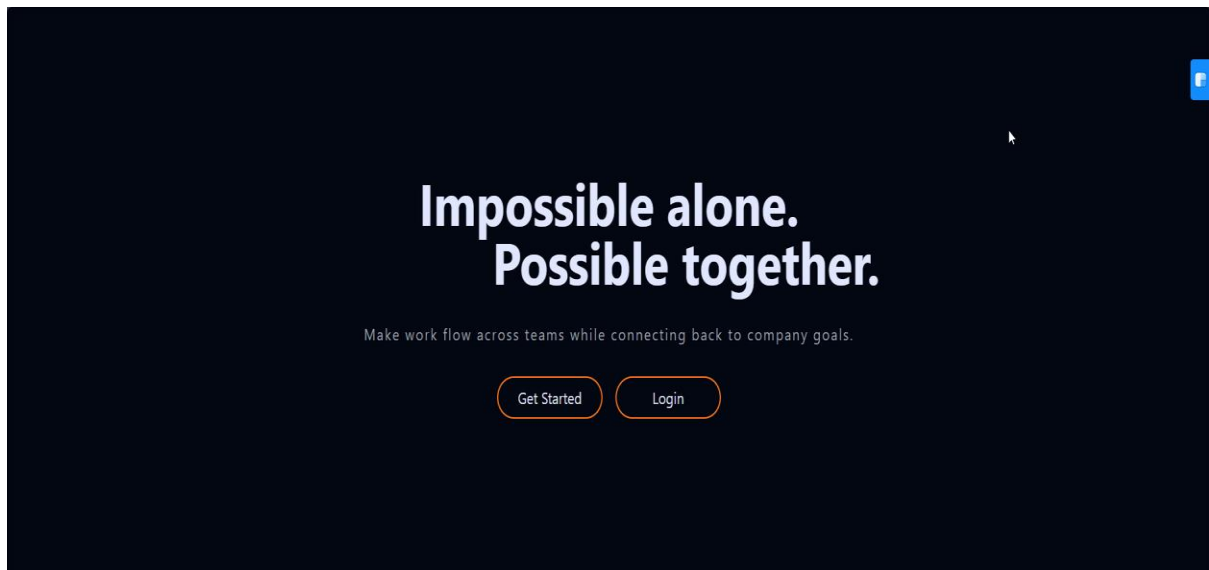


Figure B.3 Home Page

Home page of the agro center ,consists of navigation menu where the Get Started and Login is shown at center of the page.

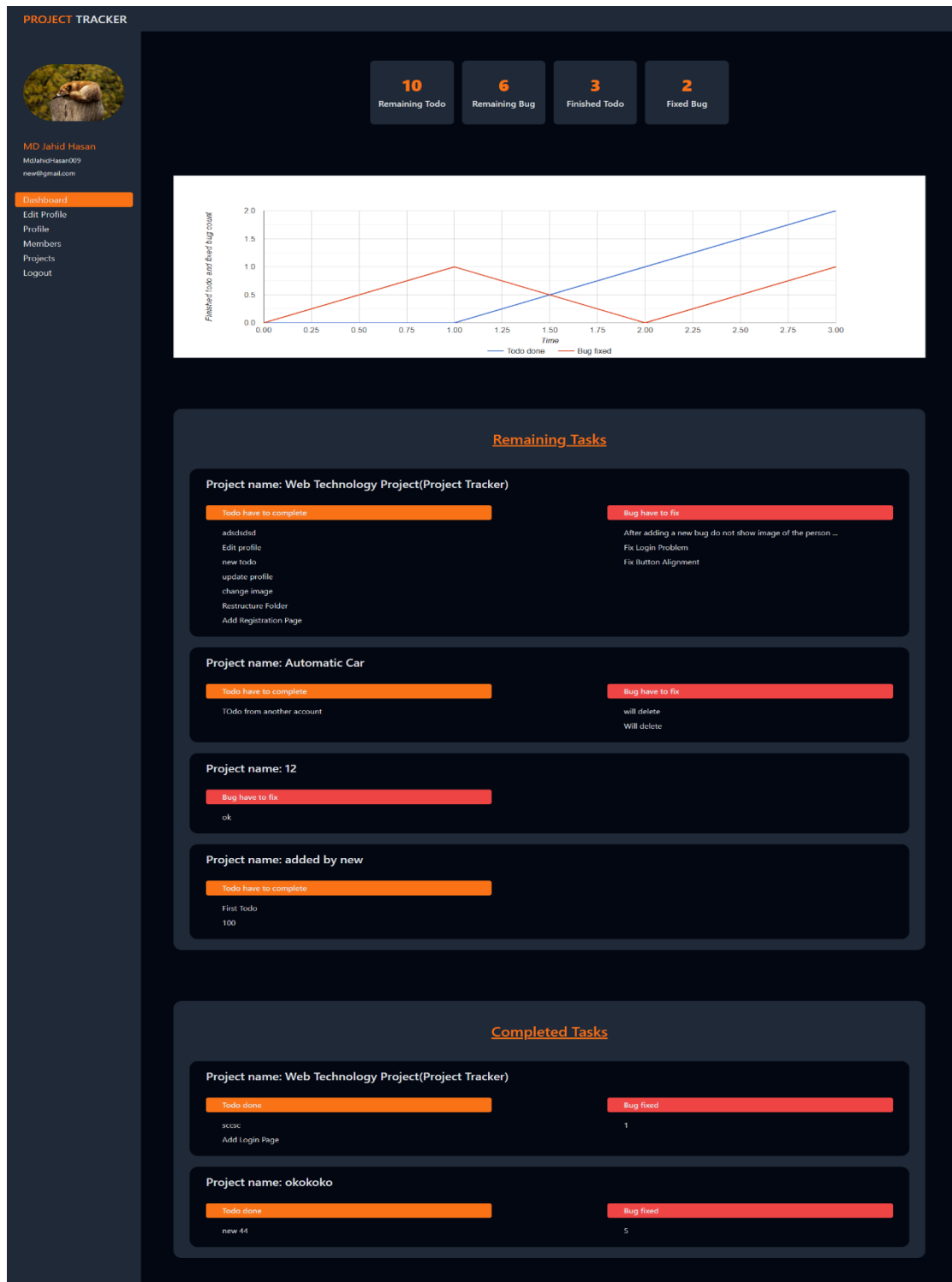


Figure B.4 Dashboard Page (Project Task)

Dashboard page, contains Remaining Task , Completed Task, are shown at bottom of the page.



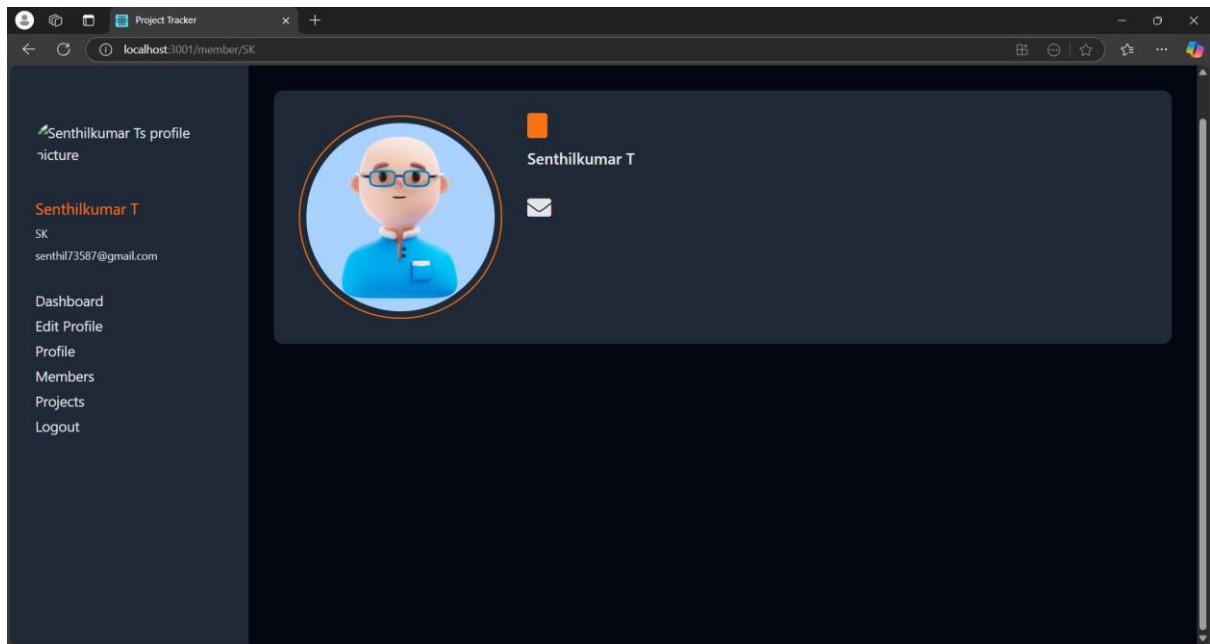


Figure B.5 User Profile Page (Member)

Profile Shows the User name and profile photo and E-mail Address.

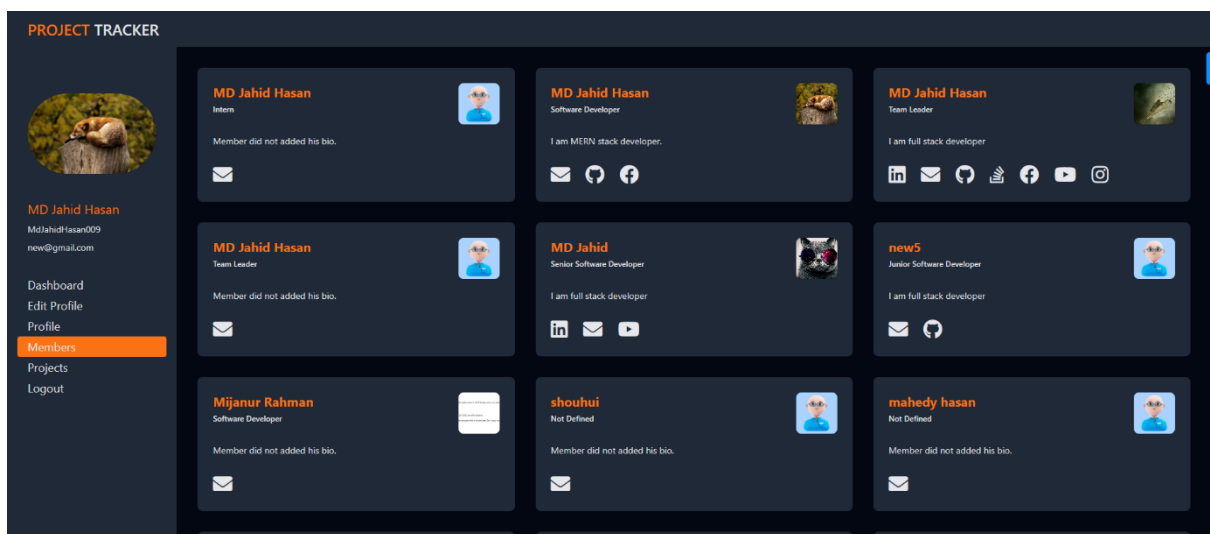


Figure B.6 Project Team Members Page

Member Pages shows how many members are work together with our projects.

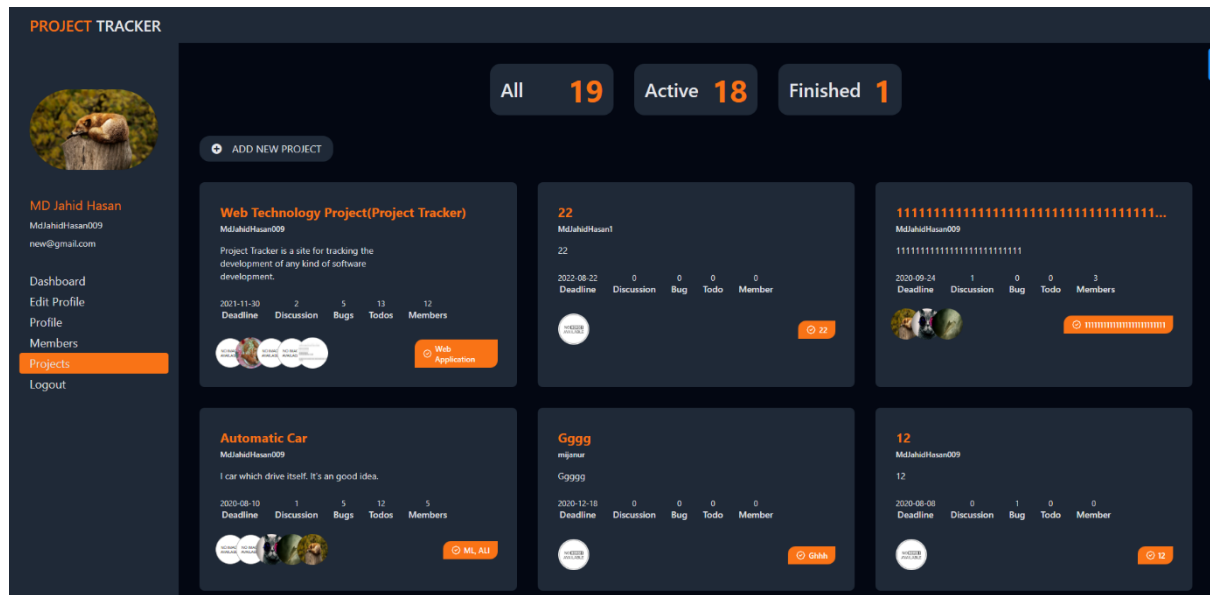


Figure B.7 Overall Project Main Page

Projects page we can able to access entire total number of projects , active number of projects, finished projects and we can add New Project

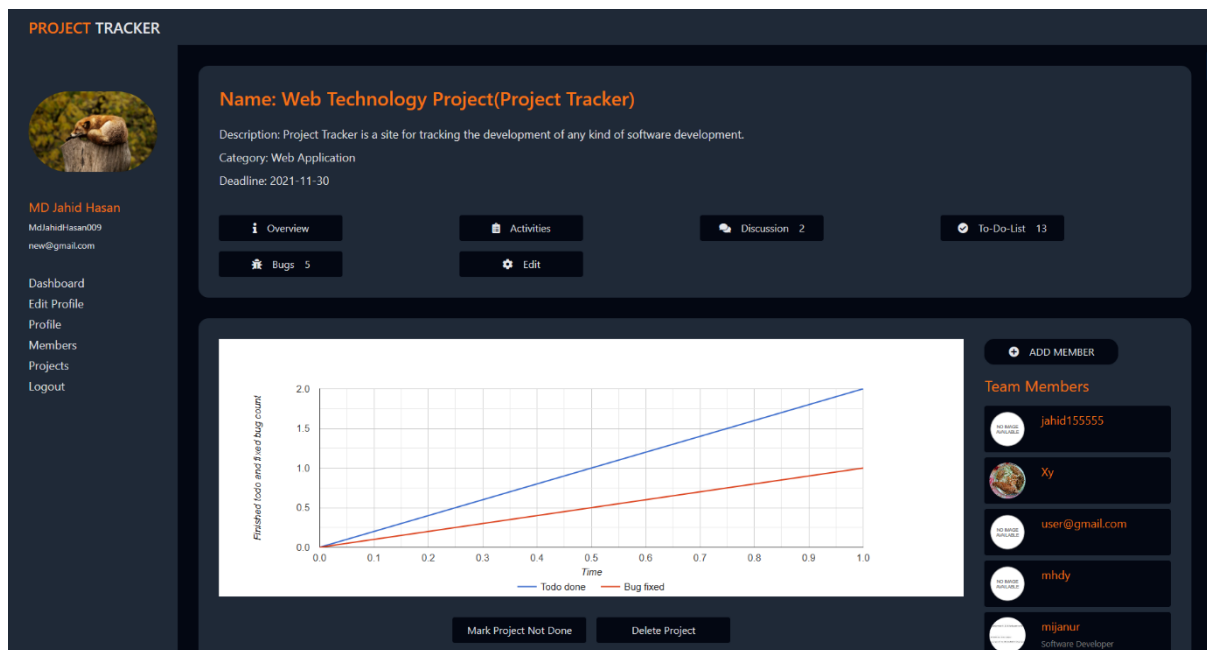


Figure B.8 Overview of Project Page

In this page shows overall projects overview like Project activities, discussion, to do list, bugs, and Graph representation.

## REFERENCES

### BOOK REFERENCE

1. Alex Banks & Eve Porcello , "Learning React", 2nd Edition, 2020
2. David Herron , "Node.js Web Development", 5th Edition, 2020
3. Jon Duckett, "HTML and CSS: Design and Build Websites", 1st Edition, 2011
4. Mark Otto and Jacob Thornton , "Bootstrap in Practice", Manning Publications, 2016
5. M. Murphy, "Firebase: Up and Running", 2nd Edition, 2019

### WEBSITE REFERENCE

1. [www.geeksforgeeks.org](http://www.geeksforgeeks.org), "Learn about Node.js and React.js".
2. [www.getbootstrap.com](http://www.getbootstrap.com), "Learn about Bootstrap".
3. [www.javatpoint.com](http://www.javatpoint.com), "Learn about JavaScript".
4. [www.tutorialspoint.com](http://www.tutorialspoint.com), "Learn about Firebase and Realtime Database".
5. [www.w3schools.com](http://www.w3schools.com), "Learn about HTML, CSS and React.js".