**THE INSTITUTE OF FINANCE MANAGEMENT**



**FACULTY OF COMPUTING INFORMATION SYSTEMS AND MATHEMATICS (FCIM)**

**DEPARTMENT OF COMPUTER SCIENCE**

**BACHELOR OF SCIENCE IN COMPUTER SCIENCE**

**THIRD YEAR-SEMESTER 2**

**CSU08216: ADVANCED WEB PROGRAMMING**

**ACADEMIC YEAR: 2023/2024**

**INDIVIDUAL ASSIGNMENT**

|  |  |
| --- | --- |
| **CANDIDATE NAME** | **REGISTRATION NUMBER** |
| 1. ABDULKARIM NATTY | IMC/BCS/2120130 |

Contents

[I) Project Description 3](#_Toc169730903)

[1. Project Overview 3](#_Toc169730904)

[2. Purpose of the Project 3](#_Toc169730905)

[2a) The User Business of the Project Effort 3](#_Toc169730906)

[2b) Goals of the Project 4](#_Toc169730907)

[3. Product Scenario 4](#_Toc169730908)

[II) Requirements 5](#_Toc169730909)

[4. Product Use Cases 5](#_Toc169730910)

[5. Use Case Diagram 8](#_Toc169730911)

[6. Functional Requirements 8](#_Toc169730912)

[Use Case 1: User Registration and Login 8](#_Toc169730913)

[Use Case 2: Create a New Project 9](#_Toc169730914)

[Use Case 3: Add and Assign Tasks 9](#_Toc169730915)

[Use Case 5: Update Task Status 10](#_Toc169730916)

[Use Case 6: View Project Progress 10](#_Toc169730917)

[7. Data Requirements 11](#_Toc169730918)

[8. Performance Requirements 13](#_Toc169730919)

[Non-functional Requirements Considerations: 14](#_Toc169730920)

[III) Design 15](#_Toc169730921)

[9. System Design 15](#_Toc169730922)

[10. Proposed Software Architecture Overview 16](#_Toc169730923)

[11. User Interface Design 18](#_Toc169730924)

[12. Database Design 23](#_Toc169730925)

[Overview 23](#_Toc169730926)

[Entities and Relationships 23](#_Toc169730927)

[Database Schema 24](#_Toc169730928)

[ER Diagram 26](#_Toc169730929)

[IV) Conclusion 27](#_Toc169730930)

# Project Description

## Project Overview

The Project Management Tool for Computer Science Students is a comprehensive software application designed to streamline task and project management for students. It offers features such as task prioritization, collaboration tools, time tracking, and resource management, all within a user-friendly interface. By facilitating efficient organization and teamwork, this tool aims to enhance academic productivity and performance. Developed using Inertia.js with React for the frontend, Laravel for the backend, and MySQL for the database, it is tailored specifically to meet the unique needs of computer science students managing multiple concurrent projects and assignments.

## Purpose of the Project

### 2a) The User Business of the Project Effort

The business logic of the Project Management Tool for Computer Science Students encompasses several key functionalities aimed at enhancing project and task management. The system allows users to create and manage projects, breaking them down into smaller tasks with assigned deadlines and priorities. It supports user authentication and role management, ensuring secure access and personalized user experiences.

The motivation behind developing the Project Management Tool for Computer Science Students stemmed from the need to address the unique challenges that students face in managing multiple academic projects and assignments. Traditional project management tools often lack features tailored to the academic environment, especially for computer science students who juggle coding projects, group assignments, and individual tasks. This tool was designed to provide a specialized platform that supports task prioritization, collaborative teamwork, and efficient resource management. By offering an intuitive interface and functionalities that cater specifically to the workflow of computer science students, this tool aims to improve their organizational skills, facilitate seamless collaboration, and ultimately enhance their academic performance

Managing multiple projects and assignments is a significant challenge for computer science students, often leading to missed tasks, poor collaboration, and inefficient resource use. Traditional tools do not cater to the specific needs of these students, resulting in disorganization and stress. By creating a specialized project management tool, we aim to provide a solution that simplifies task management, enhances teamwork, and supports academic success, addressing a critical gap in the existing tools available to students.

### 2b) Goals of the Project

The goal of the Project Management Tool for Computer Science Students is to provide an efficient and user-friendly platform for managing academic projects and tasks. It aims to enhance collaboration, organization, and resource management among students. By addressing the specific needs of computer science students, the tool seeks to improve their academic performance and reduce stress.

## Product Scenario

Jane, a computer science student, logs into the Project Management Tool at the beginning of the semester. She creates a new project for her software engineering course, dividing it into tasks such as "Design UI," "Implement Backend," and "Write Documentation." She invites her teammates to the project, assigns tasks, and uploads necessary resources like design mockups and code snippets. As they progress, the team members update their task statuses, share feedback, and collaborate through the platform. Jane uses the tool to keep track of their progress, ensuring everyone stays organized and on track, leading to the timely completion of their project with improved efficiency and less stress.

# Requirements

## Product Use Cases

#### Use Case 1: User Registration and Login

**Actors:** Student

**Description:** A new user registers for an account and logs into the system.

**Steps:**

1. The user navigates to the registration page.
2. The user enters their name, email, and password.
3. The system validates the input and creates a new user account.
4. The user receives a confirmation email and activates their account.
5. The user navigates to the login page.
6. The user enters their email and password.
7. The system verifies the credentials and logs the user in, redirecting them to the dashboard.

#### Use Case 2: Create a New Project

**Actors:** Registered User

**Description:** A user creates a new project within the system.

**Steps:**

1. The user clicks the "Create New Project" button on the dashboard.
2. The user enters the project name, description, and selects a category.
3. The user clicks "Save" to create the project.
4. The system saves the project details and displays the project on the user's dashboard.

#### Use Case 3: Add and Assign Tasks

**Actors:** Project Member

**Description:** A user adds tasks to a project and assigns them to team members.

**Steps:**

1. The user navigates to the project page.
2. The user clicks the "Add Task" button.
3. The user enters the task name, description, and priority.
4. The user assigns the task to a team member from a dropdown list.
5. The user clicks "Save" to create the task.
6. The system saves the task details and assigns it to the selected member.
7. The assigned member receives a notification about the new task.

#### Use Case 5: Update Task Status

**Actors:** Assigned Task Member

**Description:** A user updates the status of a task they are working on.

**Steps:**

1. The user navigates to the project page.
2. The user clicks on the task they want to update.
3. The user changes the task status from a dropdown list (e.g., "In Progress," "Completed").
4. The user clicks "Update" to save the new status.
5. The system updates the task status and notifies relevant project members of the change.

#### Use Case 6: View Project Progress

**Actors:** Project Member

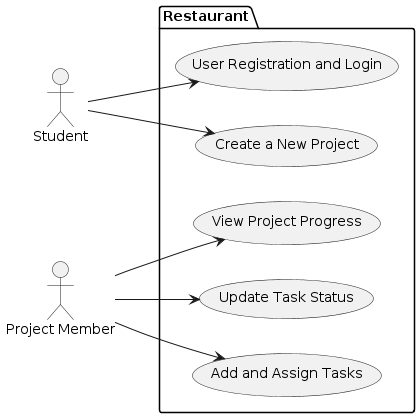
**Description:** A user views the overall progress of a project.

**Steps:**

1. The user navigates to the project page.
2. The system displays a progress bar or chart showing the completion status of all tasks.
3. The user can click on individual tasks to see more details and updates.
4. The user reviews the project progress to ensure it is on track.

These use cases outline the specific steps taken by users and the system, ensuring clarity in the functionality and interactions within the Project Management Tool.

## Use Case Diagram

fig 1.0

## Functional Requirements

### Use Case 1: User Registration and Login

**Functional Requirements:**

1. **Registration Page:**
   * FR1.1: Provide a user interface for registration where the user can enter name, email, and password.
   * FR1.2: Validate input fields (name, email format, password requirements).
   * FR1.3: Check email uniqueness to prevent duplicate accounts.
   * FR1.4: Generate and send a confirmation email with an activation link.
2. **Login Page:**
   * FR1.5: Provide a login interface where the user can enter email and password.
   * FR1.6: Verify credentials against stored user data.
   * FR1.7: Redirect authenticated users to their dashboard upon successful login.

### Use Case 2: Create a New Project

**Functional Requirements:**

1. **Dashboard:**
   * FR2.1: Display a "Create New Project" button on the user dashboard.
   * FR2.2: Provide fields for entering project name, description, and category selection.
   * FR2.3: Validate project name and description fields.
   * FR2.4: Save project details to the database upon user's save action.

### Use Case 3: Add and Assign Tasks

**Functional Requirements:**

1. **Project Page:**
   * FR3.1: Display an "Add Task" button on the project page for authorized users.
   * FR3.2: Provide fields for task name, description, priority, and a dropdown list for assigning tasks.
   * FR3.3: Validate task name, description, and priority fields.
   * FR3.4: Save task details and assignment to the database upon user's save action.
   * FR3.5: Send notification to assigned team member upon task assignment.

### Use Case 5: Update Task Status

**Functional Requirements:**

1. **Project Page:**
   * FR5.1: Display tasks with their current status (e.g., "In Progress," "Completed") on the project page.
   * FR5.2: Provide a dropdown list for updating task status.
   * FR5.3: Validate task status update.
   * FR5.4: Save updated task status to the database upon user's update action.
   * FR5.5: Send notification to relevant project members about the updated task status.

### Use Case 6: View Project Progress

**Functional Requirements:**

1. **Project Page:**
   * FR6.1: Display a progress bar or chart representing the overall project completion status.
   * FR6.2: Allow users to click on individual tasks to view detailed information and updates.
   * FR6.3: Provide functionality to review and ensure the project progress aligns with expectations.

These functional requirements outline specific behaviors and actions that the system must support to fulfill the described use cases effectively. They ensure that the system is capable of handling user interactions in a structured and reliable manner.

## Data Requirements

#### Use Case 1: User Registration and Login

1. **User Data:**
   * **Attributes:** Name, Email, Password
   * **Requirements:**
     + DR1.1: Store user's name, email, and password securely.
     + DR1.2: Ensure uniqueness of email addresses to prevent duplicate accounts.
     + DR1.3: Store activation status (activated or pending) for user accounts.
     + DR1.4: Store timestamps for account creation and last login.
2. **Session Management:**
   * **Requirements:**
     + DR1.5: Store session tokens or identifiers for authenticated users to maintain login state.

#### Use Case 2: Create a New Project

1. **Project Data:**
   * **Attributes:** Project Name, Description, Category
   * **Requirements:**
     + DR2.1: Store project details including name, description, and selected category.
     + DR2.2: Assign a unique identifier (ID) to each project for reference and retrieval.

#### Use Case 3: Add and Assign Tasks

1. **Task Data:**
   * **Attributes:** Task Name, Description, Priority, Status, Assigned Member
   * **Requirements:**
     + DR3.1: Store task details such as name, description, priority level, and current status (e.g., "In Progress," "Completed").
     + DR3.2: Associate tasks with their corresponding project ID for organizational purposes.
     + DR3.3: Track assignment of tasks to specific team members using their identifiers.

#### Use Case 5: Update Task Status

1. **Task Status Data:**
   * **Attributes:** Task ID, Updated Status
   * **Requirements:**
     + DR5.1: Maintain a history of task status changes to track progress over time.
     + DR5.2: Store timestamps for each status update to monitor task lifecycle.

#### Use Case 6: View Project Progress

1. **Progress Tracking Data:**
   * **Attributes:** Project ID, Task Progress
   * **Requirements:**
     + DR6.1: Aggregate and calculate project progress based on task completion statuses.
     + DR6.2: Provide real-time updates or refresh mechanisms to display accurate progress to users.

These data requirements ensure that the system can store and manage the necessary information to support each use case effectively. They cover aspects such as user registration details, project and task information, task assignments, status updates, and overall project progress tracking. This structured approach helps in designing a database schema and defining data storage strategies that align with the system's functional needs.

## Performance Requirements

#### Use Case 1: User Registration and Login

1. **Registration and Login Response Time:**
   * **PR1.1:** The system shall respond to user registration and login requests within 3 seconds under normal load conditions.
2. **Concurrent User Handling:**
   * **PR1.2:** The system shall support registration and login operations for up to 1000 concurrent users without degradation in response time.
3. **Email Confirmation:**
   * **PR1.3:** Confirmation emails shall be sent within 1 minute of user registration to ensure timely activation.

#### Use Case 2: Create a New Project

1. **Project Creation Response Time:**
   * **PR2.1:** The system shall save project details and update the dashboard within 2 seconds of user submission.
2. **Scalability:**
   * **PR2.2:** The system shall support the creation of up to 500 new projects per hour without performance degradation.

#### Use Case 3: Add and Assign Tasks

1. **Task Addition Response Time:**
   * **PR3.1:** Adding a new task and assigning it shall take no more than 3 seconds under normal load conditions.
2. **Task Assignment Notification:**
   * **PR3.2:** Notifications for assigned tasks shall be delivered to the team member within 30 seconds of assignment.

#### Use Case 5: Update Task Status

1. **Task Status Update Response Time:**
   * **PR5.1:** Updating the status of a task shall occur instantly (within 1 second) upon user action.
2. **Real-Time Updates:**
   * **PR5.2:** Changes in task status shall be reflected immediately across all project members' views without the need for manual refresh.

#### Use Case 6: View Project Progress

1. **Progress Display Performance:**
   * **PR6.1:** The system shall load and display project progress details (including charts or graphs) within 3 seconds of user request.
2. **Real-Time Updates:**
   * **PR6.2:** Updates to project progress shall be reflected in real-time as tasks are completed or updated, ensuring accurate and up-to-date information.

### Non-functional Requirements Considerations:

1. **Reliability:**
   * The system should be available 99.9% of the time, allowing for scheduled maintenance windows.
2. **Security:**
   * User data should be encrypted both in transit and at rest, following industry-standard practices.

These performance requirements ensure that the system remains responsive, scalable, and reliable under various user interaction scenarios, thereby enhancing user experience and system usability.

# Design

## System Design

The primary design goals include:

1. **Seamless Page Transitions:** Implementing smooth transitions between pages without full page reloads, leveraging React's state management and component re-rendering capabilities to update content dynamically.
2. **Efficient Data Fetching:** Utilizing React hooks and asynchronous data fetching techniques (e.g., using fetch API or Axios) to retrieve data in the background and update components seamlessly, ensuring minimal latency and optimal performance.
3. **Reduced Server Round-Trips:** Minimizing server round-trips by caching data locally and optimizing API calls to fetch only necessary updates, thereby reducing network latency and improving overall responsiveness.
4. **Client-Side State Management:** Leveraging React's Context API or state management libraries (e.g., Redux) to maintain application state efficiently, ensuring that UI updates reflect changes in data without unnecessary reloads.
5. **Optimized Rendering:** Employing virtual DOM (Document Object Model) rendering techniques inherent to React to efficiently update UI components based on changes in data or user interactions, thereby enhancing rendering speed and responsiveness.

By adhering to these design goals, the web application can achieve a highly fast and responsive user experience, akin to native application performance. The implementation of inertia principles with React ensures that users experience fluid navigation between pages with instant data updates, thereby enhancing usability and engagement

## Proposed Software Architecture Overview

#### Frontend Architecture

1. **React Framework:**
   * **Component-Based Architecture:** Utilize React's component-based architecture to modularize UI elements and promote reusability across the application.
   * **Virtual DOM:** Leverage React's virtual DOM for efficient rendering and updates, ensuring optimal performance by minimizing DOM manipulation.
2. **Inertia.js Integration:**
   * **Principle of Inertia:** Implement Inertia.js to handle client-side navigation and data fetching without full page reloads, thereby providing a seamless user experience similar to traditional server-rendered applications.
   * **Benefits:** Reduces overhead by preserving server-side routing while enabling single-page application (SPA) behaviors, enhancing speed and responsiveness.
3. **State Management:**
   * **React Context API or Redux:** Depending on application complexity, utilize React's Context API for simpler state management or integrate Redux for centralized state management across components.
   * **Immutable State Updates:** Implement immutability to ensure predictable state changes and facilitate debugging.
4. **UI/UX Design:**
   * **Responsive Design:** Ensure the application is responsive across devices using CSS frameworks like Bootstrap or Tailwind CSS, optimizing user experience on desktop and mobile platforms.
   * **Performance-Oriented Design:** Optimize UI rendering and animations to maintain high performance and smooth interactions, prioritizing user engagement.

#### Backend Architecture (Laravel 11)

1. **Laravel Framework:**
   * **RESTful API:** Develop RESTful API endpoints using Laravel's routing system to interact with the frontend, facilitating CRUD operations for user management, project handling, task assignments, and status updates.
   * **Middleware:** Implement middleware for handling authentication, authorization, and request validation, ensuring secure and reliable data transmission.
2. **Database Integration:**
   * **Database Management:** Utilize Laravel's Eloquent ORM to manage interactions with the MYSQL database for efficient data storage and retrieval.
   * **Migration and Seeding:** Use Laravel's migration and seeding capabilities for database schema management and initial data population, ensuring consistency across development, staging, and production environments.
3. **Authentication and Authorization:**
   * **JWT Authentication:** Implement JWT-based authentication for secure user sessions and access control, ensuring data integrity and user privacy.
   * **Roles and Permissions:** Define roles and permissions using Laravel's Gate and Policies for granular access control within the application.

The proposed software architecture leverages React on the frontend with Laravel 11 on the backend to build a high-performance web application.

## User Interface Design

#### Overview

The user interface (UI) design of the application is crafted to provide a modern and intuitive experience for users, ensuring seamless navigation and efficient interaction. The design principles prioritize responsiveness across all devices and platforms, leveraging React components for dynamic content rendering and Inertia.js for smooth page transitions without full browser refreshes.

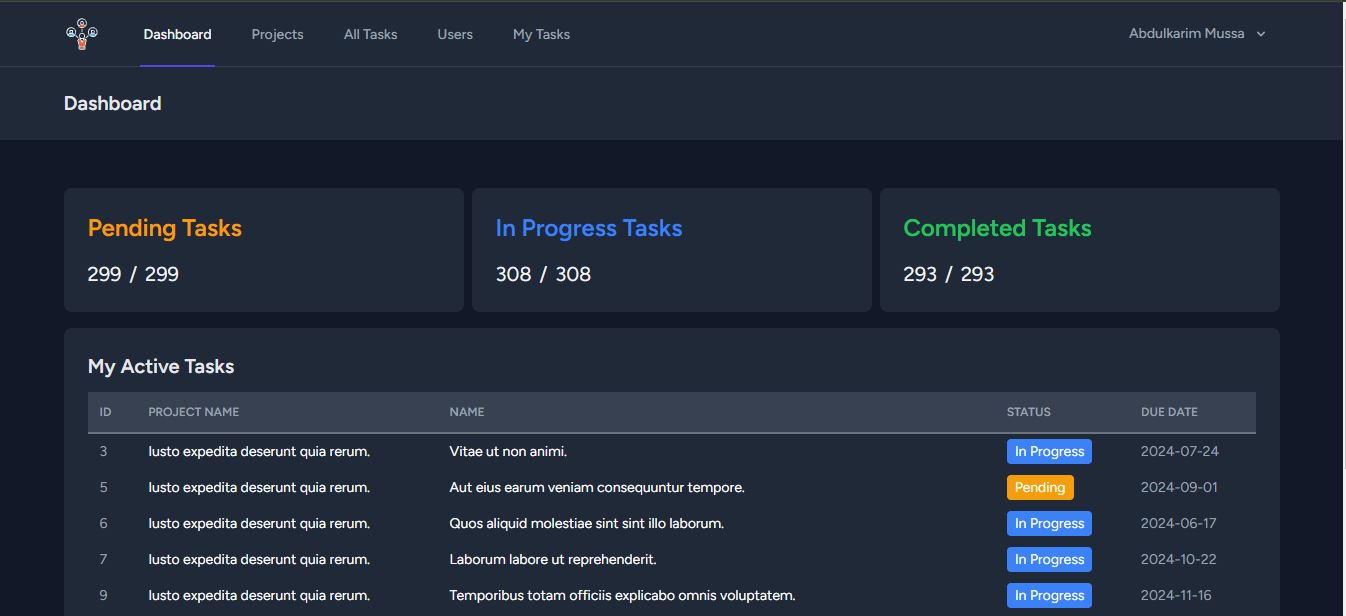
#### Design Components

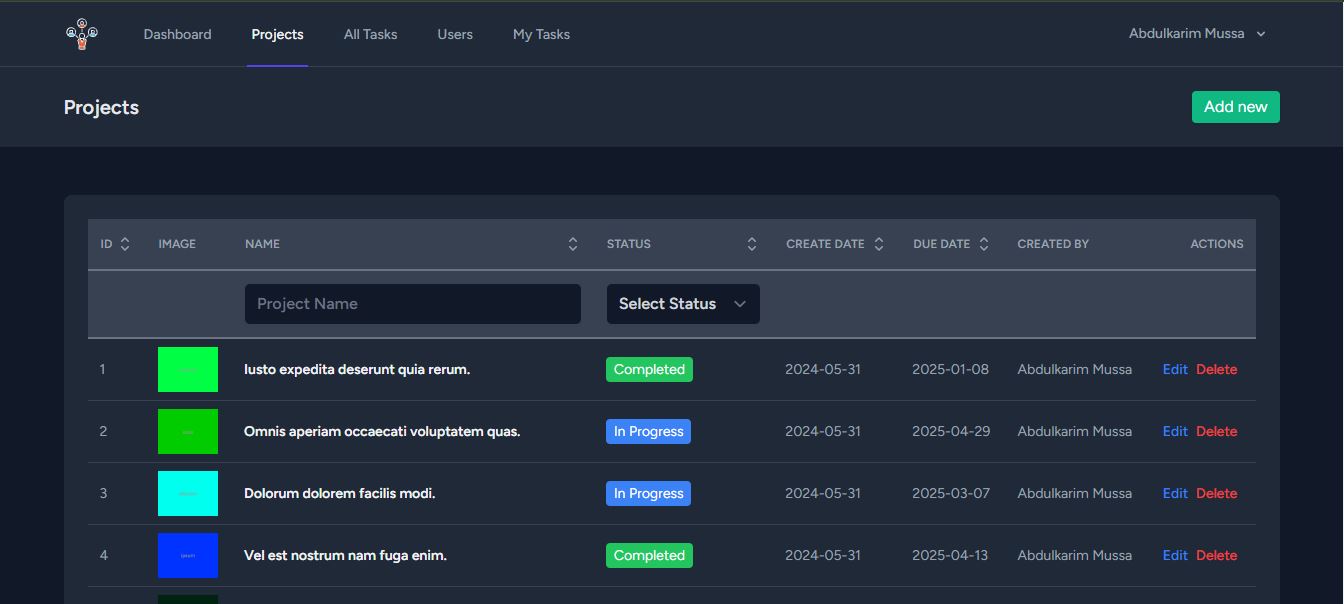
1. **Layout and Navigation**
   * **Responsive Layout:** The application features a responsive design that adapts fluidly to various screen sizes, including desktops, tablets, and mobile devices.
   * **Navigation Structure:** Clear and accessible navigation elements are strategically placed to facilitate easy traversal between different sections of the application.
2. **Visual Design**
   * **Color Scheme:** A harmonious color palette is employed to maintain brand identity and aid in visual hierarchy. Contrasting colors are used sparingly to highlight important elements and actions.
   * **Typography:** Readable and consistent typography choices enhance readability and user engagement, ensuring clarity across different screen resolutions.
3. **Form and Input Design**
   * **Input Fields:** Form inputs are designed with descriptive labels and placeholder text to guide users seamlessly through data entry processes.
   * **Validation Feedback:** Real-time validation feedback is incorporated to assist users in correcting errors promptly and efficiently.
4. **Data Presentation**
   * **Structured Display:** Data is presented in organized layouts such as tables, cards, or lists, optimizing information accessibility and user comprehension.
   * **Visualization Techniques:** Interactive charts and graphs are used where applicable to visualize complex data sets, providing users with actionable insights at a glance.

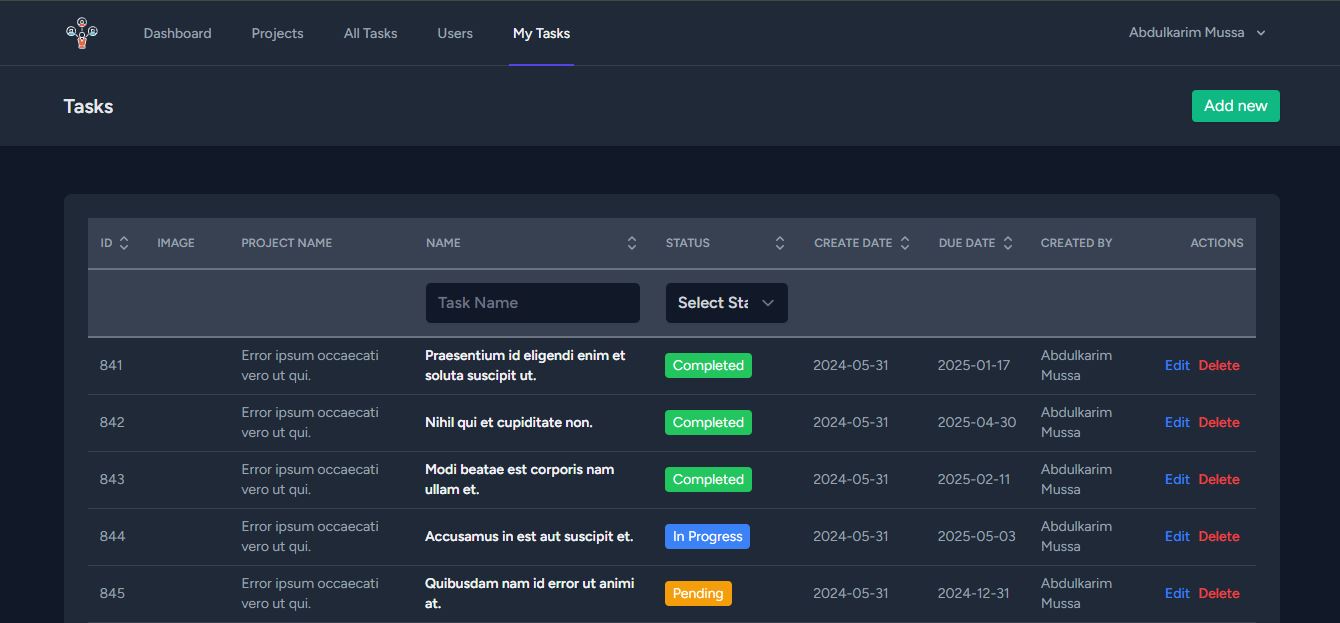
#### Implementation Strategy

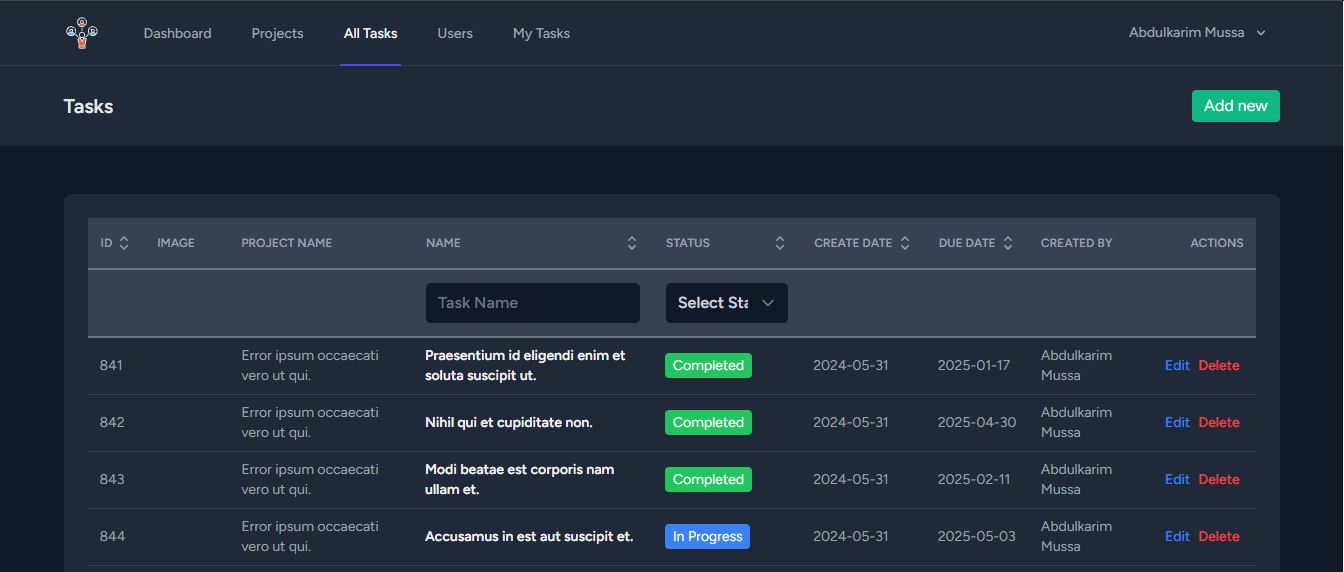
1. **Technology Stack**
   * **Frontend:** React components are structured and styled using CSS frameworks like Bootstrap or custom CSS, ensuring consistent design implementation and responsiveness.
   * **Backend Integration:** Inertia.js seamlessly integrates React components with Laravel's backend logic, maintaining SPA benefits while leveraging server-side rendering capabilities.
2. **Component Libraries**
   * Component libraries such as Material-UI or Tailwind CSS are utilized for pre-designed UI elements and icons, streamlining development and ensuring design consistency.

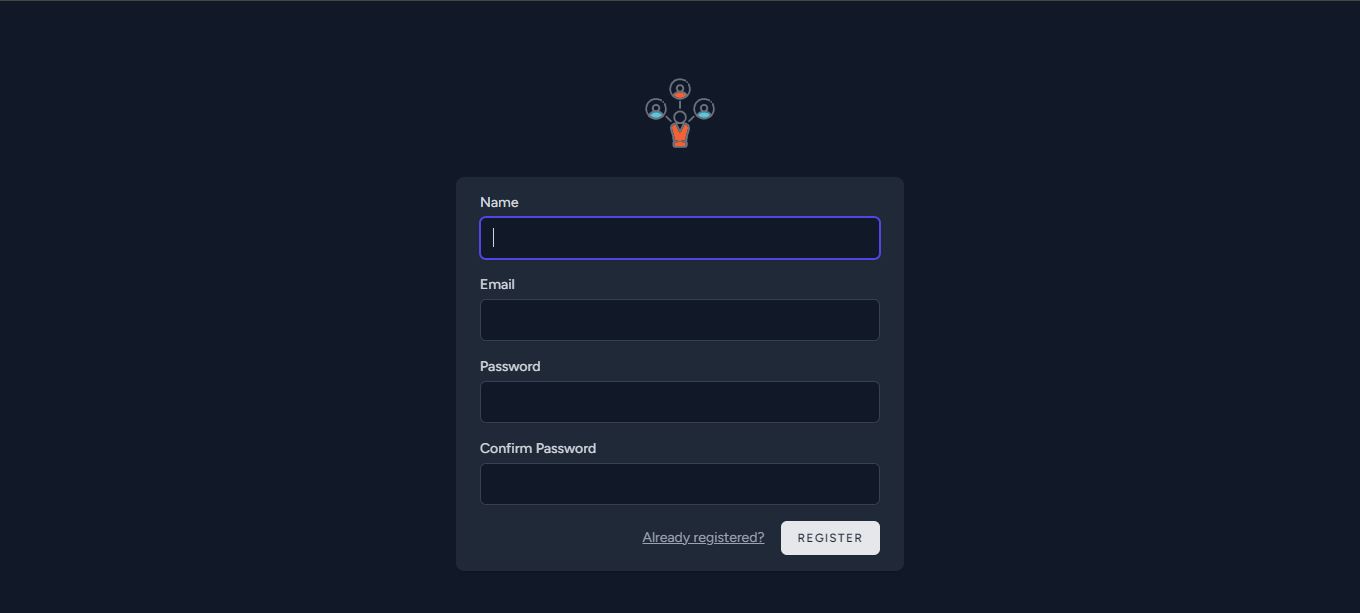
The user interface design of the application aims to deliver a visually appealing and user-centric experience, incorporating modern design practices and responsive architecture. By focusing on intuitive navigation, clear visual hierarchy, and efficient data presentation, the design ensures that users can interact with the application seamlessly across different devices, enhancing usability and overall satisfaction.

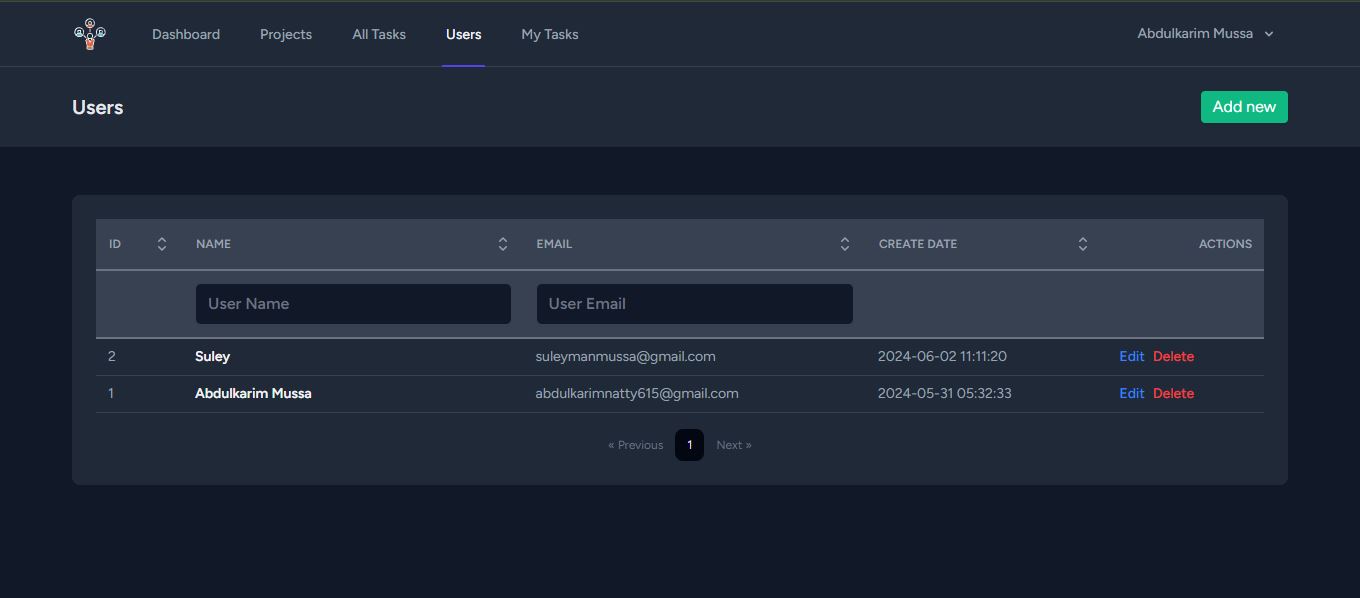
Fig 2.0 Dashboard Page.

Fig 2.1 Projects Page

Fig 2.2 My Tasks Page

Fig 2.3 All Tasks Page

Fig 2.4 Register Page

Fig 2.5 Users Page

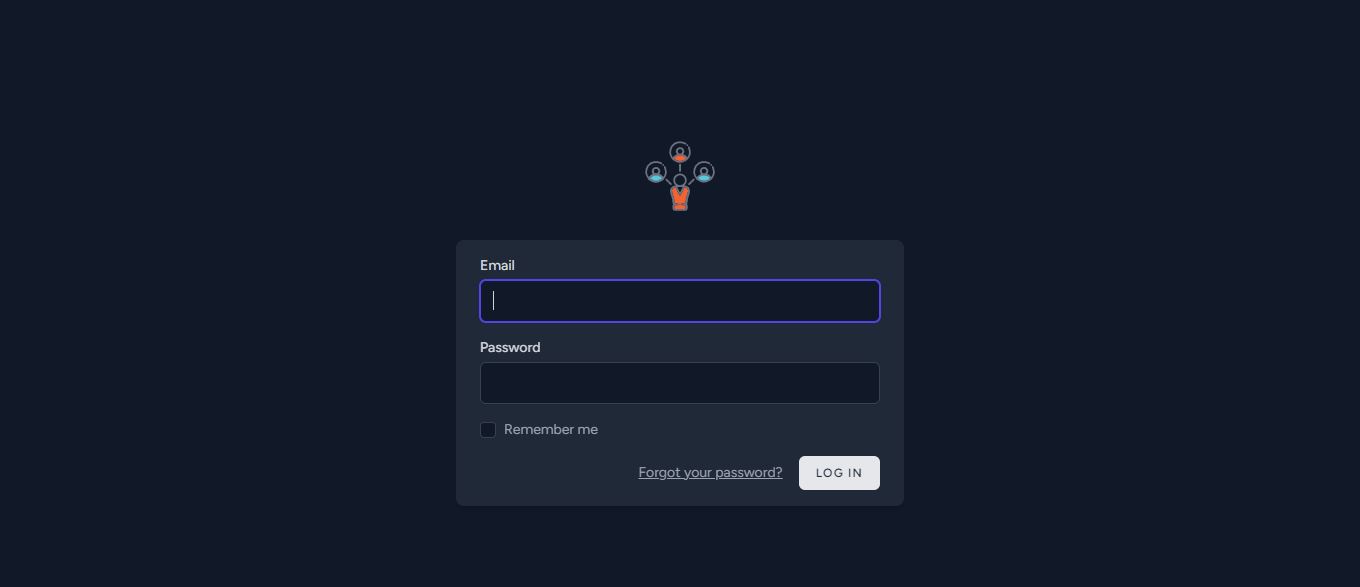


Fig 2.6 Login Page

## Database Design

### Overview

The database design for the application encompasses several interconnected entities, reflecting the relationships between users, projects, and tasks. The schema ensures efficient data management and integrity through foreign key constraints and appropriate indexing.

### Entities and Relationships

1. **Users**
   * The users table stores user information, including authentication details and timestamps for tracking account creation and updates.
   * Each user can create and update multiple projects and tasks, establishing a one-to-many relationship.
2. **Projects**
   * The projects table holds information about each project, including its name, description, status, due date, and image path.
   * Each project is associated with a user who created it and a user who last updated it, forming a many-to-one relationship with the users table.
   * Each project can have multiple tasks associated with it, forming a one-to-many relationship with the tasks table.
3. **Tasks**
   * The tasks table contains details about each task, including its name, description, status, priority, due date, and image path.
   * Each task is linked to a project, forming a many-to-one relationship with the projects table.
   * Tasks are also associated with users in three different roles: assigned user, creator, and updater, establishing multiple many-to-one relationships with the users table.

### Database Schema

1. **Users Table**

sql

Copy code

CREATE TABLE `users` (

`id` BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

`name` VARCHAR(255) NOT NULL,

`email` VARCHAR(255) NOT NULL UNIQUE,

`email\_verified\_at` TIMESTAMP NULL,

`password` VARCHAR(255) NOT NULL,

`remember\_token` VARCHAR(100) NULL,

`created\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP,

`updated\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

1. **Projects Table**

sql

Copy code

CREATE TABLE `projects` (

`id` BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

`name` VARCHAR(255) NOT NULL,

`description` TEXT NULL,

`due\_date` TIMESTAMP NULL,

`status` VARCHAR(255) NOT NULL,

`image\_path` VARCHAR(255) NULL,

`created\_by` BIGINT UNSIGNED NOT NULL,

`updated\_by` BIGINT UNSIGNED NOT NULL,

`created\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP,

`updated\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (`created\_by`) REFERENCES `users`(`id`),

FOREIGN KEY (`updated\_by`) REFERENCES `users`(`id`)

);

1. **Tasks Table**

sql

Copy code

CREATE TABLE `tasks` (

`id` BIGINT UNSIGNED AUTO\_INCREMENT PRIMARY KEY,

`name` VARCHAR(255) NOT NULL,

`description` TEXT NULL,

`status` VARCHAR(255) NOT NULL,

`priority` VARCHAR(255) NOT NULL,

`due\_date` TIMESTAMP NULL,

`image\_path` VARCHAR(255) NULL,

`assigned\_user\_id` BIGINT UNSIGNED NOT NULL,

`created\_by` BIGINT UNSIGNED NOT NULL,

`updated\_by` BIGINT UNSIGNED NOT NULL,

`project\_id` BIGINT UNSIGNED NOT NULL,

`created\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP,

`updated\_at` TIMESTAMP NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP,

FOREIGN KEY (`assigned\_user\_id`) REFERENCES `users`(`id`),

FOREIGN KEY (`created\_by`) REFERENCES `users`(`id`),

FOREIGN KEY (`updated\_by`) REFERENCES `users`(`id`),

FOREIGN KEY (`project\_id`) REFERENCES `projects`(`id`)

);

1. **Password Reset Tokens Table**

sql

Copy code

CREATE TABLE `password\_reset\_tokens` (

`email` VARCHAR(255) PRIMARY KEY,

`token` VARCHAR(255) NOT NULL,

`created\_at` TIMESTAMP NULL

);

1. **Sessions Table**

sql

Copy code

CREATE TABLE `sessions` (

`id` VARCHAR(255) PRIMARY KEY,

`user\_id` BIGINT UNSIGNED NULL,

`ip\_address` VARCHAR(45) NULL,

`user\_agent` TEXT NULL,

`payload` LONGTEXT NOT NULL,

`last\_activity` INT NOT NULL,

INDEX `user\_id\_index` (`user\_id`),

INDEX `last\_activity\_index` (`last\_activity`),

FOREIGN KEY (`user\_id`) REFERENCES `users`(`id`)

);

### ER Diagram

An Entity-Relationship (ER) diagram visually represents the database schema, showcasing the relationships between the tables.

* **Users Table:**
  + Primary Key: id
  + One-to-Many Relationships:
    - Projects (created\_by, updated\_by)
    - Tasks (assigned\_user\_id, created\_by, updated\_by)
* **Projects Table:**
  + Primary Key: id
  + Foreign Keys: created\_by, updated\_by
  + One-to-Many Relationship:
    - Tasks (project\_id)
* **Tasks Table:**
  + Primary Key: id
  + Foreign Keys: assigned\_user\_id, created\_by, updated\_by, project\_id

The database design for the application ensures robust data integrity and efficient relationships between users, projects, and tasks. By leveraging foreign key constraints and indexing, the schema promotes optimal data retrieval and management, supporting the application's requirements and scalability. This design aligns with the overall goal of delivering a highly responsive and user-friendly web application.

# Conclusion

The Project Management Tool for Computer Science Students represents a significant advancement in addressing the unique challenges faced by students in managing academic projects and tasks. Through the integration of React and Laravel 11, coupled with Inertia.js for seamless navigation, this application provides a robust and efficient platform that enhances organization, collaboration, and productivity.

Key features such as project and task management, user authentication, real-time updates, and a user-friendly interface are meticulously designed to meet the specific needs of computer science students. The implementation of fast pagination ensures that users can access and manipulate data swiftly, contributing to an overall improved user experience.

The system's architecture, which combines the strengths of client-side and server-side technologies, ensures high performance and scalability. The database design effectively supports the application's requirements, maintaining data integrity and facilitating efficient data management.

In summary, this project exemplifies a comprehensive solution tailored to improve the academic workflow of computer science students. By streamlining project management processes and enhancing user experience through innovative design and technology, the Project Management Tool is poised to make a substantial impact on students' academic success and overall productivity.