A blue and yellow logo

Description automatically generatedLebanese Universityالجامعة اللبنانية

Faculty of Engineeringكلية الهندسة

Branch-3الفرع الثالث

**Mini-Project Report**

***Prepared by:***

* Omar Ismail 6304
* Roudy Barakat 6305

**Presented for**:

Dr. Mohamad AOUDE

**Date**: 16 / 7 / 2025

# Mini-Project Report: Smart Project Management Dashboard with AI-Based KPI Evaluation

## Table of Contents

1. Introduction
2. System Architecture Overview
3. Backend Implementation
   1. Technologies Used
   2. Database Configuration
   3. Data Models
   4. Data Schemas
   5. CRUD Operations
   6. API Endpoints
4. Frontend Implementation
   1. Technologies Used
   2. Global State Management
   3. Core Components
   4. Styling
5. AI Agent: "Lama3" Implementation
   1. Input Data for Lama3
   2. Dual Approach Implementation
   3. Integration and Accessibility
   4. Explainability (XAI) and Autonomy
6. Technology Justification
7. Ethical AI Considerations
8. Conclusion

## I. Introduction

This report details the design, implementation, and functionality of a comprehensive full-stack intelligent system developed as part of the "Advanced Smart AI Agent" mini-project. The system addresses critical challenges in modern project management by providing a robust platform for efficient project oversight, task allocation, customer relationship management, and proactive, AI-driven insights. Our chosen domain for this project is Management Automation.

The core problem we aim to solve is the reactive nature of traditional project management. Projects often encounter issues (for example, budget overruns, schedule delays, resource conflicts) that are only identified after they have become significant problems. Our solution leverages an autonomous AI agent, "Lama3," to continuously analyze project Key Performance Indicators (KPIs), predict potential risks, and generate actionable alerts, thereby enabling a proactive management approach. The system is designed to be fully Dockerized, ensuring modular deployment and consistency across environments.

## II. System Architecture Overview

The system follows a classical three-tier architecture, augmented with a dedicated, locally integrated AI agent.

* **Frontend:** A responsive web application built with React, providing an intuitive User Experience (UX) and User Interface (UI) for managers, employees, and customers.
* **Backend:** A powerful and fast API developed using FastAPI, handling all business logic, data validation, and interactions with the database and the AI agent.
* **Database:** A relational database (MySQL) stores all project-related data, including customer details, employee information, project specifications, tasks, alerts, budget history, and comprehensive KPIs.
* **AI Agent ("Lama3"):** An intelligent component integrated within the backend, responsible for processing project data, predicting KPIs, and flagging risks.

The entire application stack is designed for Dockerization, facilitating easy deployment and ensuring environmental consistency.

## III. Backend Implementation

The backend serves as the brain of the application, built with a focus on performance, scalability, and maintainability.

### A. Technologies Used

* **Framework:** FastAPI
* **ORM:** SQLAlchemy
* **Database Driver:** For connecting to MySQL.
* **Authentication & Hashing:** Libraries for password hashing and handling authentication tokens.
* **Environment Management:** For loading environment variables.
* **Web Server:** Uvicorn

### B. Database Configuration

A dedicated configuration file securely manages sensitive information by loading environment variables, notably the database connection URL. Another file initializes the SQLAlchemy engine and sets up a session class for managing database sessions. It also provides a dependency injection mechanism, which ensures proper database session handling (opening and closing) for each API request. This approach promotes efficient resource management and helps prevent connection leaks.

### C. Data Models

The data models define the SQLAlchemy ORM models, mapping directly to the relational tables in the MySQL database. These models encapsulate the application's core entities and their relationships:

* **Customer**: Stores customer identification, name, company, email, and phone. It has a one-to-many relationship with Project (a customer can have multiple projects).
* **Employee**: Stores employee identification, name, email (unique), hashed password, and preferences (a text column storing structured data for notification and theme settings). It has a one-to-many relationship with Task.
* **Project**: The central entity, storing project identification, project name, customer identification (foreign key), status (an enumerated type like 'Not Started', 'In Progress', 'Completed', 'On Hold'), completion percentage, total budget, used budget, budget status (an enumerated type like 'Under Budget', 'On Budget', 'Over Budget'), start date, and launch date. It maintains relationships with Customer, Task, Alert, Budget History, and Project KPI.
* **Task**: Stores task identification, project identification (foreign key), task name, employee identification (foreign key), deadline, and status (an enumerated type like 'Pending', 'In Progress', 'Completed', 'Overdue'). It links tasks to specific projects and assigned employees.
* **Alert**: Stores alert identification, alert type (an enumerated type like 'Urgent', 'Warning', 'Info'), alert source (an enumerated type like 'Project', 'Task', 'System'), project identification (foreign key), task identification (foreign key), message, status (an enumerated type like 'Read', 'Unread'), and timestamp. Alerts are crucial for the AI agent's proactive notifications.
* **Budget History**: Stores history identification, project identification (foreign key), month (in 'YYYY-MM' format), and amount used. This table tracks historical budget consumption for projects.
* **Project KPI**: This table is pivotal for the AI agent. It stores KPI identification, project identification (foreign key), and a comprehensive set of metrics including completion percentage, milestone completion, budget utilization, schedule variance, overdue tasks, alert count, average task completion time, employee workload index, customer priority level (an enumerated type), KPI class (an enumerated type), reopened tasks, risk flag (boolean), resource allocation score, stakeholder satisfaction, team efficiency score, change request count, bug count, documentation completeness, dependency risk score, feature delivery rate, code quality score, and timestamp. This rich dataset serves as the primary input for the "Lama3" AI agent.

### D. Data Schemas

Pydantic models are used to define the data structures for request and response validation, ensuring data integrity and consistency across the API. For each entity, base, creation, update, and full schemas are defined. For instance, an employee creation schema includes password validation, while an employee update schema makes all fields optional. The configuration settings allow seamless conversion between SQLAlchemy ORM objects and Pydantic models. Nested schemas, like those for notification preferences and user preferences within the employee schema, facilitate structured data for user settings.

### E. CRUD Operations

A dedicated module encapsulates the core business logic for interacting with the database. It contains functions for create, read (single and multiple with filtering and pagination), update, and delete operations for Employees, Customers, Projects, Tasks, and Alerts. These functions abstract the database queries, allowing the API endpoints to focus solely on request handling and response formatting.

### F. API Endpoints

The FastAPI application exposes a comprehensive set of RESTful API endpoints for managing the project management system's data:

* **Employee Endpoints**: Full create, read, update, and delete operations for employee management, including email uniqueness validation during creation.
* **Customer Endpoints**: Full create, read, update, and delete operations for customer data.
* **Project Endpoints**: Full create, read, update, and delete operations for project details.
* **Task Endpoints**: Create, read, update, and delete operations for tasks, with read endpoints supporting filtering by project identification and employee identification.
* **Alert Endpoints**: Create, read, update, and delete operations for system alerts, with read endpoints supporting filtering by project identification and status.

Additionally, the main application file configures Cross-Origin Resource Sharing (CORS) to allow the frontend application (running on different local development ports) to make requests to the backend. An OPTIONS preflight handler is also included for robust CORS support. The frontend's data context management suggests the presence of an authentication endpoint, implying an OAuth2/JWT-based authentication flow for secure access.

## IV. Frontend Implementation

The frontend provides the interactive user interface, built with modern React principles for a dynamic and responsive experience.

### A. Technologies Used

* **Framework:** React
* **Routing:** React Router DOM
* **State Management:** React Context API
* **Charting:** Recharts
* **Icons:** React Icons
* **Styling:** Pure CSS

### B. Global State Management

A dedicated data context file implements the React Context API, serving as a centralized store for global application state and shared functionalities. It manages:

* Authentication state (token, authentication status, current user).
* Fetched data (customers, projects, tasks, alerts, employees).
* API interaction functions (user login, user logout, fetching data for customers, projects, tasks, alerts, and employees; adding tasks to projects; updating task statuses; updating alerts; updating user preferences; changing passwords, and so on).

This approach avoids passing properties down through many levels of components and ensures that all components have access to necessary data and functions, facilitating a cleaner and more maintainable codebase. Data fetching is triggered upon changes in authentication status, ensuring data freshness and consistency.

### D. Core Components

The React application is composed of several modular components, each responsible for a specific part of the UI and its functionality:

* **App**: The root component. It sets up the overall grid layout using CSS, includes the Header and Sidebar components, and defines the application's routing using React Router DOM.
* **Header**: Displays the application header, including a menu icon for toggling the sidebar, a search icon, and notification or message icons.
* **Sidebar**: Provides the main navigation menu, allowing users to navigate between different sections of the application (Dashboard, Projects, Tasks, Customers, Alerts, Settings). Navigation links from React Router DOM are used for active link styling.
* **Home (Dashboard)**: The application's landing page. It displays key aggregated statistics (total projects, tasks, budget, alerts) in cards and visualizes project budget overview using Recharts for interactive bar charts. While initially using sample data, it is designed to integrate with the fetched project data.
* **Projects**: Lists all projects, providing filtering capabilities by search term. It allows adding new projects via a modal and navigates to project details upon clicking a project item. Status icons (such as a check circle for "Completed") provide quick visual cues.
* **Project Details**: Displays comprehensive information for a selected project, including its name, customer, status, budget, and dates. It also lists all associated tasks, allows adding new tasks to the project, and provides functionality to update task statuses (for example, mark as "Completed" or "Pending"). The component also supports editing project details.
* **Tasks**: Provides an overview of all tasks across all projects. It categorizes tasks into "Pending," "Completed," and "Overdue" for easy monitoring. Each task item displays its name, associated project, assigned employee, deadline, and current status with corresponding icons.
* **Customers**: Presents a list of all customers, with search and filtering capabilities. It displays customer name, company, email, phone, and the number of associated projects. A modal allows for adding new customer entries. Clicking on a customer navigates to Customer Details.
* **Customer Details**: Shows detailed information for a specific customer, including contact information and a list of all projects associated with that customer. Projects are navigable, linking back to their respective project details pages.
* **Alerts**: Displays system-generated alerts. Users can filter alerts by status (all, unread, urgent, resolved). It shows alert type, source, message, timestamp, and relevant project or task details. Functionality to mark alerts as "Read" is included, which updates the backend.
* **Settings**: Allows authenticated users to view and update their profile details (name, email), notification preferences (project updates, task assignments, overdue alerts), and application theme. It also provides a secure interface for changing the user's password. Messages are displayed for feedback on save operations.

### E. Styling

The main CSS file defines the overall visual aesthetic of the application, including layout (using CSS Grid for the main container), component-specific styling, responsive adjustments, and consistent design elements like rounded corners, shadows, and status indicators.

## V. AI Agent: "Lama3" Implementation

The AI agent, "Lama3," is the intelligent core of the project management system, designed to fulfill the requirement for an autonomous, goal-driven AI that provides proactive insights. Its primary goal is to predict the KPI class and risk flag for projects based on current and historical project data.

### A. Input Data for Lama3

"Lama3" will process the various features related to projects available in the database, such as completion progress, budget utilization, schedule adherence, task statuses, and other performance metrics. Based on this comprehensive project data, it will predict the Key Performance Indicators (KPIs) for the project.

### B. Dual Approach Implementation

To satisfy the project requirement for implementing AI logic using two distinct approaches, "Lama3" will incorporate both:

1. **Traditional Rule-Based Decision-Making Engine:**
   * **Concept:** This engine will utilize predefined logical rules and heuristics established by domain experts (or derived from best practices) to evaluate project health and predict KPIs. These rules will operate on the current state of project metrics.
   * **Implementation:** Implemented as a series of conditional statements within a Python module. For instance, rules could be defined to determine a "Low" KPI class if overdue tasks exceed a certain threshold or budget utilization is too high, or to assess a "High" risk flag if multiple alerts are active and tasks are being reopened.
   * **Justification:** Provides transparency, is easy to understand and debug, and allows direct control over decision logic, making it suitable for clearly defined scenarios and initial baselines.
2. **Modern Machine Learning (ML) Engine:**
   * **Concept:** This engine will employ supervised learning models trained on historical project data to learn complex patterns and make predictions. It will be capable of classification (for KPI class) and potentially regression (for predicting numerical trends).
   * **Implementation:** Using Python libraries designed for machine learning.
     + **Data Preparation:** Features will be preprocessed (for example, numerical scaling, one-hot encoding for categorical features like customer priority level).
     + **Model Selection:** For KPI class (a classification task), models like Random Forest Classifier, Gradient Boosting Classifier, or Support Vector Machines could be used. For risk flag (a binary classification), Logistic Regression or another classifier would be suitable.
     + **Training:** The models would be trained on a dataset of historical project KPI entries, with KPI class and risk flag as target variables (which would need to be either manually labeled in historical data or derived from complex rules if no labels are present).
     + **Prediction:** Once trained, the model will take new project KPI data as input and output its predictions for KPI class and risk flag.
   * **Justification:** Capable of identifying non-linear relationships and subtle patterns that rule-based systems might miss, leading to more accurate and nuanced predictions, especially with sufficient historical data. It can adapt to new data trends through retraining.

### C. Integration and Accessibility

"Lama3" will be integrated as a module within the FastAPI backend. Dedicated API endpoints (for example, for predicting KPIs for a given project ID) will be created to trigger Lama3's predictions. When called, this endpoint will fetch the latest KPI data for a given project, pass it to both the rule-based and ML engines, and return their respective predictions.

To allow for comparison and analysis as required by the assignment, the frontend will include a section (for example, within Project Details or a dedicated "AI Insights" page) where users can view the predictions from both the rule-based and ML models for a given project. This could be presented side-by-side, or a toggle could allow switching between the "AI Models" to see their respective assessments.

### D. Explainability (XAI) and Autonomy

* **Explainability (XAI):** For the rule-based engine, explainability is inherent as the rules are human-readable. For the ML engine, techniques like feature importance (from tree-based models) or SHAP values could be employed to explain why a particular KPI prediction or risk flag was generated, enhancing user trust and understanding.
* **Autonomy:** "Lama3" operates autonomously by continuously evaluating project data (for example, on a scheduled basis or triggered by significant project updates) and automatically updating the Project KPI table with its predictions, or even generating Alerts in the Alerts table if critical thresholds are met. This moves beyond a simple chatbot interaction towards active decision support.

## VI. Technology Justification

The selection of technologies was driven by a balance of performance, development efficiency, ecosystem maturity, and alignment with modern development practices:

* **FastAPI:** Chosen for its high performance (comparable to NodeJS and Go), asynchronous capabilities, and automatic interactive API documentation (Swagger UI/ReDoc). Its strong typing with Pydantic ensures robust data validation.
* **React:** Selected for building a dynamic and responsive Single Page Application (SPA). Its component-based architecture promotes reusability, modularity, and efficient UI updates, leading to a smooth user experience.
* **MySQL:** A widely adopted, robust, and reliable relational database. Its maturity, ACID compliance, and strong community support make it an excellent choice for structured data storage in a project management system.
* **SQLAlchemy & Pydantic:** SQLAlchemy provides a powerful ORM layer, abstracting database queries and allowing Pythonic interaction with the database. Pydantic ensures data validation and serialization, crucial for maintaining data integrity between the frontend, backend, and database.
* **Docker:** Essential for modular deployment and reproducibility. Docker containers encapsulate the application and all its dependencies, ensuring that the development, testing, and production environments are consistent, simplifying deployment and troubleshooting.
* **Local AI Agent ("Lama3"):** Implementing the AI agent locally within the backend (or as a closely coupled service) allows for real-time data processing without external API latency or costs. For sensitive project data, keeping the AI logic in-house can also be a significant advantage for data privacy and security.

## VII. Ethical AI Considerations

In developing "Lama3," several ethical considerations have been prioritized:

* **Fairness and Bias:** Efforts will be made to ensure that the AI model does not inadvertently perpetuate or amplify biases present in historical data (for example, favoring certain employees or project types). Regular auditing of model predictions and their impact will be crucial.
* **Transparency and Explain ability:** As discussed, both rule-based and ML approaches will incorporate explain ability features to help users understand why a particular KPI prediction or risk flag was generated, enhancing user trust and understanding.
* **Accountability:** While the AI provides insights, the final decisions remain with human managers. The system is a tool to aid, not replace, human judgment.
* **Data Privacy and Security:** All project and employee data handled by the AI will adhere to strict data privacy protocols. The local deployment of "Lama3" further strengthens data control compared to relying on external AI services.

## IX. Conclusion

The "Advanced Smart AI Agent for Project Management" system represents a robust and intelligent solution designed to enhance project oversight through automation and predictive analytics. By combining a powerful FastAPI backend, a user-friendly React frontend, a structured MySQL database, and the innovative "Lama3" AI agent with its dual implementation approach, the project not only meets the stringent requirements of the assignment but also lays the groundwork for a truly proactive and data-driven project management experience. The focus on explainability and ethical considerations ensures that the AI serves as a transparent and trustworthy assistant to human decision-makers.