

TASK FUSION MANAGEMENT SYSTEM

*Dissertation submitted
in partial fulfillment of the requirements for the award of the degree of*

Master of Computer Applications (M.C.A)

Submitted By

BONDI PRIYANKA

(Regd.no.322203320004)

Under the esteemed guidance of

Dr N. Jaya Lakshmi

Associate Professor

Department of Computer Applications



Department of Computer Applications

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(AUTONOMOUS)**

(Affiliated to Andhra University, A.P)

VISAKHAPATNAM-530048

2022-2024

TASK FUSION MANAGEMENT SYSTEM

*Dissertation submitted
in partial fulfillment of the requirements for the award of the degree of*

Master of Computer Applications (M.C.A)

Submitted By

BONDI PRIYANKA

(Regd.no.322203320004)

Under the esteemed guidance of

Dr N. Jaya Lakshmi

Associate Professor

Department of Computer Applications



Department of Computer Applications

**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(AUTONOMOUS)**

(Affiliated to Andhra University, A.P)

VISAKHAPATNAM-530048

2022-2024

CERTIFICATE



**GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING
(AUTONOMOUS)**

This is to certify that, the dissertation titled “**TASK FUSION MANAGEMENT SYSTEM**” is submitted by **Ms. BONDI PRIYANKA** with **Regd. No 322203320004** in partial fulfillment of the requirement for the degree of Master of Computer Applications in **GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING (Autonomous)**, during the academic year 2022 – 2024 affiliated to Andhra University, Visakhapatnam is a bonafide record of project carried out by her under my guidance and supervision.

The contents of the project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree

Signature of Internal Guide

Signature of Head of the Department

Signature of External Examiner

CERTIFICATE OF PLAGIARISM CHECK



This is to certify the Master of Computer Applications (MCA) dissertation submitted by **BONDI PRIYANKA** (Regd. No 322203320004) of Department of Computer Applications under the supervision of **Dr. N. Jaya Lakshmi, Associate Professor**, has undergone plagiarism check and found to have similarity index less than 40%. The details of the plagiarism check are as under:

File Name : **TFMS_MAIN_DOCUMENTATION_copy.pdf**
(2.81M)

Dissertation Title : **TASK FUSUION MANAGEMENT SYSTEM**

Date and Time of Submission: **03-May-2024 11:57AM(UTC+0530)**

Submission ID : **2369631587**

Similarity Index : **6%**

Dean Academics Programs (PG)

DECLARATION

I hereby declare that dissertation titled “**TASK FUSION MANAGEMENT SYSTEM**” is submitted to the Department of Computer Applications, **Gayatri Vidya Parishad College of Engineering (Autonomous)** affiliated to Andhra University, Visakhapatnam in partial fulfillment of the requirements for the award of the Degree of **Master of Computer Applications (M.C.A)**. This work is done by me and authentic to the best of my knowledge under the direction and valuable guidance of **Dr. N. Jaya Lakshmi**, Associate Professor of the Department of Computer Applications.

BONDI PRIYANKA

(Regd. No. 322203320004)

ACKNOWLEDGEMENT

I take the opportunity to thank everyone who has contributed to making the project possible. I am thankful to **Gayatri Vidya Parishad College of Engineering (Autonomous)** for giving opportunity me to work on a project as part of the curriculum.

I offer my profuse thanks to **Dr. A. BalaKoteswara Rao**, the Principal, **Gayatri Vidya Parishad College of Engineering (Autonomous)**, for providing the best faculty and lab facilities throughout the completion of Master of Computer Applications course.

I offer my copious thanks to **Prof. Dr. K. Narasimha Rao Professor & Dean, Academics (PG)**, for his valuable suggestions and constant motivation that greatly helped me to complete project successfully.

I offer my copious thanks to **Dr. Y. Anuradha**, Associate Professor & Associate Head of B.Tech CSE with Data Science, for her valuable suggestions and constant motivation that greatly helped me to complete the project successfully.

My sincere thanks to **Dr. N. Jaya Lakshmi**, Associate Professor, Department of Computer Applications, my project guide, for her constant support, encouragement, and guidance. I am very much grateful for her valuable suggestions.

BONDI PRIYANKA

(Regd. No. 322203320004)

ABSTRACT

ABSTRACT

The Task Fusion Management System (TFMS) represents a comprehensive solution designed to enhance organizational efficiency through the seamless integration and management of diverse tasks. TFMS is conceived as a centralized platform that converges various task management functionalities, combining elements of project coordination, assignment, and monitoring. The system incorporates a user-friendly interface that enables the creation, coordination and tracking of tasks across teams and departments. TFMS leverages advanced features of project management tools to facilitate a holistic approach to task handling. The integration of features such as task prioritization, dependencies, ownership, access management and real-time collaboration, ensure that teams can work cohesively and efficiently. The system includes messaging services to team discussions, groups, user to user video conferencing. Additionally, TFMS provides insightful reporting and analytics, empowering decision-makers with data-driven insights into task performance and informs about task updates and deadlines. Built on a foundation of scalability and reliability, TFMS integrates seamlessly with existing features of project management tools and ensures a smooth transition for organizations. This abstract encapsulates TFMS as an innovative solution, leveraging project management tools to unify and optimize task management processes, thereby contributing to heightened productivity and collaboration within the organization.

INDEX

Certificate	i
Certificate for plagiarism check	ii
Declaration	iii
Acknowledgment	iv
Abstract	v
List of Figures	x
List of Tables	xiii

CONTENTS

S.NO	CONTENTS	PAGE NO
Chapter 1	INTRODUCTION	1-5
Chapter 2	LITERATURE SURVEY	6-7
Chapter 3	SYSTEM ANALYSIS	8
3.1	Existing system	9
3.2	Proposed system	9
Chapter 4	SYSTEM REQUIREMENTS AND SPECIFICATIONS	10
4.1	Functional Requirements	11
4.2	Non Functional Requirements	12
4.3	Hardware Requirements	13
4.4	Software Requirements	13
4.5	Libraries used	14
4.5.1	Node mailer	14
4.5.2	Material UI	14
4.5.3	JWT Token	14
Chapter 5	SYSTEM DESIGN	15
5.1	Architecture	16-18
5.2	Flow chart	19
5.3	UML Diagrams	20-21
Chapter 6	IMPLEMENTATION OF SYSTEM	22
6.1	Database schema	23-25
6.2	Signaling server	26
6.3	Mail queuing system	27

6.4	Web socket server	28-29
6.5	Flask server	30-31
6.6	React server	32-33
6.7	Sample code	34-44
Chapter 7	OUTPUT SCREENS	45-59
Chapter 8	SYSTEM TESTING	60
8.1	Functional Testing	61
8.2	Test Cases	62-64
8.3	Test Case Status	65-66
Chapter 9	CONCLUSION	67
9.1	Conclusion	68
9.2	Future Scope	69
9.3	References	70

LIST OF FIGURES

FIGURE NO	NAME OF THE FIGURE	PAGE NO
1.1	DASHBOARD	4
1.2	DASHBOARD ANALYTICS	4
1.3	BENEFITS OF TFMS	5
5.1	SYSTEM ARCHITECTURE	14
5.2	FLOW CHART	17
5.3.1	ADMIN USE CASE DIAGRAM	19
5.3.2	USER USE CASE DIAGRAM	19
6.1.1	DATABASE SCHEMA	23
6.2.1	SIGNALING SERVER	25
6.3.1	MAIL QUEUING SERVER	26
6.4.1	WEB SOCKET SERVER	27
6.5.1	FLASK APPLICATION	29
6.6.1	REACT SERVER	31
7.1	LOGIN SCREENS	46
7.1.1	SIGN IN PAGE	46
7.1.2	CREATE ORGANIZATION PAGE	46
7.2	DASHBOARD	47
7.2.1	HOME PAGE	47
7.2.2	DASHBOARD PAGE	47
7.2.3	DASHBOARD USER LIST	48
7.2.4	DASHBOARD TASK LIST	48

7.3	USER PAGE	49
7.3.1	USER LIST	49
7.3.2	USER CREATE PAGE	49
7.3.3	USER VERIFY PAGE	50
7.3.4	USER VERIFICATION MAIL	50
7.4	DEPARTMENT PAGE	51
7.4.1	DEPARTMENT LIST	51
7.4.2	CREATE DEPARTMENT	51
7.4.3	DEPARTMENT ENROLLED MAIL	52
7.4.4	DEPARTMENT DETAILS	52
7.5	PROJECT PAGE	53
7.5.1	PROJECT LIST	53
7.5.2	DETAILS OF THE PROJECT	53
7.5.3	CREATE NEW PROJECT	54
7.5.4	ENROLLED MAIL FOR PROJECT	54
7.6	TEAMS PAGE	55
7.6.1	TEAMS LIST	55
7.6.2	TEAM DETAILS	55
7.7	TASK PAGE	56
7.7.1	TASKS	56
7.7.2	TASK ASSIGNING	56
7.7.3	ASSIGNED NEW TASK MAIL	57
7.8	CHAT PAGE	57
7.9	VIDEO CONFERENCES PAGE	58

7.9.1	AUDIO CALLS	58
7.9.2	VIDEO CALLS	58
7.9.3	VIDEO CALL SCREEN SHARING	59

LIST OF TABLES

Table No.	NAME OF THE TABLE	Page No.
8.3	TEST CASE STATUS	67-68

CHAPTER 1

INTRODUCTION

1. INTRODUCTION

Task fusion management systems help organize and manage tasks across teams and projects. Integrating features of project management tools can enhance the system's capabilities.

The core components of the task fusion system would include:

- A centralized task repository to store all tasks across projects
- Dashboards and views to filter and visualize tasks
- Task automation to assign, route, and update tasks automatically
- Notifications when tasks are created, completed etc.
- Real-time collaborative (RTC) platforms that allow multiple users to engage in communication, sharing and accessibility.

Product Goals

Organize workflows of teams and individual users. The Product is mainly focuses on, Organization, Communication, Visualization and Support.

Product Features Overview:

Communication: Communication features in a project management application typically include tools for team members to collaborate on tasks, share information, and stay up-to-date on project progress. Some examples of communication features that may be included in a project management application are:

- **Comments:** Users can add comments to tasks, assignees, or the project itself. This allows for discussions and updates to be easily shared among team members.
- **Notifications:** Users can receive notifications when tasks are assigned to them, or when project milestones are reached and regarding discussions.
- **Messaging:** Users can send direct messages to each other for one-on-one conversations, or create group messages for team discussions.

- **Video conferencing:** User can access applications built-in video conferencing capabilities, allowing team members to hold virtual meetings and discuss project details in real-time.
- **User Availability:** User can check the status of other users whether they available on particular time.

Organizing: Organizing features in a project management application typically include tools that help users manage and prioritize their tasks and projects. Some examples of organizing features that may be included in a project management application are:

- **Task management:** Users can create and assign tasks to themselves or other team members. Tasks can be organized by priority, deadline, status, or other criteria.
- **Task tracking:** Users can track the progress of their tasks and projects, including time spent, status updates, and completion percentages.
- **Project planning:** Users can create project timelines and set milestones to ensure that projects are completed on time.
- **Visualization:** Visualizing features in a project management application typically include tools that help users to view and understand their tasks and projects in different formats, such as lists, boards, timelines, and charts. Some examples of visualizing features that may be included in a project management application are:
 - **Board view:** Users can organize their tasks into columns on a virtual board, often referred to as a Kanban board, for a visual representation of the progress of each task and teams.
 - **List view:** Users can view their tasks in a simple list format, sorted by criteria such as due date, priority, or status.
 - **Timeline view:** Users can create a timeline of their project, showing important dates and milestones, and allowing them to visualize the entire project from start to finish.
 - **Dashboard:** Users can view a summary of their tasks, projects, and progress on a single screen, often with customizable widgets to show the most relevant information.

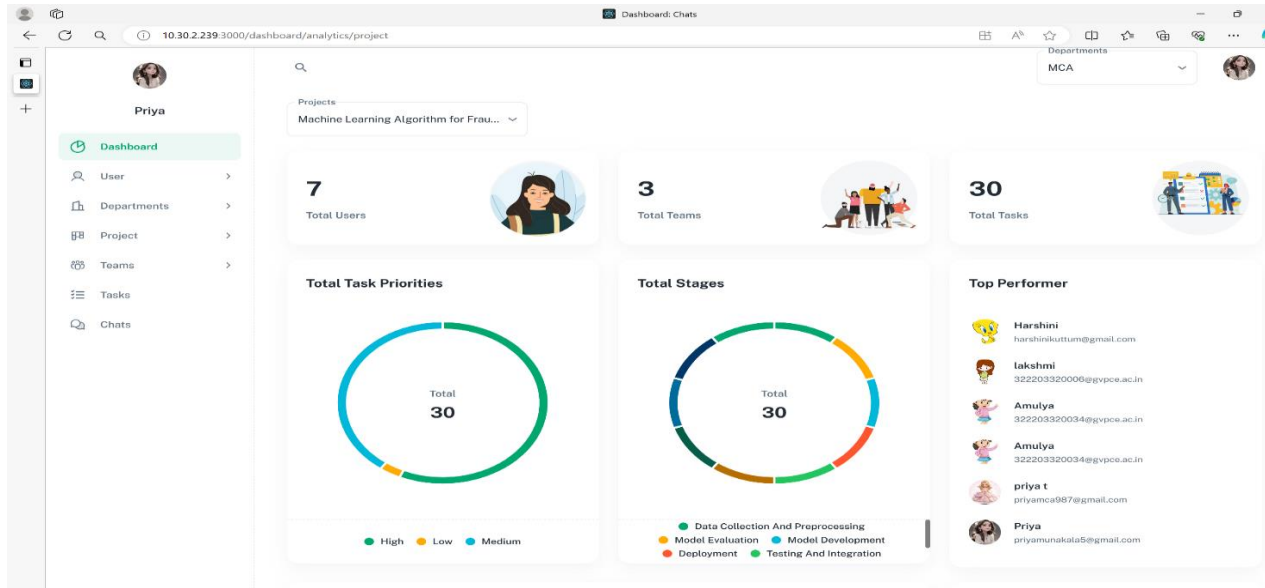


Fig : 1.1 Dashboard

Task Name	Priority	Stage	Reporter	Users	Teams	Start Date	End Date	Total Days	Task overun	Progress
Compatibility Verification	High	Testing and Integration	Priya	2	1	22-03-2024	31-03-2024	9	8	60%
Deployment Script Preparation	Medium	Deployment	Priya	2	1	26-03-2024	31-03-2024	5	8	70%
Testing Environment Deployment	High	Deployment	priya t	2	1	26-03-2024	30-03-2024	4	7	70%
Final Checks Before Deployment	Medium	Deployment	priya t	2	1	25-03-2024	31-03-2024	6	8	70%
Real-time Performance Monitoring	Medium	Monitoring and Maintenance	Priya	2	0	28-03-2024	02-04-2024	5	10	80%
False Positive/Negative Handling	Medium	Monitoring and Maintenance	priya t	2	0	27-03-2024	31-03-2024	4	8	80%
Regular Model Updates	High	Monitoring and Maintenance	priya t	2	0	29-03-2024	03-04-2024	5	11	80%
Report Writing	High	Done	Priya	2	1	28-03-2024	06-04-2024	9	14	100%

Fig : 1.2 Dash board analytics

Benefits of Task Fusion Management System:



Fig :1.3 Benefits of TFMS

CHAPTER 2

LITERATURE SURVEY

2. LITERATURE SURVEY

Md. Habibur Rahaman[1] presented the peer to peer communication using webrtc. The study comprises the understanding of Webrtc protocol, difference between networks, functionality of Webrtc, native web Api's, and signaling server. It shows the implementation of webrtc architecture and mechanism. It also provides how system can be vulnerable with different type of security attacks.

Zinah Tareq Nayyef , Sarah Faris Amer and Zena Hussain [2] proposed a system to implement real time peer to peer communication. The system is designed to implement audio calls and chat application using Webrtc for peer to peer connection and web sockets for implementing signaling server to establish connection between users. They demonstrate the system design is well suited for handling continuous stream of data and native integration with java script to run on all browsers.

Ms. Archana Nikose, Sakshi Dosani, Shreya Pardhi, Deep Nikode, Anurag Jais [3] proposed a system to implement real time communication between users using web sockets. The system is designed to handle bidirectional communication between server and clients which has a chat application as use case. Implemented in React, Node js, Mongo Db to handle user interface, server and database operations. They have concluded that the web sockets are performed better than traditional network protocols for real time applications.

Victor Velepucha and Pamela flores [4] proposed a system design to implement microservices architecture. The study comprises the understanding of microservices, patterns, practices, principles and use cases. They explained the implementation with different scenarios and patterns, difference between monolith and microservices, Object oriented principles with design patterns, advantages and disadvantages of its usage. This study demonstrates the usage of microservices in real world applications with scalable and fault tolerant applications that's used in modern industry practices.

CHAPTER 3

SYSTEM ANALYSIS

3. SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

The current task management landscape relies on disparate tools and platforms that operate independently, leading to a fragmented system where tasks, data, and communication are isolated within specific tools or teams. Access to tasks and project information is often decentralized, with varying levels of control and visibility. This lack of centralized access management poses challenges in controlling user permissions and ensuring data security. Existing solutions may lack robust analytics and reporting features, making it difficult for organizations to gain comprehensive insights into task performance, progress, and overall project health. Existing System doesn't provide centralized platform for communication between users in an organisation which leads to relying on different communication system.

3.2 PROPOSED SYSTEM:

The Task Fusion Management System (TFMS) serves as a centralized platform, unifying task management processes across teams and departments. This centralization ensures that tasks, data, and communication are streamlined and easily accessible from a single interface. TFMS incorporates advanced access management features, offering granular control over user permissions. This ensures that only authorized person can access specific tasks and project information, enhancing data security and confidentiality. The proposed system goes beyond basic task tracking by providing robust analysis and reporting functionalities. Decision-makers can leverage TFMS to gain data-driven insights into task performance, identify bottlenecks, and track progress against strategic goals. TFMS introduces a holistic approach to task tracking, incorporating features such as prioritization, dependencies, and ownership. This ensures that tasks are efficiently monitored and managed, facilitating alignment with organizational objectives. TFMS is designed to accommodate the diverse needs of organizations and departments. It allows for flexible customization to adapt to different workflows and structures, promoting a tailored approach to task management within the organization.

CHAPTER 4

SYSTEM REQUIREMENTS AND

SPECIFICATIONS

4.1 FUNCTIONAL REQUIREMENTS

A Functional Requirements (FR) describes inputting all details regarding department lists, projects in each department, teams in each project, and tasks in each team.

- **User Management**

Allow creating, updating, and deactivating user accounts, implement role-based access control to manage permissions enable single sign-on integration for authentication

- **Task Management**

Provide functionality to create, assign, update, and close tasks, allow defining task attributes like title, description, due date and priority. Implement task dependencies and subtasks

- **Analytics**

Display Statistics of tasks and users based on task stage, completion dates, overdue days, number of tasks, and task priority, in table format and pie chart and timelines.

- **Integrations**

Integrate with email clients to allow for notifying, sync tasks, and updates

4.2 NON FUNCTIONAL REQUIREMENTS

- **Scalability**

Handle increasing numbers of users, tasks, and projects without performance degradation provision for future growth and expansion of the user base.

- **Availability and Reliability**

Maintain high uptime and minimize downtime for critical operations implement redundancy and failover mechanisms for disaster recovery.

- **Security and Compliance**

Ensure end-to-end security by implementing authentication and role-based management from user interface and interface Api's to provide robust security and ensure the application mishandling.

- **Usability and Accessibility**

Provide an intuitive and user-friendly interface for diverse user personal to ensure the application is accessible to users with disabilities.

- **Performance**

Achieve fast response times for common user interactions and queries optimize database queries and caching to enhance system responsiveness.

- **Maintainability and Extensibility**

Design a modular and loosely coupled architecture for easy changes and upgrades allow configuring and customizing the system to meet evolving business requirements

4.3 SYSTEM REQUIREMENTS

4.3.1 MINIMUM HARDWARE REQUIREMENTS

Processor	: intel i5 or higher
RAM	: 4-8 GB(minimum)
HARD DISK	:10 GB

4.3.2 MINIMUM SOFTWARE REQUIREMENTS

Operating System	:Windows(10 and above)
Backend Web Framework	: Flask , Express js(Node.js)
Frontend	: Html , CSS, JavaScript, React js
Database	:Postgresql, Redis

4.3.3 PROTOCOLS

Web RTC, Web Sockets, Http, Smtip

4.3.4 ARCHITECTURE DESIGNS

Microservice Architecture

Message Queues

4.3.5 LIBRARIES USED:

4.3.5.1 Node Mailer: Node Mailer is a module for Node.js applications that allows you to send emails from within Node.js server. It provides an functionality for sending emails and supports a wide range of email transports, including SMTP, Send mail, and more. It also supports HTML emails, attachments, and other advanced features.

4.3.5.2 Material-UI: Material-UI is a popular React UI library that implements Google's Material Design principles. It provides a set of pre-built React components that follows Material Design guidelines, including buttons, cards, menus, navigation bars, and more. Material-UI components are highly customizable and responsive, making it easier to build consistent and visually appealing user interfaces for web applications.

4.3.5.3 JSON Web Tokens (JWT): JWT, or JSON Web Tokens, is an open standard for securely transmitting information between parties as a JSON object. It is commonly used for authentication and authorization in web applications. A JWT is composed of three parts: a header, a payload, and a signature. The header contains metadata about the token, the payload contains the claims (user data), and the signature is used to verify the integrity of the token

CHAPTER 5

SYSTEM DESIGN

5. SYSTEM DESIGN

5.1 SYSTEM IMPLEMENTATION:

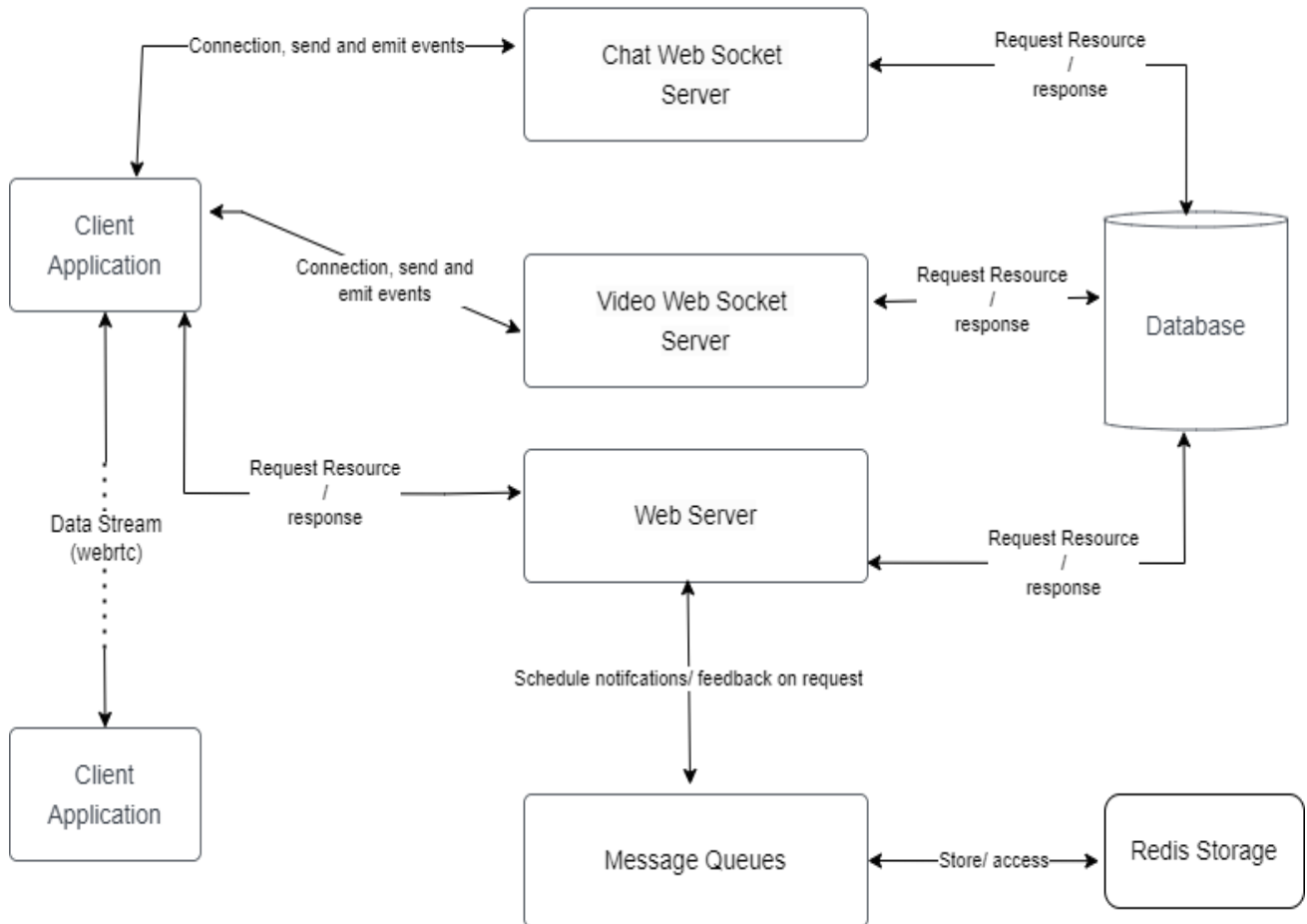


Fig : 5.1 System Implementation

5.1.1 SYSTEM IMPLEMENTATION OVERVIEW:

Microservices:

Microservices architecture is an architectural style that structures an application as a collection of services. Unlike in traditional monolithic architectures application is broken down into smaller, independent services, each responsible for a specific business capability, and also provides scalability, loose coupling, reliability and rapid development.

- **Independent Services:** Each service is self-contained and responsible for a single business capability. This allows for independent development, deployment, scaling of each service and allows for faster development cycles, as teams can work on different services simultaneously.
- **Loose Coupling:** Services communicate with each other through APIs, typically using lightweight protocols such as HTTP or messaging queues. This loose coupling reduces dependencies between services, making it easier to modify, update, or replace individual services without impacting the rest of the system. It also enables teams to use different programming languages, frameworks, or databases for each service, choosing the technology that best fits the specific requirements.
- **Scalability:** Microservices can be independently deployed and scaled. If a particular service experiences increased demand, it can be scaled horizontally (by adding more instances) or vertically (by increasing resources for each instance) without affecting other services. This allows for more efficient resource utilization and better handling of varying workloads.
- **Rapid Development and Deployment:** The independent nature of microservices allows for faster development cycles and more frequent deployments. Since changes to one service have minimal impact on others, teams can release new features or updates without waiting for the entire application to be ready.
- **Reliability and Resilience:** Microservices promote fault isolation, that failures in one service are less likely to affect the overall system. Services can be designed to handle failures gracefully, implementing mechanisms such as retry strategies, circuit breakers, and graceful degradation. This improves the overall reliability and

resilience of the application, as failures are contained within individual services.

5.1.2 Components:

- **Client Application:** This is the User Interface where users interact with the application for perform create, edit, update and delete operations on project use cases.
- **Chat Socket Server:** This server handles chat communication between users. It process operations on message routing and broadcast between clients. This server is designed using web socket protocol to provide bi-directional communication for interactive sessions.
- **Video Signaling Server:** This server acts as a signaling mechanism for WebRTC communication. It establishes connections between clients and process the exchange of necessary data (e.g., ICE candidates, session descriptions) to initialize peer-to-peer WebRTC connections. This server is designed using web socket protocol to provide bi-directional communication for share data between peer users.
- **Webserver:** This web server which handles incoming requests from client application and responds with performing business operations and sending data in different formats like JSON, HTML, Text...etc. It uses http protocol to communicate.
- **Message Queue:** This server handles message queuing and asynchronous tasks for mail notifications. It schedules the tasks to automate and trigger functions at specific times. Which takes the load and provides faster request handling of webserver.
- **Redis Storage:** This is an in-memory data store used to store scheduling sessions. It provides temporary and faster storage for session data required by the message queue for scheduling tasks.
- **Database:** This is a relational database used for storing persistent data related to the application. It stores user information, chat history, or other application-specific.

5.2 FLOW CHART

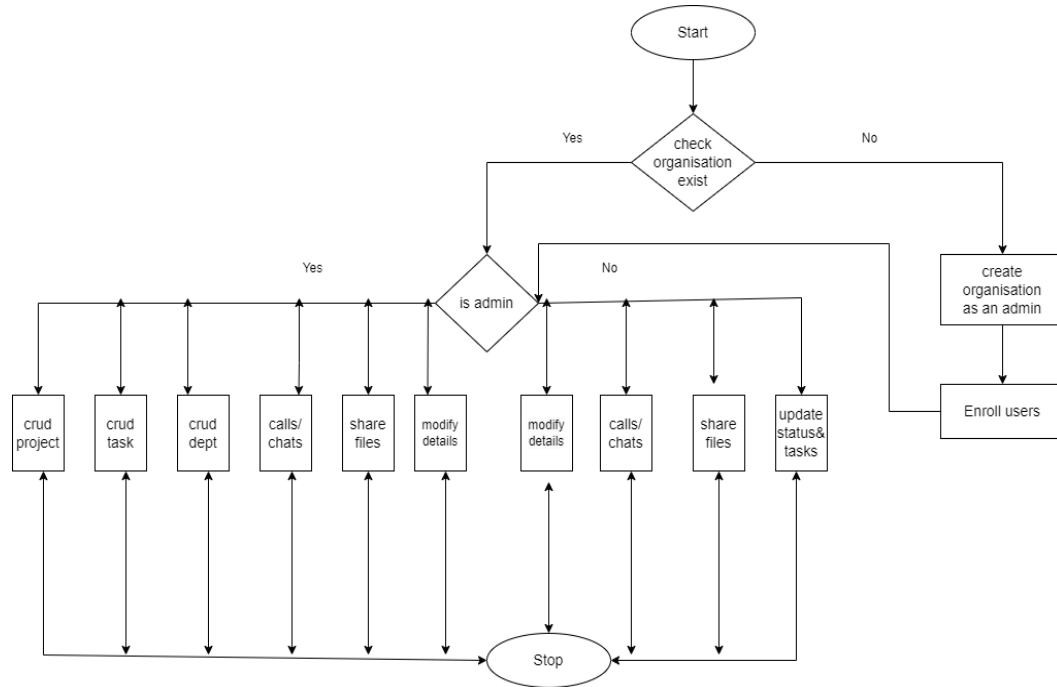


Fig : 5.2 Flow chart

5.2.1 Flow chart overview

The arrows and flow connections between these elements indicate the sequence and dependencies of the different steps in the process. The diamond-shaped decision points represent conditional branching based on the evaluation of certain conditions. This diagram represents the workflow or process related to the administration and management of an organization, including tasks such as creating the organization, enrolling users, and performing various operational activities within the system.

5.3 UML DIAGRAMS

UML (Unified Modeling Language) is a standardized graphical modeling language used to visualize, specify, construct, and document the artifacts of a software system. It provides a set of diagrams and notations to represent different aspects of a system's design and architecture.

The UML diagram shown in the image appears to be a type of UML diagram, specifically a use case diagram. Use case diagrams are used to model the functionality and interactions of a system from the perspective of the users or actors. The "Admin" element represents an actor, which is typically a user, system, or external entity that interacts with the system.

The different elements connected to the "Admin" actor (Analysis, Project, Organization, Tasks, Departments, Chat, Calls, Files) represent use cases, which are the specific functionalities or actions that the system provides to the actor.

The connections between the "Admin" actor and the use cases indicate the relationship between the actor and the system's functionality.

UML diagrams, including use case diagrams, are widely used in software engineering and system design to help understand, analyze, and communicate the requirements, structure, and behavior of a system. They provide a standardized visual language that can be used throughout the software development lifecycle, from requirements gathering to design and implementation.

The connections between the User and the various use cases (Project, Tasks, Chats, Calls, Files) indicate the interactions and relationships between the user and the system's functionalities. These connections represent the user's ability to perform .

USE CASE DIAGRAM

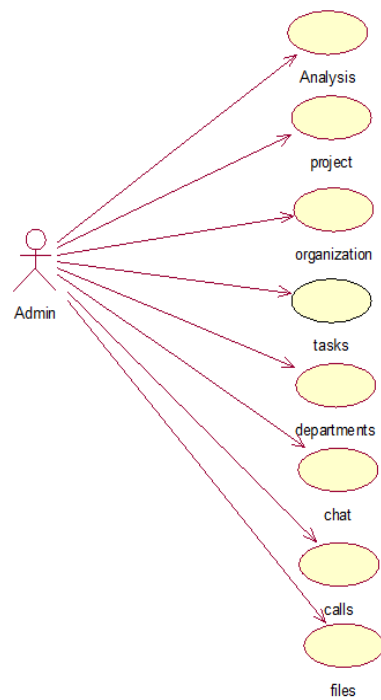


Fig : 5.3.1 Admin role

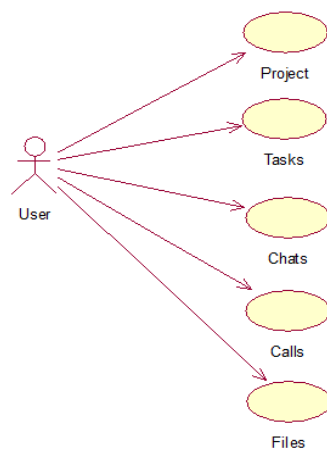


Fig : 5.3.2 User role

CHAPTER 6

IMPLEMENTATION OF SYSTEM

6.1 DATABASE SCHEMA

A Database schema is a logical structure that defines how data should be organized, stored, and related within a database management system (DBMS). It serves as a blueprint for the database, ensuring data integrity, consistency, and efficient storage and retrieval. Key components of a database schema include:

- **Tables:** Represent entities or objects and store related data in rows and columns.
- **Columns:** Define the specific data elements or attributes to be stored within a table, with defined data types.
- **Primary Keys:** Uniquely identify each row in a table, enforcing entity integrity and establishing relationships.
- **Foreign Keys:** Reference the primary key of another table, enabling data integrity and facilitating data retrieval across tables.
- **Constraints:** Rules or restrictions applied to the data to maintain integrity, such as primary key, foreign key, unique, and check constraints.
- **Indexes:** Data structures that improve query performance by allowing faster data retrieval.
- **Relationships:** Associations between tables, such as one-to-one, one-to-many, and many-to-many relationships, implemented using primary and foreign key constraints.

Proper database schema design is crucial for optimizing data storage, ensuring data integrity, facilitating efficient data retrieval, and enabling scalability and maintainability of the database system.

Tables:

- **Organization:** Stores information about organizations, such as their name, description, and creation date

- **Files:** Stores information about files uploaded within organizations, including their source, type, and creation date
- **User info:** Stores information about users within organizations, including their credentials, contact information, roles, and socket connections.
- **Group info:** Stores information about groups within organizations, including their name, description, and creator.
- **Messages:** Stores messages sent within the organization, including sender, receiver, group, attached files, and message content.
- **User Group Association:** Associates users with groups within the organization for many to many relationship.
- **Department info:** Stores information about departments within organizations, including their name, description, and associated users.
- **Projects info:** Stores information about projects within departments, including their status, description, and associated users.
- **Teams info:** Stores information about teams within departments, including their name, description, and creator.
- **Team User Associaton:** Associates users with teams within the organization for many to many relationship.
- **Project User Association:** Associates users with projects within departments and teams for many to many relationship
- **Dept User Associaton:** Associates users with departments and teams within the organization for many to many relationship.
- **Task Types:** Stores types of tasks associated with projects and departments.
- **Tasks:** Stores information about tasks associated with projects and departments, including their priority, description, and dates.
- **Task User Association:** Associates users and teams with tasks within projects
- **Comments:** Stores comments made on tasks within projects, including their content.

These tables together form a relational database schema for managing various aspects of an organization, including users, groups, projects, tasks, teams, etc..

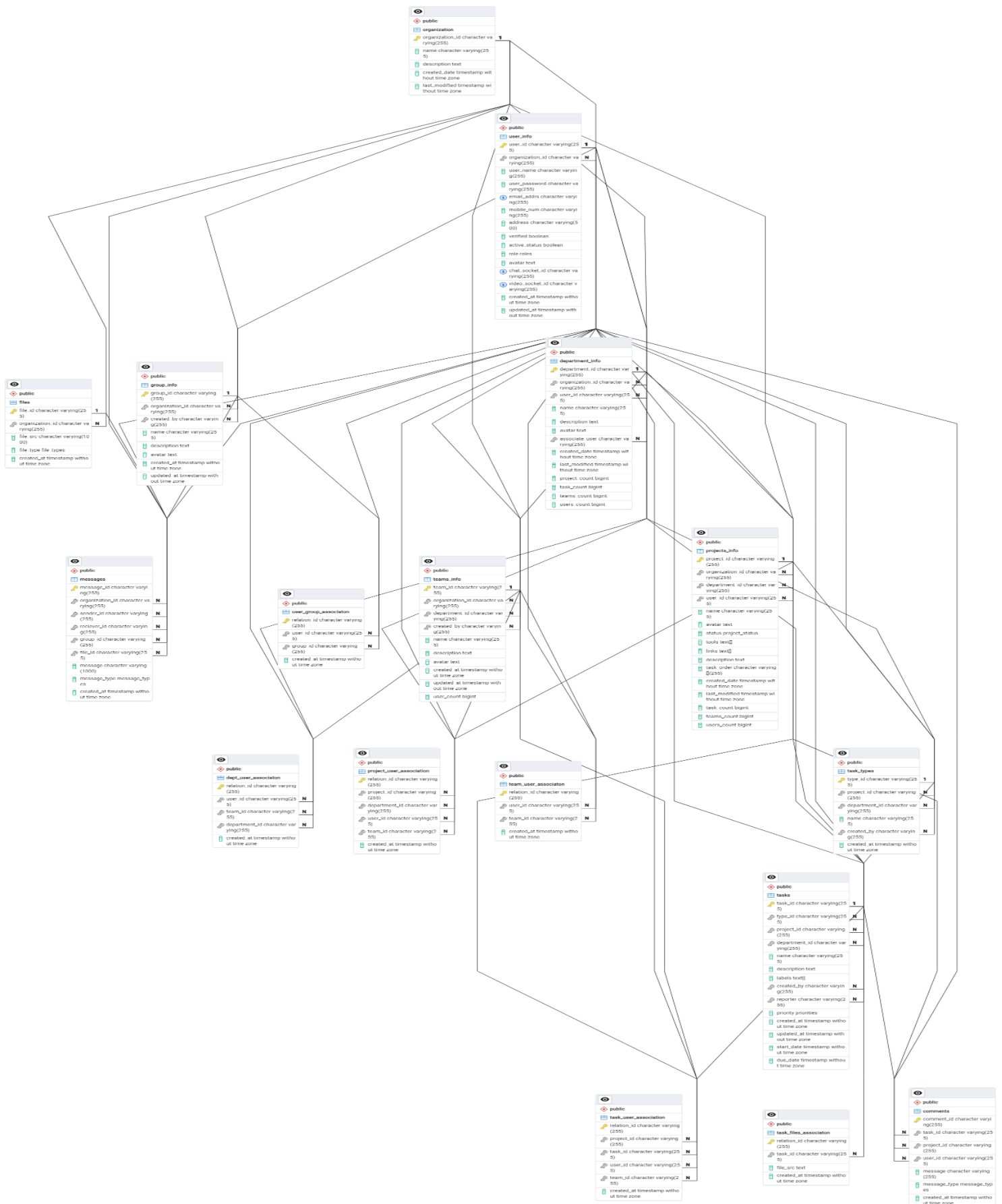


Fig : 6.1.1 Database Schema

6.2 SIGNALING SERVER

A signaling server is a crucial component in facilitating real-time communication between clients, especially in peer-to-peer (P2P) scenarios such as video/audio conferencing, online gaming, and collaborative applications. It serves as an intermediary for establishing direct connections between clients, enabling them to exchange data and media directly without going through the signaling server itself.

The primary role of a signaling server is to handle the initial signaling process, which involves exchanging metadata and negotiating the communication parameters between clients. This signaling process typically involves the following steps:

- **Client Connection:** Clients establish a connection with the signaling server, usually using Web Sockets or other bidirectional communication protocols.
- **Session Initiation:** When a client wants to initiate a communication session (e.g., a video call), it sends a session initiation request to the signaling server.
- **Session Announcement:** The signaling server forwards the session initiation request to the intended recipient(s), essentially announcing the new session to the other client(s).
- **Offer/Answer Exchange:** The clients exchange Session Description Protocol (SDP) messages, known as "offer" and "answer," through the signaling server. These messages contain information about the media types, codecs, and other parameters needed for the direct peer-to-peer connection.
- **ICE Candidate Exchange:** Clients also exchange ICE (Interactive Connectivity Establishment) candidates through the signaling server. These candidates represent potential network addresses and ports that can be used for the direct P2P connection.
- **Connection Establishment:** Once the offer/answer exchange and ICE candidate gathering are complete, the clients can establish a direct P2P connection using the negotiated parameters and ICE candidates. This direct connection allows them to exchange media and data without going through the signaling server.

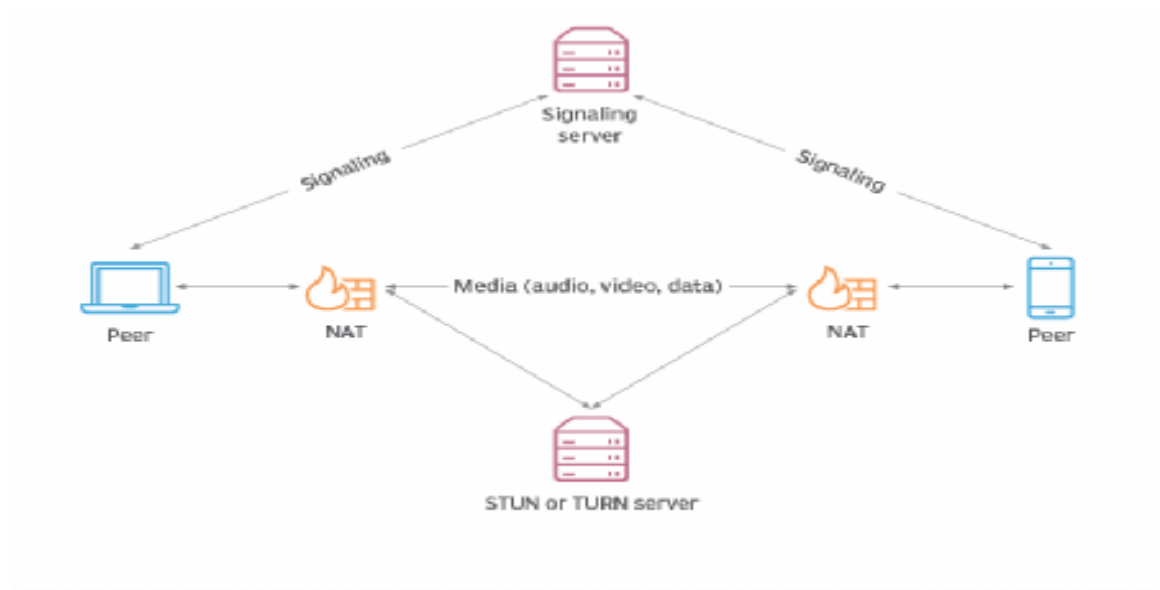


Fig : 6.2.1 Signaling Server

6.3 MAIL QUEUING SYSTEM

A mail queuing system is a software component or service that manages the process of sending email messages in an asynchronous and efficient manner. Instead of sending emails directly from the application, the email requests are added to a queue, and a separate processor service handles the actual delivery of the emails.

- **Queue:** The mail queuing system usually relies on a message queue, which is a data structure that stores and manages the email messages or requests. Popular queue systems include RabbitMQ, Amazon SQS, Apache Kafka, and Redis.
- **Producer:** When an application needs to send an email, it creates a message or task containing the email details (recipient, subject, body, attachments, etc.) and adds it to the queue. This process is often referred to as "producing" or "enqueueing" a message.
- **Consumer(s):** One or more consumer processes or workers continuously monitor the queue and retrieve (or "dequeue") messages from it. These consumers are responsible for actually sending the email messages using an email delivery service or SMTP server.

- **Retry and Failure Handling:** If an email fails to send due to a temporary issue (e.g., SMTP server unavailability, network issue), the mail queuing system can automatically retry sending the email after a specified delay or follow a configured retry policy. Failed messages can also be moved to a separate queue or storage for manual inspection or error handling.

Using a mail queuing system provides benefits such as improved application performance, increased reliability, scalability, and better resource utilization. It separates the email sending logic from the main application, allowing for asynchronous processing and better fault tolerance.

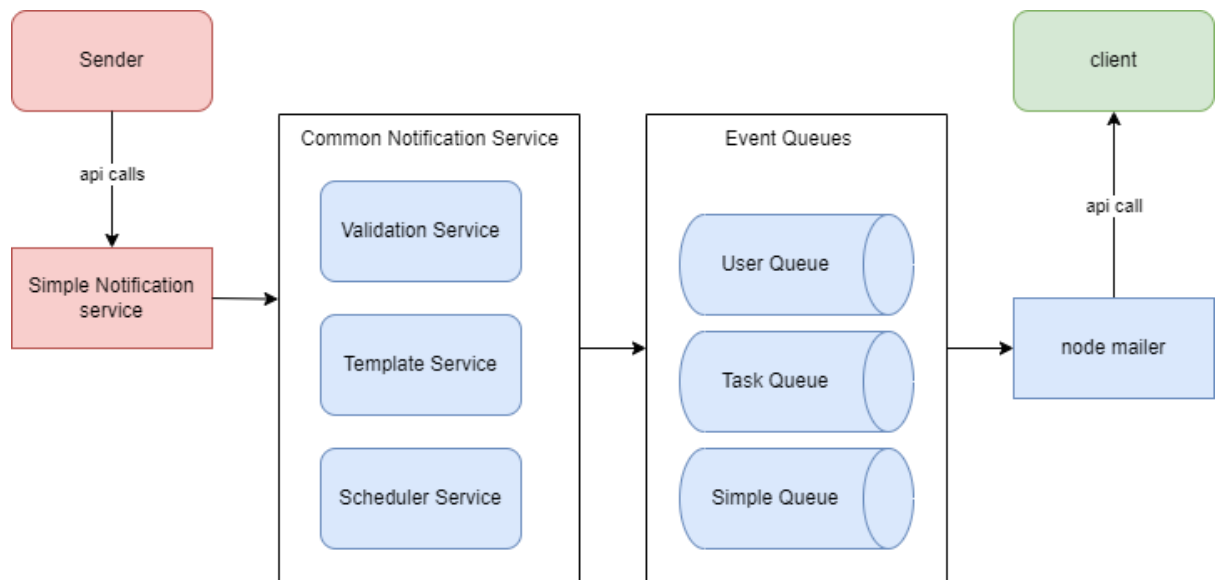


Fig : 6.3.1 Mail Queuing server

6.4 WEB SOCKET SERVER

A WebSocket server is a server-side component that enables real-time, bidirectional communication between a client (typically a web browser) and the server over a single, long-lived TCP connection. It serves as the counterpart to the WebSocket client, facilitating the establishment and management of WebSocket connections.

- **WebSocket Protocol Support:** The server must implement the WebSocket protocol, which defines the handshake process, data framing, and communication

rules between the client and server.

- **Connection Establishment:** When a client initiates a WebSocket connection, the server responds to the initial HTTP upgrade request and performs the WebSocket handshake. If the handshake is successful, the connection is upgraded from HTTP to the WebSocket protocol.
- **Bidirectional Communication:** Once the WebSocket connection is established, the server can send and receive data frames to and from the client in real-time, enabling bidirectional communication.
- **Event Handling:** The events play a crucial role in communication between the client and server.
- **Scalability and Concurrency:** WebSocket servers often employ techniques like event-driven programming, asynchronous I/O, and worker threads or processes to handle multiple concurrent connections efficiently.
- **Integration with Existing Systems:** WebSocket servers can be integrated with other server-side components, such as databases, caching systems, or message queues, to enable real-time data exchange and updates.

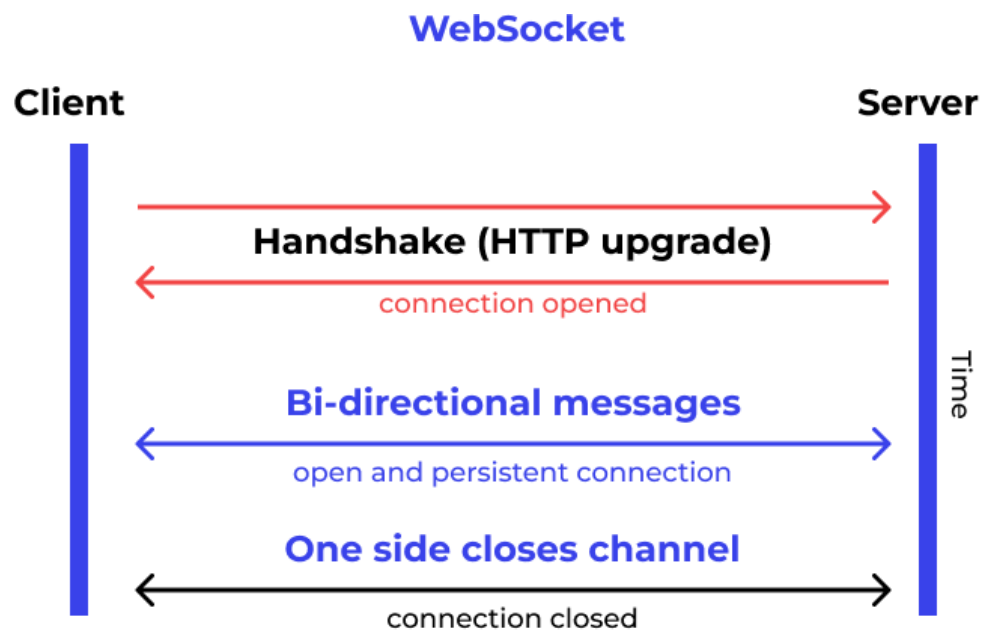


Fig : 6.4.1 Web Socket Server

6.5 FLASK SERVER

Flask is a lightweight and flexible web framework for Python, designed to make it easy to build web applications quickly and with minimal code. A Flask server is essentially an instance of the Flask application running on a server, handling HTTP requests and responses.

Here's a breakdown of the main components and features of a Flask server:

Routing: A URL is a web address that tells a browser how to find a page on the internet. URLs are made up of multiple parts, including the domain name, protocol, and path mapping the URLs to a specific function that will handle the logic for that URL using `@app.route('/url-string')`. With multiple routes the application can handle various business logics with its specific url.

Views: The functions associated with routes are called views. Views returns responses to a particular request, which performs business logic. Views can return different types of data as a response can be Json, Text, Multipart content and HTML templates.

Blueprints: A Blueprint is a way to organize a group of related views and other code. Rather than registering views and other code directly with an application, they are registered with a blueprint. Then the blueprint is registered with the application when it is available in the factory function.

Api's: For the project we created 8 endpoints these are

- **Auth:** Which consists authentication Api's such as registration, authenticate and login. Which are used user login, user registration and user token authentication.

- **User:** Which consists Api's for user operation that handles creating, editing and fetching user details.
- **Dept:** Which consists Api's for department operation that handles creating, editing, deleting, assigning and fetching department details.
- **Proj:** Which consists Api's for Project operation that handles creating, editing, deleting, assigning and fetching Project details.
- **Team:** Which consists Api's for Team operation that handles creating, editing, deleting, assigning and fetching Team details.
- **Tasks:** Which consists Api's for Task operation that handles creating, editing, deleting, assigning and fetching Task details.
- **Chat:** Which consists Api's for Chat operation that handles fetching chats, group chats and user's details.
- **Dashboard:** Which consists Api's for Dashboard operation that handles fetching users, tasks, timeline, aggregative function for count and sum for particular project details.

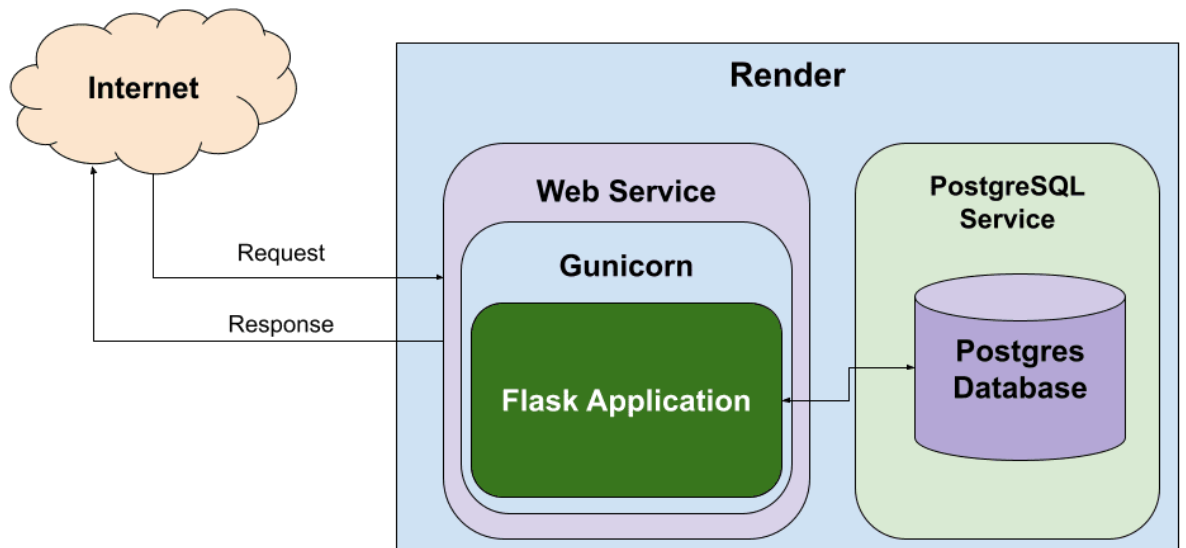


Fig : 6.5.1 Flask application

6.6 REACT

React is a JavaScript library for building user interfaces. It provides a declarative and component-based approach to building interactive UIs, making it easier to manage and update complex UIs efficiently. React allows developers to create reusable UI components and efficiently update the UI when data changes with virtual DOM (Document Object Model) rendering system.

- **Component-Based Architecture:** React applications are composed of small, reusable components. Each component encapsulates its own logic and UI, making it easier to manage and reuse code.
- **Virtual DOM:** React uses a virtual DOM to efficiently update the UI. Instead of directly manipulating the browser's DOM, React creates a lightweight representation of the DOM in memory and updates it efficiently based on changes in data or state. This minimizes DOM manipulation and improves performance.
- **React Hooks:** Hooks are functions that allow to use state and other React features without writing a class. Hooks enable a more functional approach to building components, promoting code reuse and simplifying state management.
- **React Router:** React Router is a popular routing library for React applications, allowing developers to implement dynamic routing and navigation in single-page applications.

Application contains different pages with includes:

- **Dashboard:** Which shows the information regarding analysis of projects which includes count of tasks, user's stages, timeline stages, detailed information of individual user's performance, overall performance and individual task performances with progress.
- **Authentication:** Which handles login and registration process for user's as a organization and normal users.
- **Users:** Which handles user creation, department assignment, user verification assign various roles such as super admin which has complete privileges, admin can have privileges with respective department associated with delete, create, read

and edit permissions and users can have access to view, edit and create on certain levels only such as tasks, chat and other communications.

- **Departments:** Which handles department operation showing department details, associated users, teams and able to create departments.
- **Projects:** Which handles project operation showing project details, associated users, teams, adding attachments and able to create projects.
- **Teams:** Which handles Team operations showing team details, associated users and able to create teams.
- **Tasks:** Which handle task operation for each project we create stages and tasks. Stages describes the life cycle of project and tasks are arranged as part of its life cycle. Allows users to create, edit, delete, reorder stages and tasks, additionally can add attachments associate reporter, users and teams. This implementation is designed according to kanban principles.
- **Chats:** Users can send messages to one to one or one to group that present in same organization.
- **Communication:** Users can communicate through audio, video and screensharing options to share and interact with other users in the organization.

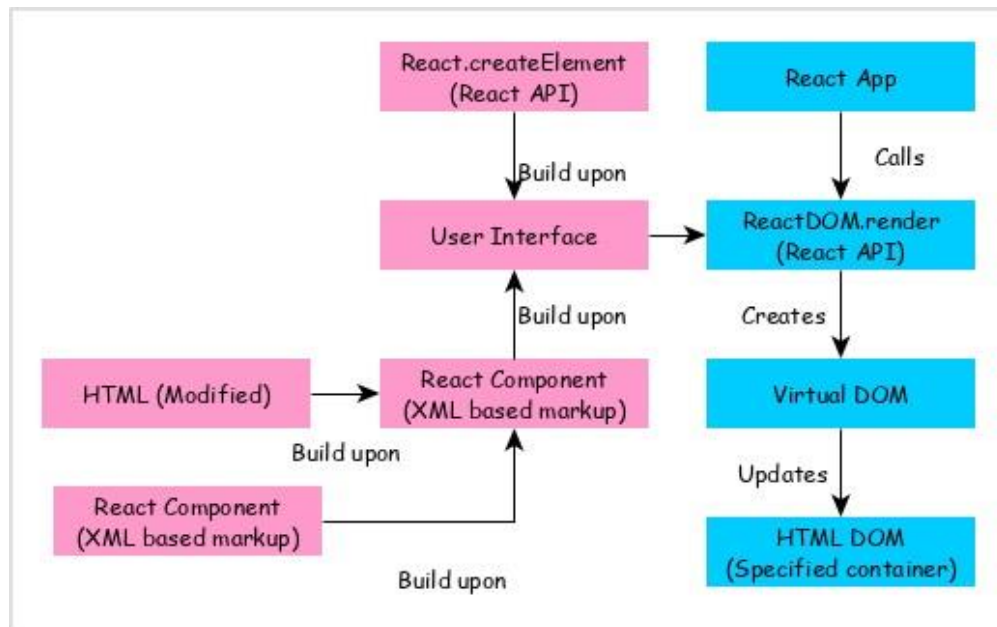


Fig : 6.6.1 React Server

6.7 SAMPLE CODES

6.7.1 FRONTEND SERVER

User-list-row.jsx

```
import {
  Avatar,
  IconButton,
  TableCell,
  TableRow,
  Typography,
  MenuItem,
  Button,
} from '@mui/material';
import CustomPopover, { usePopover } from 'src/components/custom-popover';
import Label from 'src/components/label';
import { useBoolean } from 'src/utlis/use-boolean';
import { ConfirmDialog } from 'src/components/custom-dialog';
import { useNavigate } from 'react-router-dom';
import Iconify from 'src/components/iconify/Iconify';
import { useContext } from 'react';
import { AuthContext } from 'src/auth/JwtContext';

export default function UserListRow({ row, index }) {
  const popover = usePopover();
  const confirm = useBoolean();
  const { user } = useContext(AuthContext);
  const navigate = useNavigate();

  const { user_id, user_name, avatar, email_addrs, mobile_num, verified, role } =
    row;
```

```

const editUser = (userId) => {
  navigate('/dashboard/users/create', {
    state: { userId },
  });
};

return (
  <◇
    <TableRow key={user_id} hover>
      <TableCell sx={{ display: 'flex', alignItems: 'center' }}>
        <Avatar alt={user_name} src={avatar} sx={{ mr: 2 }} />
        {user_name}
      </TableCell>
      <TableCell>{email_addrs}</TableCell>
      <TableCell>{mobile_num || '---'}</TableCell>
      <TableCell>{role === 'super_admin' ? 'SUPER ADMIN' :
role.toUpperCase()}</TableCell>
      <TableCell>
        <Label
          variant="soft"
          color={
            (verified !== false && 'success') ||
            (verified === false && 'error') ||
            'default'
          }
        >
          <Typography variant="body2" fontSize={12} fontWeight="bold">
            {verified ? 'verified' : 'pending'}
          </Typography>
        </Label>
      </TableCell>
    </◇

```

```

<TableCell>
  <IconButton onClick={popover.onOpen}>
    <Iconify icon="eva:more-vertical-fill" />
  </IconButton>
</TableCell>
</TableRow>
<CustomPopover
  open={popover.open}
  onClose={popover.onClose}
  arrow="right-top"
  sx={{ width: 180 }}
>
  <MenuItem
    disabled={user.role === 'user'}
    onClick={() => {
      popover.onClose();
      editUser(user_id);
    }}
  >
    <Iconify icon="solar:pen-bold" />
    Edit
  </MenuItem>
  <MenuItem
    disabled={role === 'super_admin' || user.role === 'user'}
    onClick={() => {
      popover.onClose();
      confirm.onTrue();
    }}
  >

```

6.7.2 BACKEND SERVER

Create_user.py

```
from sqlalchemy import text
from myapp.db import db
from utils.generate_uniqueid import generate_uniqueId
from utils.generate_rand_pass import generate_password
from utils.email_notify import notify_mail

class CreateUser:
    def __init__(self, request):
        self.request = request
        self.data = request.json

    def addNewUser(self):
        try:
            ids = generate_uniqueId(type=['user'])
            user_query = f"""
                insert into user_info (user_id, organization_id, user_name,
user_password, email_addrs, role, address, mobile_num, avatar)
                values(:user_id, :org_id, :username, :password, :email, :role, :address,
:phone, :avatar);
            """

            with db.session() as session:
                session.execute(
                    text(user_query),
                    {
                        "org_id": self.data['org_id'],
                        "user_id": ids.get('user'),
                        "username": self.data["name"],
                        "password": generate_password(),
                        "email": self.data["email"],
```

```

        "address":self.data['address'],
        "phone":self.data['phone'],
        "role":self.data['role'],
        "avatar":self.data["avatar"],
    })
    session.commit()
for dept in self.data['departments']:
    ids_dept = generate_uniqueId(type=['department_user'])
    dept_user_ads_query = f"""
        insert into dept_user_associaton (relation_id, user_id, department_id )
        values(:rel_id, :user_id, :dept_id);
    """
    with db.session() as session:
        session.execute(
            text(dept_user_ads_query),
            {
                "dept_id": dept,
                "user_id":ids.get('user'),
                "rel_id":ids_dept.get('department_user'),
            })
        session.commit()
    notify_mail('/send-user-password', {"user_id":ids.get('user')})
    notify_mail('/send-dept-mail', {"user_list":[ids.get('user')]})

    return {
        "status": True,
        "message": "registered successful",
        "errorcode": 0
    }
except Exception as e:
    print(e)

```

```
return { "status": False, "message": "failed to register", "errorcode": 2 }
```

6.7.3 CHAT SERVICE

Connection.js

```
import { queryDatabase } from "../db/queryDb.js";
import { DisconnectSocket, GroupMessage, PrivateMessage } from "../handler.js";
import { socketIO } from "../socket.js";

const SocketMiddleware = async (socket, next) => {
  const clientID = socket.handshake.auth.clientID;
  if (clientID) {
    const query = {
      name: "set-socket",
      text: "update user_info set chat_socket_id = $1,
active_status=true where user_id = $2",
      values: [socket.id, clientID],
    };
    const query2 = {
      name: "notify-online",
      text: " select user_id, active_status, chat_socket_id from
user_info where user_id != $1 and chat_socket_id notnull",
      values: [clientID],
    };
    next();
  } catch (error) {
    next(new Error(error));
  }
} else {
  next(new Error("Invalid client ID"));
}
};
```

```

const SocketConnection = (socket) => {
  socket.on(
    "private message",
    ({ recipientID, message, userId, type, username, avatar , orgId }) =>
  {
    if (type === "normal") {
      console.log("emits");
      PrivateMessage({ recipientID, message, userId ,
username, avatar, orgId});
    } else {
      GroupMessage({
        recipientID,
        orgId,
        userId,
        message,
        username,
        avatar,
      });
    }
  }
);
socket.on("join", ({ groupID, userId }) => {
  console.log("temporary checking");
});
socket.on("disconnect", () => {
  console.log("disconnect", socket.id)
  const clientID = socket.handshake.auth.clientID;
  if (clientID) {
    DisconnectSocket(clientID);
  } else {
    console.log("invalid id");
  }
}

```

```
export { SocketMiddleware, SocketConnection };
```

6.7.4 VIDEO CONFERENCE

Connection.js

```
import {
  HandleSignaling,
  clientAnswer,
  clientPickupAns,
  clientOffer,
  clientCandidate,
  clientLeave,
  DisconnectSocket,
  clientPickup,
} from "../handler.js";

try {
  const data = await queryDatabase(query);
  next();
} catch (error) {
  next(new Error(error));
}

} else {
  next(new Error("Invalid client ID"));
}

};

const SocketConnection = (socket) => {
  socket.on("newcalls", (message) => {
    const { type, receiverInfo, userInfo } = message;
    clientPickup(type, receiverInfo, userInfo, socket);
  });
  socket.on("accepincoming", (message) => {
```



```

        const { accept, receiverInfo, userInfo } = message;
        clientPickupAns(accept, receiverInfo, userInfo, socket);
    });
    socket.on("offer", (message) => {
        const { payload, receiverId } = message;
        clientOffer(payload, receiverId, socket);
    });
    socket.on("answer", (message) => {
        const { payload, receiverId } = message;
        clientAnswer(payload, receiverId, socket);
    });
    socket.on("candidate", (message) => {
        const { payload, receiverId } = message;
        clientCandidate(payload, receiverId, socket);
    });
    socket.on("leave", (message) => {
        const { payload, receiverId } = message;
        clientLeave(payload, receiverId, socket);
    });
    socket.on("disconnect", () => {
        console.log("disconnected", socket.id);
        const clientID = socket.handshake.auth.clientID;
        if (clientID) {
            DisconnectSocket(clientID);
        } else {
            console.log("invalid id");
        }
    });
};
export { SocketMiddleware, SocketConnection };

```

6.7.5 NOTIFICATION SERVER

Password-reset.js

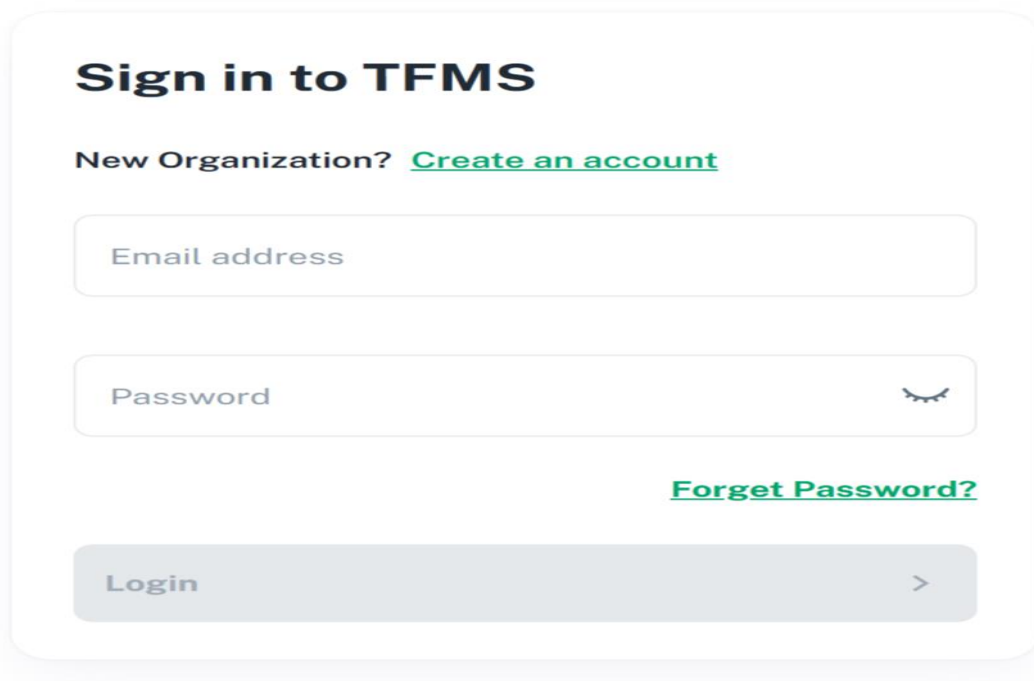
```
const PasswordReset = (action) => {
  const { user_name, user_id, token } = action;
  return {
    subject: "TFMS Password Reset",
    html: `
<div>
<table
  role="presentation"
  style="
    width: 100%;
    border-collapse: collapse;
    border: 0px;
    border-spacing: 0px;
    font-family: Arial, Helvetica, sans-serif;
    background-color: rgb(0, 0, 0);"
  >
<tbody>
<tr>
<td align="center" style="padding: 1rem 2rem; vertical-align: top;
width: 100%;">
<table
  role="presentation"
  style="max-width: 600px; border-collapse: collapse; border:
0px; border-spacing: 0px; text-align: left;">
<tbody>
<tr>
<td style="padding: 40px 0px 0px;">
<div style="text-align: left;">
<div style="padding-bottom: 20px;">
    </div>
  </div>
  <div style="padding: 20px; background-color: rgb(255,
255, 255);">
    <div style="color: rgb(43, 21, 203); text-align: left;">
      <h1 style="margin: 1rem 0">Hi
      email.</p>
      <p style="padding-bottom: 16px">Thanks,<br>The
Priya team</p>
    </div>
  </div>
  <div style="padding-top: 20px; color: rgb(255, 250,
250); text-align: center;">
    <p style="padding-bottom: 16px">Made with
TFMS</p>
  </div>
</td>
</tr>
</tbody>
</table>
</td>
</tr>
</tbody>
</table>
</div>
`,
  };
};
export { PasswordReset };

```

CHAPTER 7

OUTPUT SCREENS


7.1 LOGIN SCREENS



Sign in to TFMS

New Organization? [Create an account](#)

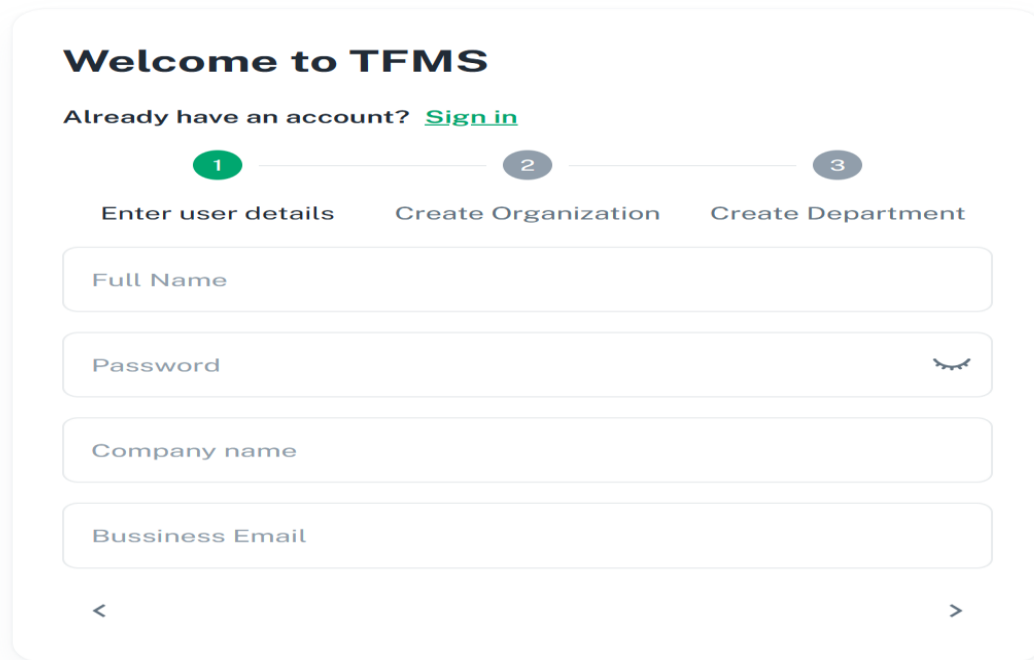
Email address

Password 

[Forget Password?](#)

Login >

Fig : 7.1.1 Sign in page




Welcome to TFMS

Already have an account? [Sign in](#)

1 2 3

Enter user details Create Organization Create Department

Full Name

Password 

Company name

Bussiness Email

< >

Fig : 7.1.2 Create page

7.2 DASHBOARD

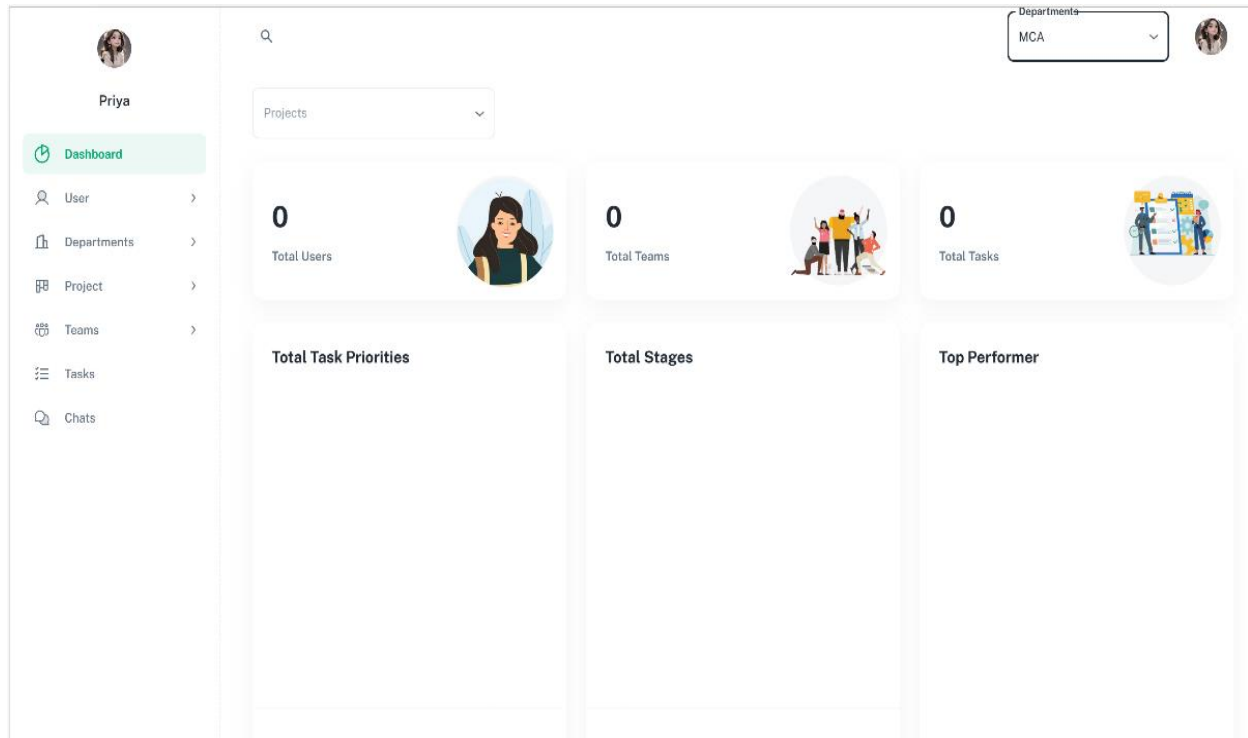


Fig : 7.2.1 Home page

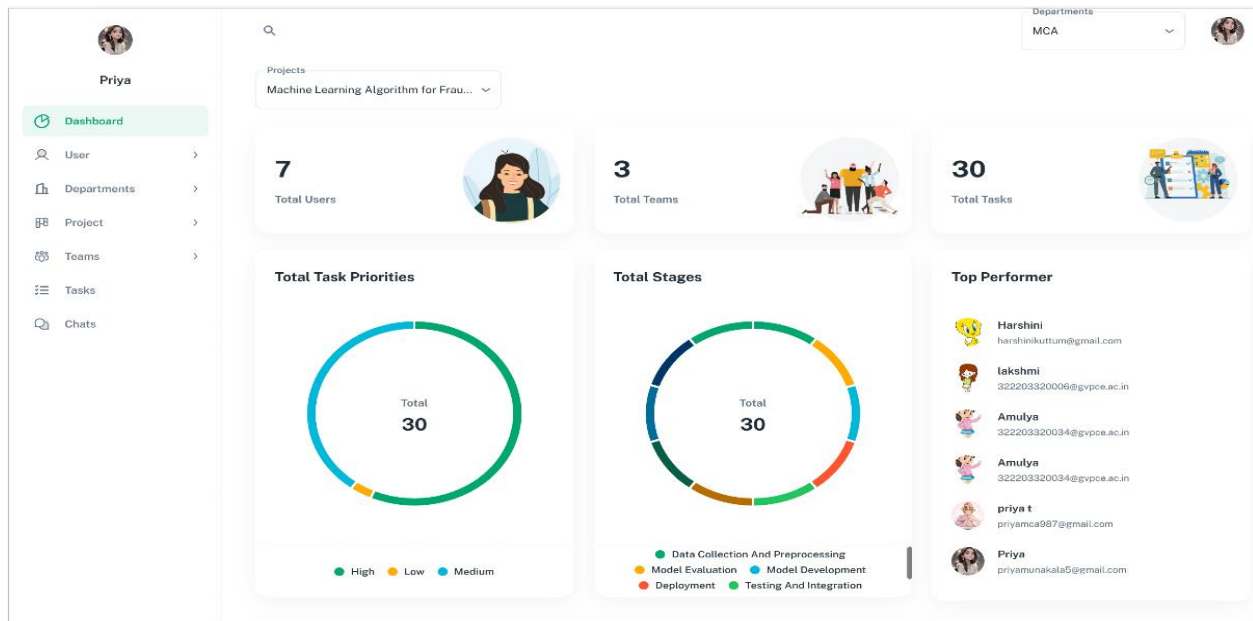


Fig : 7.2.2 Dashboard Page

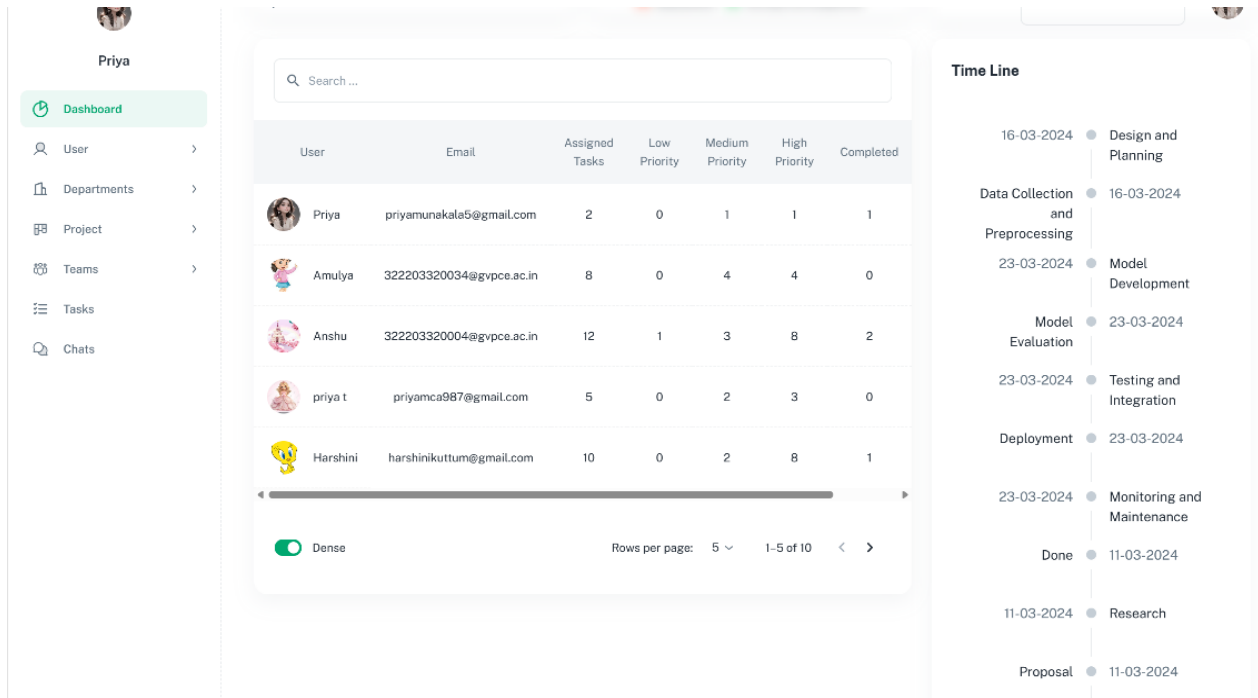


Fig : 7.2.3 Dashboard user list

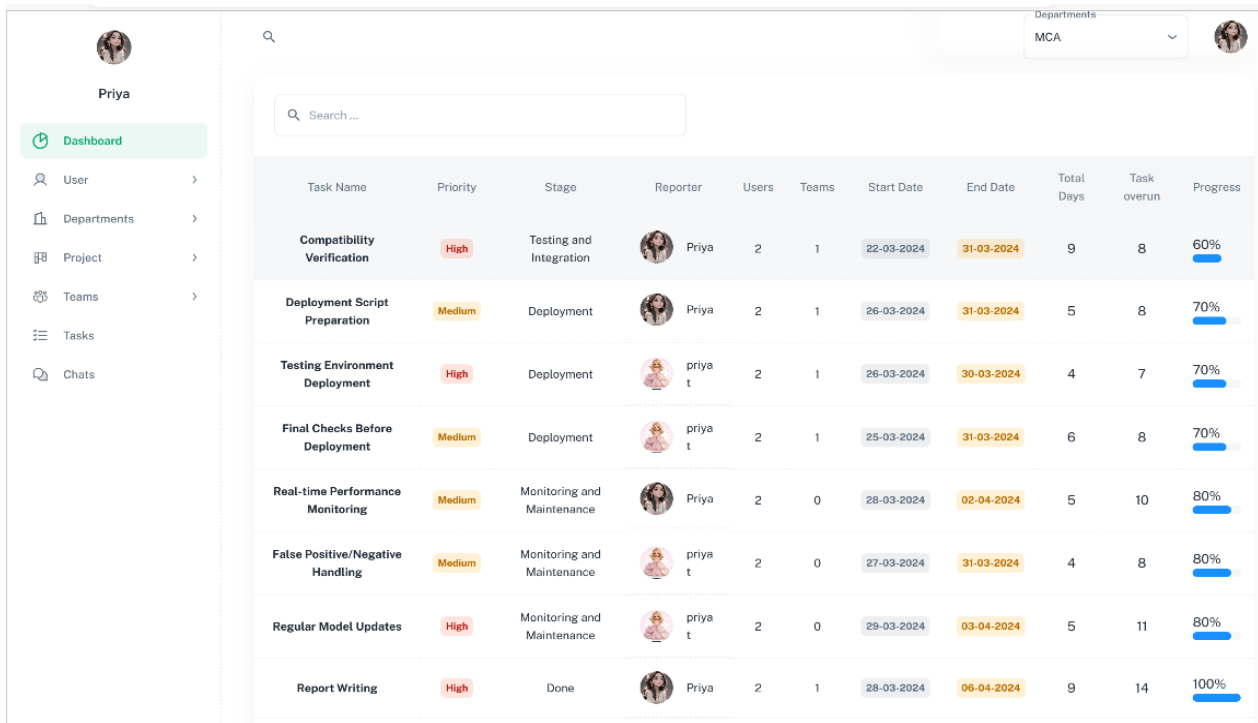


Fig : 7.2.4 Dashboard task list

7.3 User pages

The screenshot shows the 'Users' page in a web application. On the left is a sidebar with a user profile for 'Priya' and navigation links: Dashboard, User (selected), Departments, Project, Teams, Tasks, and Chats. The top right shows a search bar, a dropdown menu set to 'MCA', and another user profile. The main content area is titled 'Users' and includes a breadcrumb 'Dashboard > Users > List'. Below this are filters: 'All' (12), 'Verified' (5), 'Un Verified' (7), and 'Un Assigned' (2). A search bar is present above a table of users. The table has columns for Name, Email, Phone Number, Role, and Status. Each row includes a user icon and a three-dot menu.

Name	Email	Phone Number	Role	Status
Priya	priyamunakala5@gmail.com	9502715717	SUPER ADMIN	Verified
chandu	322203320007@gvpce.ac.in	9874335098	USER	Pending
Harshiiii	322203320033@gvpce.ac.in	9502715717	USER	Verified
Amulya	322203320034@gvpce.ac.in	9513217799	USER	Pending
lakshmi	322203320006@gvpce.ac.in	9837832833	USER	Pending
Avi	mrcroct1840@gmail.com	9569845279	ADMIN	Verified
Harshini	harshinikuttum@gmail.com	9502715717	USER	Pending

Fig : 7.3.1 User list

The screenshot shows the 'Create' page for adding a new user. The sidebar is identical to the previous page. The top right shows a search bar, a dropdown menu set to 'Health', and a user profile. The main content area is titled 'Create' with a breadcrumb 'Dashboard > Users > Create'. On the left, there is a section for 'Upload photo' with a circular placeholder and text indicating allowed file types (*.jpeg, *.jpg, *.png) and a maximum size of 3 Mb. Below this is a 'User Verification' section with a 'Verify' button and a note: 'Automatically sends the user a verification email on creating a user. You can verify it manually while editing'. On the right, there are input fields for 'Name', 'Email Address', 'Phone Number', 'Roles' (a dropdown menu), 'Select Departments', and 'Address'. A 'Create' button is located at the bottom right of the form.

Fig : 7.3.2 User create page

Create
Dashboard > Users > Create

User Verification
Automatically sends the user a verification email on creating a user. You can verify it manually while editing

Verify

Name: Anshu
Email Address: 322203320004@gvpce.ac.in
Phone Number: 9502715717
Roles: User
Address: kotturu

Select Departments: Health, Finance, IT, MCA

Save

Fig : 7.3.3 User verify page

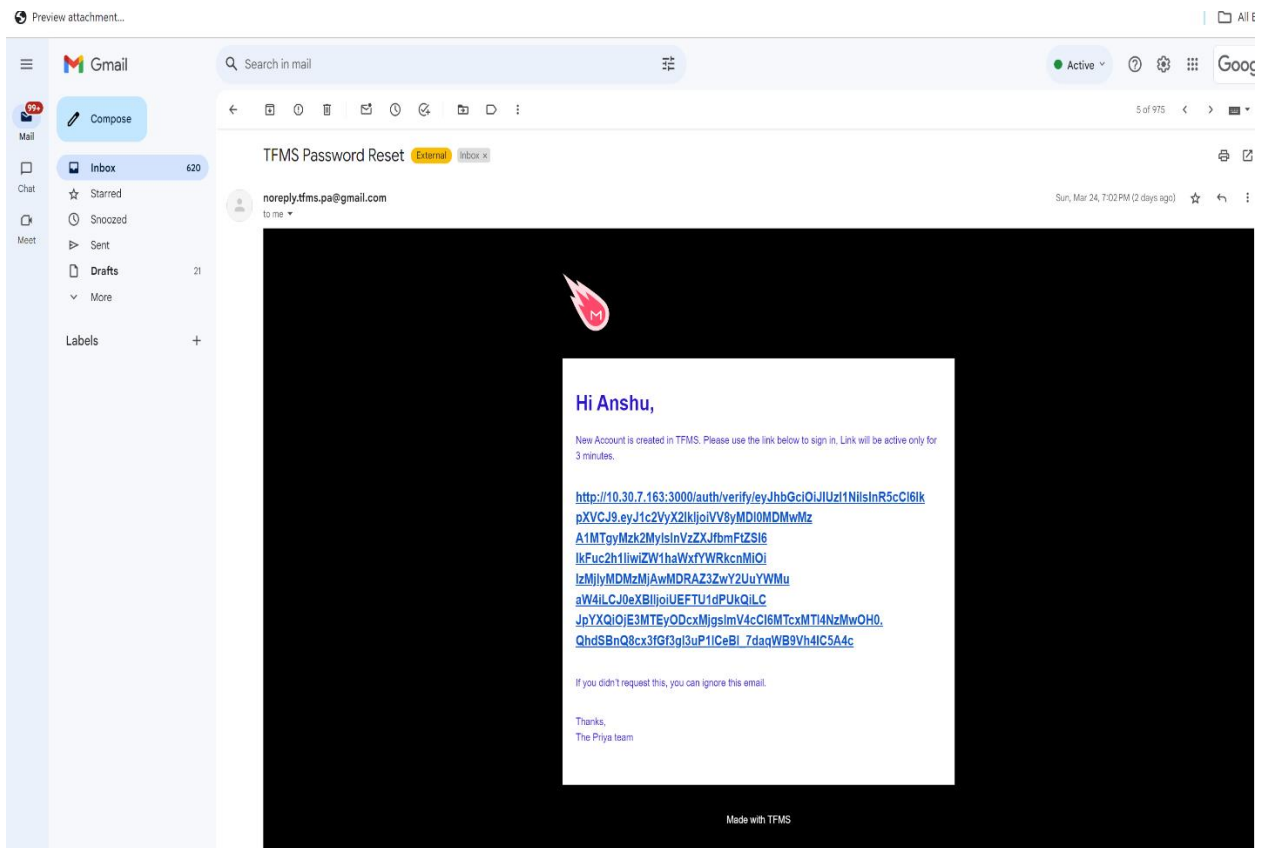


Fig : 7.3.4 User verification mail

7.4 Departments page

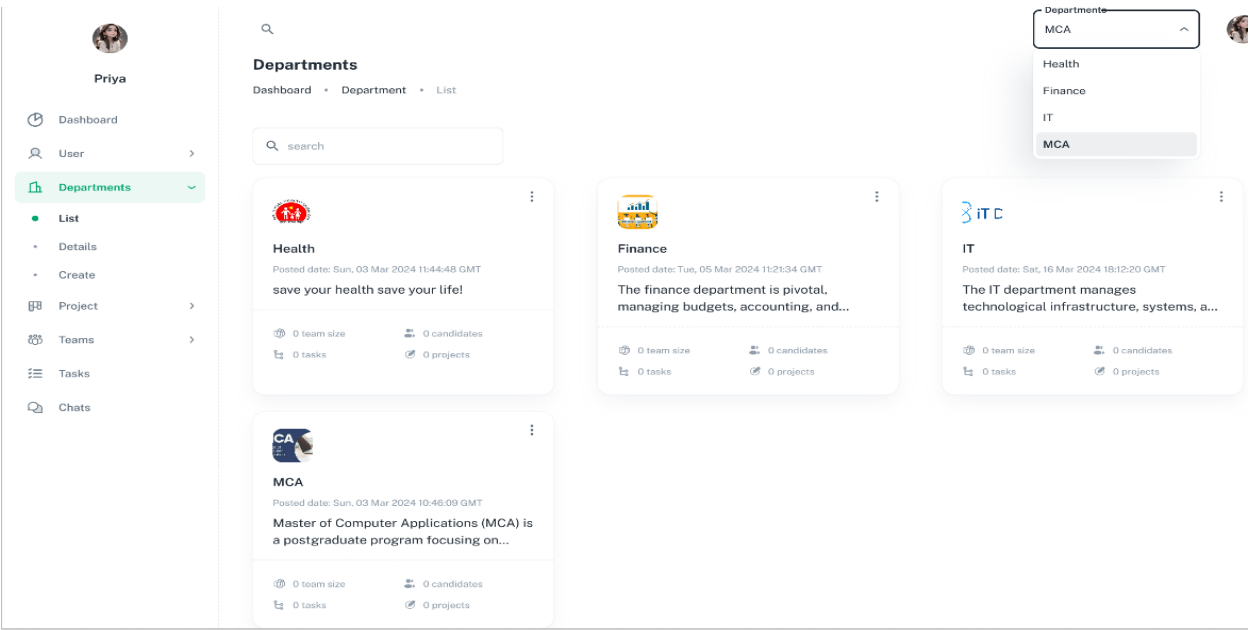


Fig : 7.4.1 Department list

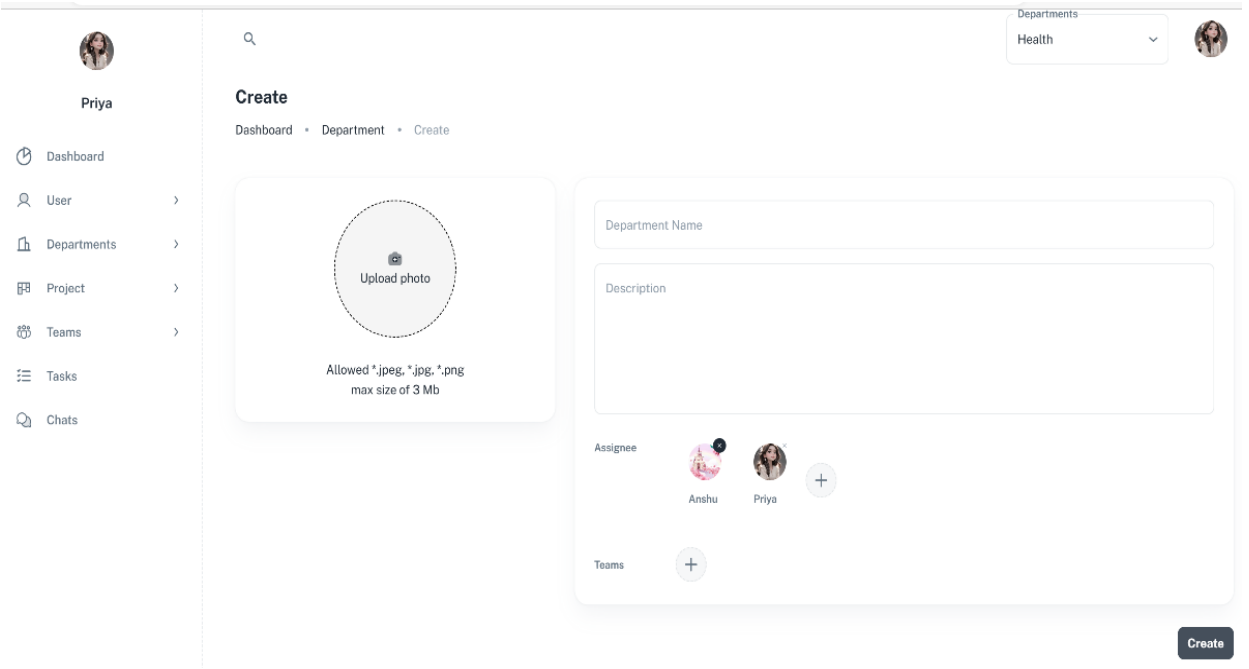


Fig : 7.4.2 Create Department

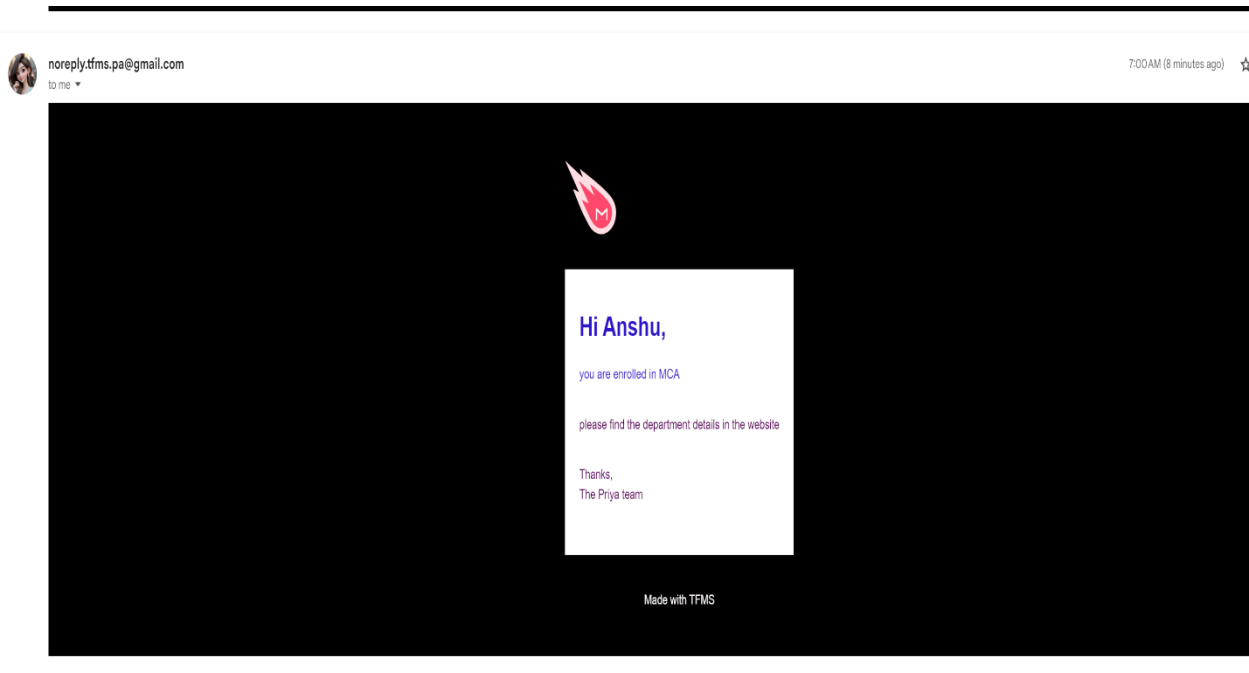


Fig : 7.4.3 Department enrolled mail

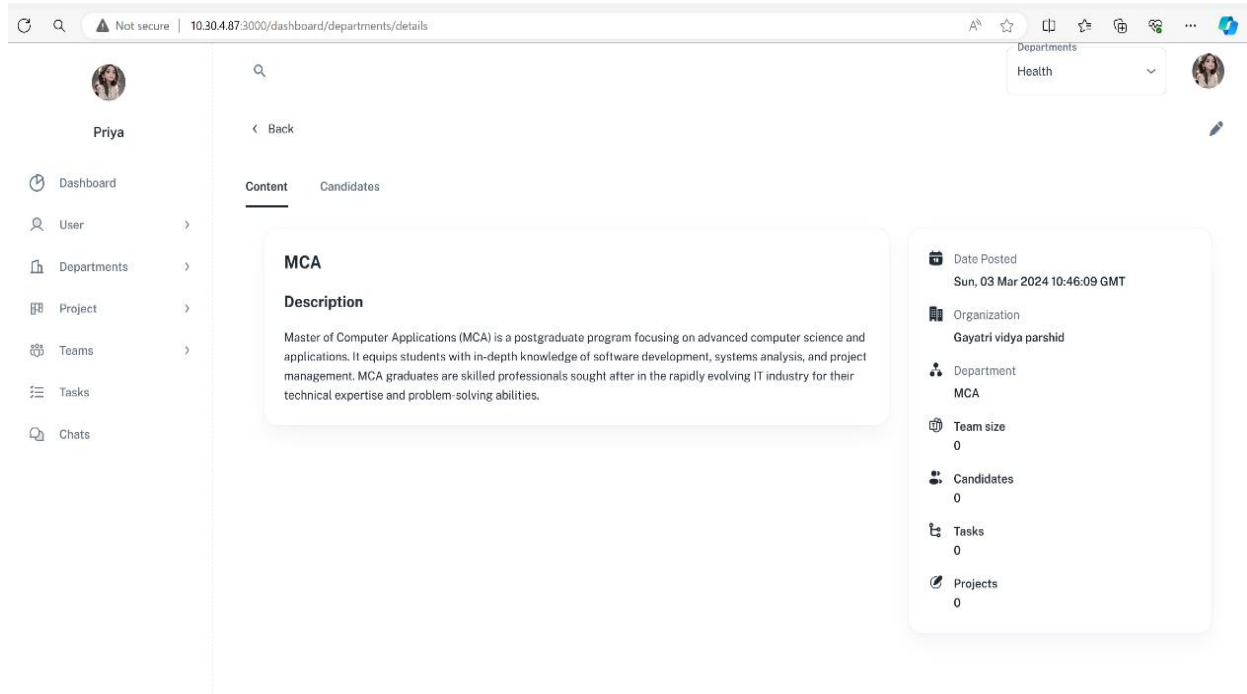


Fig : 7.4.4 Department details

7.5 Project page

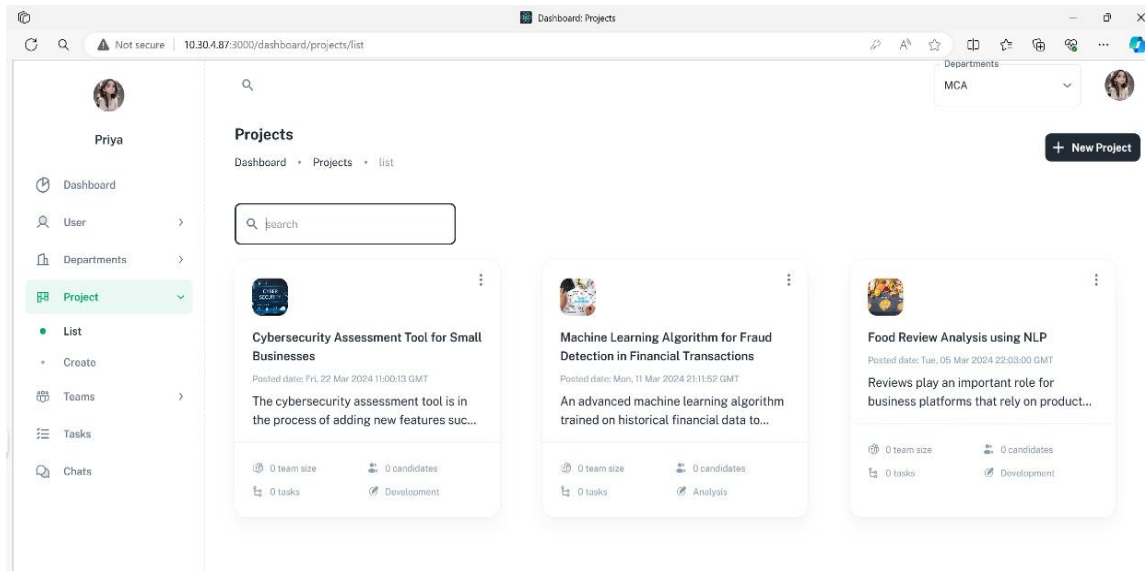


Fig : 7.5.1 Project List

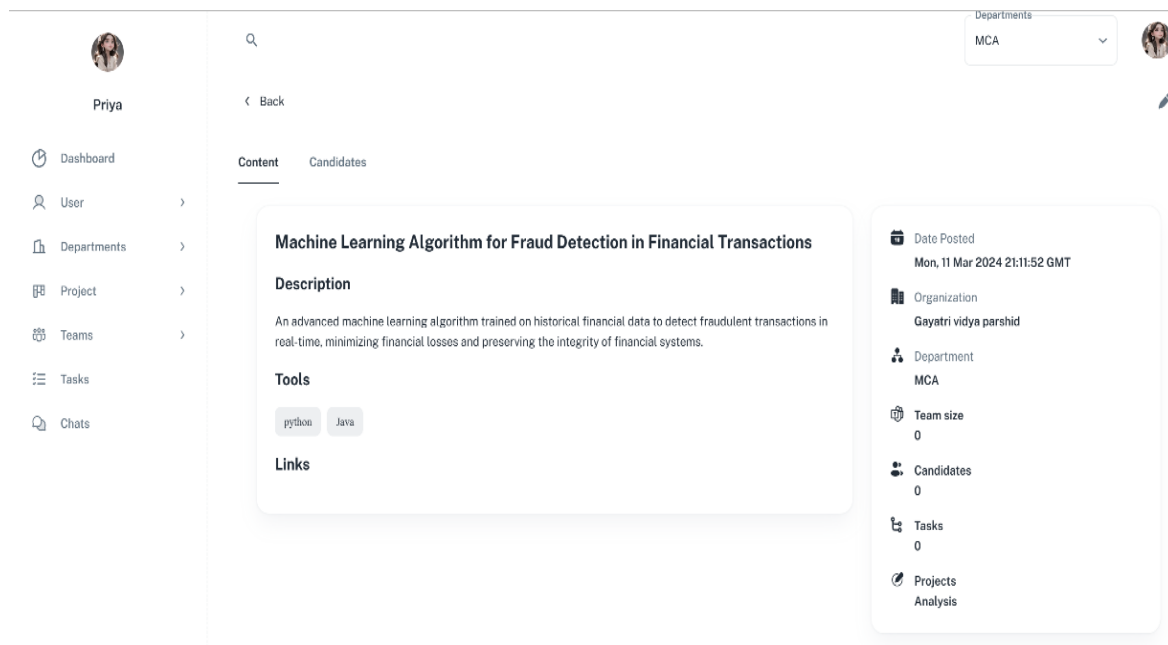


Fig : 7.5.2 Details of the project

Fig :7.5.3 Create new project



Fig : 7.5.4 Enrolled mail for project

7.6 Teams Page

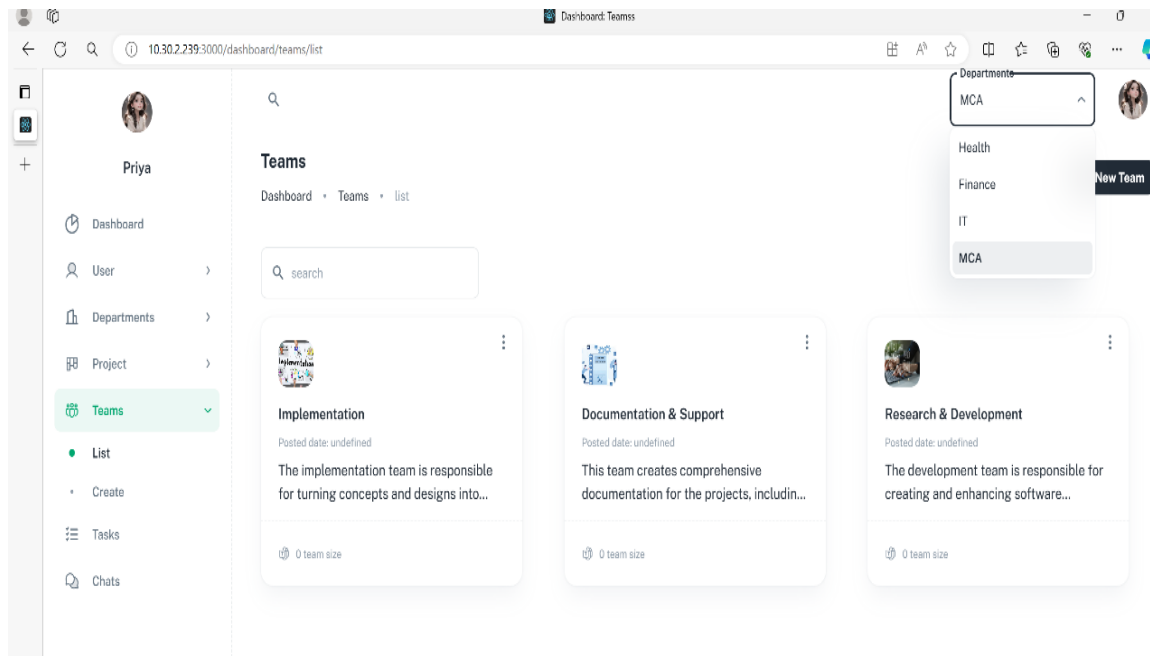


Fig : 7.6.1 Teams list

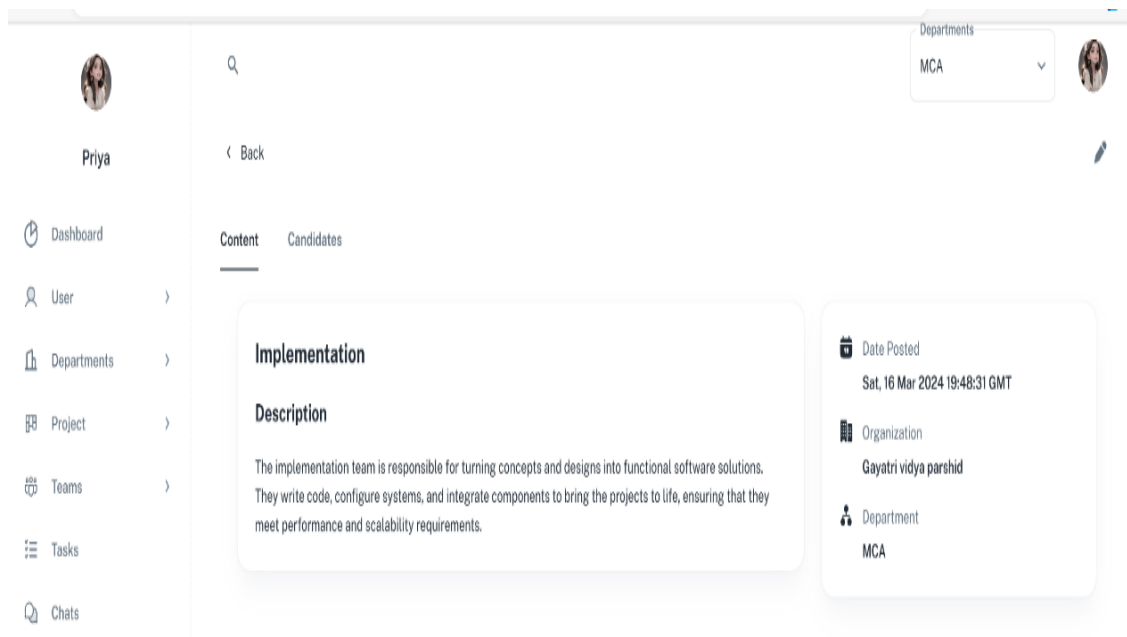


Fig : 7.6.2 Team Details

7.7 Task Page

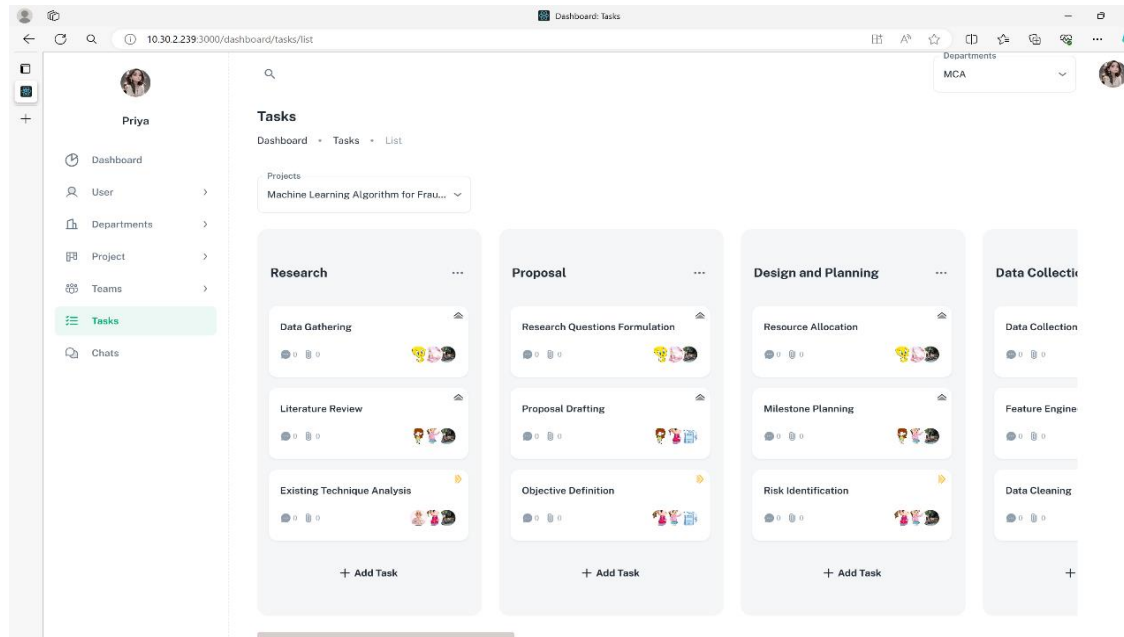


Fig : 7.7.1 Tasks

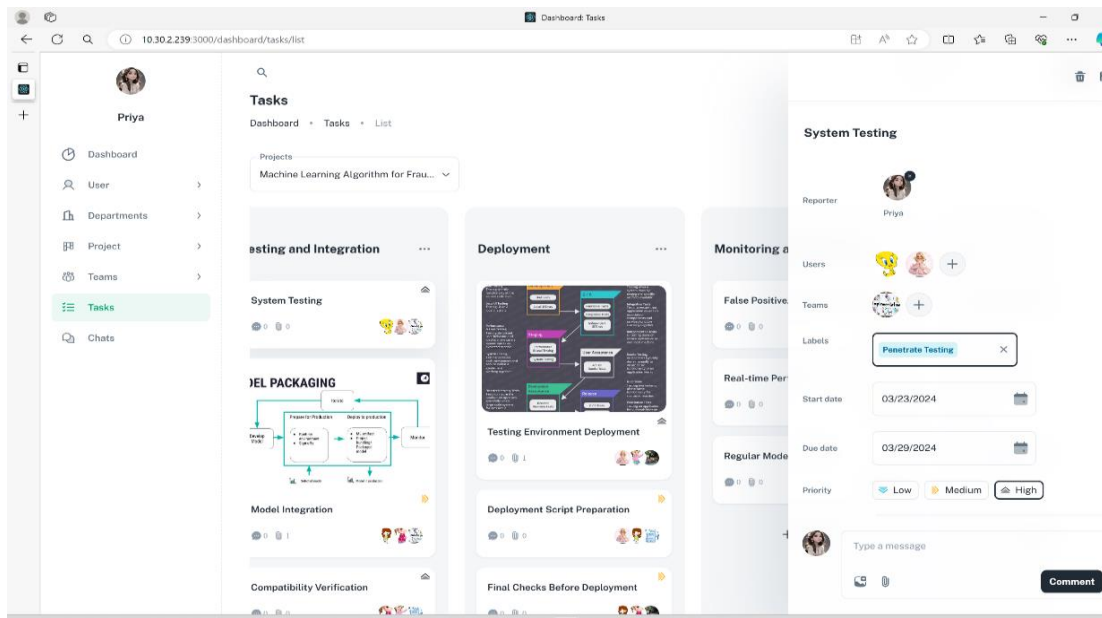


Fig : 7.7.2 Task assigning



Fig : 7.7.3 Assigned new task mail

7.8 Chat Page

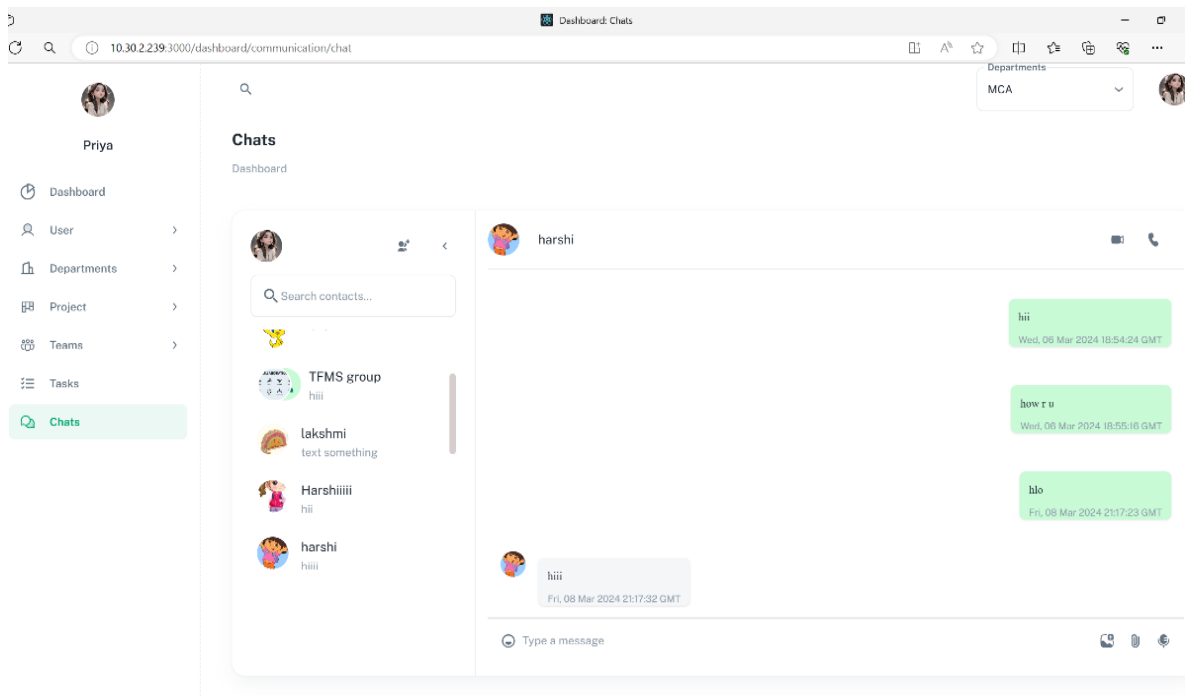


Fig : 7.8.1 Chats

7.9 video conferences page

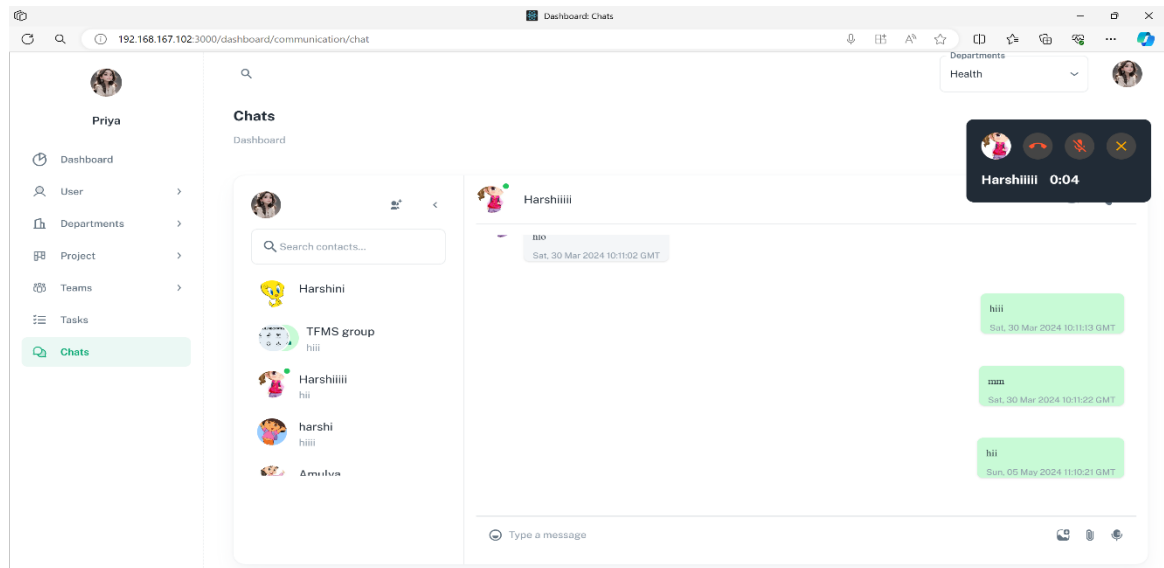


Fig : 7.9.1 Audio calls

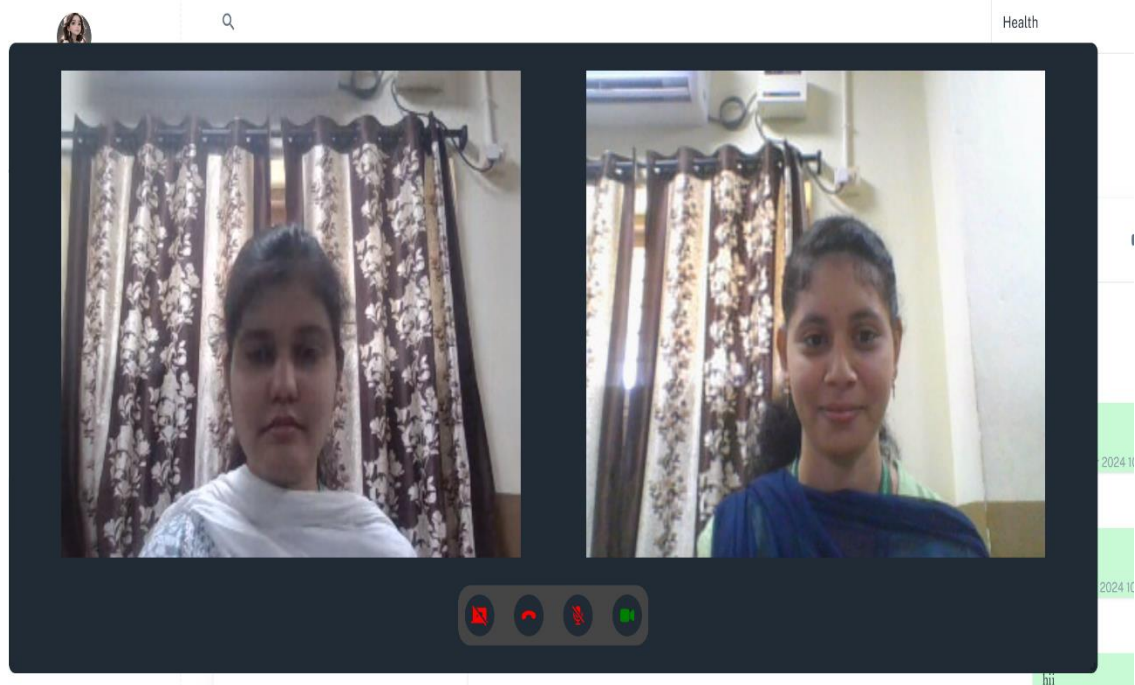


Fig7.9.2 Video calls

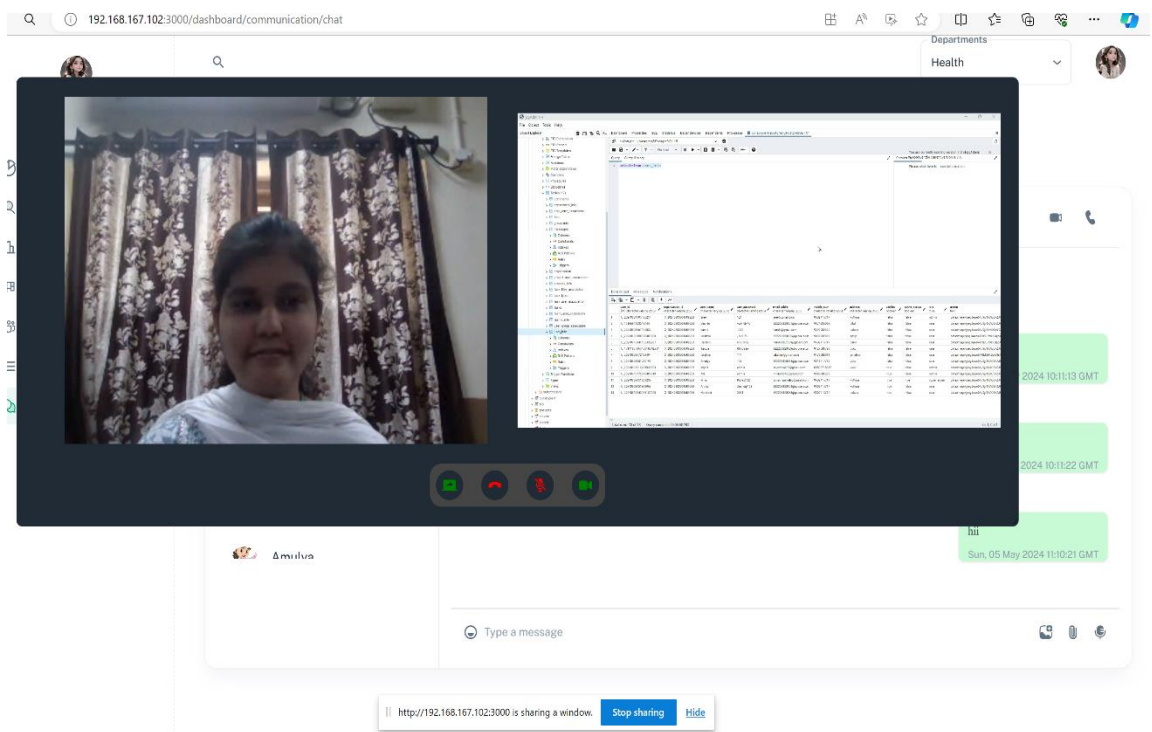


Fig 7.9.3 Video call screen sharing

CHAPTER 8

SYSTEM TESTING

8.1 FUNCTIONAL TESTING

- Functional testing is a form of software testing that concentrates on ensuring that the system or application functions properly and fulfills the functional requirements specified.
- It is an expansive concept that incorporates the assessment of a system's numerous functionalities, features, and interactions.
- Functional testing ensures that the system performs the intended functions, generates the expected outputs, and operates as specified by the requirements.
- It requires the development and execution of test cases to validate the system's functionalities, user flows, and interactions.
- Depending on the testing phase and scope, functional testing encompasses various test categories, including unit testing, integration testing, regression testing, and user acceptance testing.

User Registrations:

- Verify that users can register successfully with valid details.
- Test invalid or missing registration information to ensure appropriate error handling.
- Confirm that unique email addresses are enforced.

User login:

- Test successful login with valid credentials.
- Verify that incorrect or invalid login credentials are rejected.

Departments, Projects and Teams Creation:

- verify that super admins and admins can create with appropriate details, such as title, description and assignes.
- Test for proper validation for not allowing duplicate users and teams are assigned.

Tasks creation, updation and deletion:

- verify that roles with super admins and admins can create, update, reorder, and delete stages.
- Allow roles to create tasks and changes stages.
- Test for proper validation for mandatory details like start date, end date, atleast one reporter and task title.

8.2 TEST CASES

- Test cases are specific sets of steps and conditions that are designed to test a particular aspect or functionality of a software system.
- They are detailed instructions that outline the inputs, expected outputs, and test conditions for a specific test scenario.
- Test cases are created based on the requirements, use cases, and functional specifications of the system.
- They are used as a guide during the testing process to ensure that all necessary tests are executed and to record the actual results for comparison with the expected results.
- Test cases can cover a wide range of aspects, including functional, non functional, and user acceptance testing.

Test Case: User Registration

Description: verify the functionality of user registration.

Steps:

- Navigate to registration page.
- Enter valid details and all required fields
- click on the register button
- Expected result: The user should be successfully registered as super admin, send a users email for welcoming new users and redirect to dashboard.

Test Case: User Login

Description: verify the functionality of user Login.

Steps:

- Navigate to login page.
- Enter valid credentials(email and password).
- click on the login button
- Expected result: The user should be successfully logged with designated role and user information and if role is user will redirect to tasks or redirect to dashboard

Test Case: CRUD operations for Department, Projects, Users and Teams

Description: verify the functionality of create, delete, update for department, projects, Users and teams.

Create and update steps:

- Navigate to create page in respective sections.
- Enter valid details and all required fields.
- click on the create button to create new record or save button to update the fields of existing record.
- Expected result: If the the operation is successfull show success notification and redirect to list page. If unsuccessful show error notification.

Delete steps:

- Navigate to list page in respective sections.
- Click on menu item to delete will be enabled for super admins and roles.
- Click on ok button on delete confirmation dialogue to continue for deletion.

Expected result:

- If the deletion on department then delete all tasks, projects, departments and associated users/teams if operations are successfull show success notification or show error notification with message.
- If the deletion on projects then delete all tasks, projects, and associated users/teams if operations are successfull show success notification or show error notification with message.
- If the deletion on teams then delete all associated users if operations are successfull show success notification or show error notification with message.

Test Case: CRUD operations for Task stages.

Description: verify the functionality of create, delete, update for Task stages.

Create and update steps:

- Navigate to Tasks and select project.
- Click on add stage will be enabled for super admins and roles.
- Enter valid Stage name.
- Listen to enter event to create the new stages.
- Expected result: If the the operation is successfull show success notification. If unsuccessful show error notification.

Delete steps:

- Navigate to Tasks and select project.
- Click on menu item to delete
- Click on ok button on delete confirmation dialogue to continue for deletion.
- Expected result: If the deletion on department then delete all tasks, attachments and associated users/teams if operations are successful show success notification or show error notification with message.

Test Case: CRUD operations for Tasks.

Description: verify the functionality of create, delete, update for Task stages.

Create and update steps:

- Navigate to Tasks and select project.
- Click on add stage will be enabled for super admins and roles.
- Enter valid Stage name.
- Listen to enter event to create the new stages.
- Expected result: If the the operation is successful show success notification and refreshes the page. if unsuccessful show error notification.

Delete steps:

- Navigate to Tasks and select project.
- Click on Task item to view task details
- Click on delete button followed with delete confirmation dialogue to continue for deletion.
- Expected result: If the the operation is successful show success notification and refreshes the page. if unsuccessful show error notification.

Test case: Dashboard

Description: verify the functionality of dashboard calculation.

Steps:

- Navigate to dashboard page.
- select the project to view analysis of dashboard.
- Expected result: The analysis should reflect the individual task and user performance, able to handle null information on calculations.

8.3 TEST CASE STATUS

TestCase ID	TestCase Condition	Expected Output	Actual Output	Status
TC001	User Registration	User should be successfully registered as super admin, email sent for welcoming new users,and redirected to dashboard	User registered successfully, email sent, redirected to dashboard	Passed
TC002	User Login	User should be successfully logged in with designated role, and redirected to tasks or dashboard based on the role	User logged in successfully, redirected to tasks	Passed
TC003	CRUD operations for Department, Projects, Users, and Teams	Creation/Update: Success notification and redirect to list page. Deletion: Success notification or error message	CRUD operations successful, no errors encountered	Passed
TC004	CRUD operations for Task Stages	Creation/Update: Success notification.	CRUD operations successful, no errors	Passed

		Deletion: Success notification or error message	encountered	
TC005	CRUD operations for Tasks	Creation/Update: Success notification and refresh page. Deletion: Success notification or error message	CRUD operations successful, no errors encountered	Passed
TC006	Dashboard	Analysis should reflect individual task and user performance, handle null information on calculations	Dashboard analysis accurate, handles null information	Passed

CHAPTER 9

CONCLUSION

9.1 CONCLUSION

The Task Management system is a web-based application. This system organizes the departments, projects, tasks, users in hierarchal way to provide organization structure. This helps in managing the departments and its projects work independently which removes the complexity of managing users and teams only to work on associated departments or projects only.

The system's intuitive kanban interface allows users to easily track, organize, and prioritize tasks, enhancing productivity and workflow management. Additionally, the analysis capabilities provided, such as tracking users performance, task progress, timeline for task stages and aggregative counts, users, tasks, priorities and stages .

Moreover, the integration of audio, video calling, screensharing, and chatting functionalities facilitates seamless communication and collaboration among team members, further improving efficiency and teamwork.

By offering all these essential features in a single platform, the Task Management system simplifies organizational processes and enables users to connect and organize according to their specific needs. Overall, it serves as a powerful tool for enhancing productivity, project management, and collaboration within the organization.

9.2 FUTURE SCOPE

File sharing: Users can upload and share files with team members directly within the project management application.

Private Sessions: Users can create Private sessions like groups and meetings with specific other users.

Calendar integration: Users can integrate their project management application with their calendar, allowing for better time management and scheduling.

Access management: Users can set access rules over tasks and its dependencies. for providing more control over task which are important for organization and maintaining access based on hierarchy.

Third party integration: Application should integrates closely with Online Storages. It allows users to store and sync files across devices, collaborate on files, and access them within application. Users can integrate their project management application with other tools and applications they use, such as email, calendars, or chat applications, to streamline their workflow and reduce manual work.

9.3 REFERENCES

- [1] VICTOR VELEPUCHA AND PAMELA FLORES, “A Survey on Microservices Architecture: Principles, Patterns and Migration Challenges”, Digital Object Identifier 10.1109/ACCESS.2023.3305687, 15 August 2023.
- [2] Md. Habibur Rahaman, “A Survey on Real-Time Communication for Web”, ISSN 2201-2796, 7 July 2015.
- [3] Zinah Tareq Nayyef, Sarah Faris Amer, Zena Hussain¹, “Peer to Peer Multimedia Real-Time Communication System based on WebRTC Technology”, International Journal of Engineering & Technology, 7 (2.9) (2018) 125-130, February 2019.
- [4] Ms. Archana Nikose, Sakshi Dosani, Shreya Pardhi, Deep Nikode, Anurag Jais, “Real-Time Chat Application”, ISSN: 2395-1990, 15 April 2023.