VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY

INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Project: Task Management System Database

Team's Members:

Nguyễn Quỳnh Anh - ITCSIU24005

Nguyễn Dương Đức Thịnh - ITCSIU24081

Noh Anh Kiệt - ITDSIU23011

Tôn Duy Thành - ITCSIU21234

Đoàn Hiển Long - ITCSIU23023

Course: IT079IU - Principles of Database Management

Instructor: Assoc. Prof. Dr. N.T.T.Loan

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1. Abstract

For our course on Principles of Database Management, our group proposes the development of the project: Task Management System Database. The goal of this project is to design and implement a relational database that effectively supports the organization, assignment, and monitoring of tasks within a collaborative team environment. In a real-world situation, many projects require teams to manage and allocate responsibilities, set deadlines, and track progress to ensure efficiency and accountability of members within a group. Our proposed system will address these issues by providing a structured database where users can create and manage projects, assign tasks to specific members, set priorities and due dates, and generate reports on task completion status.

This project will put emphasis on the application of database concepts learned in this course, such as entity-relationship (ER) modeling, data normalization, and SQL query design to ensure consistency, efficiency, and scalability of this system. By building a functional backend for this project, our goal is to demonstrate not only our theoretical knowledge but also our ability to create a solution to practical data management. Ultimately, this project highlights how database systems play a crucial role in supporting productivity and collaboration in modern organizations.

2. System Overview

The Task Management System is designed to provide a robust foundation for managing tasks, users, and team collaboration through a comprehensive digital platform. Built using Spring Boot and MySQL, the system serves as a centralized hub that streamlines project workflows by offering features for user authentication, task lifecycle management, and activity tracking.

Key Features & Functionalities:

Task Lifecycle Management: Enables complete CRUD (Create, Read, Update, Delete) operations for tasks, supporting workflows from "Pending" to "Done" with distinct priority levels (Low to Urgent).

Role-Based Access Control (RBAC): Differentiates between "Group Leaders" and "Members," ensuring appropriate access rights where leaders can manage team assignments while members focus on execution.

Categorization & Organization: Utilizes a flexible system of categories and tags to label and organize tasks effectively, aiding in quick retrieval and filtering.

Activity Tracking & Audit: Maintains a comprehensive activity_log that records all task-related activities (updates, deletions, assignments), ensuring transparency and accountability.

Data Integrity & Safety: Implements "soft delete" functionality to prevent accidental permanent data loss and utilizes specific constraints to maintain referential integrity between users and tasks.

Dual Interface Support: Designed with a layered architecture that supports both a direct web interface (via Thymeleaf for testing) and RESTful API endpoints to serve modern frontend frameworks.

3. Project Goals

The Task Management System is developed with the following objectives:

Streamlining Team Collaboration

- Provide a transparent platform where team members can view assigned tasks and track overall progress.
- Facilitate clear assignment of duties using a many-to-many relationship model between tasks and users.

Ensuring Data Integrity & Reliability

- Implement a normalized database schema (3NF) to minimize redundancy and prevent data anomalies.
- Enforce strict referential integrity via foreign keys and type safety through Enum-based status and priority systems.

Establishing a Scalable Architecture

- Create a maintainable backend using Spring Boot's layered architecture (Controller, Service, Repository, Entity).
- Ensure the system is extensible, allowing for future integrations such as file uploads and email notifications.

Enhancing Security & User Management

- Secure user credentials using BCrypt password encryption during registration and authentication.
- Provide robust input validation to prevent malformed data entry.

Optimizing Performance

- Address common performance pitfalls such as the "N+1 query problem" through strategic fetching strategies (Eager vs. Lazy loading).
- Implement distinct Data Transfer Objects (DTOs) to optimize API responses and avoid circular reference issues.

4. Techniques and Tools used

To develop the Task Management System, we selected a technology stack that prioritizes rapid development, type safety, and industry-standard best practices.

- **Programming Languages**
    - **Java:** Used for backend logic, utilizing Spring Boot 3.2.0 for rapid application development.

- **SQL:** Used for defining the database schema and managing relational data within MySQL.
- **TypeScript:** Used for building the frontend web interface to ensure type safety and interactivity.

- **Database Management System (DBMS)**
  - **MySQL 8.0:** The primary relational database system used for persistent storage of users, tasks, and logs.
- **Development Environment**
  - **IntelliJ IDEA:** The primary IDE for Java backend development, offering advanced debugging and Spring support.
  - **Visual Studio Code (VS Code):** Utilized for frontend development and script editing.
  - **MySQL Workbench:** Used to design the database schema, create tables, and execute verification queries. [9]
- **Frameworks & Libraries**
  - **Spring Boot 3.2.0:** The core framework provides embedded server capabilities and dependency management.
  - **Spring Data JPA / Hibernate:** Handles Object-Relational Mapping (ORM) for seamless database interactions.
  - **Lombok:** Reduces boilerplate code (getters, setters, constructors) through annotations.
  - **Thymeleaf:** A server-side template engine used for initial UI rendering and backend logic validation.
- **Database Design Tools**
  - **ERDPlus:** Used to create the conceptual Entity-Relationship Diagrams.
  - **MermaidChart:** Employed to visualize the Relational Schema and query trees.
- **Version Control & Collaboration**
  - **GitHub:** Hosted the source code repository for version control and team collaboration.

    - **Google Docs:** Facilitated collaborative report writing and documentation.

5. Scope and Limitations

Scope: The current implementation covers the core backend infrastructure, including user registration, secure authentication, full CRUD capabilities for tasks, and an audit logging system. It supports a standard task workflow (Pending to Done) and categorization.

Limitations:

- **Advanced Security:** While basic authentication is in place, full JWT-based stateless authentication is currently a planned enhancement.
- **Feature Set:** Advanced features like file attachments, email notifications, and recurring tasks are outlined for future phases and not yet implemented.
- **Frontend Integration:** The current React frontend integration is in early stages; Thymeleaf was used for immediate backend validation to overcome initial integration challenges.

6. Structure of the report

Chapter 1: Introduction: Outlines the project background, objectives, and the selected technology stack.

Chapter 2: Database Design: Details the Entity-Relationship Diagram (ERD), Relational Model, and data integrity constraints.

Chapter 3: Architecture & Implementation: Describes the Spring Boot layered architecture, key code components (Entities, Repositories, Controllers), and design decisions.

Chapter 4: Results & Analysis: Presents the implemented features, testing results, and analysis of the system's performance.

Chapter 5: Conclusion & Future Work: Summarizes the project outcomes and discusses potential future enhancements such as JWT authentication and API documentation.

Chapter 6: Reference: Lists all the materials the group have referred to complete the whole project.

CHAPTER 2: TASK TIMELINE & DIVISION

1. Project Timeline

- **Phase 1: Research, Planning, and Initial Design (Weeks 1-4)**

We spent three weeks on Requirement Analysis and Topic Selection, defining key features and understanding timelines. The project flow consists of two user flows. The leader is responsible for managing, creating, editing, and assigning tasks, while the members can only view, update statuses, and write comments on the task. The goal of the whole application is to provide a user-friendly, transparent workflow. The final requirements were completed by Week 4.

- **Phase 2: Database Implementation and Query Development (Weeks 5-7)**

In weeks 5 to 7, we designed the Entity Relationship Diagram (ERD) and then converted it into a Relational Model, following Lecture 3's mapping rules (detailed in Chapter 3, Sections 2 and 3).

Table Creation involved using SQL CREATE TABLE statements based on the relational model. Tables included appropriate data types, primary/foreign keys, and constraints (UNIQUE, NOT NULL, ENUM). InnoDB was used. **Data Collection and Insertion** involved manually generating sample data (4 Users, 4 Categories, 8 Tags, 3 Tasks, and 6 Comments) to ensure referential integrity, inserted in a dependency-based order. **SQL Queries, Relational Algebra, and Tree Queries** were developed for application features (authentication, filtering, tracking). Each SQL query was accompanied by its Relational Algebra equivalent and a conceptual Query Tree.

- **Phase 3: Development of Web Application, Test, and Project Finalization (Weeks 8, 10-11)**

Website Structure Development included a Java/Spring Boot REST API backend using a layered architecture and a Typescript/Next.js (React) frontend with a minimalistic design, implementing essential functions like Task Assignments, Kanban Board, and Task History.

This phase also involved Methodology Review and Refinement (Chapter 3) and Implementation Process Documentation and Results Analysis to confirm the system met initial requirements.

2. Contribution Breakdown

Each member's final contribution percentage will be calculated as the sum of the scores achieved across those below categories, and then re-calculated so that the sum of all members of the group equals 100%.

1. Phase Contribution (60%): This ensures each member's existence and participation within the group for each phase of the proposal.

- Proposal (15%)
- Midterm Report (20%)
- Final Report (25%)

2. Individual Contribution (30%): This criteria shows the specific technical and managerial efforts contributed by the individual member.

- **Contribution to Coding (15%):** Evaluates the functionality, logic, efficiency, and volume of code developed by the member.
- **Contribution to Report (10%):** Assesses the thoroughness, academic writing style, and structure of the written sections assigned to the member.
- **Contribution to Team Management (5%):** Measures active participation in task distribution, conflict resolution, meeting organization, and supporting other members.

3. Presentation Slide Preparation (10%): Credits the design, visual appeal, logical flow, and content organization of the final presentation slides.

Table 2.1: Contribution Breakdown

Member's Name	Student's ID	Evaluation in the respective	Total	Final

Nguyễn Quỳnh Anh	ITCSIU24005	15%, 20%, 25%, 5%, 10%, 5%, 10%	90%	20.5%
Nguyễn Dương Đức Thịnh	ITCSIU24081	15%, 20%, 25%, 15%, 5%, 0%, 10%	90%	20.5%
Noh Anh Kiệt	ITDSIU23011	15%, 20%, 25%, 15%, 10%, 0%, 10%	95%	21%
Tôn Duy Thành	ITCSIU21234	15%, 20%, 25%, 10%, 5%, 0%, 10%	85%	19.5%
Đoàn Hiển Long	ITCSIU23023	5%, 20%, 25%, 15%, 5%, 0%, 10%	80%	18.5%

3. Task Allocation Table

Table 2.2: Task Allocation Table

TASKS	ASSIGNED MEMBER	DETAILS
WEEK 01		
Choose a topic and discuss	All members	Discuss about each others' experiences Choose topic in the given list
Manage proposal report	Noh Anh Kiệt	Write abstract and brief introduction of the topic Manage and format the report according to the guideline

Task Allocation Table	Nguyễn Quỳnh Anh	A detailed breakdown of tasks by week and by team member. The more specific the allocation, the higher the score.
Tools and Programming Languages	Tôn Duy Thành	List the software, tools, and languages that will be used.
	Nguyễn Dương Đức Thịnh	
Final Proposal Report Compilation	All members	Make final adjustments and submit the report
WEEK 02		
Research application ideas	Noh Anh Kiệt	List features: list pages, features needed, and use cases for each
	Nguyễn Quỳnh Anh	
Sketch ERD	Nguyễn Dương Đức Thịnh	Sketch the entities and attributes
Coding Research	Đoàn Hiển Long	Know how to use each tech stack for different parts
	Tôn Duy Thành	
Github Setup	Nguyễn Quỳnh Anh	Create Github repository for team

WEEK 03		
Identify entities	All members	From use cases, • Define scope & rules • Identify entities (include weak entities) • List attributes & pick key • Identify relationships & their attributes • Set cardinality & participation • Model hierarchies
Learn foundations	All members	Learn about relational algebra, Tree, SQL, and clearly understand the topic
WEEK 04		
Design Entity-Relationship Diagram (ERD)	Nguyễn Dương Đức Thịnh	Build an ER Diagram representing the entities and relationships
	Đoàn Hiển Long	
Design a Schema Diagram	Noh Anh Kiệt	Build a schema diagram showing the logical structure of the database
WEEK 05		
Write Midterm Report	Tôn Duy Thành	Update information - revisions and improvements to the proposal
Check and finalize	Nguyễn Quỳnh Anh	Check the accuracy of the report Make final adjustments and submit the report

WEEK 06		
Finalize ER Diagram	Nguyễn Dương Đức Thịnh	Finalize the draft versions of ERD
Data Collection and Insertion	Noh Anh Kiệt	Research on methods used to populate the database
WEEK 07		
Table Normalization	Đoàn Hiển Long	Normalize table format, ensuring Boyce-Codd Normal Form (BCNF) - a stricter format of Third Normal Form (3NF)
	Nguyễn Quỳnh Anh	
Finalize Schema Diagram	Noh Anh Kiệt	Finalize the draft versions of Schema Diagram
Write Methodology - Website Structure	Nguyễn Dương Đức Thịnh	List the tech stack and discuss how to integrate into the project
	Tôn Duy Thành	
WEEK 08 - 09		
Build User Interface	Nguyễn Dương Đức Thịnh	Build, test, and debug for each use case that has been discussed
	Tôn Duy Thành	
	Nguyễn Quỳnh Anh	
Database Connection	Noh Anh Kiệt	Connect to Database and work on implementing database for each feature

Implementation	Đoàn Hiển Long	of the website
WEEK 10		
Finalize Methodology - Database Design	Tôn Duy Thành	Finalize the final report and check for errors
Finalize Methodology - Website Structure	Nguyễn Quỳnh Anh	
Finalize Final Report	All members	
WEEK 11		
Prepare Presentation	Noh Anh Kiệt	Prepare slides, scripts for representation
	Tôn Duy Thành	
	Đoàn Hiển Long	
	Nguyễn Dương Đức Thịnh	
Record a small demo (optional)	All members	Record a small demo for each use case in case that there is any errors occur on the demo day
Evaluate contribution	Nguyễn Quỳnh Anh	Evaluate the members’ contributions and write a report on their responsibilities

CHAPTER 3: METHODOLOGY

1. Entity Relationship Diagram (ERD)

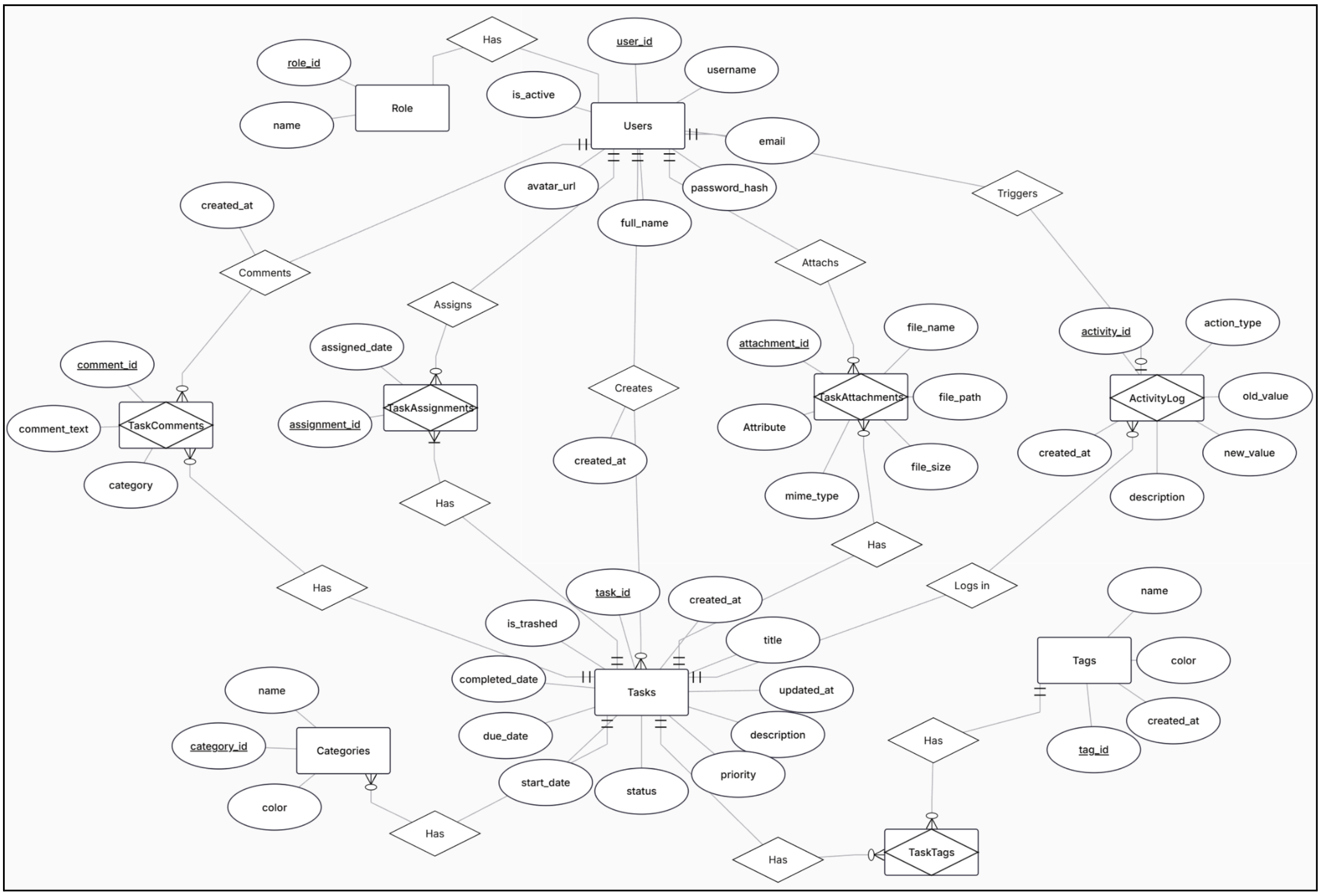


Figure 3.1: ERD for Task Management System Database

1.1. Entities and Attributes Descriptions

Users

- **Description:** Represents system users with login credentials, personal details, and access roles.
- **Key Attribute:** user_id - User ID (Primary Key).
- **Basic Attributes:**

    - username - Unique identifier for logging in.
    - email - Unique email address for communication and recovery.
    - password_hash - Encrypted password for security.
    - full_name - User's legal or display name.
    - role - Access level (categotized into: 1 - 'GROUP_LEADER', 2 - 'MEMBER').
    - status - Current account state (ENUM: 'ACTIVE', 'INACTIVE').
    - avatar_color - Hex color code for the user's default avatar.
    - created_at - Timestamp when the account was registered.

Categories

- **Description:** Represents the classification groups for tasks (e.g., Development, Design).
- **Key Attribute:** category_id - Category ID (Primary Key).
- **Basic Attributes:**
    - name - Unique category name.
    - color - Hex color code for visual distinction.

Tasks

- **Description:** Represents work items created, assigned, and tracked within the system.
- **Key Attribute:** task_id - Task ID (Primary Key).
- **Basic Attributes:**
    - title - A brief summary of the task.
    - description - Detailed requirements or instructions.
    - status - Workflow state (ENUM: 'PENDING', 'TO_DO', 'IN_PROGRESS', 'DONE').
    - priority - Urgency level (ENUM: 'LOW', 'MEDIUM', 'HIGH', 'URGENT').
    - start_date - Scheduled start date.
    - due_date - Deadline for completion.

    - category_id - Foreign key linking to the Category entity.
    - is_deleted - Flag indicating if the task is in the trash (Soft Delete).
    - created_by - ID of the user who created the task.
    - created_at - Timestamp of creation.
    - updated_at - Timestamp of the last modification.

TaskAssignment

- **Description:** A junction entity representing the assignment of tasks to specific users.
- **Key Attribute:** assignment_id - Assignment ID (Primary Key).
- **Basic Attributes:**
    - assigned_at - Timestamp when the user was assigned to the task.

TaskComments

- **Description:** Represents discussions, questions, or feedback related to a specific task.
- **Key Attribute:** comment_id - Comment ID (Primary Key).
- **Basic Attributes:**
    - comment_text - The content of the comment.
    - category - Classification of the comment (e.g., 'QUESTION', 'FEEDBACK').
    - created_at - Timestamp when the comment was posted.

Tags

- **Description:** Labels used to organize tasks with specific keywords (e.g., 'Frontend', 'Urgent').
- **Key Attribute:** tag_id - Tag ID (Primary Key).
- **Basic Attributes:**
    - name - Unique tag text.
    - color - Hex color code for the tag.

TaskAttachments

- **Description:** Stores metadata for files uploaded and attached to tasks.
- **Key Attribute:** attachment_id - Attachment ID (Primary Key).
- **Basic Attributes:**
    - file_name - Original name of the uploaded file.
    - file_path - Storage location/URL of the file.
    - file_size - Size of the file in bytes.
    - uploaded_at - Timestamp of the upload.

Role

- **Description:** Store the detailed of the specific access role.
- **Key Attribute:** role_id - Role ID (Primary Key).
- **Basic Attributes:**
    - role_name - Name of the role.

ActivityLog

- **Description:** Records a history of actions taken within the system for audit trails (replacing simple notifications).
- **Key Attribute:** activity_id - Activity ID (Primary Key).
- **Basic Attributes:**
    - action_type - Type of action (e.g., 'CREATED', 'UPDATED', 'STATUS_CHANGED').
    - description - Human-readable summary of the event.
    - old_value / new_value - Data snapshots before and after the change.
    - created_at - Timestamp when the action occurred.

1.2. Relationship Descriptions

Users - TaskAssignment (One-to-Many)

- **Description:** A User can be assigned to multiple tasks via the assignment records. Conversely, each TaskAssignment record identifies exactly one responsible User.
- **Constraints:**
    - TaskAssignment[user_id] ⊆ Users[user_id].
    - ON DELETE CASCADE: If a user is deleted, their assignments are removed.

Tasks - TaskAssignment (One-to-Many)

- **Description:** A Task can have multiple assignees (users). Each assignment record links one specific Task to one User.
- **Constraints:**
    - TaskAssignment[task_id] ⊆ Tasks[task_id].
    - Unique Constraint: A specific pair of (task_id, user_id) can exist only once.

Categories - Tasks (One-to-Many)

- **Description:** A Category can group multiple Tasks (e.g., "Development" has 10 tasks). However, a Task belongs to only one Category at a time.
- **Constraints:**
    - Tasks[category_id] ⊆ Categories[category_id].
    - ON DELETE SET NULL: If a category is deleted, the task remains but has no category.

Users - Tasks (One-to-Many)

- **Description:** A User (specifically a Leader) creates Tasks. One User can create multiple Tasks, but a Task is created by exactly one User.
- **Constraints:**
    - Tasks[created_by] ⊆ Users[user_id].

Tasks - TaskComments (One-to-Many)

- **Description:** A Task acts as a container for a discussion thread. It can have multiple Comments, but each Comment belongs to exactly one Task.
- **Constraints:**
    - TaskComments[task_id] ⊆ Tasks[task_id].

Users - TaskComments (One-to-Many)

- **Description:** A User can post multiple Comments across different tasks. Each Comment acts as a record of who said what.
- **Constraints:**
    - TaskComments[user_id] ⊆ Users[user_id].

Tasks - Tags (Many-to-Many)

- **Description:** A Task can be labeled with multiple Tags, and a Tag can be applied to multiple Tasks. This relationship is resolved physically via the task_tags junction table.
- **Constraints:**
    - TaskTags links Tasks[task_id] and Tags[tag_id].

Tasks - ActivityLog (One-to-Many)

- **Description:** Changes made to a Task trigger entries in the Activity Log. A Task can have a comprehensive history of multiple log entries.
- **Constraints:**
    - ActivityLog[task_id] ⊆ Tasks[task_id].

Users - Roles (Many-to-One)

- **Description:** A user must have one role. A role is a label for many users.
- **Constraints:**
    - Users[role_id] ⊆ Roles[role_id]

2. Schema Diagram

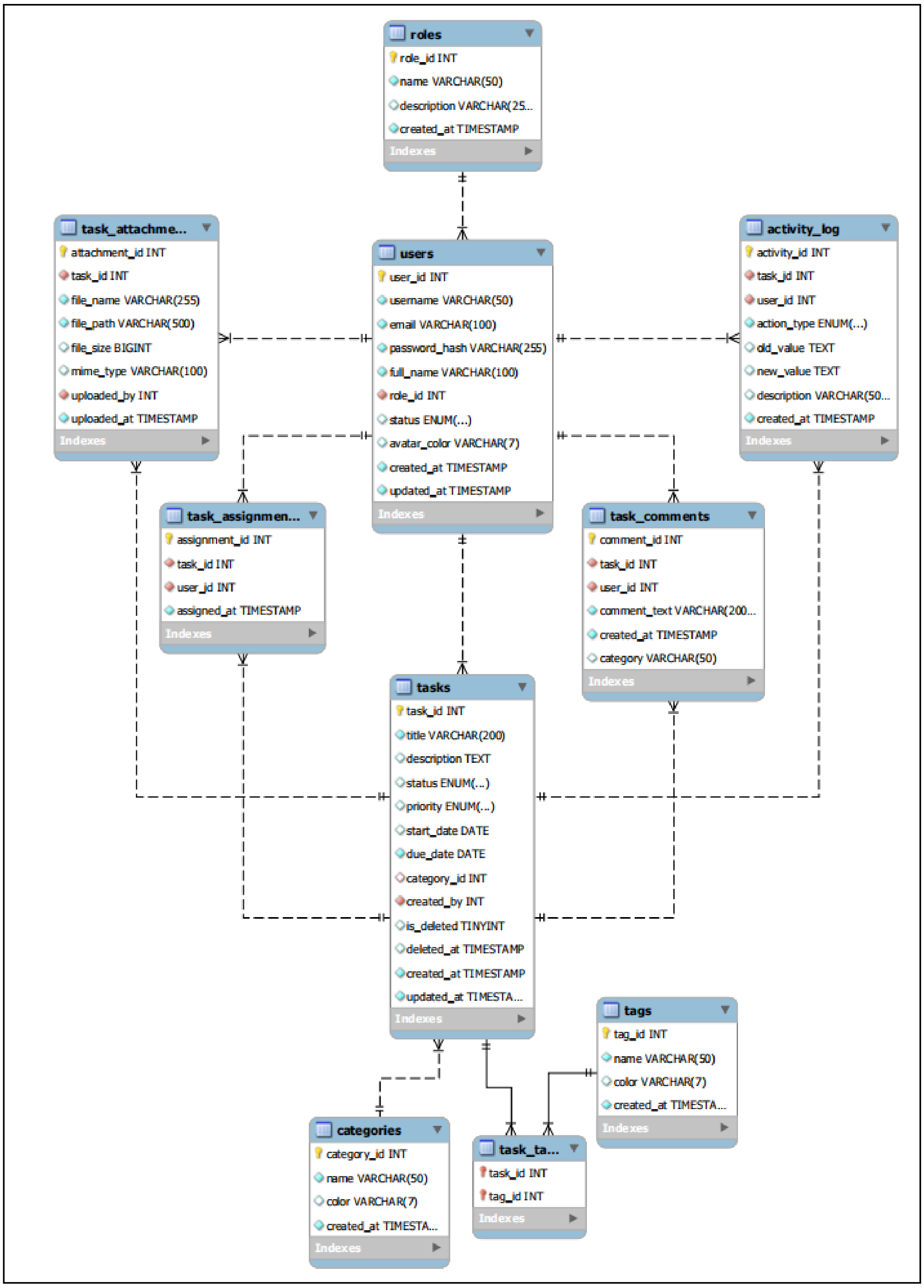


Figure 3.2: Schema Diagram for Task Management System Database [10]

2.1. Schema Definition

- **Roles** (role_id (PK), name, description, created_at)
- **Users** (user_id (PK), username, email, password_hash, full_name, role_id (FK), status, avatar_color, created_at, updated_at)
- **Categories** (category_id (PK), name, color, created_at)
- **Tasks** (task_id (PK), title, description, status, priority, start_date, due_date, category_id (FK), created_by (FK), is_deleted, deleted_at, created_at, updated_at)
- **Task_Assignments** (assignment_id (PK), task_id (FK), user_id (FK), assigned_at)
- **Tags** (tag_id (PK), name, color, created_at)
- **Task_Tags** (task_id (PK), tag_id (PK)) *[Composite PK: task_id, tag_id]*
- **Task_Attachments** (attachment_id (PK), task_id (FK), file_name, file_path, file_size, mime_type, uploaded_by (FK), uploaded_at)
- **Activity_Log** (activity_id (PK), task_id (FK), user_id (FK), action_type, old_value, new_value, description, created_at)
- **Task_Comments** (comment_id (PK), task_id (FK), user_id (FK), comment_text, parent_comment_id (FK), category, created_at)

2.2. Referential Integrity Constraints

The following inclusion dependencies ensure data consistency across relationships:

- Users[role_id] ⊆ Roles[role_id]
- Tasks[category_id] ⊆ Categories[category_id]
- Tasks[created_by] ⊆ Users[user_id]
- Task_Assignments[task_id] ⊆ Tasks[task_id]
- Task_Assignments[user_id] ⊆ Users[user_id]
- Task_Tags[task_id] ⊆ Tasks[task_id]

- Task_Tags[tag_id] ⊆ Tags[tag_id]
- Task_Attachments[task_id] ⊆ Tasks task_id]
- Task_Attachments[uploaded_by] ⊆ Users[user_id]
- Activity_Log task_id] ⊆ Tasks[task_id]
- Activity_Log[user_id] ⊆ Users[user_id]
- Task_Comments[task_id] ⊆ Tasks[task_id]
- Task_Comments[user_id] ⊆ Users[user_id]
- Task_Comments[parent_comment_id] ⊆ Task_Comments[comment_id]

2.3. Unique Keys

To enforce business rules and prevent duplicate data, the following unique constraints were applied:

- Roles[name]
- Users[username]
- Users [email]
- Categories[name]
- Tags[name]
- Task_Assignments[task_id, user_id] (Ensures a user is not assigned the same task twice)

3. Table Creation and Normalization

3.1. Table Creation

- Role Table:

```
CREATE TABLE roles (
   role_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(50) UNIQUE NOT NULL,
   description VARCHAR(255),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
```

```
    ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Users Table:

```
CREATE TABLE users (
    user_id INT AUTO_INCREMENT PRIMARY KEY,
    username VARCHAR(50) UNIQUE NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    full_name VARCHAR(100) NOT NULL,
    role_id INT NOT NULL,
    status ENUM('ACTIVE', 'INACTIVE') DEFAULT 'ACTIVE',
    avatar_color VARCHAR(7),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (role_id) REFERENCES roles(role_id),
    INDEX idx_username (username),
    INDEX idx_email (email),
    INDEX idx_role_id (role_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Categories Table:

```
CREATE TABLE categories (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    color VARCHAR(7),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Tasks Table

```
CREATE TABLE tasks (
    task_id INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(200) NOT NULL,
    description TEXT,
    status ENUM('PENDING', 'TO_DO', 'IN_PROGRESS', 'DONE') DEFAULT
'PENDING',
    priority ENUM('LOW', 'MEDIUM', 'HIGH', 'URGENT') DEFAULT 'MEDIUM',
    start_date DATE,
    due_date DATE NOT NULL,
    category_id INT,
    created_by INT NOT NULL,
    is_deleted BOOLEAN DEFAULT FALSE,
    deleted_at TIMESTAMP NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES categories(category_id) ON DELETE
SET NULL,
    FOREIGN KEY (created_by) REFERENCES users(user_id),
    INDEX idx_status (status),
    INDEX idx_priority (priority),
    INDEX idx_due_date (due_date),
    INDEX idx_is_deleted (is_deleted),
    INDEX idx_created_by (created_by)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Task Assignments Table

```
CREATE TABLE task_assignments (
    assignment_id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    user_id INT NOT NULL,
    assigned_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (task_id) REFERENCES tasks(task_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    UNIQUE KEY unique_assignment (task_id, user_id),
    INDEX idx_task_id (task_id),
    INDEX idx_user_id (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Tags Table

```
CREATE TABLE tags (
    tag_id INT AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(50) UNIQUE NOT NULL,
    color VARCHAR(7),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Task Tags Table

```
CREATE TABLE task_tags (
    task_id INT NOT NULL,
    tag_id INT NOT NULL,
    PRIMARY KEY (task_id, tag_id),
    FOREIGN KEY (task_id) REFERENCES tasks(task_id) ON DELETE CASCADE,
    FOREIGN KEY (tag_id) REFERENCES tags(tag_id) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Task Attachment Table

```
CREATE TABLE task_attachments (
    attachment_id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    file_name VARCHAR(255) NOT NULL,
    file_path VARCHAR(500) NOT NULL,
    file_size BIGINT,
    mime_type VARCHAR(100),
    uploaded_by INT NOT NULL,
    uploaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (task_id) REFERENCES tasks(task_id) ON DELETE CASCADE,
    FOREIGN KEY (uploaded_by) REFERENCES users(user_id),
    INDEX idx_task_id (task_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Activity Log Table

```
CREATE TABLE activity_log (
    activity_id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    user_id INT NOT NULL,
    action_type ENUM('CREATED', 'UPDATED', 'STATUS_CHANGED', 'ASSIGNED',
                'DELETED', 'RESTORED', 'FILE_UPLOADED', 'FILE_REMOVED')
NOT NULL,
    old_value VARCHAR(255),
    new_value VARCHAR(255),
    description VARCHAR(500),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (task_id) REFERENCES tasks(task_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id),
    INDEX idx_task_id (task_id),
    INDEX idx_user_id (user_id),
    INDEX idx_created_at (created_at)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

- Task Comments Table

```
CREATE TABLE task_comments (
    comment_id INT AUTO_INCREMENT PRIMARY KEY,
    task_id INT NOT NULL,
    user_id INT NOT NULL,
    comment_text VARCHAR(2000) NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    category VARCHAR(50),
    FOREIGN KEY (task_id) REFERENCES tasks(task_id) ON DELETE CASCADE,
    FOREIGN KEY (user_id) REFERENCES users(user_id) ON DELETE CASCADE,
    INDEX idx_task_id (task_id),
    INDEX idx_user_id (user_id)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

3.2. Normalization Analysis

The design of the Task Management System database adheres to the following normal forms:

Table 3.1: Definitions normal forms [1] [7]

Normal Form	**Description**
1NF	A relation is in first normal form if every attribute in every row can contain only one single (atomic) value.
2NF	A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key

3NF	A table is in 3NF if it is in 2NF and for each functional dependency X → Y at least one of the following conditions hold: ● X is a super key of table ● Y is a prime attribute of table An attribute that is a part of one of the candidate keys is known as a prime attribute.
BCNF	A table complies with BCNF: ● It is in 3NF and ● for every functional dependency X → Y, X should be the superkey of the table.

- **First Normal Form (1NF):**
  - **Requirement:** The table must have a primary key, and all attributes (columns) must contain atomic (indivisible) values. There should be no repeating groups of columns.
  - **Compliance:** All tables (users, tasks, categories, etc.) have one PRIMARY KEY (e.g., user_id, task_id). Each attribute in each table only holds one value (e.g., users.email holds one email address, tasks.priority holds one specific ENUM value). To handle multiple values, such as a task having multiple tags or multiple assignees, we created junction tables (task_tags and task_assignments) where they store the multiple values from two tables, rather than storing comma-separated lists in the tasks table. Thus, the schema satisfies 1NF.
- **Second Normal Form (2NF):**

- **Requirement:** The table must be in 1NF, and all non-key attributes must be fully functionally dependent on the entire primary key. This form primarily addresses potential issues in tables with composite primary keys.
- **Compliance:** The schema meets 1NF requirements. Furthermore, the core entity tables use single-attribute surrogate keys (e.g., users.user_id, categories.category_id, tasks.task_id). Because these primary keys consist of a single column, it is impossible for a non-key attribute to depend on only "part" of the key.
- For the table with a composite primary key, such as task_tags (PK: task_id, tag_id), there are no non-key attributes in that table to violate partial dependency. It strictly serves as a linking table. Therefore, the schema automatically satisfies 2NF.

- **Third Normal Form (3NF):**
  - **Requirement:** The table must be in 2NF, and there should be no transitive dependencies. A transitive dependency exists when a non-key attribute is functionally dependent on another non-key attribute, rather than directly on the primary key.
  - **Compliance:** The schema adheres to 2NF. We analyze for transitive dependencies in key tables:
    - **Example: Tasks Table:** The primary key is task_id. Attributes like title, description, status, priority, and due_date depend directly on the task_id. The category_id and created_by are foreign keys. Importantly, details about the category (like category.name or category.color) are not stored in the tasks table; they reside in the categories table. This prevents a transitive dependency where tasks.category_name would depend on tasks.category_id, which in turn depends on tasks.task_id.

        - **Example: Task_Assignments Table:** The primary key is assignment_id. The attribute assigned_at depends directly on this specific assignment instance. The user_id is a foreign key linking to the users table. Information about the user (such as full_name or role) is stored in the users table, ensuring that user details are not redundantly stored within the assignment record.
    - **Conclusion for 3NF:** A similar analysis confirms that in all tables (tags, task_attachments, task_comments), non-key attributes are dependent only on the primary key. Foreign keys are strictly used to link related data residing in other tables. Therefore, the database design satisfies the Third Normal Form (3NF).
- **Boyce-Codd Normal Form (BCNF):**
    - **Requirement:** The table must satisfy 3NF, and for every non-trivial functional dependency $X \rightarrow Y$, the determinant X must be a superkey (or candidate key). In simpler terms, the only entity that can determine other columns must be a valid unique key of the table. This addresses specific anomalies in tables with multiple overlapping candidate keys that 3NF might miss.
    - **Compliance:** The database schema adheres to BCNF standards. We analyzed tables with multiple candidate keys to ensure compliance:
        - **Example: Users Table:** This table has a Primary Key (user_id) and two candidate keys defined by UNIQUE constraints (username and email). Any functional dependency, such as username determining the password_hash or email determining the full_name, is valid because both username and email are candidate keys. There are no dependencies where a non-key attribute determines part of a candidate key.

- **Example: Task Assignments Table:** This table has a surrogate Primary Key (assignment_id) and a composite Unique Key (task_id, user_id). The attribute assigned_at is dependent on assignment_id (the PK) and also fully dependent on the pair (task_id, user_id). Since both the assignment_id and the combination of task_id + user_id are valid candidate keys, the table satisfies BCNF.
- **Conclusion:** Since all determinants in our schema are candidate keys, the design satisfies Boyce-Codd Normal Form.

4. Data Collection and Insertion

4.1. Data Collection

All the data used in this Task Management System database is completely synthetic, created by our team members and testers. We made up realistic user accounts, tasks with different priorities and deadlines, assignments for one or more people, comments and replies, notifications for reminders and updates, and even some deleted tasks to show the trash feature.

4.2. Data Insertion [2] [6]

The following SQL queries must be executed in the same order because mixing them up causes errors and prevents the entire data or table schema from being created correctly.

- Role Table:

```
INSERT INTO roles (name, description) VALUES

('GROUP_LEADER', 'Quản lý nhóm, có quyền phân công task'),

('MEMBER', 'Thành viên, thực hiện task được giao');
```

- Users Table:

```
-- Insert demo users

INSERT INTO users (username, email, password_hash, full_name, role, avatar_color)
VALUES

('leader_a', 'leader.a@example.com',
'$2a$10$xn3LI/AjqicFYZFruSwve.681477XaVNaUQbr1gioaWPn4t1KsnmG', 'Nguyễn
Văn A', 'GROUP_LEADER', '#5B8DEF'),

('member_b', 'member.b@example.com',
'$2a$10$xn3LI/AjqicFYZFruSwve.681477XaVNaUQbr1gioaWPn4t1KsnmG', 'Trần
Thị B', 'MEMBER', '#5ECFB1'),

('member_c', 'member.c@example.com',
'$2a$10$xn3LI/AjqicFYZFruSwve.681477XaVNaUQbr1gioaWPn4t1KsnmG', 'Lê Văn
C', 'MEMBER', '#F5A864'),

('leader_d', 'leader.d@example.com',
'$2a$10$xn3LI/AjqicFYZFruSwve.681477XaVNaUQbr1gioaWPn4t1KsnmG', 'Phạm
Thị D', 'GROUP_LEADER', '#F56565');
```

- Tasks Table

```
-- Insert demo tasks

INSERT INTO tasks (title, description, status, priority, start_date, due_date, category_id,
created_by) VALUES

('Phân tích yêu cầu hệ thống', 'Thu thập và phân tích chi tiết yêu cầu người dùng cho dự án
mới.', 'DONE', 'HIGH', '2025-11-10', '2025-11-15', 1, 1),
('Thiết kế cơ sở dữ liệu', 'Xây dựng mô hình ERD và tạo script database.',
'IN_PROGRESS', 'URGENT', '2025-11-15', '2025-11-20', 1, 1),

('Phát triển module đăng nhập', 'Xây dựng giao diện và logic xử lý đăng nhập/đăng ký.',
'TO_DO', 'HIGH', '2025-11-20', '2025-11-25', 1, 2);
```

- Task Assignments Table

```
-- Assign tasks to users

INSERT INTO task_assignments (task_id, user_id) VALUES

(1, 1),

(2, 2),

(2, 3),

(3, 2);
```

- Categories Table

```
-- Insert categories

INSERT INTO categories (name, color) VALUES

('Development', '#3182CE'),

('Design', '#9F7AEA'),

('Testing', '#48BB78'),

('Documentation', '#ED8936');
```

- Tags Table

```sql
-- Insert tags

INSERT INTO tags (name, color) VALUES

('urgent', '#F56565'),

('frontend', '#4299E1'),

('backend', '#48BB78'),

('database', '#9F7AEA'),

('design', '#ED8936'),

('documentation', '#38B2AC'),

('analysis', '#DD6B20'),

('authentication', '#667EEA');
```

- Tag Tasks Table

```sql
-- Add tags to tasks

INSERT INTO task_tags (task_id, tag_id) VALUES

(1, 6), -- documentation

(1, 7), -- analysis

(2, 3), -- database

(2, 5), -- design
```

```
(2, 1), -- urgent

(3, 1), -- urgent

(3, 2), -- frontend

(3, 3), -- backend

(3, 8); -- authentication
```

```
INSERT INTO task_comments (task_id, user_id, comment_text, category) VALUES
-- Bình luận cũ
(1, 2, 'Phần yêu cầu phi chức năng cần làm rõ thêm về hiệu năng.', 'QUESTION'),
(1, 3, 'Tôi đã cập nhật tài liệu version 2.0, mọi người kiểm tra nhé.', 'UPDATE'),
(2, 1, 'Cấu trúc bảng Users cần thêm field avatar_color không?', 'QUESTION'),

-- Các bình luận trước đây là Reply, giờ thành bình luận thường
(1, 1, 'Đồng ý, chúng ta sẽ họp về phần này vào sáng mai.', 'REPLY'),
(2, 2, 'Có nhé, nên thêm vào để UI hiển thị đẹp hơn.', 'REPLY'),

-- Một bình luận khác
(2, 3, 'Đã thêm field avatar_color vào thiết kế.', 'RESOLVED');
```

5. Database Queries: Relational Algebra, Tree, and SQL Queries

This task showcases the process of retrieving and manipulating data within the Task Management System database. We provide several example queries that highlight typical application functionalities, illustrating their implementation in SQL (Structured Query Language), their theoretical representation through Relational Algebra, and their conceptual execution plan depicted as Query Trees.

Below are examples of common data retrieval tasks relevant to the system. For each example, we provide:

1. **Purpose:** A description of the information being retrieved and its relevance to an application feature.
2. **SQL Query:** The standard SQL statement used to execute the query against the MySQL database.
3. **Relational Algebra:** The equivalent expression using relational algebra notation (σ: selection, π: projection, ⋈: join, γ: aggregation).
4. **Query Tree:** A representation of the execution plan, showing the order of operations (Tables are leaves, operations are internal nodes).

Example Query 1: Retrieve User Information by Username

- **Purpose:** (Related to Feature: Authentication) To fetch basic user details (ID, email, role, full name) based on a username. This is essential to identify current user session and permission levels.
- **SQL Query:**

```
SELECT user_id, email, full_name, role
FROM users
WHERE username = 'leader_a';
```

- **Relational Algebra:**

$$\pi_{user\_id,\, email,\, full\_name,\, role}(\sigma_{username='leader\_a'}(users))$$

- **Query Tree:**

```
π (user_id, email, full_name, role)
            |
      σ (username = 'leader_a')
            |
          users (U)
```

Example Query 2: Filter Tasks by Status (To-Do List)

- **Purpose:** (Related to Feature: Kanban Board) To retrieve all active tasks that are currently in the 'TO_DO' column. This allows the system to render the specific column of the task board.
- **SQL Query:**

```sql
SELECT task_id, title, priority, due_date
FROM tasks
WHERE status = 'TO_DO' AND is_deleted = FALSE;
```

- **Relational Algebra:**

$$\pi_{task\_id,\, title,\, priority,\, due\_date}(\sigma_{status\,=\,'TO\_DO' \wedge is\_deleted=FALSE}(tasks))$$

- **Query Tree:**

```
π (task_id, title, priority, due_date)
                   |
 σ (status = 'TO_DO' AND is_deleted = FALSE)
                   |
          tasks (T)
```

Example Query 3: View Tasks Assigned to a Specific Member

- **Purpose:** (Related to Feature: "My Tasks" View) To show a specific user (identified by `user_id`) all the tasks currently assigned to them. This requires joining the tasks table with the many-to-many assignment table.
- **SQL Query:**

```
SELECT T.task_id, T.title, T.status, T.due_date
FROM tasks T
JOIN task_assignments TA ON T.task_id = TA.task_id
WHERE TA.user_id = 2; -- Example User ID
```

- **Relational Algebra:**

$$\pi_{T.task\_id,\, T.title,\, T.status,\, T.due\_date}(\sigma_{TA.user\_id\,=\,2}(tasks\ T \bowtie_{T.task\_id\,=\,TA.task\_id} task\_assignments\ TA))$$

- **Query Tree:**

```
π (T.task_id, T.title, T.status, T.due_date)
                 |
         σ (TA.user_id = 2)
                 |
        ⋈ (T.task_id = TA.task_id)
        /                \
   tasks (T)       task_assignments (TA)
```

Example Query 4: Find High-Priority Tasks in 'Development'

- **Purpose:** (Related to Feature: Filtering and Search) To identify urgent work within a specific department (Category). This helps project managers focus on bottlenecks in specific areas like 'Development'.
- **SQL Query:**

```
SELECT T.title, T.status, T.due_date, C.name AS category_name
FROM tasks T
JOIN categories C ON T.category_id = C.category_id
WHERE C.name = 'Development' AND T.priority = 'HIGH';
```

- **Relational Algebra:**

$$\pi_{T.title,\, T.status,\, T.due\_date,\, C.name}(\sigma_{C.name='Development' \wedge T.priority = 'HIGH'}(tasks\ T \bowtie_{T.category\_id = C.category\_id} categories\ C))$$

- **Query Tree:**

```
π (T.title, T.status, T.due_date, C.name)
                    |
  σ (C.name = 'Development' AND T.priority = 'HIGH')
                    |
       ⋈ (T.category_id = C.category_id)
        /                    \
   tasks (T)            categories (C)
```

Example Query 5: List Tasks with 'Urgent' Tags

- **Purpose:** (Related to Feature: Tag Filtering) To view all tasks associated with a specific tag (e.g., 'urgent'). This requires a 3-way join between tasks, the junction table `task_tags`, and the `tags` definition table.
- **SQL Query:**

```sql
SELECT T.title, T.priority, TG.name AS tag_name
FROM tasks T
JOIN task_tags TT ON T.task_id = TT.task_id
JOIN tags TG ON TT.tag_id = TG.tag_id
WHERE TG.name = 'urgent';
```

- **Relational Algebra:**

$$\pi_{T.title,\, T.priority,\, TG.name}(\sigma_{TG.name = 'urgent'}(tasks\ T \bowtie_{T.task\_id = TT.task\_id} task\_tags\ TT \bowtie_{TT.tag\_id = TG.tag\_id} tags\ TG))$$

- **Query Tree:**

```
π (T.title, T.priority, TG.name)
                    |
            σ (TG.name = 'urgent')
                    |
          ⋈ (TT.tag_id = TG.tag_id)
          /                        \
   ⋈ (T.task_id = TT.task_id)    tags (TG)
   /                 \
tasks (T)     task_tags (TT)
```

Example Query 6: Retrieve Discussion History for a Task

- **Purpose:** (Related to Feature: Task Collaboration) To display all comments associated with a specific task (e.g., Task ID 1), showing the content, the author's name, and the timestamp. This allows team members to track conversation context.
- **SQL Query:**

```sql
SELECT U.username, C.comment_text, C.category, C.created_at
FROM task_comments C
JOIN users U ON C.user_id = U.user_id
WHERE C.task_id = 1
ORDER BY C.created_at ASC;
```

- **Relational Algebra:**

$$\pi_{U.username,\, C.comment\_text,\, C.category,\, C.created\_at}(\sigma_{C.task\_id=1}(task\_comments\ C \bowtie_{C.user\_id\, =\, U.user\_id} users\ U))$$

- **Query Tree:**

```
π (U.username, C.comment_text, C.category, C.created_at)
                    |
            σ (C.task_id = 1)
                    |
         ⋈ (C.user_id = U.user_id)
         /                      \
  task_comments (C)          users (U)
```

6. Website Structure

GitHub Back-end: Task Manager Database Backend

GitHub Front-end: Task Manager Database Frontend

6.1. Project Architecture [3] [4] [5]

a) High-Level System Architecture:

- **Backend:** A Java-based application built with **Spring Boot 3.2.0**, serving as the core engine for business logic and data persistence.
- **Frontend:** A web application built with **Next.js (React)** and TypeScript, providing the user interface for task management and progress tracking.
- **Database: MySQL 8.0** is used as the Relational Database Management System (RDBMS) for persistent storage of users, tasks, and activity logs.
- **Communication:** The frontend and backend communicate via **RESTful API endpoints** using standard HTTP methods, transferring data primarily in JSON format.

b) Backend Architecture (Spring Boot):

- **Layered Structure:** The backend adopts a standard Spring Boot layered architecture to ensure separation of concerns:

    - **Controller Layer:** Handles incoming HTTP requests, validates input using Jakarta validation, and returns appropriate HTTP responses or views.
    - **Service Layer:** Encapsulates the core business logic, such as task status workflows and assignment rules.
    - **Repository Layer:** Interfaces extending JpaRepository that handle all database interactions.
    - **Entity Layer:** Defines Java classes mapped directly to database tables using JPA annotations (e.g., @Entity, @Table).
    - **DTO Layer:** Data Transfer Objects are used to structure API responses and prevent circular reference issues common in bidirectional relationships.
- **Security & Validation:**
    - **Password Encryption:** Utilizes BCryptPasswordEncoder (via Spring Security Crypto) to hash user passwords before storage.
    - **Input Validation:** Implements constraints (e.g., @NotNull, @Size) to ensure data integrity at the API level.

c) Frontend Architecture (Next.js/React):[3] [9]

- **Framework:** Built using **Next.js**, leveraging its file-based routing and server-side rendering capabilities.
- **Component-Based UI:** The interface is constructed using reusable React components for the Kanban board, task lists, and forms.
- **State Management:** Utilizes React hooks (useState, useEffect) to manage local UI state and API data fetching.
- **Role-Based Access:** Implements logic to differentiate views for "Group Leaders" (who can assign/delete tasks) and "Members" (who update status).

6.2. Class Structure

The system's class structure follows Object-Oriented Programming (OOP) principles, ensuring modularity and reusability.

- **Entity Classes (Domain Model):** The core business entities are modeled as POJOs (Plain Old Java Objects) annotated with Jakarta Persistence (JPA) annotations. Key classes include:
  - User: Stores credentials, roles (GROUP_LEADER, MEMBER), and status.
  - Task: Represents work items, containing attributes like priority, status, and timestamps.
  - TaskAssignment: Manages the many-to-many relationship between users and tasks.
  - ActivityLog: Records the audit trail for system actions.
- **Repository Interfaces:** Instead of writing raw SQL queries in classes, the project defines interfaces (e.g., TaskRepository, UserRepository) that extend JpaRepository. This provides built-in methods for standard CRUD operations while allowing for custom derived query methods.
- **DTO Classes:** Classes such as TaskDTO or UserDTO are implemented to decouple the internal database structure from the external API, solving JSON serialization issues like infinite recursion.

6.3. Database Connection Implementation [6] [8] [11]

Unlike legacy systems using manual JDBC connection management, this project leverages **Spring Data JPA** and **Hibernate** for robust database connectivity.

3. **ORM (Object-Relational Mapping):** Hibernate is used as the JPA implementation to map Java objects directly to MySQL tables. This eliminates the need for manual SQL string concatenation and boilerplate code.
4. **Connection Configuration:** Database connection details (URL, username, password) and the MySQL driver (mysql-connector-j) are managed centrally via Spring Boot's configuration (typically application.properties), allowing for easy environment switching.
5. **Transaction Management:** The system utilizes the InnoDB storage engine and Spring's transaction management to ensure data consistency, particularly for operations involving junction tables like task_assignments.

6. **Fetching Strategies:** To optimize performance and prevent "Lazy Loading" errors, critical relationships (like User within a Task) are configured with FetchType.EAGER, while others use FetchType.LAZY to conserve resources

6.4. Graphical User Interface (GUI Design)

- **Layout & Responsive Design:** The site features a clean, minimalistic interface optimized for desktop productivity while maintaining responsiveness. The layout adapts dynamically to window resizing using a flexible grid system, ensuring usability across different screen resolutions.
- **Navigation Sidebar:** A persistent sidebar on the left side provides quick access to core modules: Dashboard, Tasks, Team, Trash, User Account, and Log out.
- **Kanban Visualization:** The central task view utilizes a Kanban Board metaphor. Tasks are represented as cards moving through columns (Pending, To Do, In Progress, Done), allowing users to visualize workflow bottlenecks and progress at a glance.The task view follows the Kanban Board, allowing users to better visualize the progress of tasks.
- **Functionality & User Experience:**
  - Offers features and dashboards suited to various users.
  - Users can quickly access relevant sections through intuitive navigation (e.g., registration, login, dashboard, task…).

1. The Leader Workspace (Administrative Control): The Leader interface is designed for management and oversight. Leaders possess full **CRUD (Create, Read, Update, Delete)** privileges:

- **Task Management:** Leaders can create new tasks, set priority levels (Low to Urgent), and define deadlines.
- **Assignment Logic:** Only Leaders have the authority to assign or reassign tasks to specific members via the "Task Assignments" module.
- **System Cleanup:** Leaders retain exclusive rights to delete tasks or manage the "Trash" (soft delete) to maintain data hygiene.

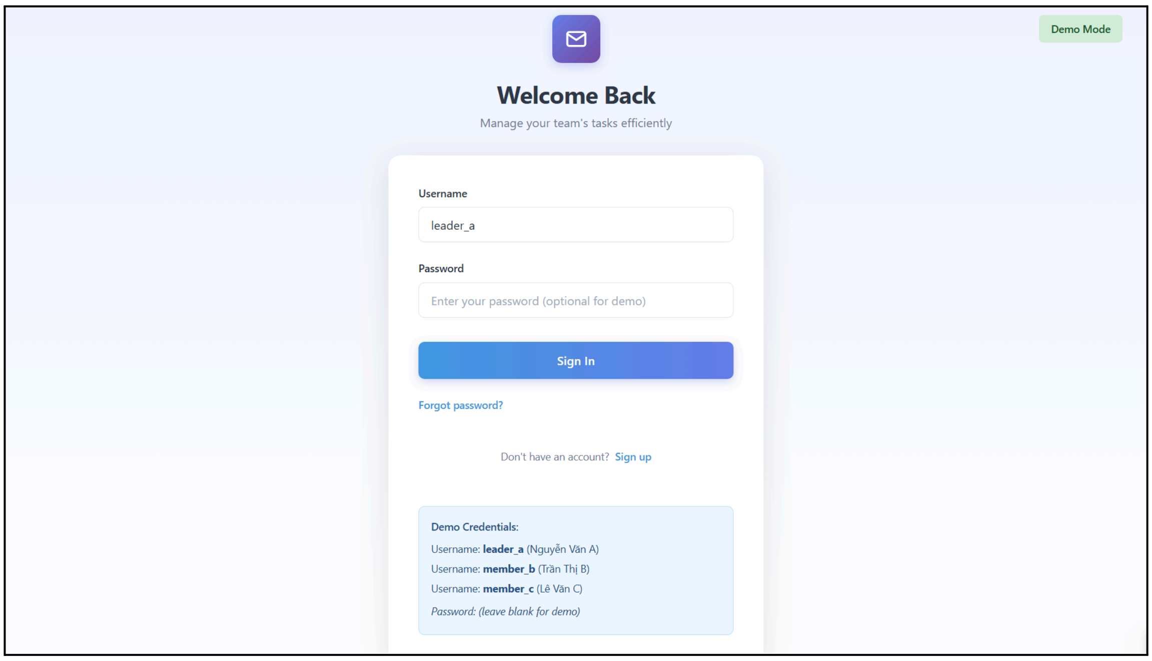
2. The Member Workspace (Execution & Collaboration): The Member interface focuses on task execution and communication, restricting actions that could alter the project scope:

- **View-Only Architecture:** Members can view all task details, attachments, and statuses but are **restricted from editing** core task properties (Title, Description, Due Date) or deleting records.
- **Collaboration & Updates:** Interaction is limited to the **Comment System**. Members can post updates, ask questions, or reply to threads within the "Task Comments" section.
- **Status Updates:** (Optional depending on your logic) Members may be permitted to drag their assigned tasks on the Kanban board (e.g., moving a task from "To Do" to "In Progress"), but cannot reassign the task to others.

3. Navigation & Accessibility: Users can quickly access relevant sections through intuitive navigation paths. Upon login, the system automatically routes the user to their specific dashboard - Leaders see a project overview, while Members see a personalized list of "Upcoming Deadline”

- **Forms & Buttons:**
    - ■ **Button and form implementation**
- **Registration/Login Page**

- A login screen is displayed at the beginning with an account-creation option for those who have not registered yet. Users who already have an account but have forgotten their password or would like to change it can use the "Forgot password" section.


Demo Mode
Welcome Back
Manage your team's tasks efficiently
Username
leader_a
Password
Enter your password (optional for demo)
Sign In
Forgot password?
Don't have an account? Sign up

Demo Credentials:
Username: **leader_a** (Nguyễn Văn A)
Username: **member_b** (Trần Thị B)
Username: **member_c** (Lê Văn C)
Password: (leave blank for demo)

Figure 3.3: Login Page

- Clicking "Sign up" will bring up the registration screen. Users must enter their username, email address, and password in this field to make an account. When finished, a successful creation message will show up.

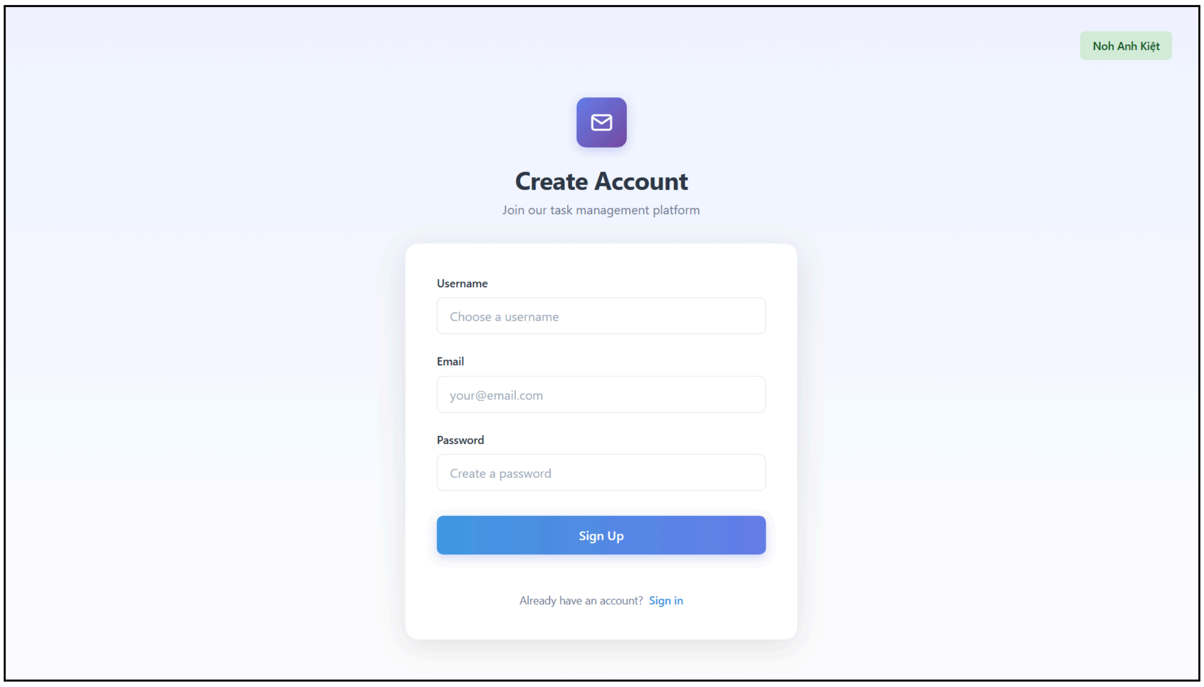
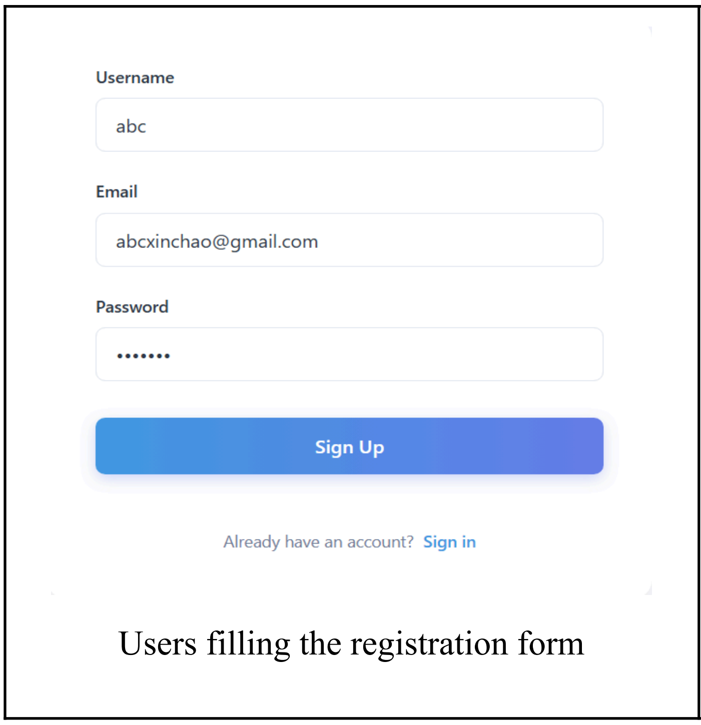


Figure 3.4: User Registration Page


Username

abc

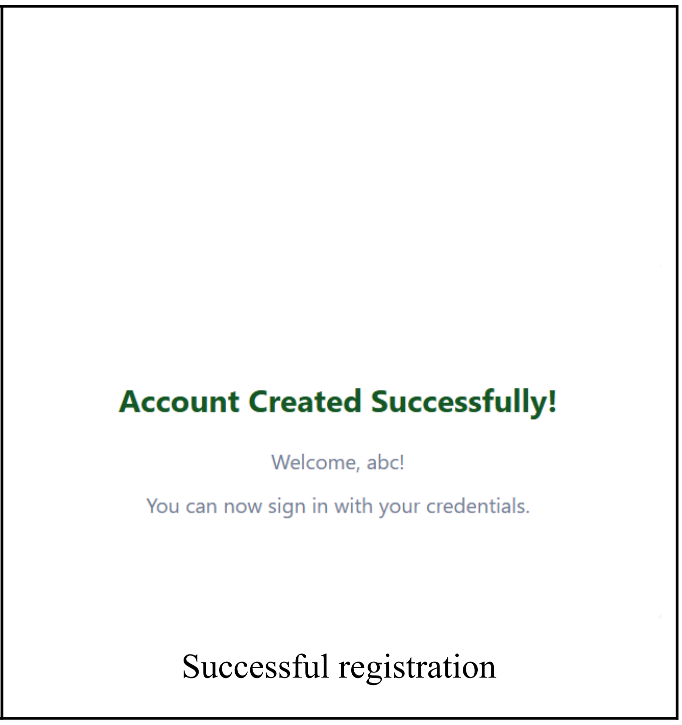
Email

abcxinchao@gmail.com

Password

•••••••

Users filling the registration form


Account Created Successfully!

Welcome, abc!

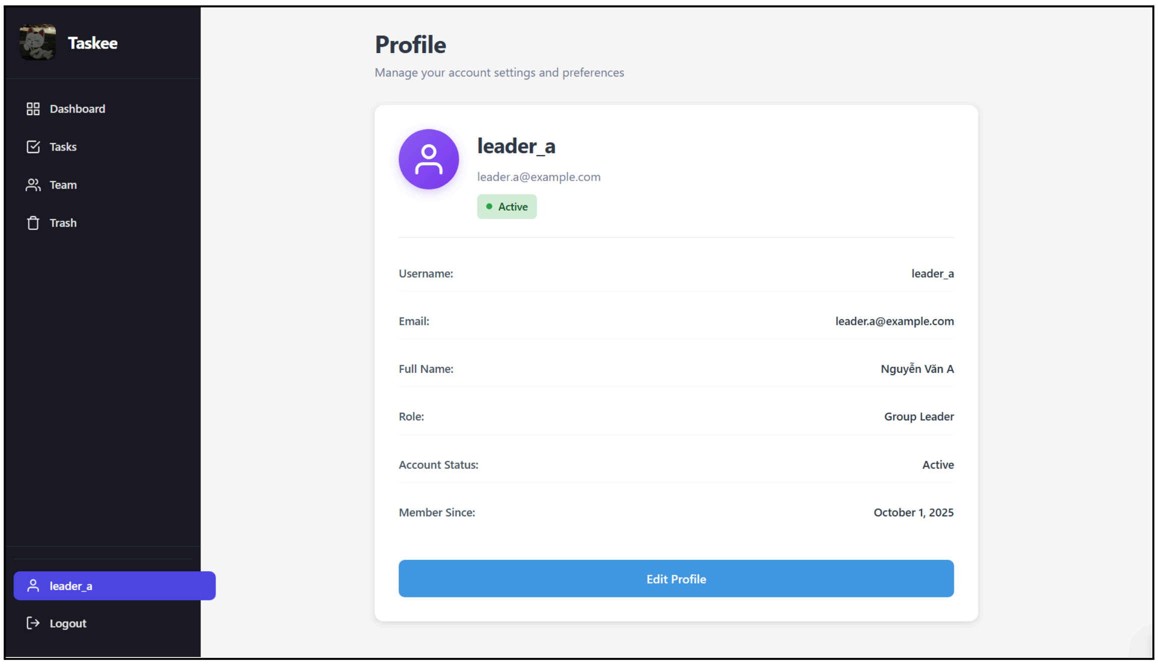
You can now sign in with your credentials.

Successful registration

Figure 3.5: Registration Form

- **User Profile**

- User Profile provides username, email, full name, account status, and account creation date, all displayed in a clear and easy-to-read format.



Profile

Manage your account settings and preferences

leader_a

leader.a@example.com

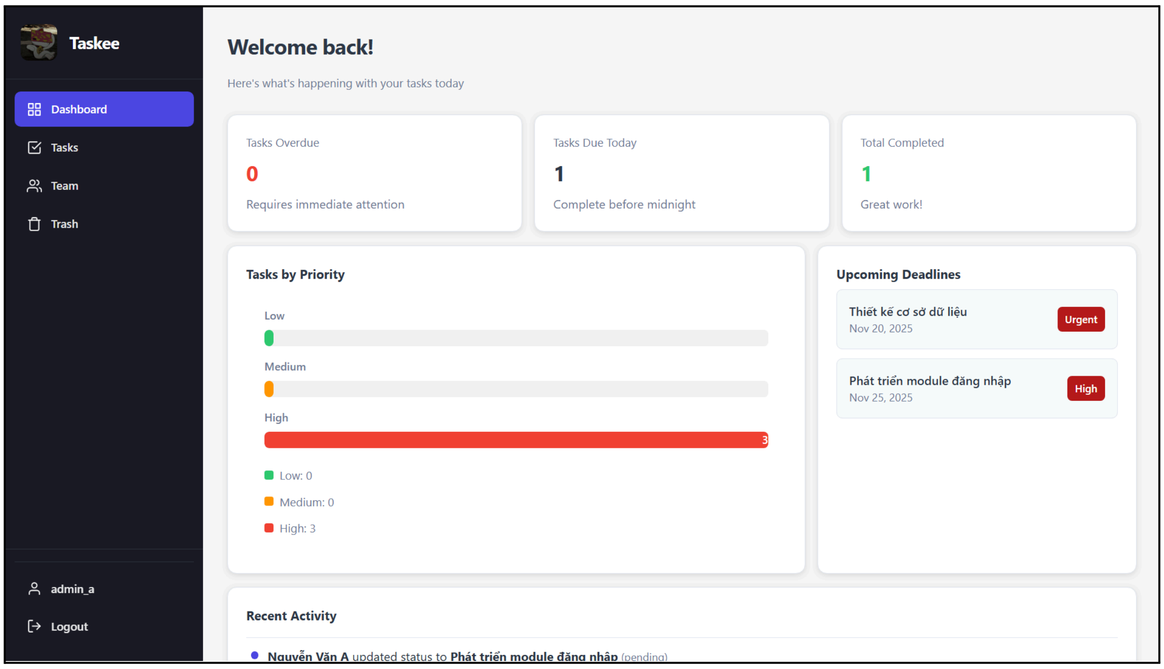
Active

Username:	leader_a
Email:	leader.a@example.com
Full Name:	Nguyễn Văn A
Role:	Group Leader
Account Status:	Active
Member Since:	October 1, 2025

Figure 3.6: User Account Page

- **Task Dashboard**

- The dashboard provides a brief summary of a user’s work status by including task statistics, priority charts, deadlines, and recent activity updates.



Welcome back!

Here's what's happening with your tasks today

Tasks Overdue	Tasks Due Today	Total Completed
0	1	1
Requires immediate attention	Complete before midnight	Great work!

Tasks by Priority

Low
Medium
High

🟩 Low: 0
🟧 Medium: 0
🟥 High: 3

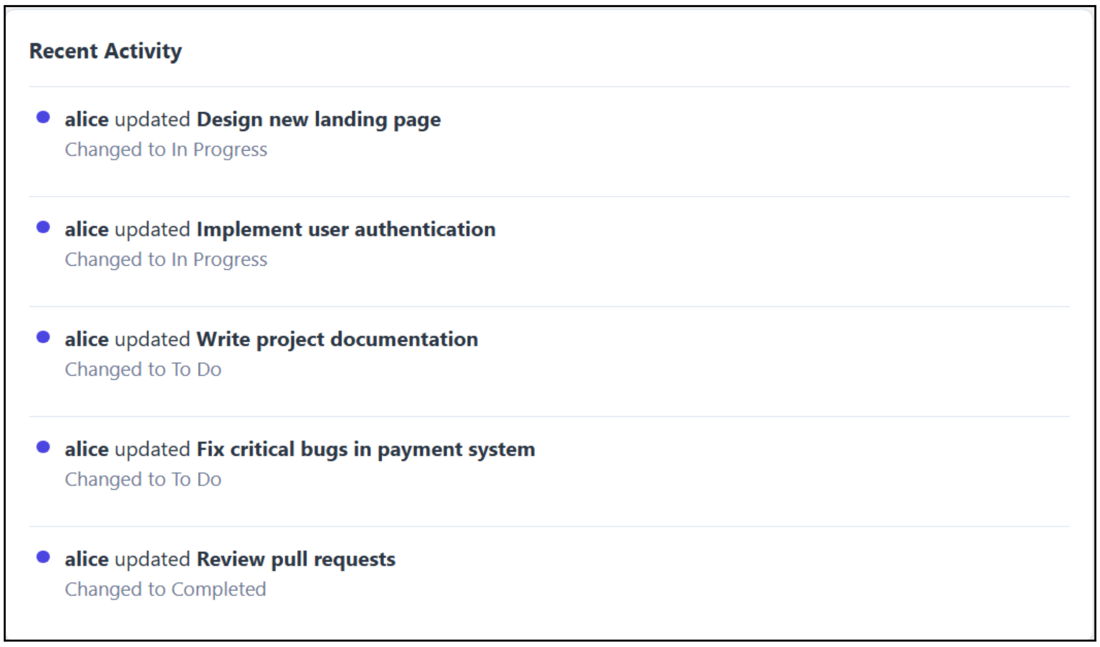
Upcoming Deadlines

Thiết kế cơ sở dữ liệu — Nov 20, 2025 — Urgent
Phát triển module đăng nhập — Nov 25, 2025 — High

Recent Activity

● Nguyễn Văn A updated status to Phát triển module đăng nhập (pending)

Figure 3.7: Dashboard Page



Recent Activity

● **alice** updated **Design new landing page**
Changed to In Progress

● **alice** updated **Implement user authentication**
Changed to In Progress

● **alice** updated **Write project documentation**
Changed to To Do

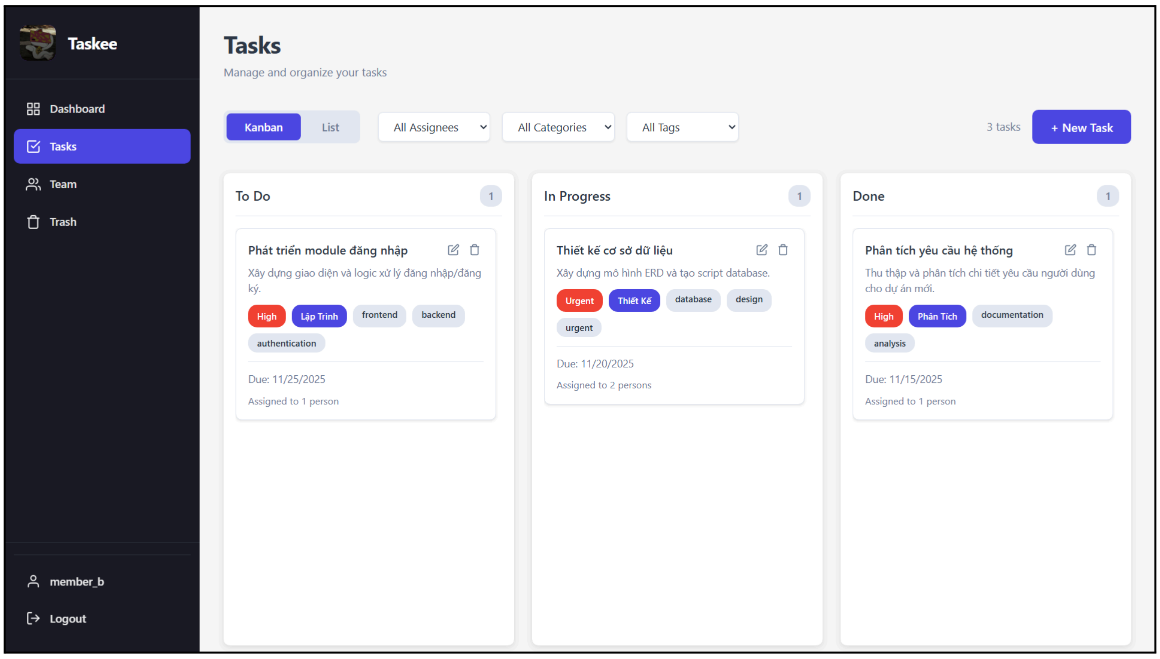
● **alice** updated **Fix critical bugs in payment system**
Changed to To Do

● **alice** updated **Review pull requests**
Changed to Completed

Figure 3.8: Notifications of all updates

- **Process Tracking and Task Creation**

- Task provides a Kanban-style board with filters, task details, status grouping, and options to create or manage tasks (for the Group Leader side). While members cannot create new tasks, members can drag tasks from 'To Do' to 'In Progress' and to 'Done'.



Phát triển module đăng nhập
Xây dựng giao diện và logic xử lý đăng nhập/đăng ký.
High · Lập Trình · frontend · backend · authentication
Due: 11/25/2025
Assigned to 1 person

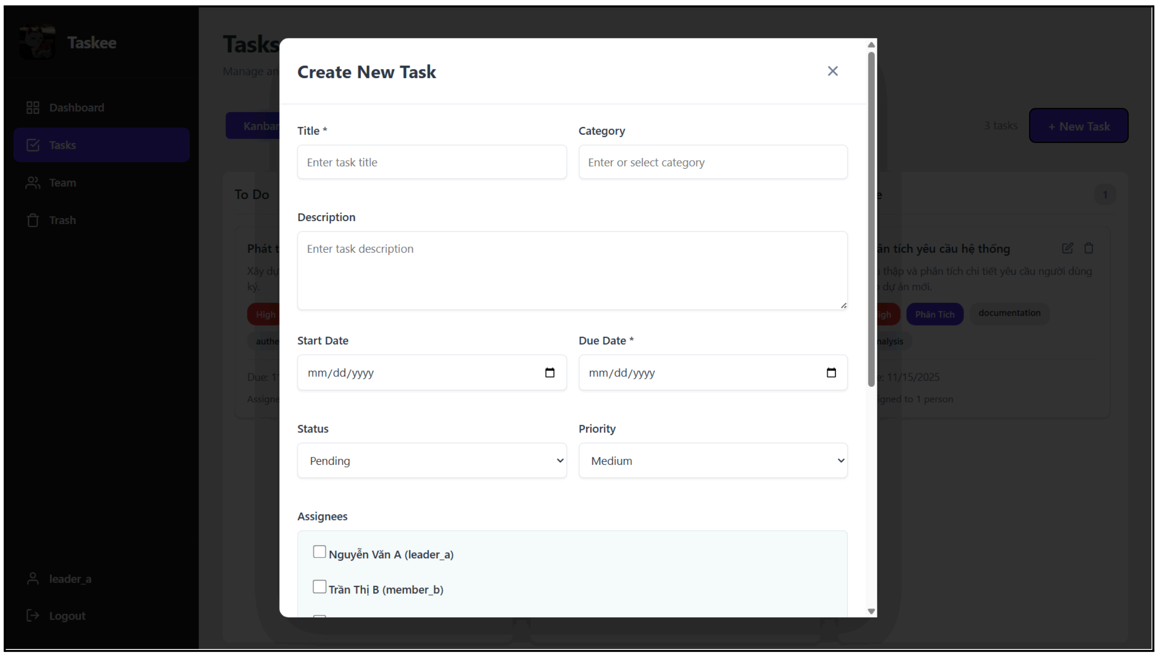
Thiết kế cơ sở dữ liệu
Xây dựng mô hình ERD và tạo script database.
Urgent · Thiết Kế · database · design · urgent
Due: 11/20/2025
Assigned to 2 persons

Phân tích yêu cầu hệ thống
Thu thập và phân tích chi tiết yêu cầu người dùng cho dự án mới.
High · Phân Tích · documentation · analysis
Due: 11/15/2025
Assigned to 1 person

Figure 3.9: Process Tracking and Task Creation

- **Task Creation and Assignment**

- After clicking on '+New Task', a pop-up where users can create a new task is shown. Where the manager/ the team leader can create and assign tasks to other members in the group.


Create New Task

Title *

Enter task title

Category

Enter or select category

Description

Enter task description

Start Date

mm/dd/yyyy

Due Date *

mm/dd/yyyy

Status

Pending

Priority

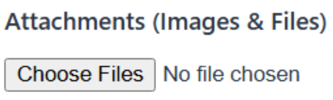
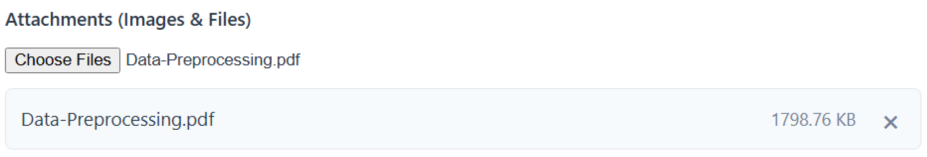
Medium

Assignees

☐ Nguyễn Văn A (leader_a)

☐ Trần Thị B (member_b)

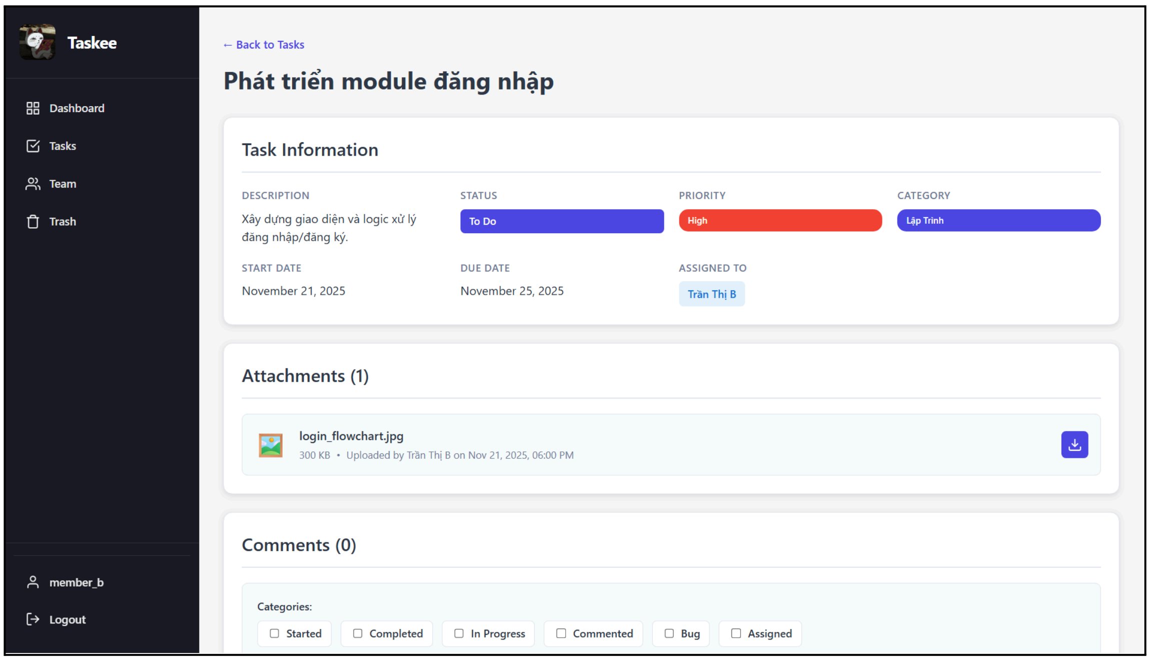
Figure 3.10: Task Creation Form

 Attachments (Images & Files) Choose Files No file chosen *Figure 3.11: Attaching files*	 Attachments (Images & Files) Choose Files Data-Preprocessing.pdf Data-Preprocessing.pdf 1798.76 KB *Figure 3.12: After attaching files*

- Once completed, a new task will show up in the to-do or in progress column, depending on the user or their group progress, along with the title, description, tags, due date, and assignment details.

- **Task Information**

- When users click on each task, a page pops up to show the detailed information of the task, where they can comment on the task.



← Back to Tasks

Phát triển module đăng nhập

Task Information

DESCRIPTION
Xây dựng giao diện và logic xử lý đăng nhập/đăng ký.

STATUS
To Do

PRIORITY
High

CATEGORY
Lập Trình

START DATE
November 21, 2025

DUE DATE
November 25, 2025

ASSIGNED TO
Trần Thị B

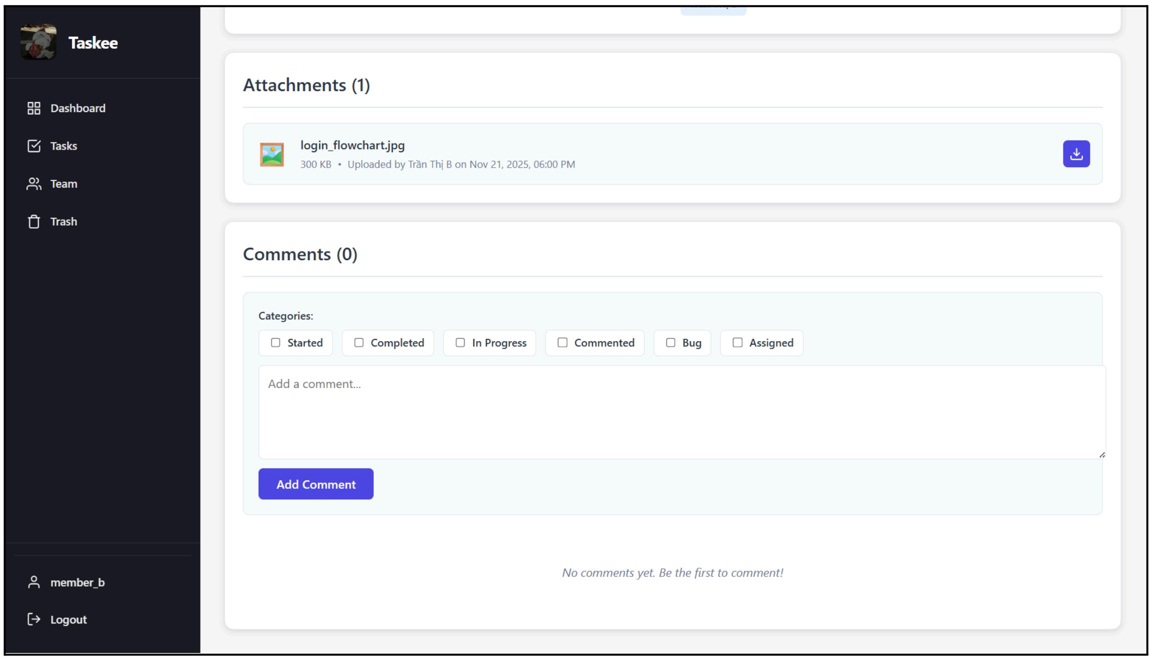
Attachments (1)

login_flowchart.jpg
300 KB • Uploaded by Trần Thị B on Nov 21, 2025, 06:00 PM

Comments (0)

Categories:
☐ Started ☐ Completed ☐ In Progress ☐ Commented ☐ Bug ☐ Assigned

Figure 3.13: Task Information



Attachments (1)

login_flowchart.jpg
300 KB • Uploaded by Trần Thị B on Nov 21, 2025, 06:00 PM

Comments (0)

Categories:
☐ Started ☐ Completed ☐ In Progress ☐ Commented ☐ Bug ☐ Assigned

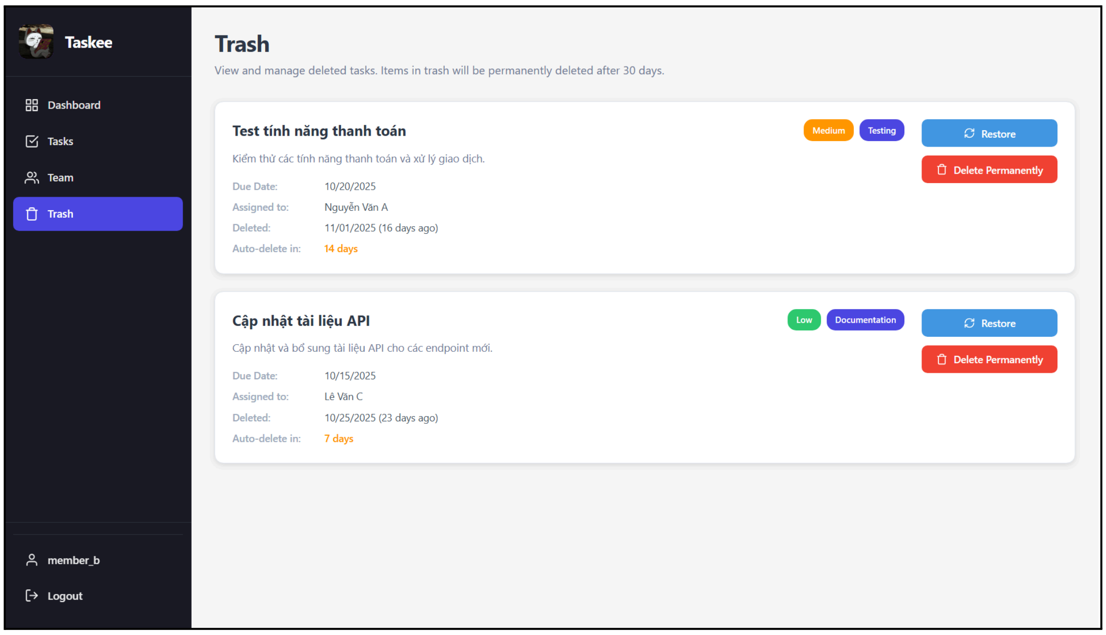
Add a comment...

No comments yet. Be the first to comment!

Figure 3.14: Comments Form

- **Task History**

- In addition to providing options to restore or permanently delete deleted tasks prior to their auto-deletion, Trash page lists deleted tasks and their details.


Taskee

Dashboard
Tasks
Team
Trash

member_b
Logout

Trash

View and manage deleted tasks. Items in trash will be permanently deleted after 30 days.

Test tính năng thanh toán — Medium — Testing — Restore — Delete Permanently

Kiểm thử các tính năng thanh toán và xử lý giao dịch.

Due Date:	10/20/2025
Assigned to:	Nguyễn Văn A
Deleted:	11/01/2025 (16 days ago)
Auto-delete in:	14 days

Cập nhật tài liệu API — Low — Documentation — Restore — Delete Permanently

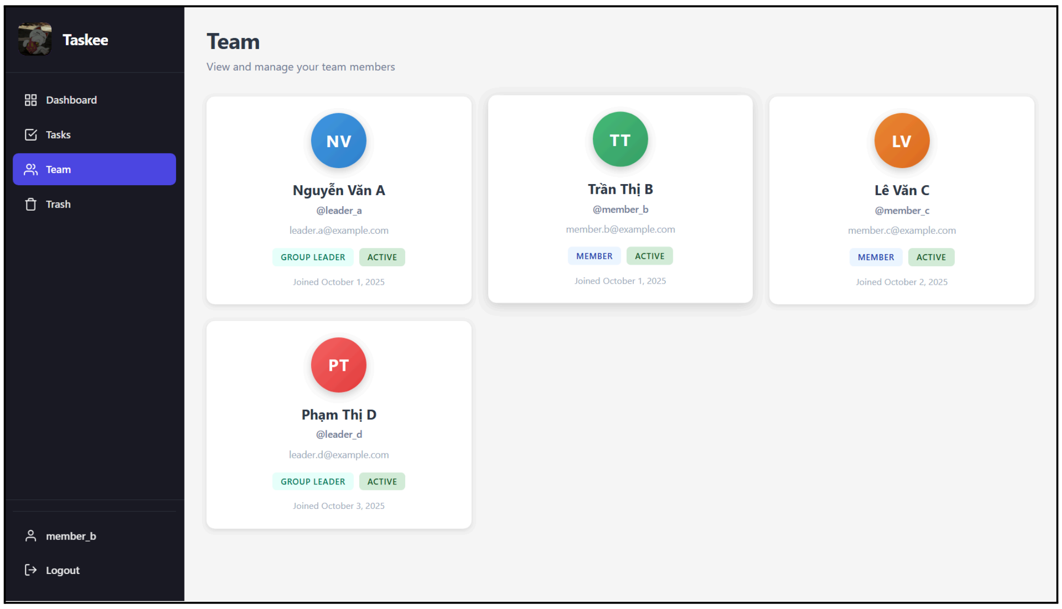
Cập nhật và bổ sung tài liệu API cho các endpoint mới.

Due Date:	10/15/2025
Assigned to:	Lê Văn C
Deleted:	10/25/2025 (23 days ago)
Auto-delete in:	7 days

Figure 3.15: Figure Task history (after deleting a task)

- **Team Member View and Management**

- Team gives a clear overview for team management by showing team members along with their roles, status, and basic contact information.


Taskee

Dashboard
Tasks
Team
Trash

member_b
Logout

Team

View and manage your team members

NV — Nguyễn Văn A — @leader_a — leader.a@example.com — GROUP LEADER — ACTIVE — Joined October 1, 2025

TT — Trần Thị B — @member_b — member.b@example.com — MEMBER — ACTIVE — Joined October 1, 2025

LV — Lê Văn C — @member_c — member.c@example.com — MEMBER — ACTIVE — Joined October 2, 2025

PT — Phạm Thị D — @leader_d — leader.d@example.com — GROUP LEADER — ACTIVE — Joined October 3, 2025

Figure 3.16: Figure Team View and Management

CHAPTER 4: IMPLEMENTATION & DISCUSSION

1. Implementation Process

The Task Management System follows the guidelines provided by the Professor, starting from topic selection, requirement review, followed by ERD design and Relational Model conversion, then Data Processing, and GUI development at the end.

Phase 1: Research, Planning, and Initial Design (Week 1 - 4)

- **Requirement Analysis and Topic Selection:** The project started by defining the requirements and given topics, finding out the most suitable topic that every member of the team favored. The process continues with key features identification, reference findings, requirements, and submission timelines understanding. This took the team three weeks to fully figure out what the team aims to do with the project.
- **System Overview and Project Goals**: The initial goal of the final product was to enable team members to effectively manage incoming teamwork projects. This led the application to develop two primary user flows: one for team leaders and one for regular members. From the leader's side, they are able to assign tasks, edit existing tasks, restore deleted tasks, and oversee the overall progress of the team. Members, on the other hand, can view their assigned tasks, update task statuses, and collaborate through supporting features such as comments, attachments, and activity logs.
- **ERD Design and Relational Model:** From week 4 to week 6, based on the system requirements, our team designed an Entity-Relationship Diagram. The ERD also specifies the cardinality relationship between entities. The ERD was then transformed into a relational schema, following the seven steps in the Mapping Rules of Lecture 3: ERD to Relational [1].

Phase 2: Database Implementation and Query Development (Week 5 - 7)

- **Table Creation:** Based on the structure of the relational model, our team members implemented SQL queries in MySQL using CREATE TABLE statements. Each table was structured with appropriate data types, primary keys (user_id, task_id, etc.), and foreign keys (linking tasks to categories and users) to enforce referential integrity. Specific constraints were applied, including UNIQUE for usernames and emails, NOT NULL for mandatory fields, and ENUM types for defining role, status, and priority. The storage engine InnoDB was specified to support transactions. A critical aspect of this phase was ensuring the database schema adhered to at least the Third Normal Form (3NF); for example, many-to-many relationships between tasks and users, as well as tasks and tags, were resolved using the junction tables task_assignments and task_tags to minimize data redundancy and prevent anomalies.
- **Data Collection and Insertion:** Sample data was manually generated by team members to represent typical scenarios within the Task Management System, covering various user roles, task categories, priority levels, and assignment workflows. Data was inserted in a specific order—starting with independent entities like users, categories, and tags, followed by tasks, and finally the dependent junction tables—to satisfy foreign key constraints.
- **SQL Queries, Relational Algebra, and Tree Queries:** A set of SQL queries was developed to demonstrate data retrieval and manipulation for common application features. These included queries for user authentication, filtering tasks by priority, tracking assignments, and a specialized VIEW (task_statistics) to summarize critical metrics such as "To-Do" "In Progress," "Done," and "Overdue" counts. Each SQL query was accompanied by its theoretical representation in Relational Algebra and a conceptual Query Tree to illustrate the optimization of data access paths.

Phase 3: Development of Web Application, Test, and Project Finalization (Week 8, Week 10 - 11)

- **Website Structure Development:**
    - **Backend:** A Java-based REST API using Spring Boot, employing a layered architecture including controller, data transfer object (dto), config, repository, and direct Java Database Connectivity (JDBC) for database connectivity.
    - **Frontend:** The website was built with Typescript and Next.js (React) with a minimalistic style, ensuring the essential functions of the project requirement, including Task Assignments, Task Information, User Accounts, Progress Tracking (Kanban Board), and Task History, where the managers can delete and restore tasks whenever needed.
- **Methodology Review and Refinement:** The methodology section (Chapter 3) was reviewed to ensure clarity, completeness, and accurate documentation of the database design and implementation steps.
- **Implementation Process Documentation and Results Analysis:** This involved documenting the overall implementation process and analyzing how the developed database and conceptual application structure met the project's initial requirements.

2. Results & Analysis

The project successfully culminated in the deployment of a robust database schema and a scalable web application architecture for a Collaborative Task Management System. The system achieved its core objectives: streamlining project workflows, enforcing clear accountability through Role-Based Access Control (RBAC), and enhancing team communication via an integrated commenting module.

3. Challenges & Solutions

Throughout the development lifecycle, the team encountered technical and structural challenges, which were mitigated through analytical problem-solving and adherence to software engineering best practices:

- **Complex Schema Modeling & Normalization:**
  - **Challenge:** Accurately representing the many-to-many relationships inherent in project management (specifically task_assignments and task_tags) while ensuring the schema remained flexible and normalized.
  - **Solution:** We employed an iterative ERD refinement process. Junction tables were explicitly defined to resolve many-to-many associations. A systematic dependency analysis was conducted to eliminate transitive dependencies, ensuring the schema satisfied 3NF requirements.
- **Advanced SQL Aggregation & Optimization:**
  - **Challenge:** Constructing efficient queries for the "Dashboard" view, which requires calculating metrics (e.g., tasks "Overdue," "To-Do," or "Done") in real-time without compromising performance.
  - **Solution:** We encapsulated complex aggregation logic within a dedicated MySQL VIEW (task_statistics). This abstraction simplified the application layer's data retrieval and optimized the query execution plan using underlying indices on status and due_date.
- **Security & Role-Based Access Control (RBAC):**
  - **Challenge:** strict segregation of duties was required to ensure Members could not perform destructive actions (like deleting tasks) restricted to Group Leaders.
  - **Solution:** We implemented a dual-layer security strategy. At the database level, distinct user roles were defined via ENUM. At the application level, secure password hashing (BCrypt) was mandated, and API endpoints were secured to ensure only authorized roles could trigger specific INSERT or DELETE operations.

CHAPTER 5: CONCLUSION & FUTURE WORK

1. Summary of Key Findings

The Task Management System project demonstrates the entire process of creating and putting into use a database schema (up to BCNF) based on practical needs. Our main achievements include database implementation with advanced features such as activity logs, task tags, attachments, and multi-user task assignments. Finally, a working website that demonstrated how the backend database facilitates task assignment status tracking, user roles, activity update and document attachments.

2. Reflection on Objectives

Every project goal was met, especially in creating, controlling and completing tasks as required, significantly improving the efficiency and productivity of individuals or groups using our system, especially in the field of work progress management. Throughout the development, we tend to use knowledge from lectures, including ERD design, schema conversion rules, functional dependencies, and normalization principles. In addition, the project enhanced communication, cooperation, and planning abilities. Problems were fixed through group discussion and incremental improvement.

3. Implications

This project demonstrates the importance of database normalization (3NF) and structured data modeling (ERD to relational schema) for a task management system.It illustrates database normalization to reduce redundancy, minimize anomalies, and guarantee data integrity with transparent task ownership, progress tracking, and user interactions. Lastly, the design offers a platform that can be expanded into an application that can be utilized by individuals, groups, or businesses in both professional and non-professional settings.

4. Future Work

Although the project has satisfied the requirements, there are still a number of possible improvements:

- **Interface Improvement**: Advanced filtering features, search, and user-customizable interface elements like dark mode or any theme.
- **Application & Program**: Develop into a software that can be installed directly on operating systems such as Windows, Mac, or Linux, and make the application available on the Android or iOS store instead of using the web as it is now.
- **User Authentication**: Use captcha, Server-side, IP or JavaScript & browser environment to make sure that a real person is using this service.
- **Notification System**: Integrate real-time notifications using WebSocket or Firebase for instant updates.
- **Task Reminders**: When the due date draws near, automatically send out emails or SMS reminders about incomplete tasks.

REFERENCE

1. International University. (n.d.). *PDM lecture slides* (Course: Principles of Database Management; Instructor: Nguyen Thi Thuy Loan).
2. International University. (n.d.). *PDM lab assignment* (Course: Principles of Database Management).
3. CodeWave. (n.d.). *MERN stack task manager* [YouTube playlist]. YouTube. https://youtube.com/playlist?list=PLeR-H2WqA8sNzkIhNA0TwW44oitkknc8G&si=NCrdtb1CAseYaevS
4. Code with Projects. (n.d.). *Task management system using Spring Boot, Angular, MySQL* [YouTube playlist]. YouTube. https://youtube.com/playlist?list=PLgYFT7gUQL8HSoaSoaYU2-rSG-N_DnSeS&si=V3V0gKPZX4565Zo3
5. Sandesh300. (n.d.). *Task management system* [GitHub repository]. GitHub. https://github.com/sandesh300/Task-Management-System
6. Oracle. (n.d.). *MySQL documentation*. https://dev.mysql.com/doc/
7. GeeksforGeeks. (n.d.). *Normal forms in DBMS*. https://www.geeksforgeeks.org/normal-forms-in-dbms/
8. Oracle. (n.d.). *JDBC: Database access*. https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/
9. Vercel. (n.d.). *Designing UI with Next.js*. https://nextjs.org
10. Oracle. (n.d.). *MySQL Workbench manual*. https://dev.mysql.com/doc/workbench/en/
11. Spring. (n.d.). *Mapping support in Spring Data JDBC*. https://docs.spring.io/spring-data/relational/reference/jdbc/mapping.html