TaskCraft Pro: Empower Your Workforce with Intelligent Task Management

DB - Project Report

Muhammad Tahir (K214503)

Muhammad Talha (K213349)

Introduction

Project Background:

In today's dynamic business environment, effective task management is crucial for organizational success. The project stems from the observed challenges faced by many organizations in efficiently organizing and tracking tasks assigned to employees, leading to missed deadlines and a lack of transparency. The need for a centralized and user-friendly solution has prompted the development of the Employee Task Management Dashboard. This system aims to streamline task assignment, tracking, and completion, fostering a more organized and productive work environment.

Project Scope

The Employee Task Management Dashboard project focuses on developing a web-based platform to facilitate efficient task management within the organization. The primary functionalities include task assignment, tracking, a bonus points system, task editing, task deletion, user authentication, and database operations. The system's scope encompasses providing a comprehensive solution for authorized personnel to manage tasks effectively while ensuring transparency and timely completion. The system will not address unrelated functionalities beyond the specified task management features.

Not in Scope

The following functionalities are explicitly not in the scope of the current project:

- Advanced project management features beyond individual task tracking.
- Integration with external project management tools.
- Financial transactions or invoicing functionalities.

Project Objectives

The project aims to achieve the following objectives:

- Streamline task assignment and tracking processes.
- Enhance transparency in task management within the organization.
- Motivate employees through a bonus points system for timely task completion.
- Provide a user-friendly Employee Task Management Dashboard for authorized personnel.
- Demonstrate efficient database operations for task management.

External Interface Requirements

Hardware Interfaces

The Employee Task Management System is a web-based application and does not have direct hardware dependencies. It is designed to operate on standard computing devices, including desktops, laptops, and tablets. The hardware interfaces are generic and include devices with web browser capabilities and internet connectivity.

Software Interfaces

The system interacts with the following software components:

- **MySQL Database (Version 8.2.0):** The Employee Task Management System relies on MySQL for database management. It includes interactions for data storage, retrieval, and management. The MySQL database is connected through the XAMPP server.
- **Express (Node.js Framework):** The backend of the system is built using Express, a Node.js framework. It handles API requests, authentication, and database interactions.
- **React (Version 18):** The frontend of the system is developed using React, providing a dynamic and responsive user interface for the Employee Task Management Dashboard.
- **Node.js (Version 18.16.1):** Node.js is used as the server-side JavaScript runtime to execute server operations and facilitate communication between the frontend and the database.

Data Items or Messages Exchanged:

- **Between Frontend and Backend:** JSON data structures for task details, user authentication, and system responses.
- **Between Backend and Database:** SQL queries and responses for database operations.

Communications Interfaces

The Employee Task Management System relies on standard web communication protocols. The communication interfaces include:

- **HTTP/HTTPS Protocols:** The system uses HTTP/HTTPS protocols for communication between the client (web browser) and the server (Node.js/Express backend). HTTPS is employed to ensure secure data transmission.
- **RESTful API:** The backend exposes RESTful API endpoints to facilitate communication between the frontend and backend. This includes endpoints for task management, user authentication, and other system functionalities.
- **Communication Security:** Secure Socket Layer (SSL) or Transport Layer Security (TLS) protocols will be implemented to ensure the security of data during transmission.
- **Data Transfer Rates:** The data transfer rates will depend on the internet connection speed of the user. However, efforts will be made to optimize data transfer for efficient performance.

- **Synchronization Mechanisms:** The system will employ asynchronous communication for non-blocking interactions, providing a responsive user experience.

Design Strategy

The design strategy for the Employee Task Management System is rooted in principles that prioritize flexibility, scalability, and maintainability. Several key considerations have influenced the overall organization of the system and its high-level structures:

Future System Extension or Enhancement:

- **Strategy:** The system is designed to be modular, allowing for easy extension or enhancement of functionalities in the future.
- **Reasoning:** Modular design promotes the addition of new features or modifications to existing ones without significant disruption to the entire system. This ensures that the Employee Task Management System can adapt to evolving organizational needs.

System Reuse:

- **Strategy:** The use of a three-tier architecture promotes system reuse by separating the UI, business logic, and data access layers.
- **Reasoning:** Each layer can be reused independently, enabling the frontend to be updated without affecting the backend, and vice versa. This separation of concerns facilitates code reuse and maintenance.

User Interface Paradigms:

- **Strategy:** The system employs a responsive and dynamic UI using React, adhering to modern user interface paradigms.
- **Reasoning:** A user-friendly and intuitive interface enhances user experience and encourages adoption. The use of React allows for the creation of a single-page application, reducing page reloads and providing a smoother, more interactive user experience.

Data Management (Storage, Distribution, Persistence):

- **Strategy:** MySQL is chosen as the relational database management system for its robust data management capabilities.
- **Reasoning:** MySQL provides a reliable and efficient solution for data storage, ensuring that employee and task-related information can be stored, retrieved, and managed effectively. The relational model supports complex queries and relationships, aligning with the structured nature of task management data.

Concurrency and Synchronization:

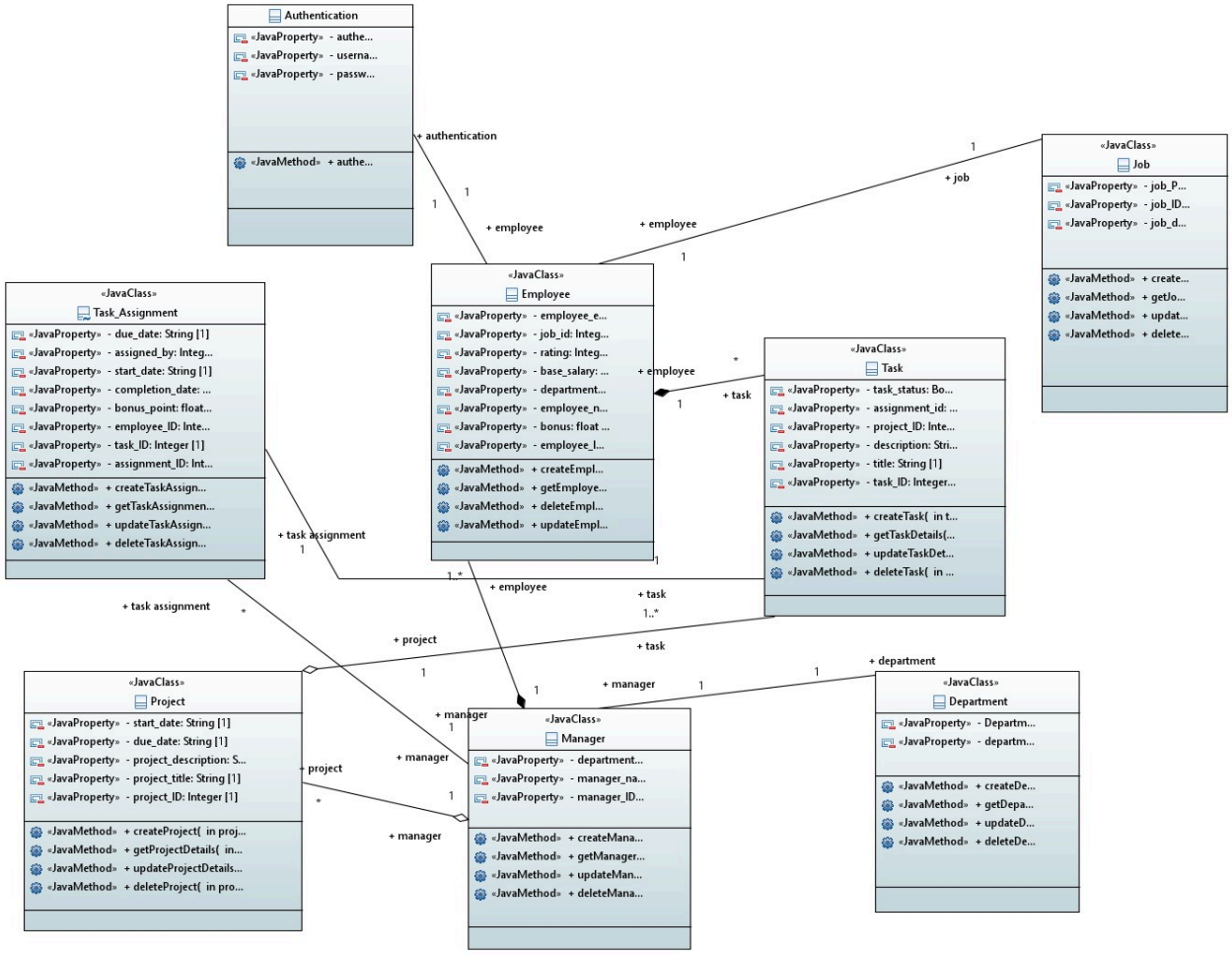
- **Strategy:** The system employs asynchronous communication and RESTful API endpoints to manage concurrency.
- **Reasoning:** Asynchronous communication allows for non-blocking interactions, ensuring a responsive user experience. RESTful API endpoints provide a standardized and stateless approach to communication, simplifying synchronization between the frontend and backend. This approach enhances the system's ability to handle multiple concurrent users.

Trade-offs:

- While the three-tier architecture provides modularity and separation of concerns, it introduces additional complexity in terms of deployment and maintenance.
- The use of React for the frontend requires a modern browser environment, potentially limiting compatibility for users with older browsers.

Detailed System Design

Class Diagram



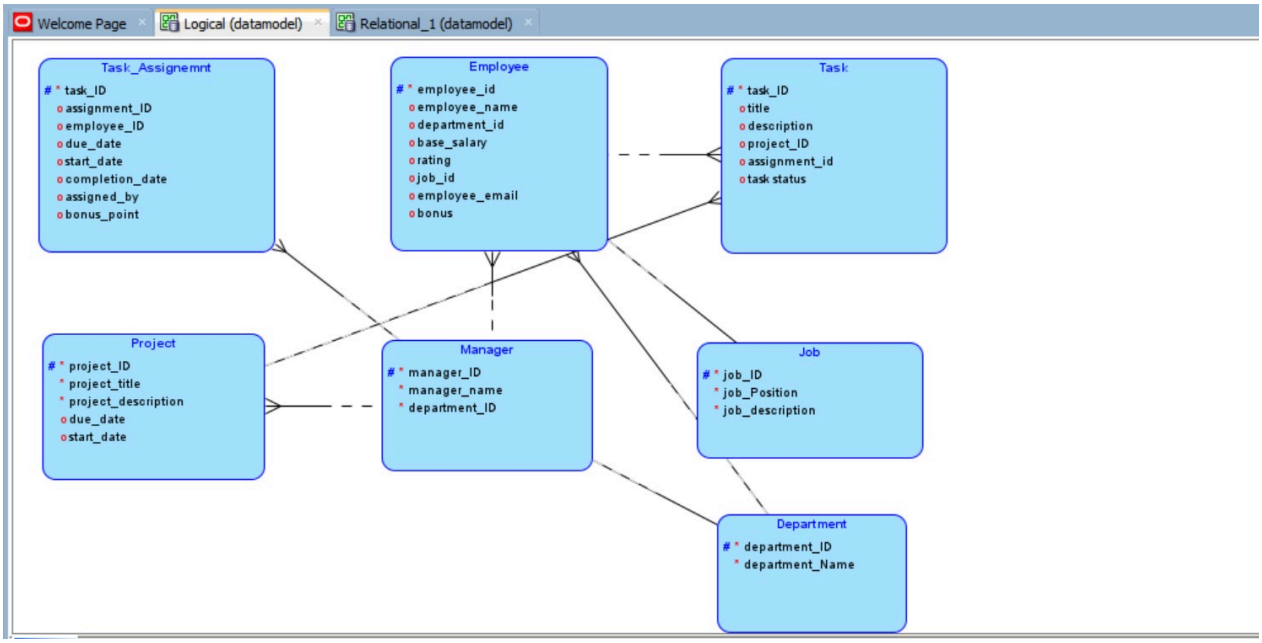
Here's a detailed description of the classes and their attributes, methods, and interactions:

1. **Employee**: This class represents an employee in the company. It has attributes such as employeeID, name, employeeRating,baseSalary,bonus,departmentID,jobID,rating, and methods like addEmployee(), updateEmployee(), deleteEmployee(). These methods are used to manage the employee data.
2. **Manager**: This class could represent a manager in the company. It is an instance of the Employee class. It has additional methods like assignTask(), manageProject(), manageDepartment().
3. **Department**: This class represents the various departments within the company. It has attributes like departmentID, departmentName, and methods like addDepartment(), deleteDepartment().

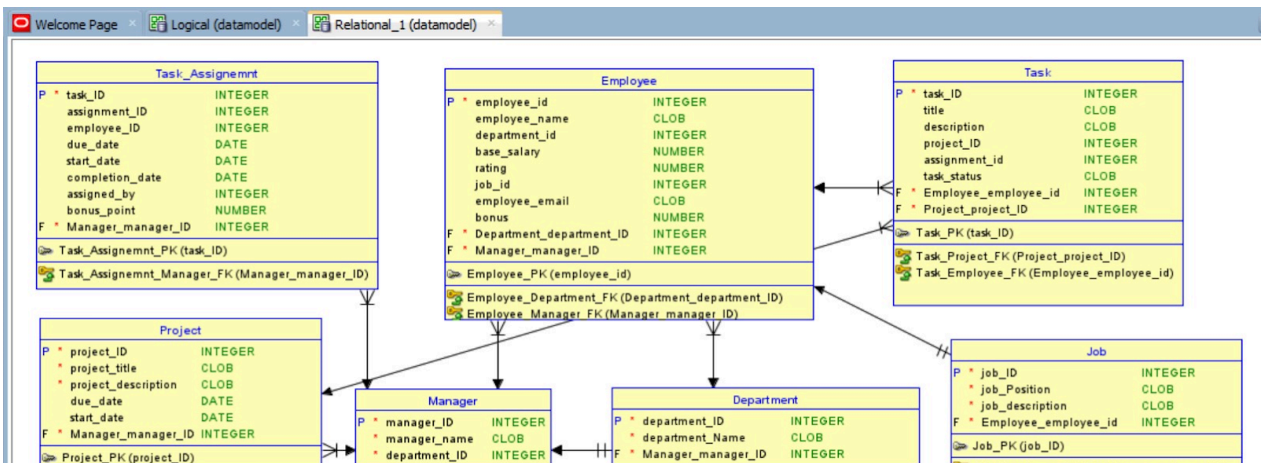
4. **Project**: This class represents the different projects within the company. It has attributes like projectID, projectName,startDate, completionDate, dueDate, and methods like addProject(), updateProjectDetails(), deleteProject().
5. **Task**: This class represents the different tasks within a project. It has attributes like taskID, taskTitle, status, taskDescription, projectID, assignmentID and methods like addTask(), updateTask(), deleteTask().

The lines connecting the classes represent the relationships between them. For example, an Employee could be assigned to a Department, represented by a line connecting the Employee and Department classes. Similarly, a Manager could assign a Task to an Employee, represented by a line connecting the Manager and Task classes.

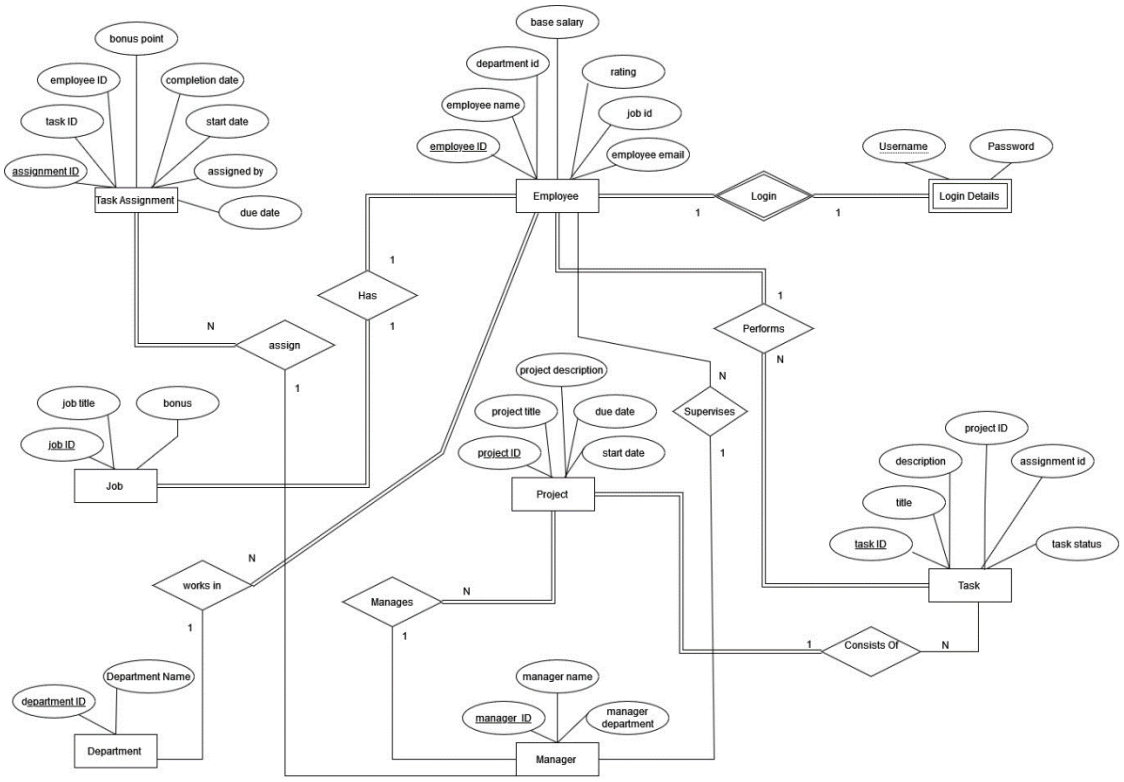
Logical Model



Relational Data Model:



8.1.2. ER Diagram



1. **Employee Entity**: This entity represents an employee in the system. Each employee has the following attributes:
    - **Name**: The name of the employee.
    - **Job**: The job title or role of the employee.
    - **Department**: The department in which the employee works.
    - **Manager**: The manager of the employee. This could be a reference to another employee entity who is the manager.
2. **Project Entity**: This entity represents a project in the system. Each project has the following attributes:
    - **Name**: The name of the project.
    - **Description**: A brief description of the project.
    - **Task**: The tasks associated with the project.
    - **Status**: The current status of the project (e.g., in progress, completed, etc.).
3. **Relationships and Multiplicity**: The relationships between entities are represented by lines connecting them. The multiplicity (or cardinality) of a relationship is indicated by the numbers or symbols near the entities. For example, if there's a "1" near the Employee entity and an "N" near the Project entity, it means one employee can be

associated with multiple (N) projects. Conversely, if there's a "1" near both entities, it means each employee is associated with exactly one project, and each project is associated with exactly one employee.

Un-Normalized DB schema:

1. Employee (employee ID, employee name, department id, employee email, job id, rating, salary)
2. Task (task ID, title, description, project ID, assignment id)
3. Manager (manager ID, manager name, manager department)
4. Project (project ID, project title, project description, due date, start date)
5. Department (department ID, Department name)
6. Job (job ID, job title, bonus)
7. Task Assignment (assignment id, employee ID, manager ID, assigned by, start date, completion date, due date, task status, bonus point)
8. Admin(employee ID, email, password)

Normalized Schema:

1. ActiveUsers(AdminID, Email, Password, Designation)
2. TaskAssignment (AssignmentID *EmployeeID*, *TaskID*, *AssignedBy*, StartDate, CompletionDate, DueDate, TaskStatus, BonusPoint)
3. Job (JobID, JobTitle)
4. Department (DepartmentID, DepartmentName, *ManagerID*)
5. Project (ProjectID, ProjectTitle, ProjectDescription, DueDate, StartDate)
6. Manager (ManagerID, ManagerName, *Email, Password*)
7. Task (TaskID , Title, Description, *ProjectID*, *AssignmentID*)
8. Employee (EmployeeID, EmployeeName, *DepartmentID*, *EmployeeEmail*, *JobID*, Rating, Salary, *Password*)
9. ApprovalRequest (ApprovalID, *AssignmentID, EmployeeID,* RequestDate, Status)