Task Management System Using React

1.Project Planning & Management

Overview of the Project:

This project aims to develop a Task Management System using React, Node.js, Redux, and MongoDB. The system will allow users to create, assign, track, and manage tasks efficiently. Users will be able to filter and search tasks based on status

Objectives:

1.Develop an intuitive and responsive task management platform.

2.Implement user authentication and role-based access control.

3.Provide real-time task updates using WebSocket.

4.Ensure a seamless user experience with efficient state management (Redux).

5.Optimize performance and security.

Scope:

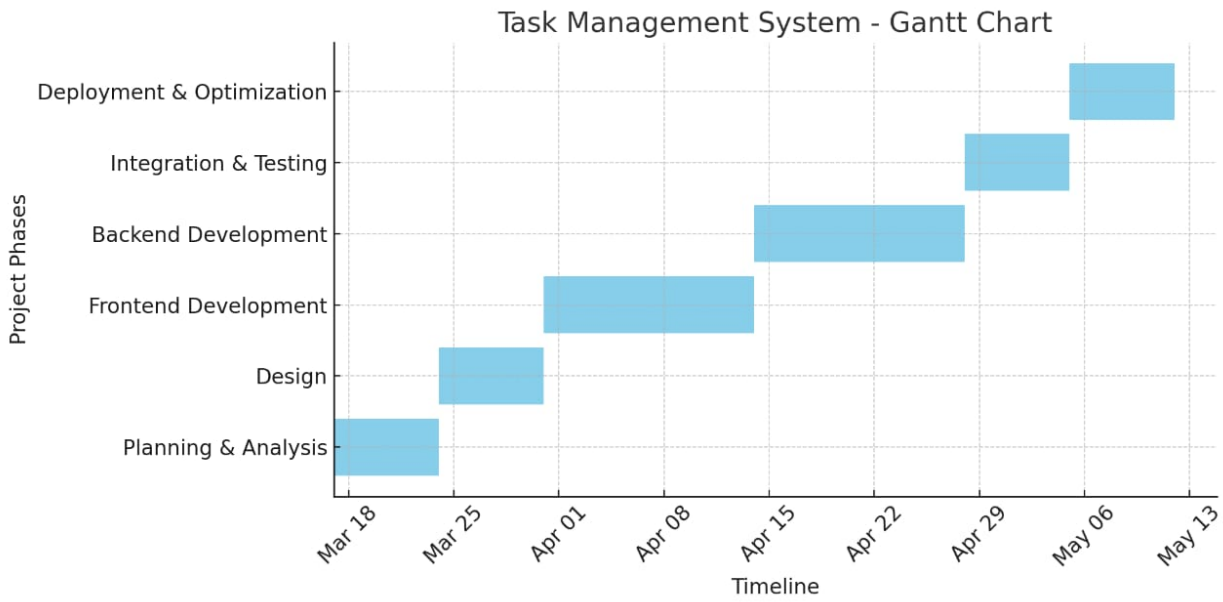
Task Management: Create, edit, delete, assign tasks, set priorities, and deadlines.

Task Tracking: Status updates (To-Do, In Progress, Completed), search, filter, notifications.

Collaboration: Assign tasks to team members, track progress.

2. Project Plan

Timeline (Gantt Chart)



Milestones

Milestone	Description	Deadline
Project Kickoff	Finalize project scope, requirements, and timeline.	Week 1
UI/UX Design Completion	Finish wireframes and database schema.	Week 2
Frontend MVP Ready	Basic UI components and state management in place.	Week 4
Backend MVP Ready	REST APIs, authentication, and database setup completed.	Week 6
System Integration Complete	Frontend and backend fully connected and functional.	Week 7
Final Testing & Bug Fixes	Run unit, integration, and user testing.	Week 8
Project Deployment	Deploy on Vercel (frontend) & Render/Heroku (backend).	Week 8

3. Task Assignment & Roles

Team Member	Role	Responsibilities
Shorok Hassan , Toka Sedik Ahmed Samir , Omar Rabie Alaa Maghraby	Frontend Developer	Develop React components, manage Redux state, oversee project progress.
Shorok Hassan , Toka Sedik Ahmed Samir , Omar Rabie Alaa Maghraby	Backend Developer	Build REST APIs using Node.js & Express.js, handle authentication and database.
Shorok Hassan , Toka Sedik Ahmed Samir , Omar Rabie Alaa Maghraby	Database Administrator	Design, optimize, and manage the MongoDB database.
Shorok Hassan , Toka Sedik Ahmed Samir , Omar Rabie Alaa Maghraby	UI/UX Designer	Create wireframes, design user interface, and ensure responsive design.
Shorok Hassan , Toka Sedik Ahmed Samir , Omar Rabie Alaa Maghraby	QA Tester	Perform unit testing, integration testing, and bug tracking.

4. Risk Assessment & Mitigation Plan

Risk	Likelihood	Impact	Mitigation Strategy
Scope Creep	High	High	Clearly define and document requirements.
Delays in Development	Medium	High	Implement Agile methodology with sprints.
Security Vulnerabilities	High	High	Follow best security practices & testing.
Performance Issues	Medium	Medium	Optimize code and conduct load testing.
Team Member Availability	Medium	High	Maintain backup resources and clear task documentation.

5. Key Performance Indicators (KPIs)

1. System Performance Metrics

Response Time: API response time should be ≤ 200ms for optimal performance.

System Uptime: Ensure 99.9% uptime to maintain availability.

Page Load Speed: Web app should load within 3 seconds.

2. Security & Reliability

Error Rate: Number of failed API calls should be < 2%.

Authentication Success Rate: Ensure at least 98% successful logins without issues.

3. Development & Deployment Efficiency

Code Quality Score: Maintain a high-quality codebase with low bug density.

Issue Resolution Time: Bugs should be fixed within 48 hours of reporting.

Deployment Success Rate: Successful deployment with minimal rollbacks.

2.Literature Review

1. Feedback & Evaluation

The system will include a frontend built with React.js and Redux for state management. The backend will utilize Node.js and Express for handling API requests, with MongoDB as the database solution. The application will support user authentication, task prioritization, and advanced filtering options.

2. Suggested Improvements

-
-
-
-
-

3. Final Grading Criteria

Criteria	Percentage (%)
Documentation	20%
Implementation	40%
Testing	20%
Presentation	20%

3. Requirements Gathering

1. Stakeholder Analysis

Stakeholder	Role	Needs & Expectations
Project Team	Developers, Designers, Testers	Clear requirements, well-defined tasks, collaboration tools.
End Users	Team members, project managers	User-friendly UI, task tracking, notifications, mobile compatibility.
Client/Product Owner	Oversees project development	Functional system, timely delivery, security, scalability.
QA & Testing Team	Ensures system quality	Bug-free application, effective test cases, performance stability.
Deployment Team	Handles hosting & maintenance	Smooth deployment, system uptime, security compliance.

2. User Stories & Use Cases

- As a user, I want to create tasks so that I can organize my work.

- As a user, I want to set deadlines and priorities so that I can manage my workload efficiently.

- As a manager, I want to assign tasks to team members so that work is distributed effectively.

- As an administrator, I want to generate reports so that I can analyze system usage.

- Create Task: A team member creates a task with a title, description, and deadline.

- Assign Task: A project manager assigns a task to a specific team member.

- Update Task Status: A team member updates the task status (To Do, In Progress, Done).

- Receive Notifications: The system sends alerts for task updates and deadlines.

3. Functional Requirements

- User authentication (Sign-up, Login, Logout).

- Task creation, editing, and deletion.

- Task assignment and status tracking.

- Real-time notifications for task updates.

- Dashboard with task analytics.

- Role-based access control.

4. Non-functional Requirements

- The system should have 99.9% uptime.

- Response time should be less than 200ms for most operations.

- User-friendly and intuitive UI/UX.

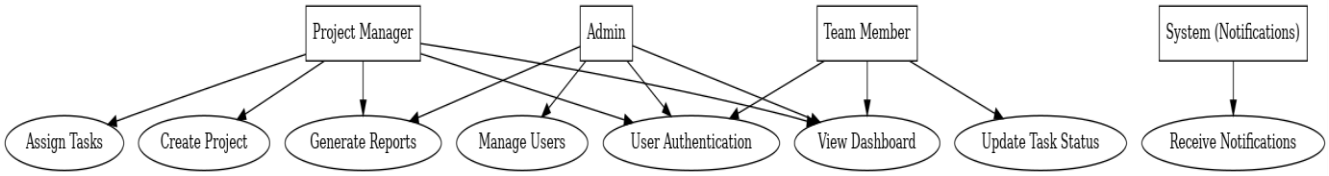
- Data encryption and secure authentication.

- Scalable to support up to 10,000 users.

- Cross-platform compatibility (Desktop & Mobile).

4. System Analysis & Design

1. Case Diagram & Descriptions



Use Case	Actors Involved	Description
User Authentication	Admin, Project Manager, Team Member	Users log in or register using credentials.
Manage Users	Admin	Admin can create, update, or delete user accounts.
Create Project	Project Manager	The manager creates a new project and defines its scope.
Assign Tasks	Project Manager	Assigns tasks to team members with deadlines.
Update Task Status	Team Member	Updates the progress (To-Do, In Progress, Completed).
View Dashboard	All Users	Displays project progress, deadlines, and notifications.
Receive Notifications	System, All Users	Sends email or in-app reminders for deadlines.
Generate Reports	Admin, Project Manager	Provides analytics on project progress and team performance.

2. Functional & Non-Functional Requirements

Functional Requirements (FRs)

- The system must allow user authentication and role-based access.

- The project manager should be able to create and assign tasks.

- Team members should be able to update task statuses.

- The system should send automated notifications for deadlines.

- Users should be able to view a dashboard summarizing their tasks.

- Admins should have the ability to manage users and system settings.

- The system must support real-time collaboration and comments.

- Reports and analytics should be generated for performance tracking.

Non-Functional Requirements (NFRs)

- Scalability: The system should support multiple teams and projects efficiently.

- Performance: Task updates and notifications should be processed in real-time.

- Security: User authentication and role-based access control must be enforced.

- Usability: The UI should be intuitive and responsive across devices.

- Availability: The system should ensure 99.9% uptime for continuous access.

3. Software Architecture

The Task Management System follows a Microservices-based architecture using the MERN stack (MongoDB, Express, React, Node.js) with Redux for state management.

Architecture Components

Frontend (React + Redux)

- User Interface (UI)
- Task Dashboard
- Authentication

Backend (Node.js + Express)

- RESTful API to handle requests
- Task, Project, User management
- Authentication & Authorization (JWT)

Database (MongoDB)

- Stores users, projects, tasks, comments, and notifications

Notification Service (Microservice)

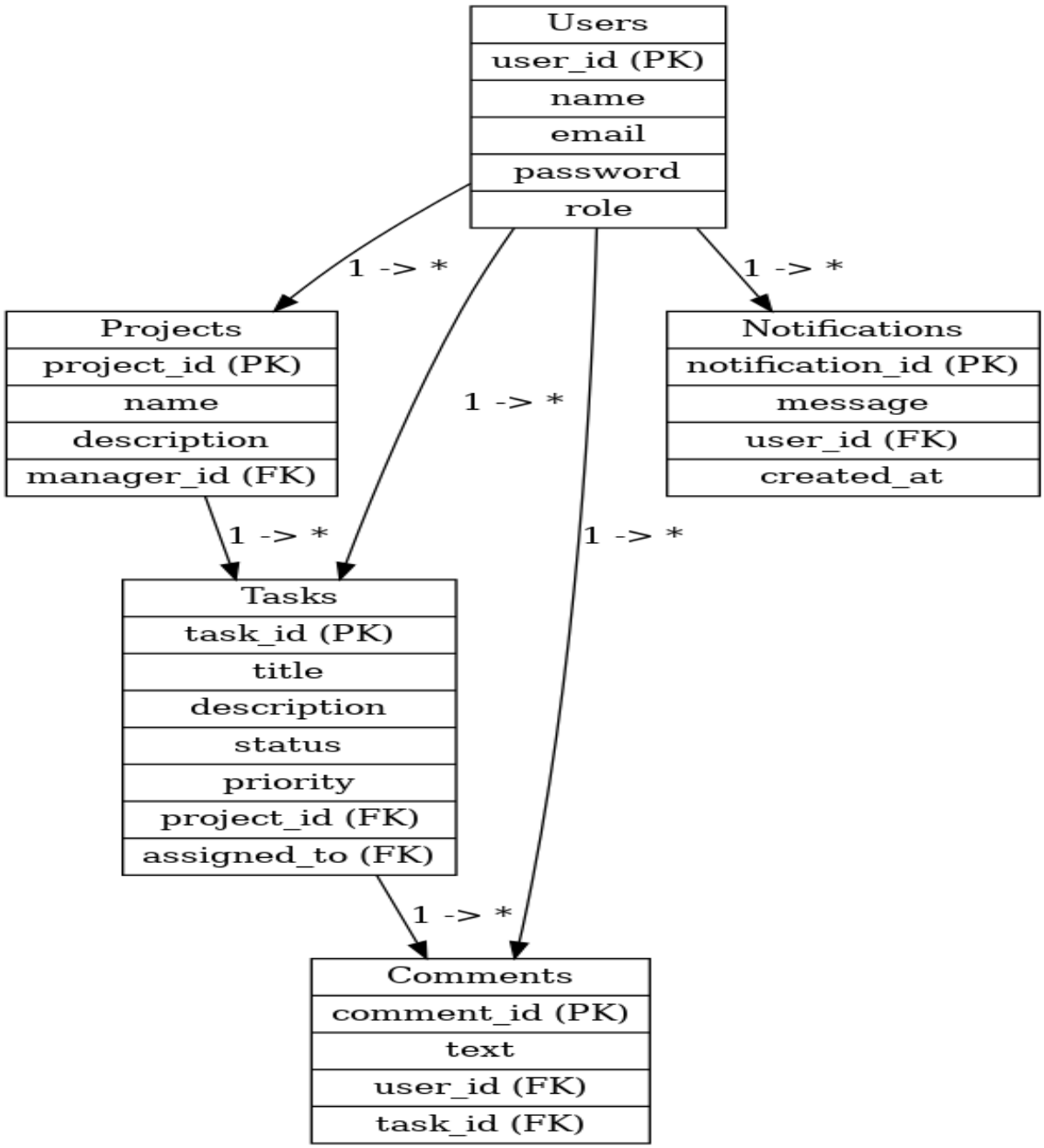
- Manages email & push notifications for deadlines

Integration & Deployment

- Hosted on AWS/GCP
- CI/CD with GitHub Actions
- Docker containers for microservices

2. Database Design & Data Modeling

1. ER Diagram (Entity-Relationship Diagram)



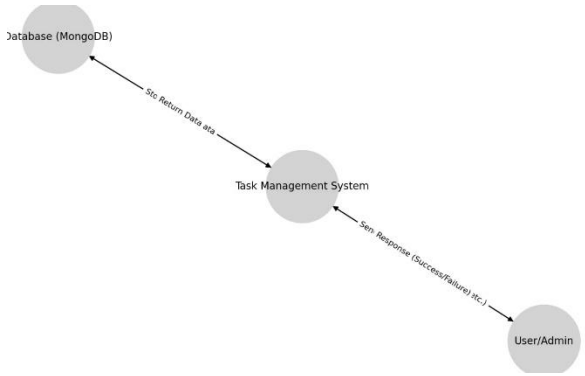
2. Logical & Physical Schema

Tables & Attributes

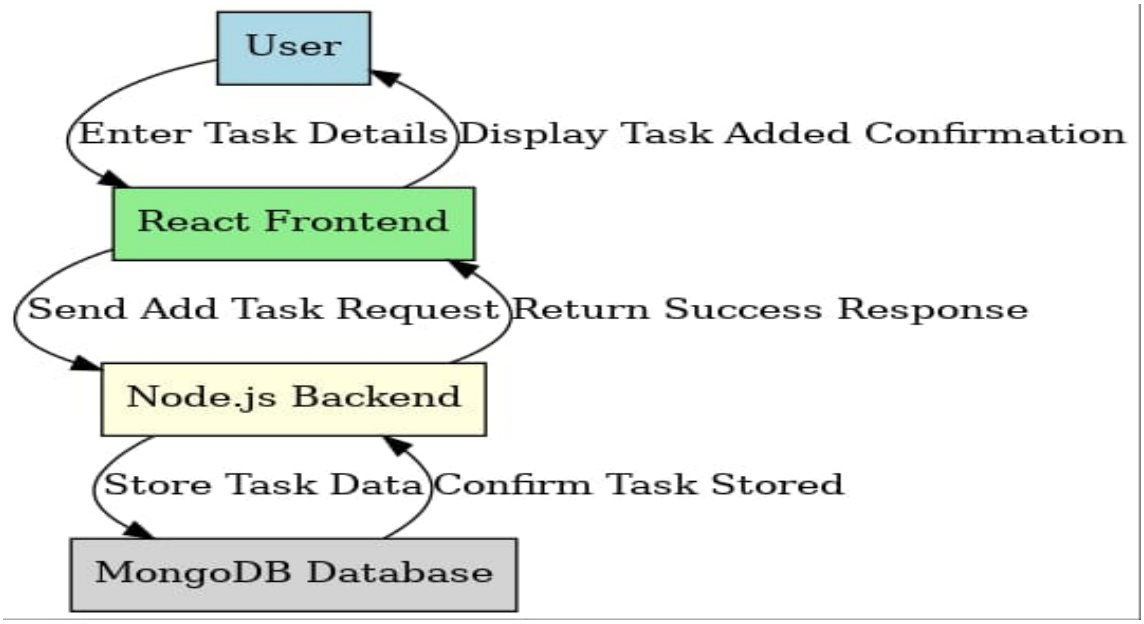
Table Name	Attributes
Users	user_id (PK) name email (Unique) password role
Projects	project_id (PK) name description manager_id (FK) -> Users
Tasks	task_id (PK) title description status priority project_id (FK) -> Projects assigned_to (FK) -> Users
Comments	comment_id (PK) text user_id (FK) -> Users task_id (FK) -> Tasks
Notifications	notification_id (PK) message user_id (FK) -> Users created_at

3. Data Flow & System Behavior

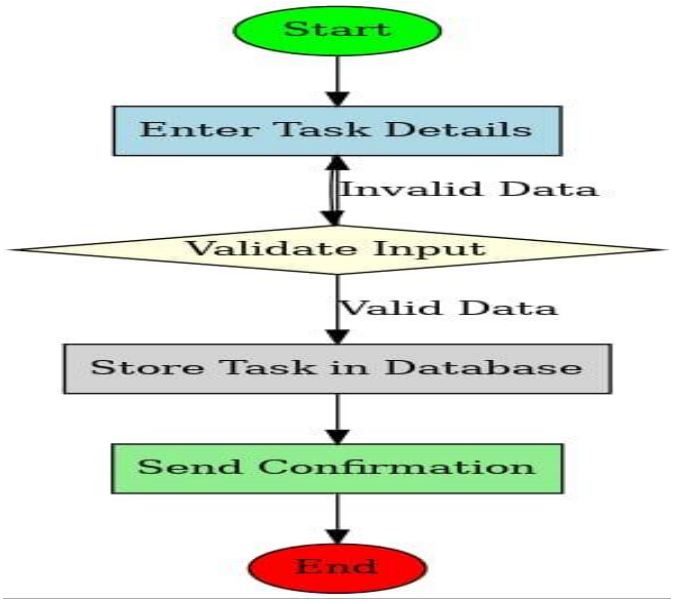
1.DFD (Data Flow Diagram)



2. Sequence Diagram

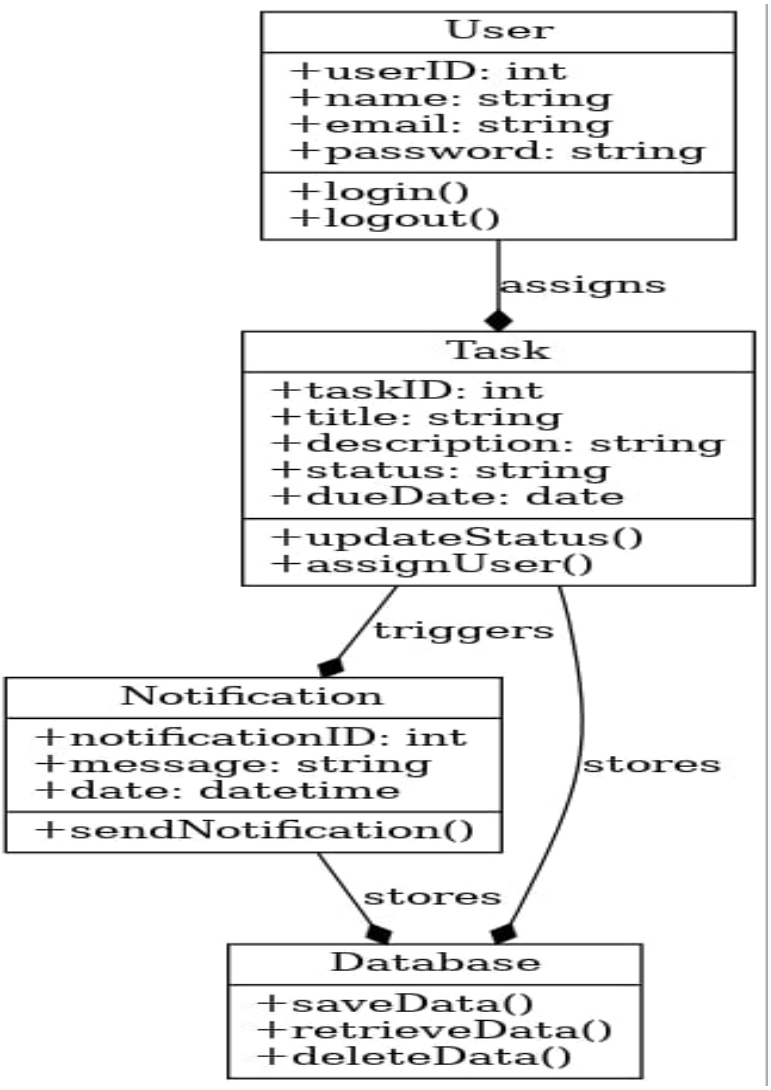


3. Activity Diagram

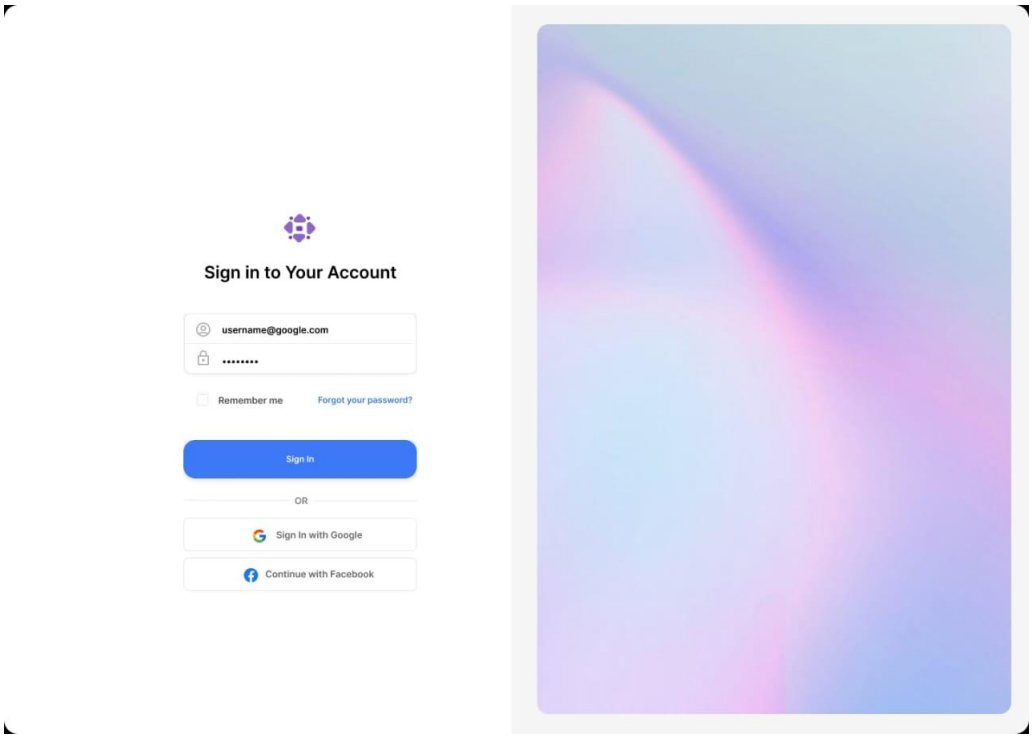


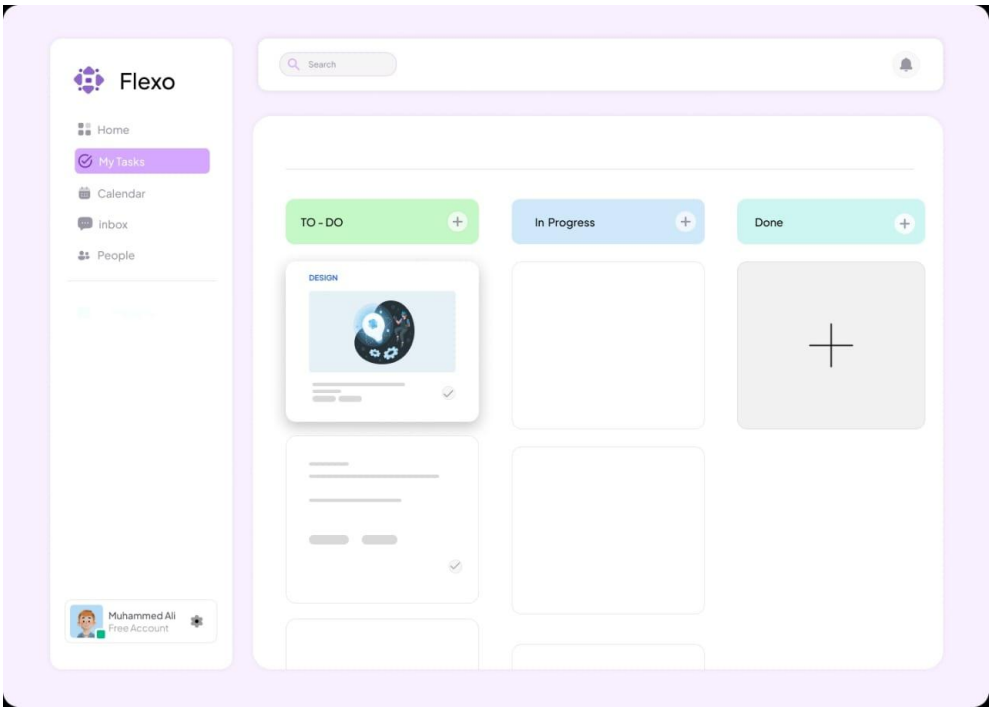
4. State Diagram

5. Class Diagram



4. UI / UX Design & Prototyping





5. System deployment & Integration

1. Technology Stack

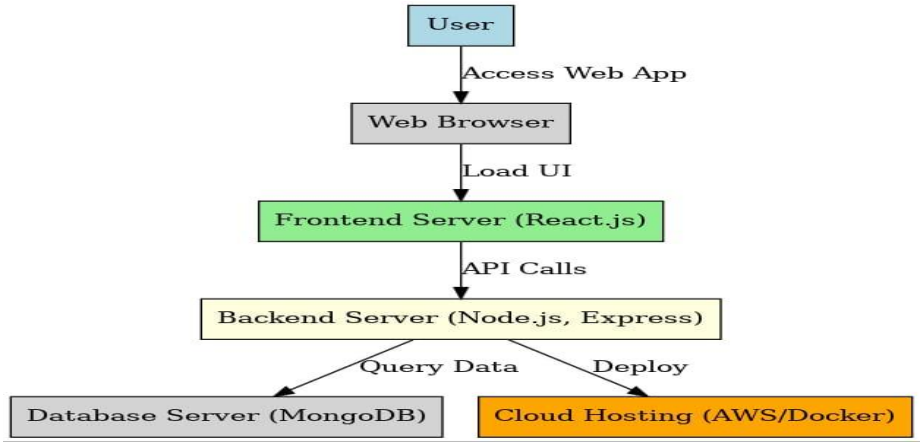
Frontend: React.js, Redux, Bootstrap

Backend: Node.js, Express.js

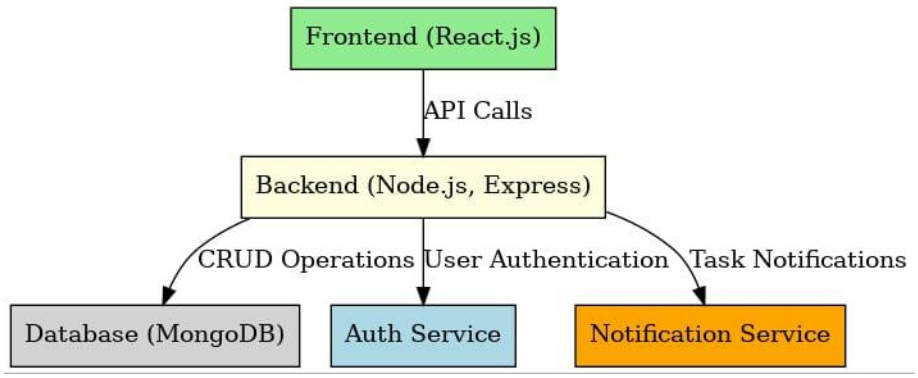
Database: MongoDB

Deployment: Docker, Nginx, AWS

2. Deployment Diagram



3. Component Diagram



6. Additional Deliverables

2. Testing & Validation

A. Unit Testing

Unit tests are conducted using Jest and Mocha to ensure individual components function correctly.

B. Integration Testing

Integration testing ensures proper interaction between the Frontend and Backend using

Postman and Supertest.

C. User Acceptance Testing (UAT)

User testing is conducted to ensure the system meets user expectations and requirements.

3. Deployment Strategy

A. Hosting Environment

The system is hosted on AWS using EC2 and Docker, with the database managed via MongoDB Atlas.

B. Deployment Pipelines

Deployment is automated using GitHub Actions to ensure clean and tested code before deployment.

C. Scaling Considerations

Load balancers and auto-scaling groups are used to maintain system stability as user traffic increases.

https://github.com/a7med741911/Task-Management-System.git