

**IT 309 SOFTWARE ENGINEERING**

PROJECT DOCUMENTATION

PROJECT NAME

Prepared by:

**Kerim Senderovic**

**Rijad Basic**

Proposed to:

**Nermina Durmić, Assist. Prof. Dr.**

**Lejla Brescic, Teaching Assistant**

**Ajla Korman, Teaching Assistant**

**TABLE OF CONTENTS**

1. Introduction ……………………………………………………………………..
   1. About the project ……………………………………………………………..
   2. High-Level plan ……………………………………………………………….
   3. Goal-Oriented Roadmap …………………………………………………..
   4. Release plan …………………………………………………………………….
   5. Project requirements………………………………………………………
   6. User stories……………………………………………………………………..
   7. UML diagrams………………………………………………………………….
   8. Activity diagram……………………………………………………………….
   9. class diagram……………………………………………………………………
   10. Sequence diagram…………………………………………………………….

2. Project Structure………………………………………………

2.1. Technologies……………………………………………………

2.2. Architectural Pattern………………………………………….

2.3. Database Entities……………………………………………….

2.4. Design Patterns…………………………………………………

2.5. Project Functionalities and Screenshots……………………….

2.6. Tests…………………………………………………………….

3. Conclusion……………………………………………………..

4. Individual Contributions……………………………………….

# Introduction

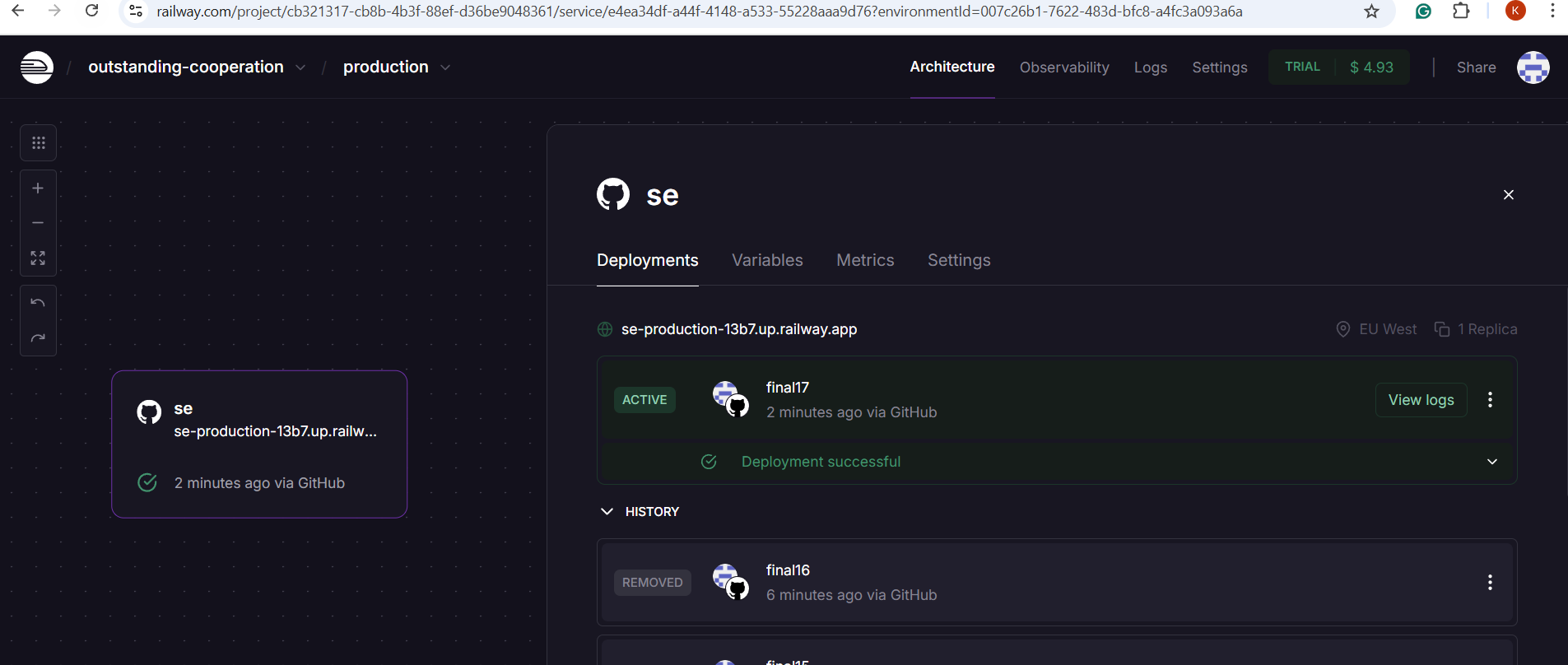
## The Task Management System is a productivity application designed to help individuals and teams organize, track, and manage their tasks efficiently. It provides a structured environment for creating tasks, assigning responsibilities, setting deadlines, and monitoring progress. The system aims to improve workflow, enhance collaboration, and ensure timely completion of work. Whether used for personal projects or team-based assignments, the Task Management System offers essential features such as task categorization, status tracking, and user role management to streamline daily operations and boost productivity.

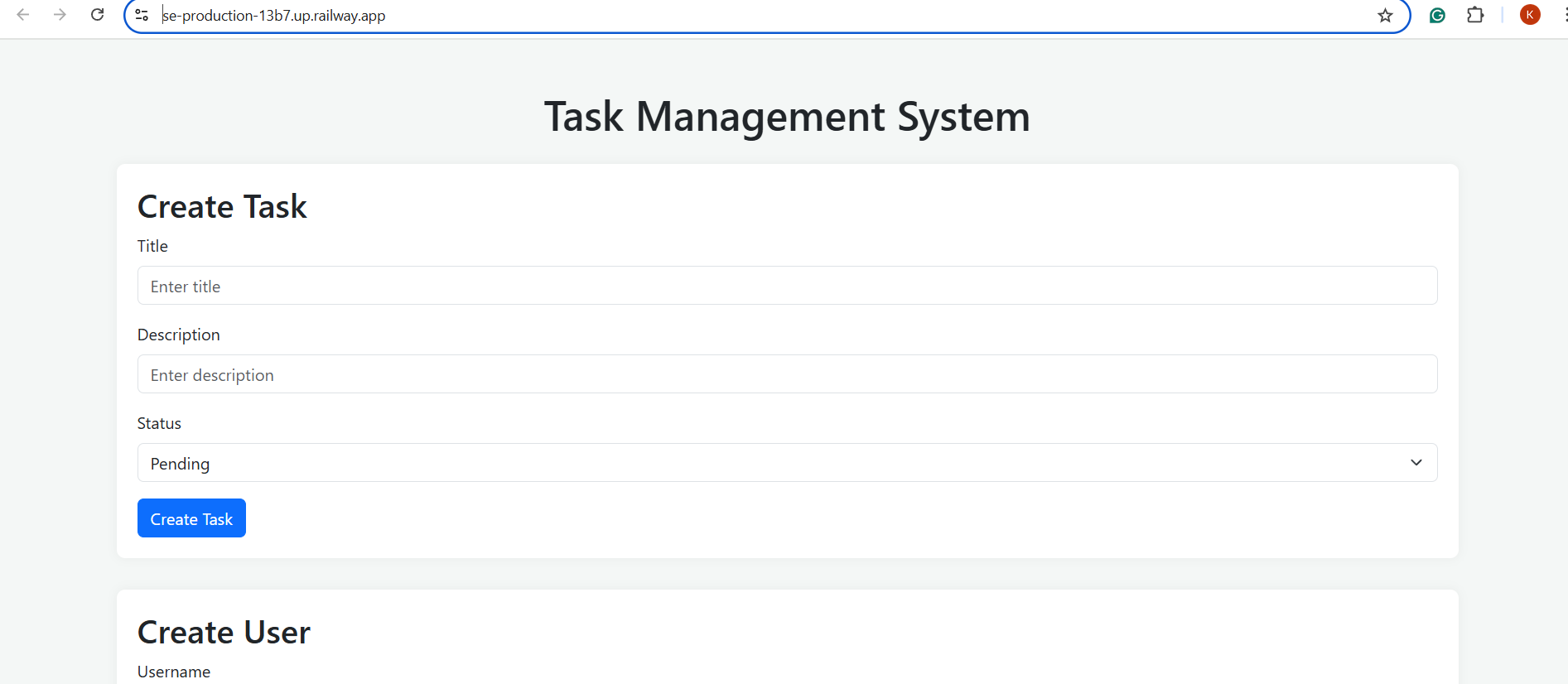
## https://github.com/kerimsenderovic/se.git

## 1.1. About the Project

## This project is a Task Management System built using Node.js (Express) for the backend and HTML, CSS, JavaScript, and Bootstrap for the frontend. It allows users to create, view, update, and delete tasks, manage projects, and assign tasks to specific users. The application supports user authentication, project-level task organization, and a responsive interface for seamless usability across devices. It is designed to improve personal or team productivity through organized task tracking and project coordination.

## ^ [https://se-production-13b7.up.railway.app/] DEPLOYED USING RAILWAY

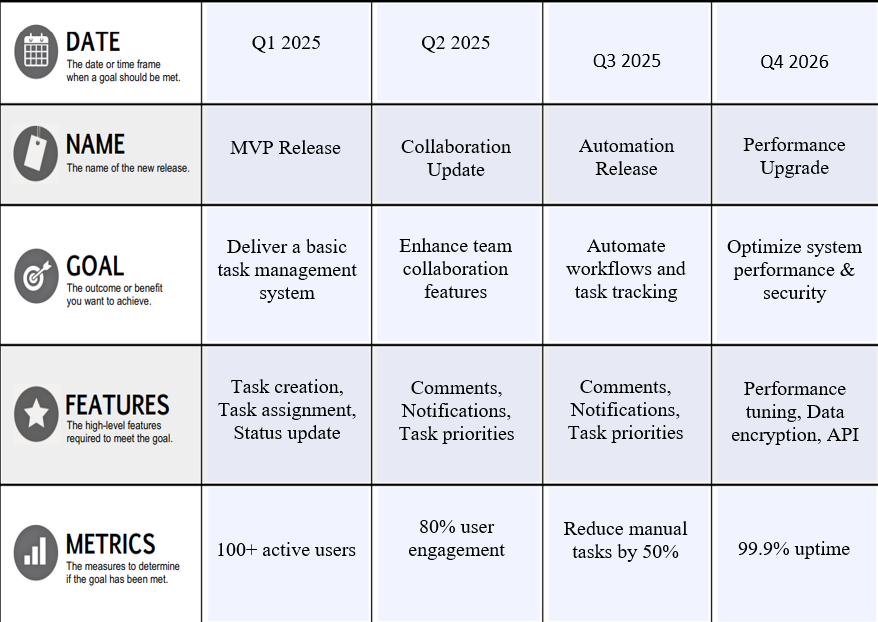




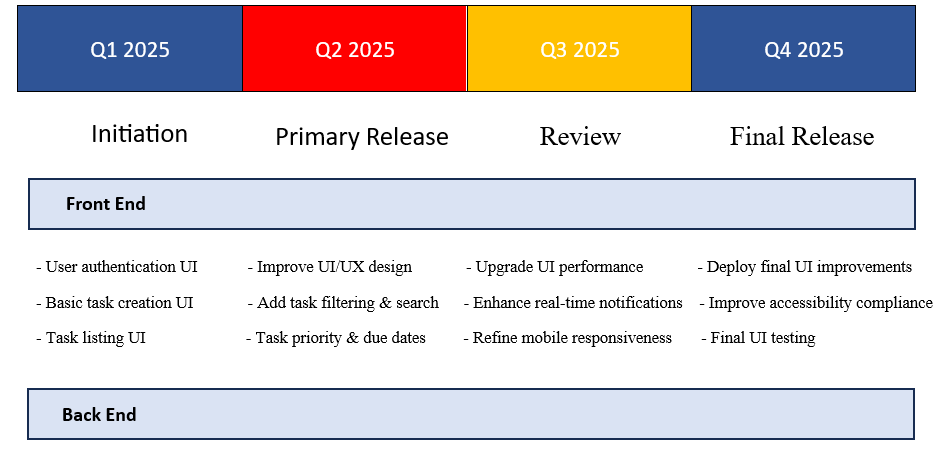


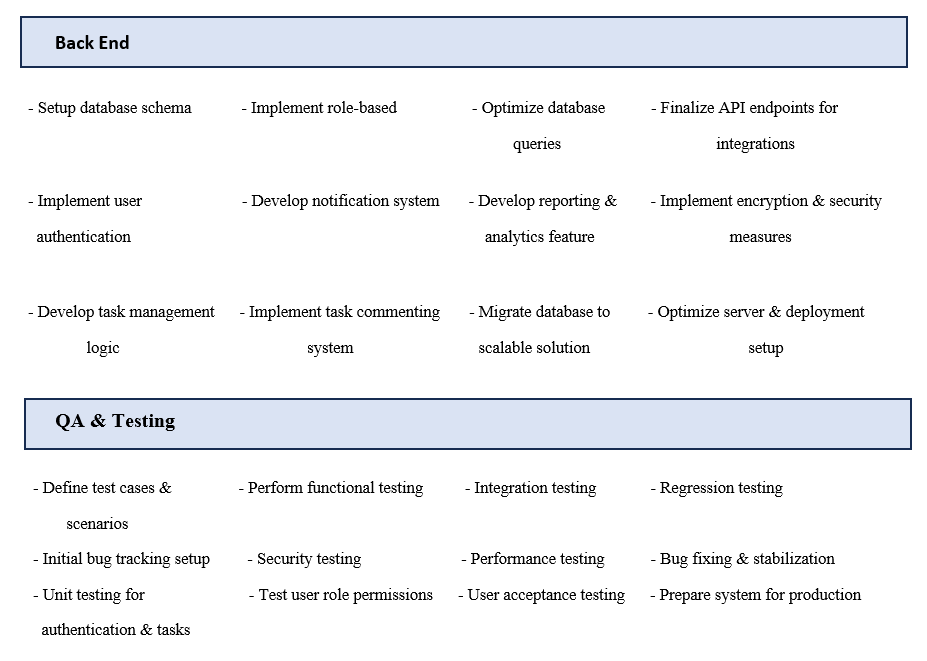
## 1.2. High-level Plan

1.3 Goal-Oriented RoadMap



1.4 Release Plan





## Project Requirements

**1.6 User Stories**

**Theme 1: User Account Management (Epic 1)**

1. **User Story:** As an unregistered user, I want to create an account so that I can save my tasks and access them later.
   * **Acceptance Criteria:**
     + A "Create Account" button will be visible on the homepage and login page. The system validates email format and password strength (min. 6 characters).
     + Clicking the "Create Account" button redirects the user to the registration page.
     + On the registration page, there will be input fields for name, email, and password.
     + The email input will be validated to ensure it follows the correct email format (e.g., [user@example.com](mailto:user@example.com)).
     + The password input will enforce a minimum of 6 characters and will check for strength (including a mix of letters, numbers, and special characters).
     + After submitting the form, a verification email will be sent to the user's email address.
     + Clicking the verification link in the email will activate the account and redirect the user to the login page.
2. **User Story:** As a registered user, I want to log into my account so that I can manage my tasks.
   * **Acceptance Criteria:**
     + The "Login" button will be displayed on the homepage, login, and account pages.
     + Clicking the "Login" button will redirect the user to the login page.
     + On the login page, there will be email and password inputs with a "Login" button below them.
     + The user is required to fill in both inputs before logging in.
     + When the user clicks "Login," the system will check if the email exists in the database.
     + If the email exists, the system will verify if the password matches.
     + If both are correct, the user will be redirected to the Home Feed page.
     + If incorrect, an alert message ("Wrong email address or password") will be displayed, and the user will not be logged in.
     + After 5 failed login attempts, the system will lock the account and display a message "Too many attempts. Please try again later."
3. **User Story:** As a logged-in user, I want to log out so that my session is securely ended.
   * **Acceptance Criteria:**
     + A "Logout" button will be displayed on the top right of the home and account pages.
     + Clicking "Logout" will end the user's session.
     + The user will be redirected to the login page.
     + All session data will be cleared to prevent unauthorized access.
4. **User Story:** As a logged-in user, I want to update my profile details so that my information remains up-to-date.
   * **Acceptance Criteria:**
     + The "My Account" button will be visible on the top right of the application..
     + Clicking on "My Account" will redirect the user to their account page.

4

* + - On the account page, there will be an option to view and edit the profile.
    - The profile page will have fields for name, email, and password.
    - After making changes, the user can click "Save" to update their profile.
    - Changes will be validated before being saved (email format and password strength).
    - Upon successful update, a confirmation message "Profile updated successfully" will be displayed.

1. **User Story:** As a logged-in user, I want to delete my account so that I can remove all my data permanently.
   * **Acceptance Criteria:**
     + The "Delete Account" button will be visible on the account settings page.
     + Clicking "Delete Account" will prompt the user to confirm their decision with a message: "Are you sure you want to delete your account?".
     + If the user confirms, all their data (tasks, profile) will be permanently deleted from the database.
     + After account deletion, the user will be redirected to the home page with a message: "Your account has been deleted."

**Theme 2: Task Management (Epic 2)**

1. **User Story:** As a user, I want to create a new task so that I can track my work.
   * **Acceptance Criteria:**
     + A "New Task" button will be visible on the task dashboard and profile page.
     + Clicking the "New Task" button will open a form with fields for title, description, due date, priority, and category.
     + The user will be able to enter a task title (mandatory), description (optional), select a due date (date picker), set priority (High, Medium, Low), and choose a category (Work, Personal, etc.).
     + Clicking "Save" will store the task in the database and display it on the task list.
2. **User Story:** As a user, I want to view my list of tasks so that I can keep track of my workload.
   * **Acceptance Criteria:**
     + A task list displays task title, due date, priority, and status.
     + Each task will have buttons to edit or delete it.
     + Overdue tasks will be highlighted in red.
3. **User Story:** As a user, I want to edit a task so that I can update its details.
   * **Acceptance Criteria:**
     + Clicking a task from the task list will open an edit form.
     + The user can update the task's title, description, due date, priority, or category.
     + Changes will be saved to the database once the user clicks "Save."
4. **User Story:** As a user, I want to delete a task so that I can remove unnecessary tasks.
   * **Acceptance Criteria:**
     + A "Delete" button will be visible on each task in the task list.
     + Clicking "Delete" will prompt the user for confirmation: "Are you sure you want to delete this task?"
     + Once confirmed, the task will be removed from the database and the list.
5. **User Story:** As a user, I want to mark a task as completed so that I can track my progress.
   * **Acceptance Criteria:**

5

* + - Clicking a checkbox updates the task status to "Completed".

1. **User Story:** As a user, I want to set task priorities so that I can focus on important tasks first.
   * **Acceptance Criteria:**
     + Task priority options: High, Medium, Low.
     + Tasks are color-coded based on priority.
2. **User Story:** As a user, I want to categorize tasks so that I can organize them better.
   * **Acceptance Criteria:**
     + Users can assign tasks to categories (Work, Personal, etc.).
3. **User Story:** As a user, I want to set task deadlines so that I can complete them on time.
   * **Acceptance Criteria:**
     + A date picker allows users to set deadlines.
     + Overdue tasks are highlighted.
4. **User Story:** As a user, I want to receive reminders for upcoming deadlines so that I don’t miss important tasks.
   * **Acceptance Criteria:**
     + Users receive email or app notifications before deadlines.
5. **User Story:** As a user, I want to assign tasks to other users so that we can collaborate.
   * **Acceptance Criteria:**
     + A task can be assigned to a registered user via email.
     + The assignee receives a notification.

**Theme 3: Team Collaboration & Reporting (Epic 3)**

1. **User Story:** As a user, I want to add comments to tasks so that I can communicate with my team.

6

* + **Acceptance Criteria:**
    - Each task will have a comment section where users can enter and submit comments.
    - Comments will be timestamped and displayed with the user's name.
    - Comments will be stored in the database and associated with the specific task.

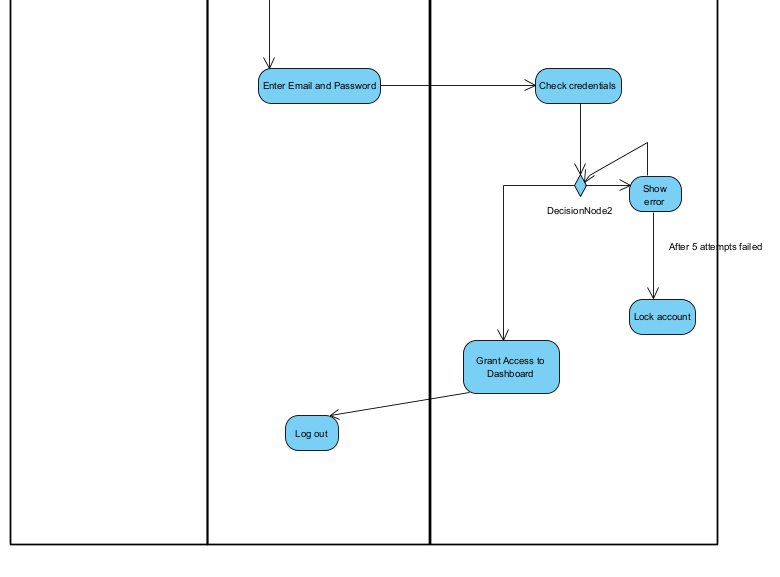
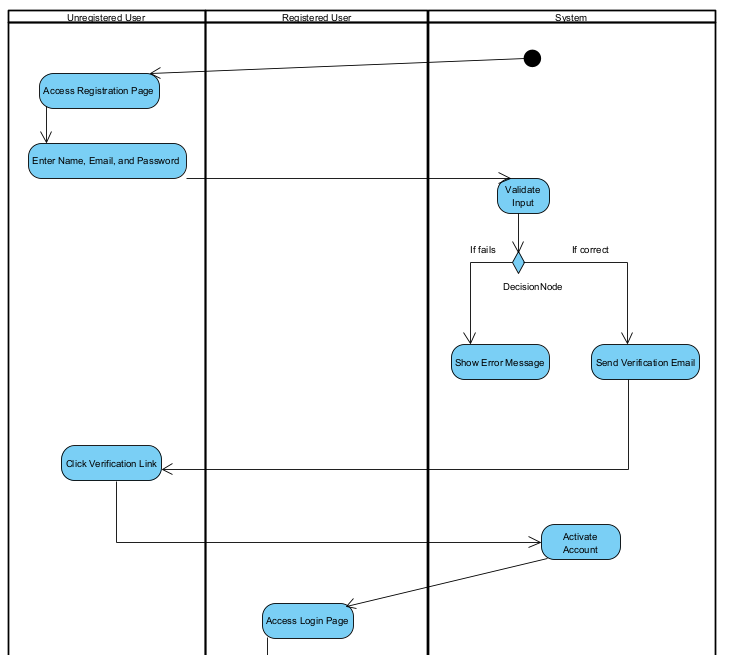
1. **User Story:** As a user, I want to attach files to tasks so that I can share important documents.
   * **Acceptance Criteria:**
     + An "Attach File" button will be visible on the task form.
     + Clicking the button will allow the user to upload files (PDFs, images, etc.).
     + Uploaded files will be displayed as downloadable links within the task details.
2. **User Story:** As a user, I want to view tasks assigned to me so that I can focus on my responsibilities.
   * **Acceptance Criteria:**
     + A dedicated section shows tasks assigned to the logged-in user.
3. **User Story:** As a user, I want to filter tasks by priority, category, and due date so that I can organize my work.
   * **Acceptance Criteria:**
     + Task list includes filter and sort options.
4. **User Story:** As a user, I want to search for tasks so that I can quickly find specific ones.
   * **Acceptance Criteria:**
     + A search bar allows users to enter keywords to find tasks.
5. **User Story:** As an admin, I want to create user roles so that I can manage permissions.
   * **Acceptance Criteria:**
     + Roles: Admin, Member, Guest.
     + Admins can add and remove users.
6. **User Story:** As a team lead, I want to generate task completion reports so that I can analyze productivity.
   * **Acceptance Criteria:**
     + A "Generate Report" button creates a PDF or CSV file of completed tasks.

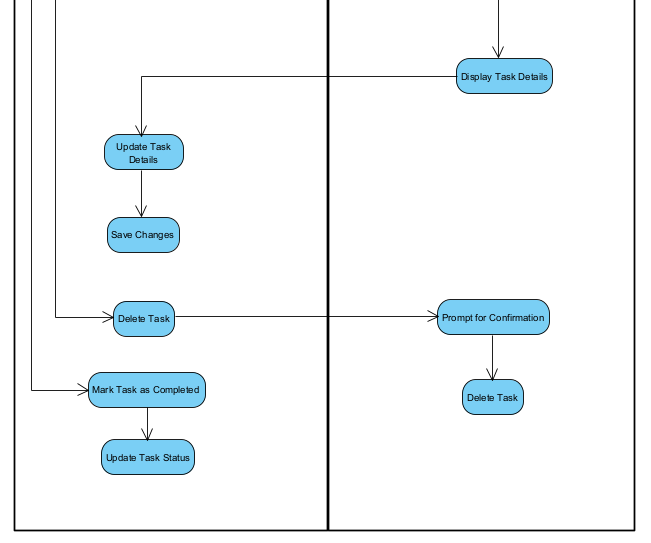
7

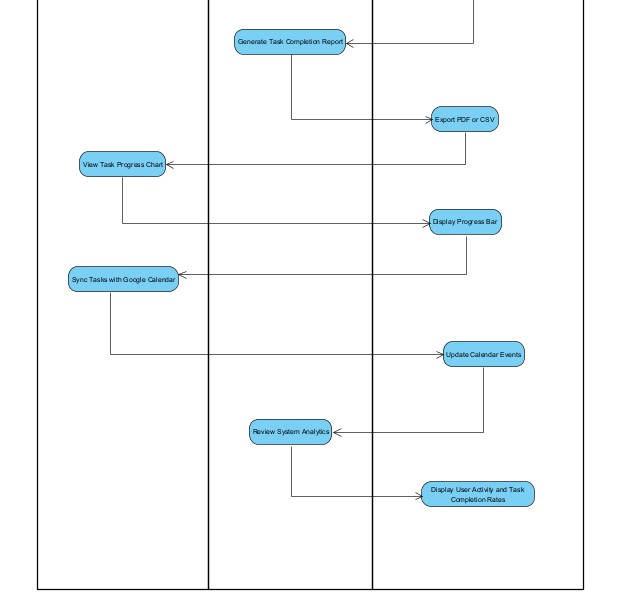
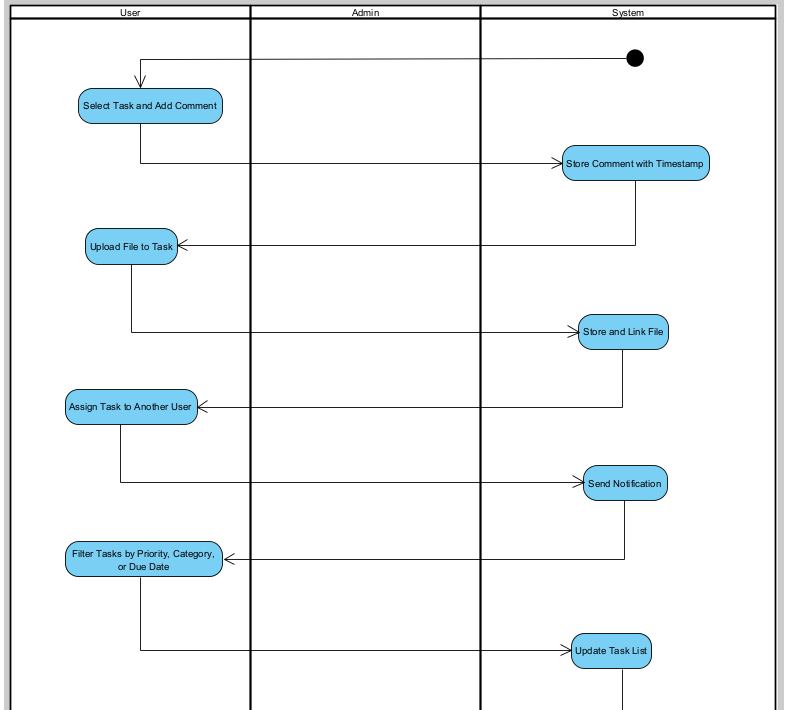
1. **User Story:** As a user, I want to view a task progress chart so that I can see my progress visually.
   * **Acceptance Criteria:**
     + A dashboard shows a progress bar for completed tasks.
2. **User Story:** As a user, I want to integrate my tasks with Google Calendar so that I can sync deadlines.
   * **Acceptance Criteria:**
     + Clicking "Sync with Calendar" adds tasks to the user’s Google Calendar.
3. **User Story:** As an admin, I want to receive system usage analytics so that I can monitor performance.
   * **Acceptance Criteria:**
     + An analytics dashboard displays user activity and task completion rates.

## 1.7. UML diagrams

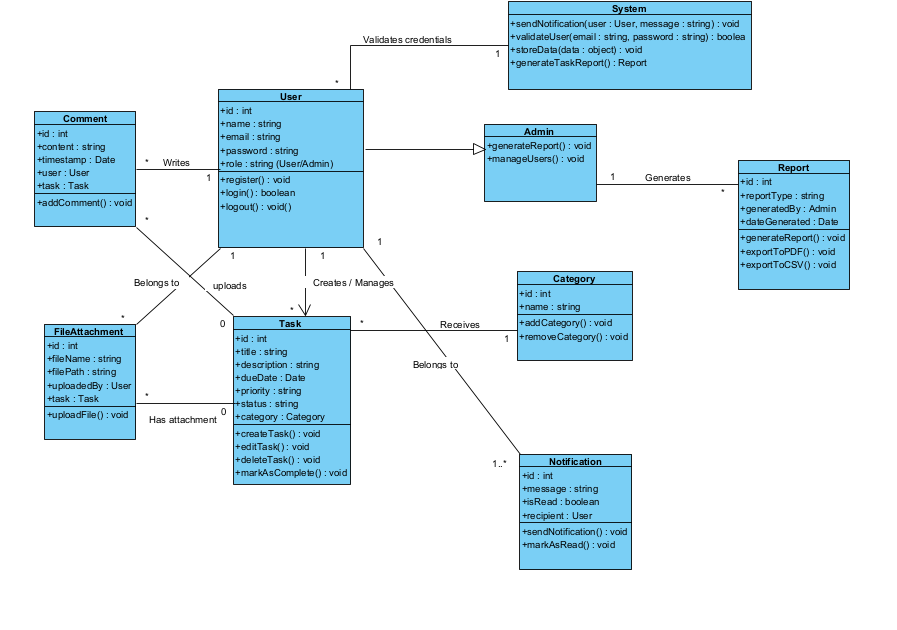
**1.8Activity Diagram**







**1.9 Class Diagram**



1. Class Diagram 1: User Registration & Authentication(System)

Interacts with

User --------------🡪System

1. Class Diagram 2: Task Creation & Management

Creates/Manages

Has attachment

Belongs to

User ------ 🡪Task------🡪 Category ------- 🡪 FileAttachment

1. Class Diagram 3: Team Collaboration & Reporting

Generates

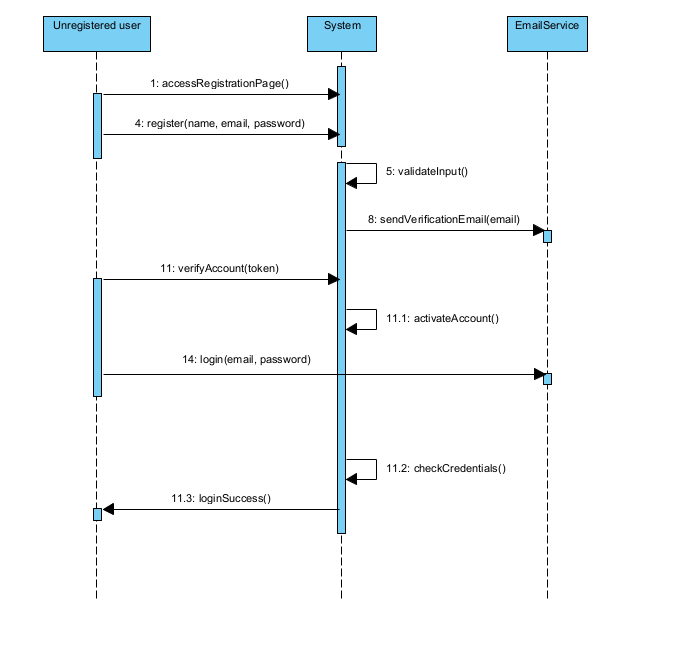
Generates

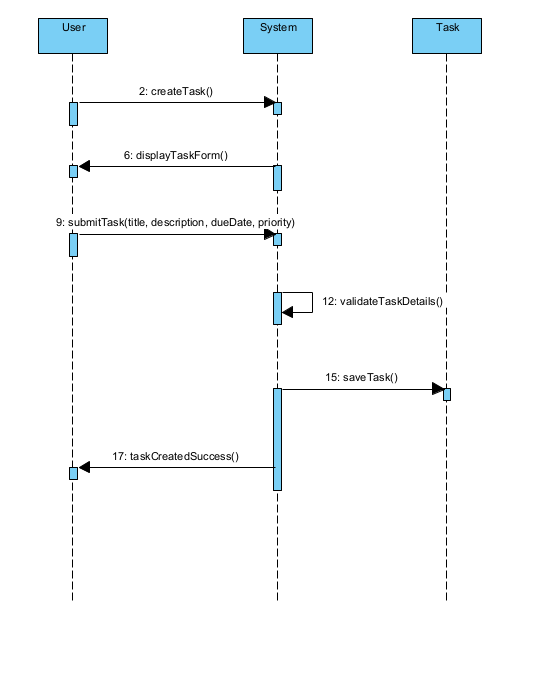
Belongs to

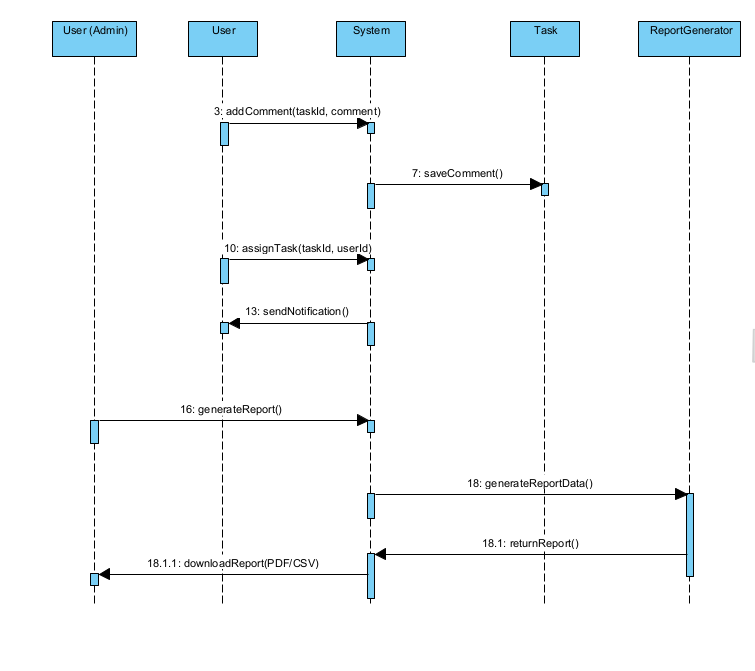
Creates/Manages

User-----🡪Comment----- 🡪 Task---- 🡪Admin------🡪Report

**1.10 Sequence Diagram**







# 2. Project Structure

## 2.1. Technologies

## Technologies Used

## Backend: Node.js with Express framework

## Database: MySQL (using Singleton design pattern for database connection management)

## Frontend: HTML, CSS, and JavaScript (responsive design for multiple devices)

## Architecture: MVC (Model-View-Controller) pattern, separating concerns into models, views (frontend), and controllers (business logic & routing)

## Communication: RESTful APIs enabling frontend-backend interaction

## Design Patterns Implemented

## Singleton Pattern: Used to create and manage a single database connection instance, ensuring efficient and consistent database access throughout the app.

## Observer Pattern: Applied to handle real-time updates or task status changes — for example, triggering logs or notifications when a task’s status changes.

## Database Entities and CRUD Operations

## Our system has three main entities represented as MySQL tables with standard CRUD (Create, Read, Update, Delete) operations:

| Entity | Fields | Purpose |
| --- | --- | --- |
| Users | id, username, email, password,role | User management and authentication |
| Projects | id, name, description, | Manage projects belonging to specific users |
| Tasks | id,title,description,status,user\_id, project\_id,created\_at, updated\_at | Tasks linked to projects, track details & status |

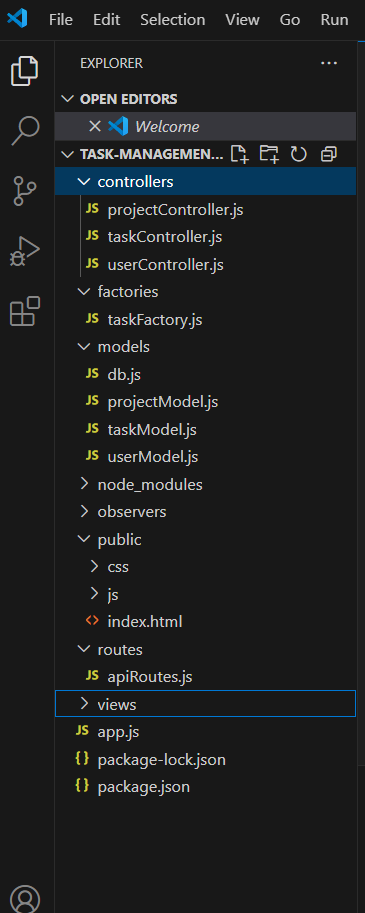
## 2.2. Architectural Pattern

## Description: The MVC pattern divides the application into three interconnected components:

## Model: Handles data and business logic (e.g., database entities like Users, Projects, Tasks).

## View: The user interface, displaying data and capturing user input (your HTML/CSS/JavaScript frontend).

## Controller: Manages communication between Model and View, processing user requests and updating data accordingly.



## 2.3. Database Entities

## Database Entities

## users Stores user information such as username, email, and password.

## projects Contains project details like project name, description, and the user who owns the project.

## tasks Holds tasks linked to projects, including task title, description, status, due date, and associated project.

## 2.4. Design Patterns

1. Singleton Pattern

* Location: models/db.js
* Explanation:  
  The Singleton pattern is applied in the Database class to ensure only one instance of the MySQL database connection is created and reused throughout the application. The static method getInstance() controls the instantiation, preventing multiple connections that could waste resources or cause conflicts.
* Benefit:  
  This pattern provides a single point of access to the database connection, ensuring efficient resource usage and consistent database state across all models and controllers.

2. Factory Pattern

* Location: factories/taskFactory.js
* Explanation:  
  The Factory pattern is used to create different types of Task objects (UrgentTask and NormalTask) based on the provided task type. This encapsulates object creation logic, so client code (e.g., controllers) can request tasks without worrying about the specific task subclass details.
* Benefit:  
  It improves code modularity and makes adding new task types easier without changing existing creation logic in multiple places.

3. Model-View-Controller (MVC) Pattern

* Location:
  + Models: models/projectModel.js, models/taskModel.js, models/userModel.js, models/db.js
  + Controllers: controllers/projectController.js, controllers/taskController.js, controllers/userController.js
  + Routes: routes/apiRoutes.js
* Explanation:  
  The application follows the MVC architecture to separate concerns:
  + Models manage data access and represent database tables.
  + Controllers handle the business logic, receiving requests from routes and interacting with models.
  + Routes define the API endpoints and map HTTP requests to controllers.
* Benefit:  
  This separation enhances maintainability, testing, and scalability by organizing code into distinct layers with clear responsibilities.

## 2.5. Project Functionalities and Screenshots

## User Management

## Register new users with username, email, password, and role (e.g., admin, regular user).

## View, update, and delete user profiles.

## Role-based access control can be implemented to restrict actions based on user roles.

## Project Management

## Create, read, update, and delete projects.

## Each project has a name, description, and ownership by a user.

## List all projects or view project details by ID.

## Task Management

## Create tasks linked to specific projects.

## Tasks include title, description, status (e.g., pending, in progress, completed), and due date.

## Update task details or delete tasks as needed.

## Retrieve all tasks or specific tasks by ID.

## RESTful API

## Clean API endpoints for tasks, projects, and users following REST conventions.

## Supports CRUD operations through HTTP methods (GET, POST, PUT, DELETE).

## Database Layer

## Centralized database connection using Singleton pattern to ensure efficient resource management.

## Separate model classes for Users, Projects, and Tasks encapsulating database queries.

## MVC Architecture

## Separation of concerns between data models, business logic (controllers), and API routing.

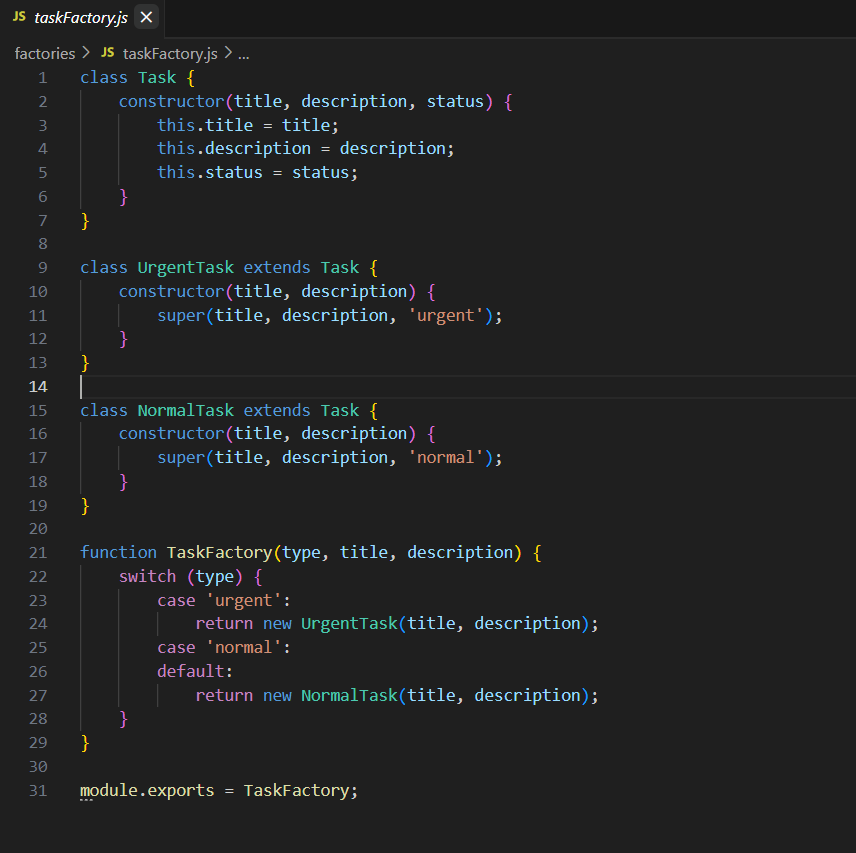
## Enhances maintainability and scalability of the application.

## Basic Input Validation and Error Handling

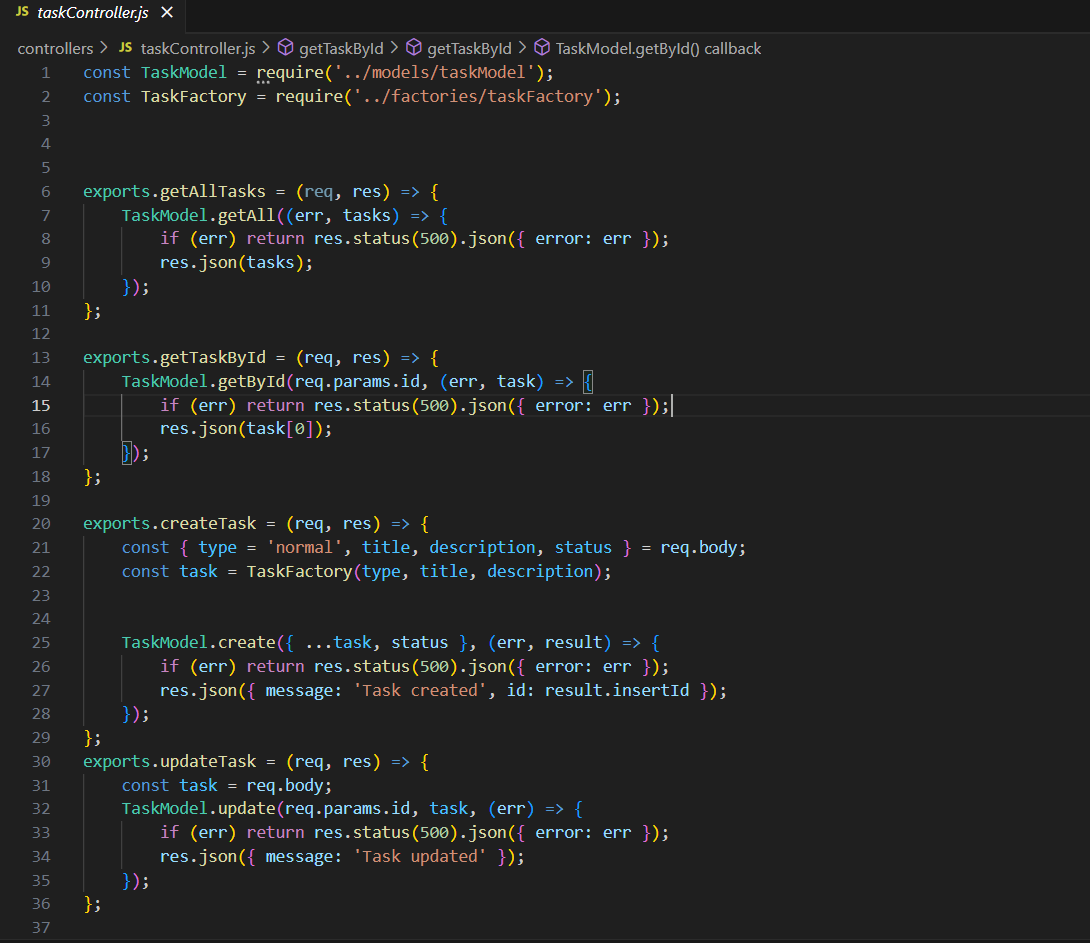
## Ensures data integrity before saving to the database.

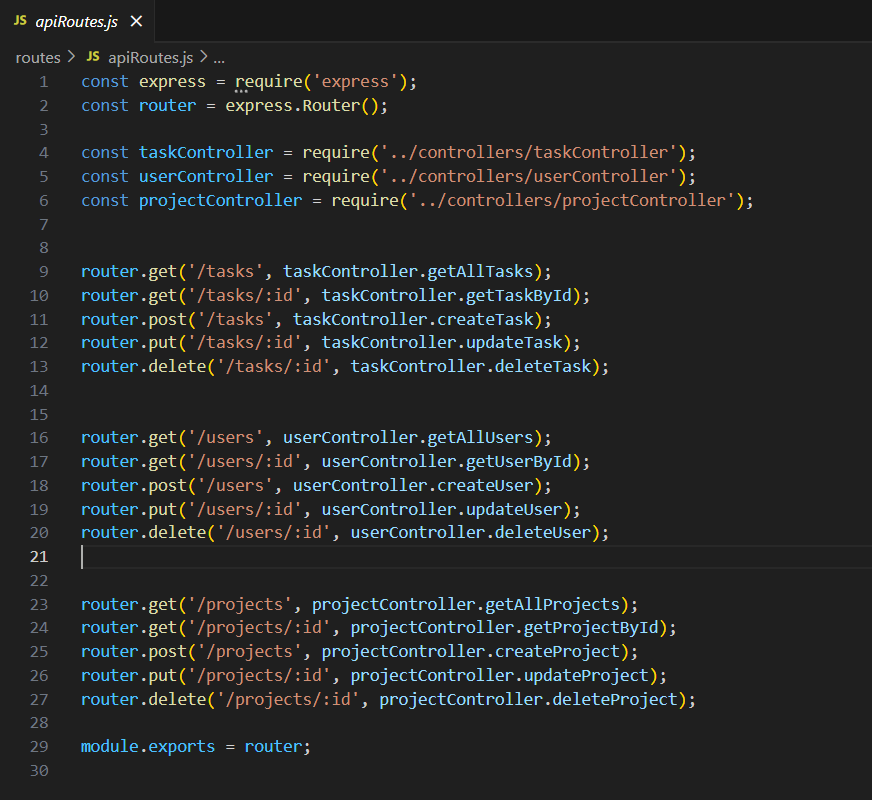
## Returns appropriate success or error responses on API calls.

## 

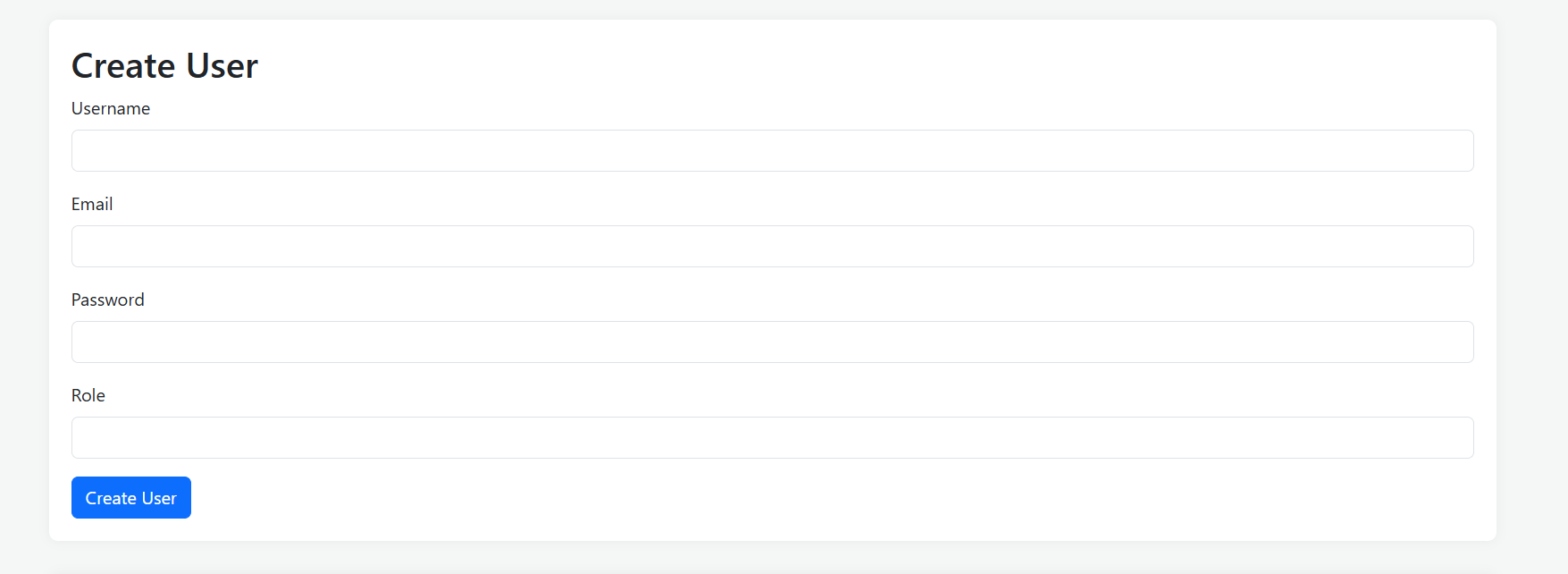


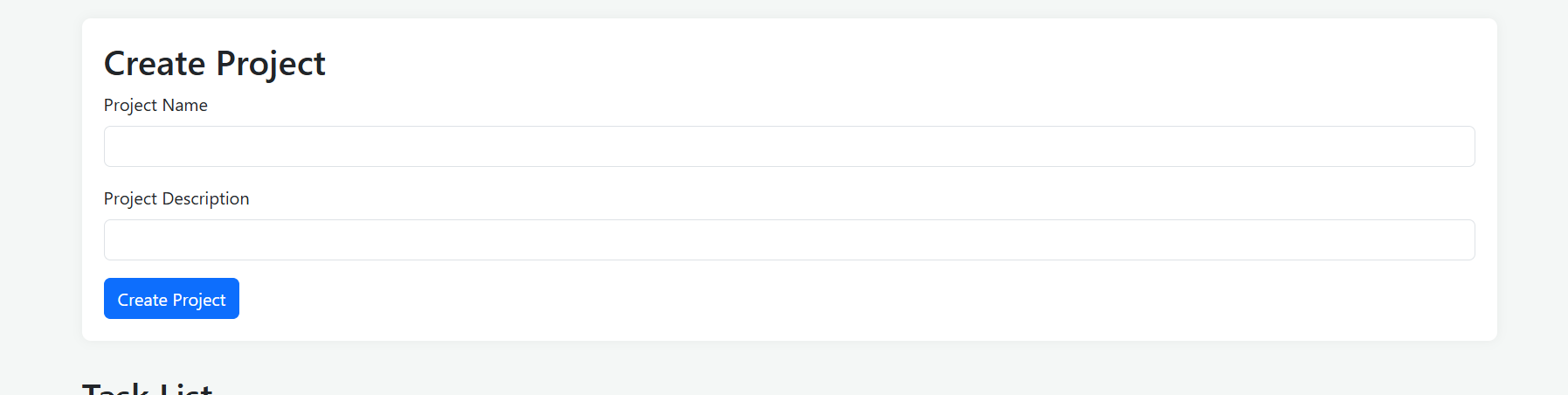


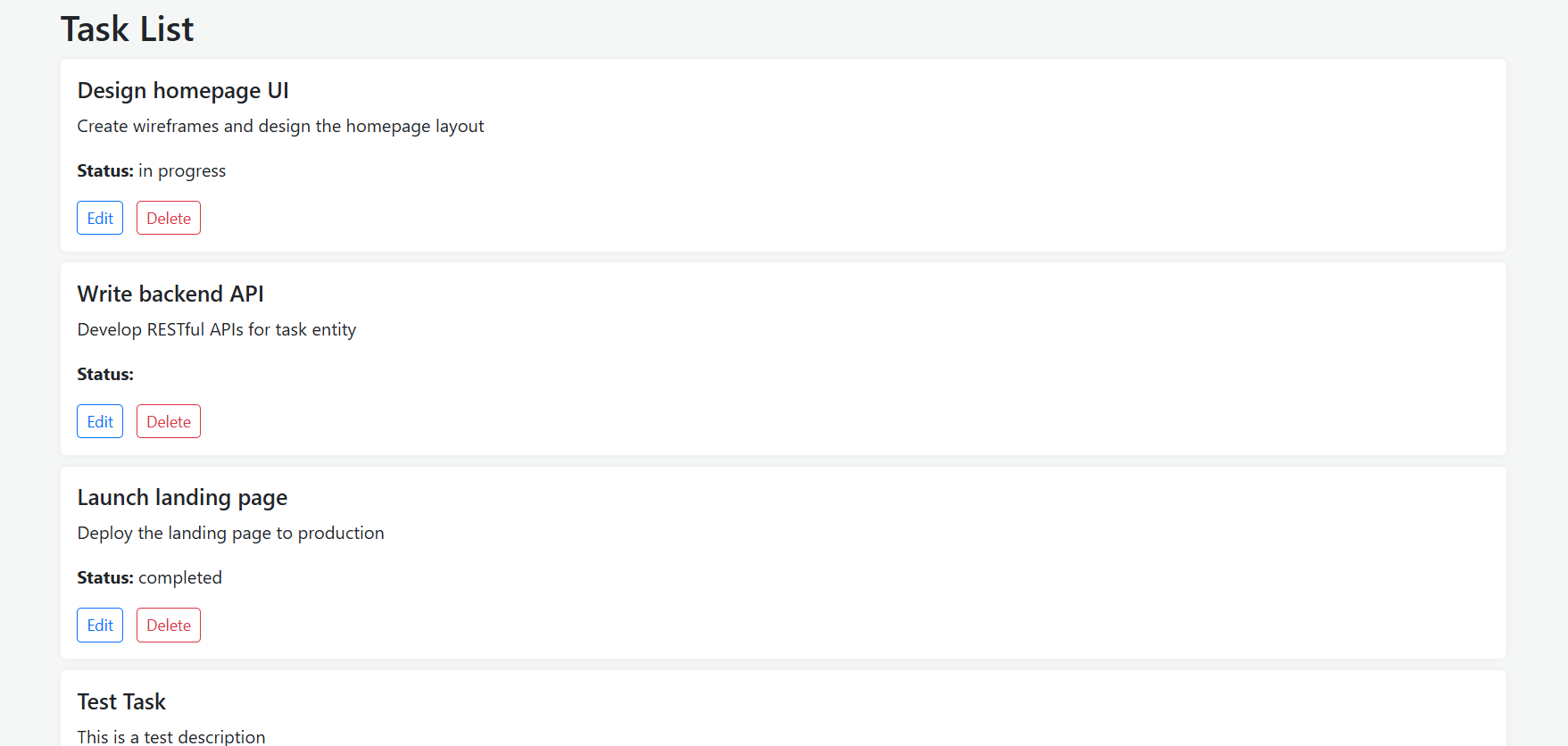


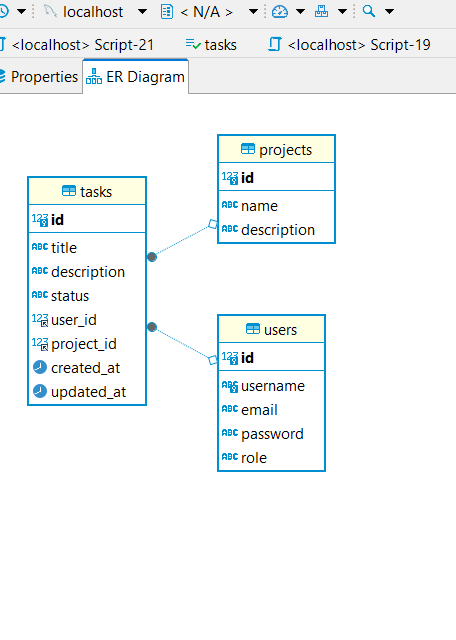












## 2.6. Tests

# For this application, unit tests were written using the Jest testing framework. These tests are designed to validate the core functionality of the model layer, specifically the logic that interacts with the MySQL database.

# 🔹 Types of Tests:

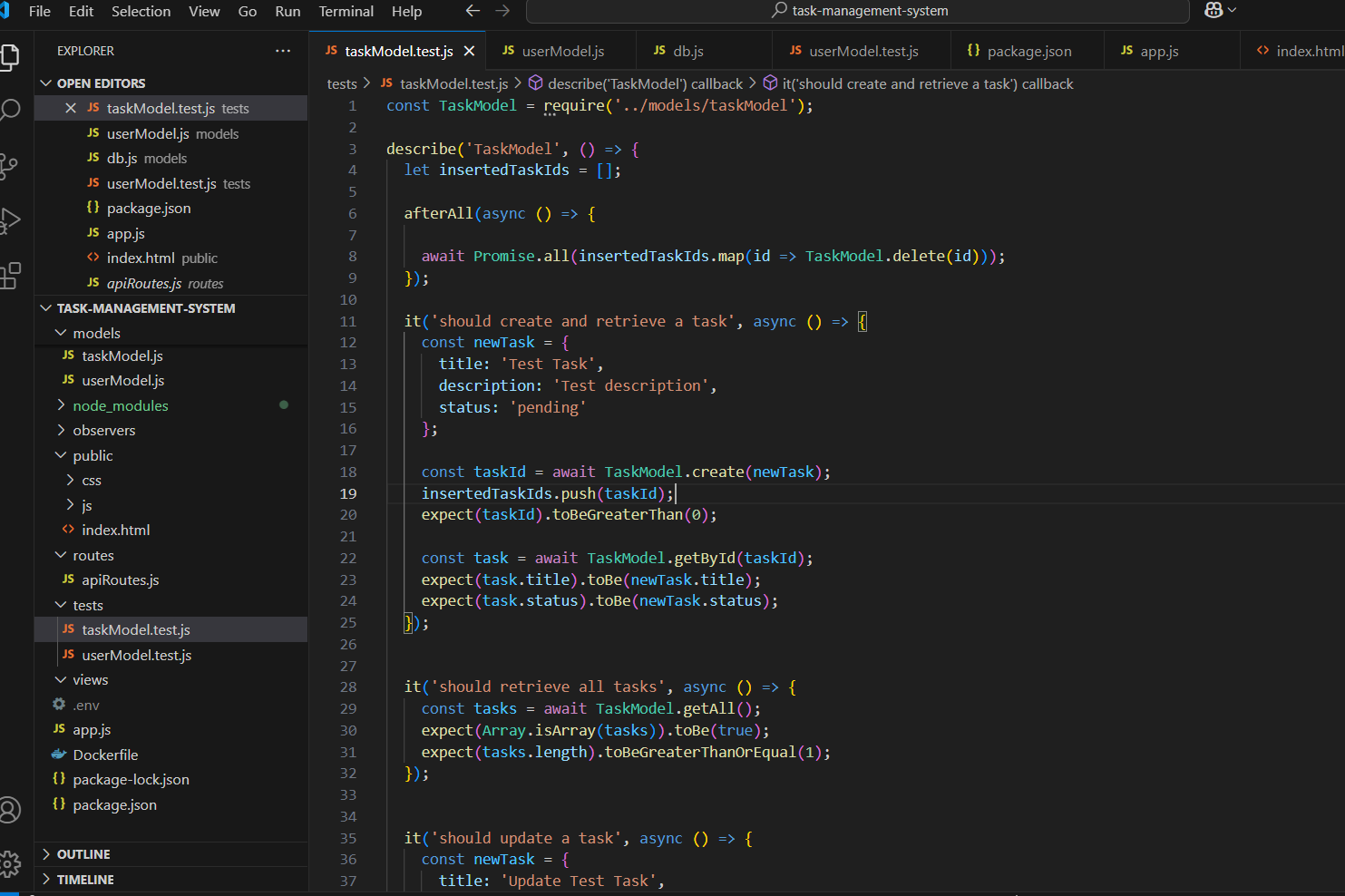
# Unit Tests Focused on testing individual functions in the UserModel and TaskModel. These tests ensure that basic CRUD operations (Create, Read, Delete) work as expected without involving the entire application stack.

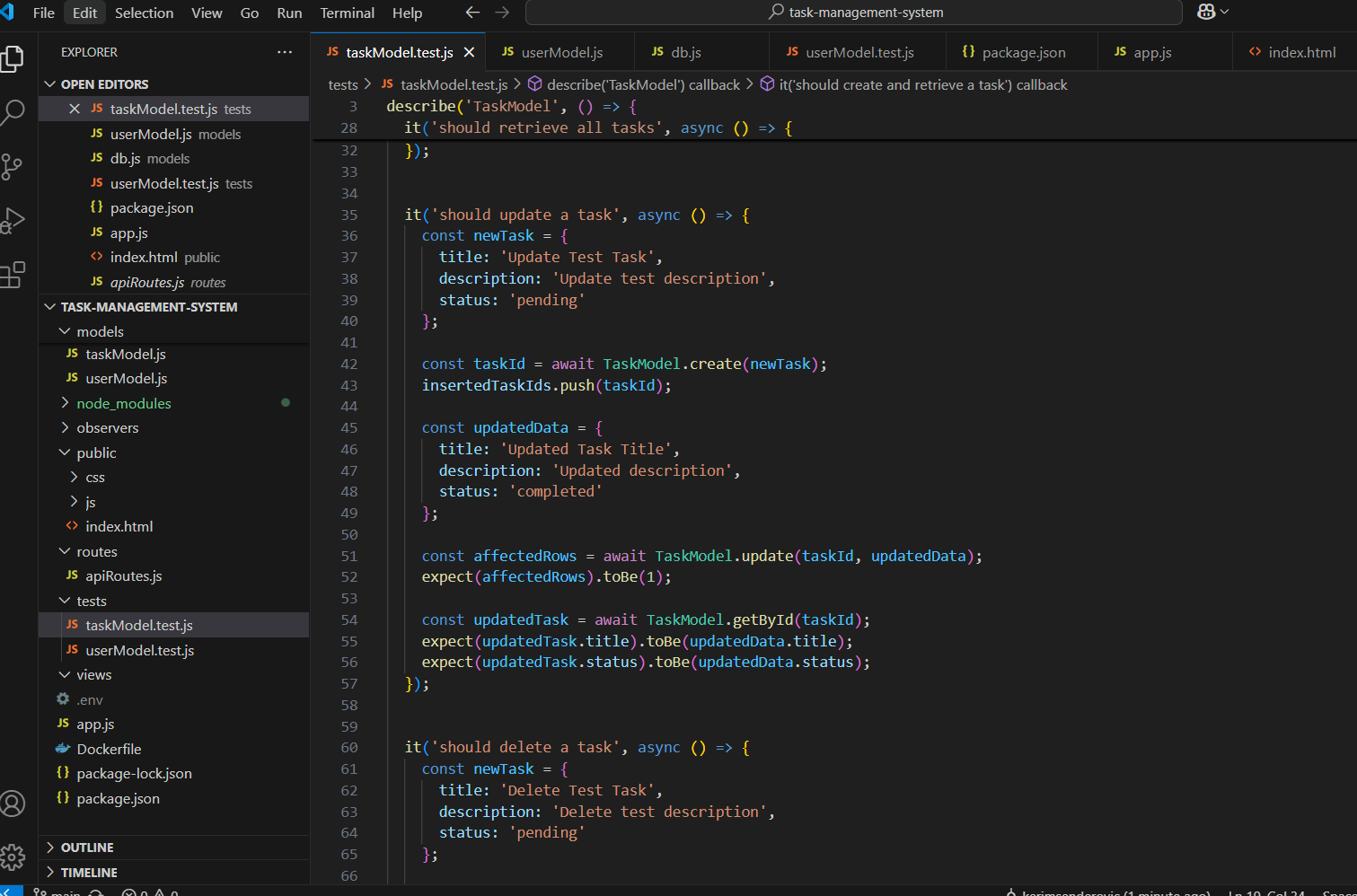
# 🔹 Test Location:

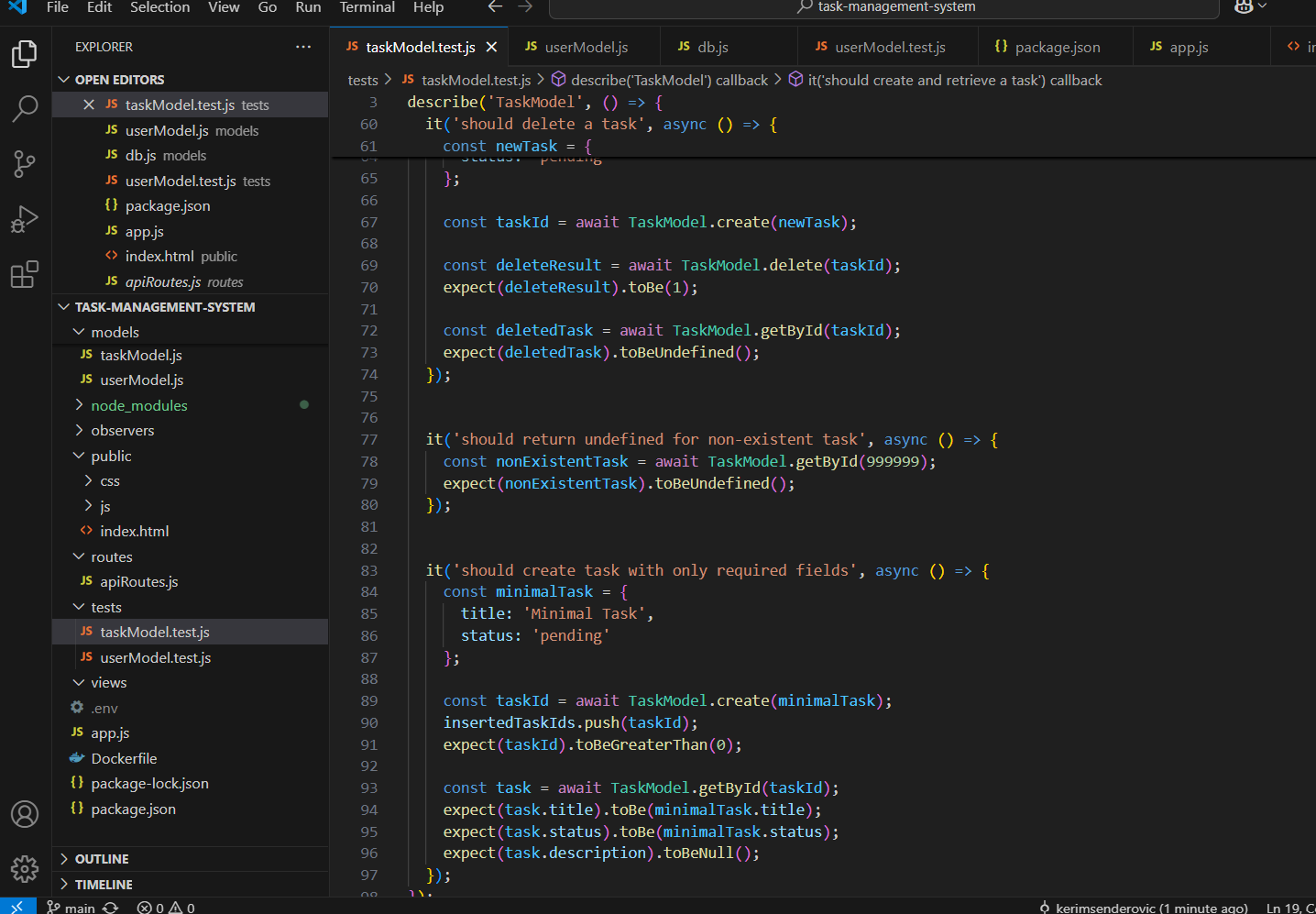
# All test files are located in the tests/ directory at the root of the project.

# tests/userModel.test.js — tests for user creation, retrieval, and deletion.

# tests/taskModel.test.js — tests for task creation, retrieval, and deletion.









# 3. Conclusion

Overall, we are satisfied with the implementation of our Task Management System. This project allowed us to deepen our understanding of full-stack development, particularly in applying the MVC architecture, setting up RESTful APIs, integrating a MySQL database, and writing unit tests using Jest.

One of the more challenging aspects was ensuring a stable and reusable database connection using the Singleton pattern and correctly handling asynchronous operations in our test cases. It took some troubleshooting to manage connection issues and test reliability.

In the future, we would like to enhance the user interface for better usability, implement authentication with JWT and role-based access control, and expand our test coverage to include integration and possibly end-to-end tests. Incorporating CI/CD pipelines would also be a valuable next step.

This project has been a meaningful learning experience and has helped us gain practical skills in building, organizing, and testing a maintainable backend system.

# 4. Individual Contributions

In this section, each team member should describe their specific contributions to the project. This includes the tasks they were responsible for, and the work they completed. Be as detailed as possible to clearly outline your role and efforts in the project.

Consider mentioning:

* The specific features, research, or components you worked on.
* Collaboration with other team members and how you contributed to the team's overall progress.

This section helps document the contributions of each member, ensuring transparency and recognition of individual efforts. Please make sure to provide a thoughtful and comprehensive summary rather than leaving this section empty.

**Student 1 – Kerim Senderović (50%)**

Kerim was primarily responsible for backend development, testing, and system planning. His contributions included:

* Backend Development: Designed and implemented the server-side logic using Node.js and Express. Developed RESTful API endpoints for users, tasks, and projects, enabling full CRUD operations and smooth client-server communication.
* Database Design: Created the MySQL database schema and implemented models using the MVC structure. Applied the Singleton pattern to manage database connections efficiently.
* Design Patterns: Implemented design patterns including Singleton for the database and Factory for generating task types.
* System Planning & Architecture: Contributed to writing functional and non-functional requirements, and created key architecture components including the Agile roadmap and release plan.
* Testing: Wrote backend unit tests using Jest for models (UserModel, TaskModel), ensuring reliable database operations and code quality.
* Collaboration: Maintained frequent coordination with Rijad to ensure the frontend consumed the API correctly. They worked together during debugging and database testing phases.

**Student 2 – Rijad Bašić (50%)**

Rijad was primarily responsible for frontend development, user experience, and project documentation. His contributions included:

* Frontend Development: Built the frontend using HTML, CSS, JavaScript, and Bootstrap. Designed responsive forms and UI components for user, project, and task management.
* API Integration: Connected frontend elements to backend endpoints using fetch() to enable data retrieval, submission, and dynamic rendering.
* System Analysis & Documentation: Led the effort in preparing System Analysis and Design (SAD) diagrams, including Use Case, Class, and Sequence diagrams. Contributed to writing the functional and non-functional requirements.
* Agile & Release Planning: Helped draft the release plan, defining version features and sprint goals. Participated in sprint reviews and retrospectives to reflect on progress.
* Deployment Preparation: Worked on preparing the project for local deployment and ensured the app ran on localhost without errors. Helped document how to start and test the application.
* Testing & Debugging: Collaborated with Kerim to run and troubleshoot unit tests for the backend. Verified that frontend interactions triggered expected backend behavior.
* Collaboration: Worked closely with Kerim on syncing frontend features with backend logic and database models, ensuring that both ends functioned cohesively.

Together, both members contributed equally and collaboratively to a full-stack Task Management System following good software engineering practices like modularity, testing, and documentation.