CIMCON Software (India) Pvt. Ltd.
Global Leadership Through Technology Innovation

2024

Project Report

PROJECT DASHBOARD

CIMCON Software (India) Pvt. Ltd.
Global Leadership Through Technology Innovation

Project Report for
WEB APPLICATION - PROJECT DASHBOARD

Document No. PMDB002

Document Control

Prepared By	Reviewed By	Approved By
Pratyush Pilli	Lokesh Chainani	Jutan Sharma
Date:	Date:	Date:

TABLE OF CONTENTS

Project Dashboard

1. Introduction

1.1. Project Overview:

The Project Dashboard Web Application is a web-based platform designed to streamline and enhance the data management processes within the company. The primary objective is to centralize data and provide real-time insights through an interactive that allows for data manipulation, viewing and editing. The dashboard will facilitate various operational services, initially focusing on core operational data. A key feature of the application is its role-based access control, allowing the admin to dynamically manage permissions for different users within the organization. This ensures that sensitive data is accessed and modified only by authorized personnel, while still providing necessary information to users according to their roles.

1.2. Tech Stack:

1.2.1. Frontend Tech Stack:

Framework: React.JS

Core Libraries

- React: 18.3.1
- React DOM: 18.3.1
- React Router DOM: 6.25.0
- TypeScript: 5.4.5
- Vite: 4.5.3
- UI Libraries and Components
- @mui/material: 5.15.20 – Material-UI components for a React-based UI.
- @mui/icons-material: 5.15.19 – Material-UI icons.
- @mui/x-data-grid-pro: 7.13.0 – Advanced data grid for material UI with pro features.
- @mui/x-data-grid: 7.13.0 – Material-UI data grid for handling large datasets.
- @mui/x-date-pickers: 7.14.0 – Material-UI date pickers for handling date inputs.
- Styling
- @emotion/react: 11.11.4 – Emotion library for styling React components.
- @emotion/styled: 11.11.5 – Styled-components API using Emotion.
- Sass: 1.77.4 – A CSS preprocessor to make stylesheets more dynamic and organized.

Charting Libraries

- @amcharts/amcharts4: 4.10.39 – AmCharts for interactive charts and graphs.
- Echarts: 5.5.1 – ECharts for rich, interactive visualizations.
- Echarts-for-react: 3.0.2 – A wrapper for integrating ECharts in React.
- Data Handling and Spreadsheets
- ag-grid-community: 32.2.0 – High-performance grid handling large datasets.
- ag-grid-react: 32.2.0 – React wrapper for ag-Grid.
- React Spreadsheet: 0.9.5 – Excel-like spreadsheet functionality in React.
- XLSX: 0.18.5 – Library for reading and writing Excel files.
- Date-fns: 3.6.0 – Modern JavaScript date utility library.
- Day.js: 1.11.13 – JavaScript library for parsing, validating, manipulating, and formatting dates.

HTTP Requests and Utilities

- Axios: 1.7.7 – Promise-based HTTP client for the browser and Node.js.
- Cross-env: 7.0.3 – Run scripts that set environment variables across platforms.
- Development Tools
- @vitejs/plugin-react: 4.3.0 – Vite plugin for React applications.
- @types/node: 22.1.0 – TypeScript definitions for Node.js.
- @types/react: 18.3.3 – TypeScript definitions for React.
- @types/react-dom: 18.3.0 – TypeScript definitions for React DOM.
- ESLint: 8.57.0 – Linter for identifying and fixing problems in JavaScript/TypeScript code.
- GH-pages: 6.1.1 – Easily publish your project to GitHub Pages.

1.2.2. Backend Tech Stack:

Core Libraries

- asgiref: 3.8.1 – ASGI server reference implementation used by Django for asynchronous support.
- certifi: 2024.8.30 – Certificates for SSL/TLS verification.
- charset-normalizer: 3.3.2 – Library used to detect and normalize character sets.
- idna: 3.10 – Internationalized domain names in applications.
- six: 1.16.0 – Python 2 and 3 compatibility utilities.
- typing-extensions: 4.12.2 – Additional type hinting features for Python.
- tzdata: 2024.1 – Time zone database for Python.

Django and Extensions

- Django: 5.0.6 – High-level Python web framework.
- django-cors-headers: 4.3.1 – Handling Cross-Origin Resource Sharing (CORS) in Django applications.

- **django-filter:** 24.2 – Django package for filtering query sets dynamically.
- **djangorestframework:** 3.15.1 – Toolkit for building Web APIs in Django.

Database Management

- **dj-database-url:** 2.2.0 – Utility to allow database configuration using URL strings.
- **psycopg2:** 2.9.9 – PostgreSQL database adapter for Python.
- **psycopg2-binary:** 2.9.9 – Standalone version of the PostgreSQL adapter.

Data Handling

- **et-xmlfile:** 1.1.0 – Minimal library for writing XML files.
- **numpy:** 1.26.4 – Package for numerical computations in Python.
- **openpyxl:** 3.1.3 – Library for reading/writing Excel files.
- **pandas:** 2.2.2 – Data manipulation and analysis library.
- **python-dateutil:** 2.9.0.post0 – Utilities for manipulating dates.
- **pytz:** 2024.1 – World time zone definitions for Python.

HTTP and Web Requests

- **requests:** 2.32.3 – Simplified HTTP library for Python.
- **urllib3:** 2.2.3 – HTTP client for Python with enhanced support for secure connections.

JSON Handling

- **simplejson:** 3.19.3 – Fast JSON encoder and decoder.

Financial Data Library

- **forex-python:** 1.8 – Foreign exchange rates and Bitcoin price library.

Other

- **sqlparse:** 0.5.0 – SQL formatting and parsing tool.
- **Werkzeug:** 3.0.4 – Comprehensive WSGI web application library.

2. Installation Guide

2.1. System Requirements:

- **Operating System**: Windows, macOS, or Linux
- **Memory**: At least 4 GB of RAM (8 GB recommended)

- **Storage**: Minimum 500 MB of free space
- **Processor**: Dual-core CPU or higher

2.2. Backend:

- **Python**: Version 3.9 or higher
- **Django**: Version 5.0.6
- **PostgreSQL**: Version 13 or higher
- **Dependencies**: Install Python packages listed in the requirements.txt file:
- **Command**: pip install -r requirements.txt

2.3. Frontend:

- **Node.js**: Version 18 or higher
- **React**: Version 18.3.1
- **Dependencies**: Install frontend dependencies by running the following command in the frontend directory:
- **Command**: npm install

3. API Documentation

3.1. Projects API

- **URL:** /api/projects/
- **Method:** GET
- **Description:** Retrieves a list of all projects.
- **Response:** JSON array containing project details.
- **URL:** /api/projects/<int:pk>/
- **Method:** GET
- **Description:** Retrieves detailed information of a project by its ID.
- **Response:** JSON object of the project.

3.2. Milestones API

- **URL:** /api/milestones/
- **Method:** GET
- **Description:** Retrieves a list of milestones for projects.
- **Response:** JSON array containing milestone details.
- **URL:** /api/milestones/<int:pk>/
- **Method:** GET
- **Description:** Retrieves detailed information of a milestone by its ID.
- **Response:** JSON object of the milestone.

3.3. Save Projects

- **URL:** /api/save_projects/
- **Method:** POST
- **Description:** Saves new project data.
- **Request Body:** JSON object with project details.
- **Response:** Confirmation of project saved.

3.4. Save AMC

- **URL:** /api/save_amc/
- **Method:** POST
- **Description:** Saves new AMC data.
- **Request Body:** JSON object with AMC details.
- **Response:** Confirmation of AMC saved.

3.5. Save Indent

- **URL:** /api/save_indent/

- **Method:** POST
- **Description:** Saves new indent data.
- **Request Body:** JSON object with indent details.
- **Response:** Confirmation of indent saved.

3.6. Save Milestones

- **URL:** /api/save_milestones/<str:pk>/
- **Method:** POST
- **Description:** Saves new milestone data for a given project.
- **Request Body:** JSON object with milestone details.
- **Response:** Confirmation of milestone saved.

3.7. Update Milestones

- **URL:** /api/save_milestones/<int:pk>/
- **Method:** PUT
- **Description:** Updates milestone data for a given milestone ID.
- **Request Body:** JSON object with updated milestone details.
- **Response:** Confirmation of milestone updated.

3.8. Filter Projects

- **URL:** /api/filter/
- **Method:** GET
- **Description:** Filters projects based on the criteria provided.
- **Response:** Filtered project data as a JSON array.

3.9. File Upload

- **URL:** /api/upload/
- **Method:** POST
- **Description:** Uploads files related to projects.
- **Request Body:** File data.
- **Response:** Confirmation of successful file upload.

3.10. Radar Chart Data

- **URL:** /api/radar-chart/<str:project_code>/
- **Method:** GET
- **Description:** Retrieves radar chart data for a given project code.
- **Response:** JSON object with radar chart metrics.

3.11. Financial Dashboard

- **URL:** /api/financial-dashboard/
- **Method:** GET
- **Description:** Retrieves financial dashboard data.
- **Response:** JSON array with financial data and metrics.

3.12. Financial Overview

- **URL:** /api/financial-overview/
- **Method:** GET
- **Description:** Retrieves an overview of the financial status across projects.
- **Response:** JSON array with financial overview data.

3.13. Project Financial Overview

- **URL:** /api/project-financial-overview/
- **Method:** GET
- **Description:** Retrieves the financial status of specific projects.
- **Response:** JSON array with financial details for selected projects.

3.14. Filter Financial Data

- **URL:** /api/filter-finance/
- **Method:** GET
- **Description:** Filters financial data based on the criteria provided.
- **Response:** Filtered financial data as a JSON array.

3.15. AMC Project Codes

- **URL:** /api/amc/project-codes/
- **Method:** GET
- **Description:** Retrieves project codes for AMC.
- **Response:** JSON array with AMC project codes.

4. Features and Functionality

4.1. Frontend (React):

4.1.1. User Interface:

- An intuitive and user-friendly interface built using React, enabling seamless navigation and interaction.
- Responsive design that adapts to various screen sizes for accessibility on desktops, tablets, and mobile devices.

4.1.2. Component-Based Architecture:

- Modular component structure for reusable UI components, enhancing maintainability and scalability.
- Use of Material-UI for consistent design and styling across the application.

4.1.3. State Management:

- Implemented state management using React's Context API or Redux to manage application state efficiently.
- Allows for easy data sharing among components without prop drilling.

4.1.4. API Integration:

- Axios or Fetch API used to make asynchronous requests to the Django backend for data retrieval and submission.
- Error handling and loading states implemented for a better user experience during data fetch operations.

4.1.5. Data Visualization:

- Integration of data visualization libraries (e.g., Chart.js, AmCharts, or ECharts) to represent data through interactive charts and graphs.
- Visualizations provide insights into project performance, financial metrics, and other key indicators, aiding decision-making processes.

4.1.6. Dynamic Data Management:

- Data entry forms designed for easy input and management of project details, milestones, and operational information.
- Real-time updates on dashboards to reflect changes in data, ensuring users have access to the latest information.

4.2. Backend (Django):

4.2.1. RESTful API Development:

- Django REST Framework (DRF) used to create RESTful APIs that handle CRUD operations for various resources (projects, milestones, AMC, indent).
- Implementation of serializers for data validation and transformation before sending responses to the frontend.

4.2.2. Authentication & Authorization:

- User authentication implemented with JWT or session-based authentication, ensuring secure access to API endpoints.

- Role-based access control to restrict functionalities based on user roles (e.g., admin, user).

4.2.3. Business Logic:

- Modular views and services that encapsulate business logic, making it easier to maintain and test.
- Support for various utility functions, such as filtering, saving, and updating data.

4.2.4. File Handling:

- Capabilities to handle file uploads for relevant project documents (e.g., reports, images).

4.3. Database (PostgreSQL):

4.3.1. Data Modelling:

- Use of PostgreSQL for robust data storage, with normalized tables for projects, milestones, AMC, and indent information.
- Relationships established among tables to maintain data integrity and optimize queries.

5. Future Improvements

5.1. Error Handling:

- Graceful error handling mechanisms on both frontend and backend to inform users of any issues during data processing.

5.2. Client-Side Validation:

- Validation rules implemented on the frontend to ensure user inputs are correct and complete before submission.
- Use of libraries like Formik or Yup for structured validation of forms.

5.3. User Authentication and Roles:

- Enhanced Role-Based Access Control: **Implement more granular access controls to restrict data visibility based on user roles, ensuring sensitive information is only accessible to authorized users.**
- Single Sign-On (SSO): **Integrate SSO capabilities to streamline user authentication across different platforms and improve security.**