Technical Documentation

STC Marketing Work Management Solution

Project Name	**STC Marketing Work Management Solution (SMWMS)**
Author	Ha Chu, Alexander Gu, Alexander Lap, Alexander Siracusa
Date	1 March 2025
Version	v1.0

Version History:

Version	**Author**	**Updated Date**
v1.0	Ha Chu, Alexander Gu, Alexander Lap, Alexander Siracusa	1 March 2025

Table of Contents

1. Introduction

This web application (to be hosted on STC's domain) provides the following main functionalities:

- Project-Task Management Interface: An interface where project and task data entries can be easily added, edited, or removed.
- Project Visualizations: The visualization options (e.g., Gantt, CPM, EVM) are there to aid project planning and monitoring.
- Internationalization: Only English and Simplified Chinese are currently supported, but the infrastructure allows for more languages to be added.
- Email Notifications: These notify the relevant people of any task-related information.

Scope-wise, the project aims to provide a fully integrated project management solution that is tailored to STC's requirements. However, due to time and feasibility constraints, initial development will concentrate on core functionalities with extensibility for future enhancements.

The purpose of this project is to develop a tailored project management software that can reduce the cognitive load on project management from the current approach of using Excel sheets and improve team collaboration. By centralizing task management, enhancing visualization capabilities, and providing a multilanguage interface, the system increases efficiency and decision-making accuracy for the marketing team.

2. Document Outline

This document will include information on what the SMWMS does, all layers of functionality, what tools were used to develop it, and how to get the application up and running from start to finish.

3. System Overview

3.1 Introduction

Provide here the overview of your software design document.

3.2 System Overview

Provide here the information about your software that includes functionalities and features.

3.3 Design Considerations

Describe the issues that need to be addressed before completing your software design.

3.3.1 Assumptions and Dependencies

3.3.1.1 Assumptions

- Users have an understanding of how to navigate browsers and websites.
- Users are familiar with project management methods - Critical Path Method, Earned Value Management - along with terminology related.
- Users have the ability to interpret charts and graphs.

3.3.1.2 Dependencies

[...]

3.3.2 General Constraints

The constraint that affected this enterprise mostly falls under implementation feasibility. This project is to be completed in the short span of 7 weeks. With such a limited time frame the development structure has been organised into a Minimal Viable Product (MVP), with additional features time permitting. The MVP consists of successfully achieving the primary goals: Task Management Interface, Project Visualizations Suites, Multi-Language Support, Documentation. Additional features may include exporting project to external csv file for meeting purpose, [...list of additional features].

3.3.3 Goals and Guidelines

Describe the strategy of design and development software and software development life cycle. This part includes goals, guidelines, principles, and priorities of system design.

3.4 Architectural Strategies

Describe here your design strategies and decisions that impact your overall software.

We opted for a mono-repo structure (i.e., having the frontend, backend, and database logic all in one repository) for ease of access and setup during development; instead of starting these individual components on separate terminals, all we do is run one script (see Section 3.8 Developing, Building, and Deploying).

Also, we decided to use Python for the backend and React + Typescript for the frontend in part because these are popular programming languages and frameworks, which ease the process of finding a contractor that has the language know-how to further develop this project (see Section [LINK TO SECTION]).

3.5 System Architecture

The system of this application is divided into three principal subsystems: Database, Backend, and Frontend. Each subsystem is designed with a clear separation of concerns, ensuring modularity and maintainability.

3.5.1 Database

The Database subsytem is the foundation for data persistence and integrity. It uses PostgreSQL as the relational database engine. The database is responsible for storing and managing all core data such as projects, tasks, and accounts.

Tables: Project, Task, Task_Depends_On, Account, Session

3.5.2 Backend

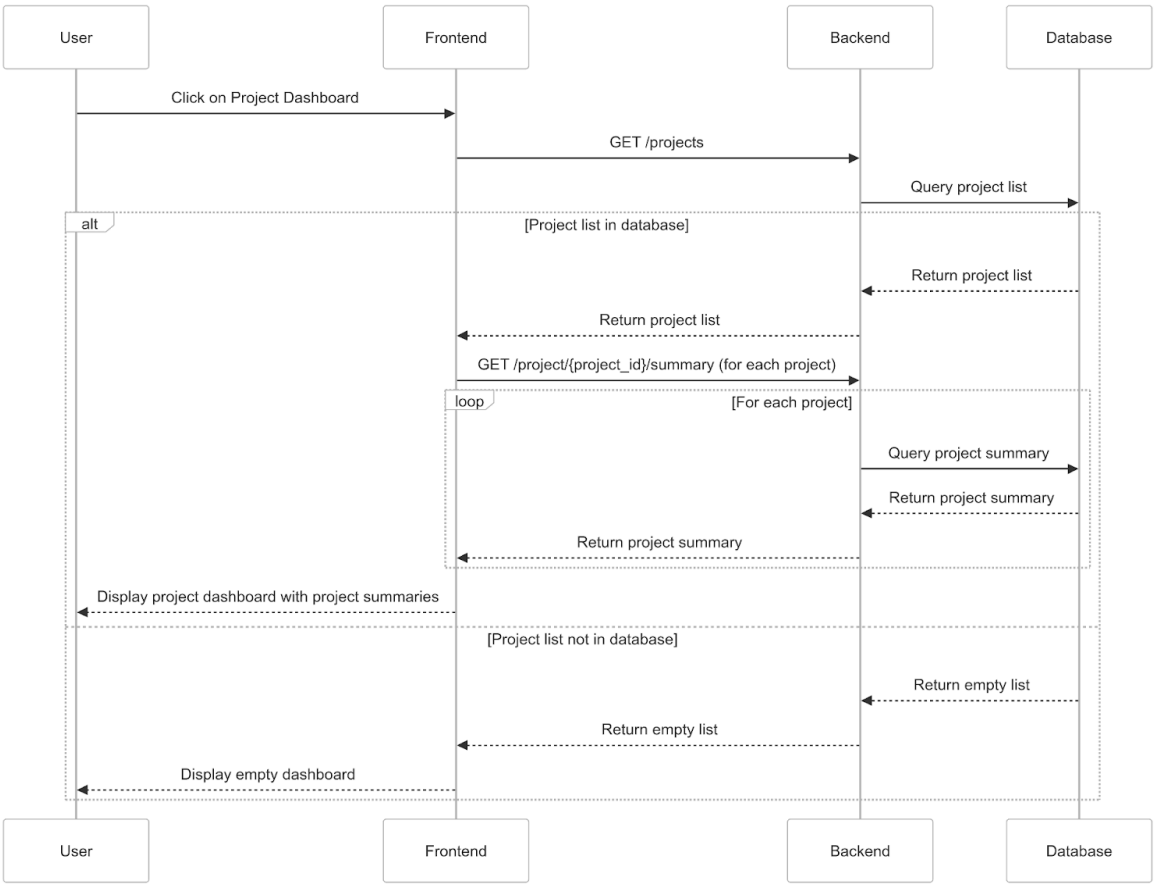
The Backend subsystem handles the business logic, data processing, and communication with the Database. The Backend is built using FastAPI. It exposes endpoints to the frontend for performing CRUD operations. Refer to section 3.10 API Documentation for detail information about endpoints.

3.5.3 Frontend

The Frontend subsystem provides the user interface for interacting with the application. The Frontend communicates with the backend via HTTP requests to perform actions such as viewing project summaries, updating tasks, and managing dependencies.

3.5.4 Example

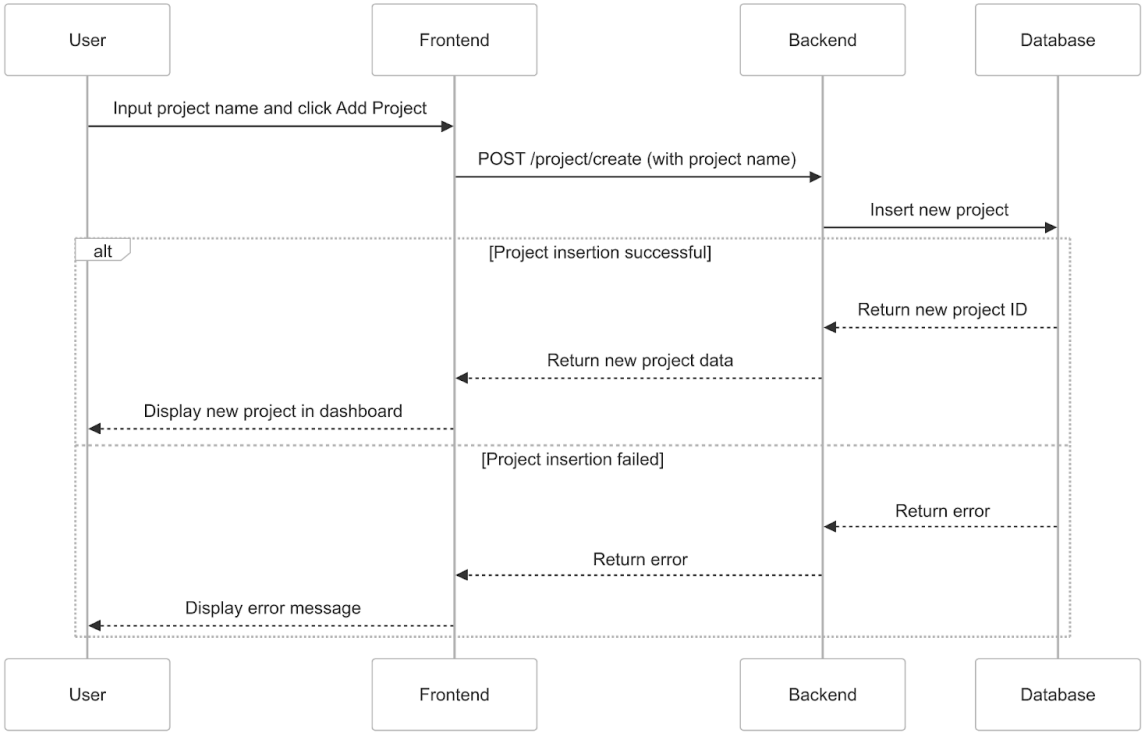
This is an example of the flow between Frontend, Backend, and Database when User clicks on Project Dashboard



1. User clicks on Project Dashboard.
2. The frontend sends a GET /projects request to the backend.
3. The backend queries the PostgreSQL database for the project list.
4. If there are projects in the database, the backend returns the project list to the frontend.
5. The frontend then sends a GET /project/{project_id}/summary request to the backend for each project to fetch detailed project summaries.
6. For each project, the backend queries the PostgreSQL database for the project summary.
7. The backend returns the project summary to the frontend.
8. The frontend displays the project dashboard with the project summaries to the user.

9. If there is no project in the database, the database returns empty list to the backend.
10. The backend returns empty list to the frontend.
11. The frontend display empty dashboard.

This is another example when User input a project name and click on Add Project Button



1. User inputs project name and clicks on Add Project.
2. The frontend sends a Post /project/create request with project name in the request body to the backend.
3. The backend inserts the new project into the PostgreSQL database.
4. If the project insertion is successful, the database returns the new project ID to the backend.
5. The backend returns the new project data to the frontend.
6. The frontend displays the new project in the dashboard.
7. If the project insertion fails, the database returns an error to the backend.
8. The backend returns the error to the frontend.
9. The frontend displays an error message to the user.

3.6 Policies and Tactics

3.6.1 Coding Standards

Naming:

- Camel case for variable, function, and class names
- PascalCase for file names.

Styling:

- CSS classes are used for styling components
- Inline styles are used for conditional styling

Type Annotations:

- TypeScript interfaces and types are used to define the shape of props and state.
- Type annotations are used for function parameters and return types.

3.6.2 Design Choices

PostgreSQL for database layer: PostgreSQL is an advanced open-source relational database management system (RDBMS) known for its robustness, scalability, and support for complex queries.

FastAPI with asyncpg: this combination provides seamless and efficient interactions with PostgreSQL database. It can help with making plain SQL as text to make database calls. FastAPI provides high performance due to asynchronous capabilities and automatic generation of interactive API documentation.

Python for backend: Python is a very readable and versatil language that provides high performance and concurrency. Python works perfectly with FastAPI and PostgreSQL.

React with typescript for frontend: React provides a strong foundation for responsive and great user interfaces. Using typescript not only improves code readability but also facilitates collaboration with teams, as strong typing provides a clear definition of components and props. Typescript has a rich ecosystem of libraries and tools, which can help implementing features much more efficient.

3.7 Detailed System Design

3.7.1 Resources

[...]

3.7.X Detailed Subsystem Design

Write here the detailed subsystem design. Describe each sub-section in a detailed description.

Chart.js
Classification: Data Visualization Library
Definition: Chart.js is a flexible charting library for JavaScript.
Responsibilities: Render charts, provide interactive data visualization.
Constraints: Limited to predefined chart types and limited chart components customization.
Composition: Chart Components, Plugins, Scales.
Uses/Interactions: Interacts with data sources to render charts.
Resources: See bibliography for full documentation.
Processing: Processes data and renders charts using canvas.
Interface/Exports: Chart Components, Configuration Options.

Cytoscape
Classification: Data Visualization Library
Definition: Cytoscape is a visualization library for a complex network of data.
Responsibilities: Render CPM graph, provide interactive data visualization.
Constraints: Need to add
Composition: Nodes, Edges, Layouts, Styles
Uses/Interactions: Interacts with data sources to render graphs.
Resources: See bibliography for full documentation.
Processing: Processes data and renders graphs.
Interface/Exports: Graph Components.

FastAPI
Classification: Backend Framework
Definition: FastAPI is a modern and high-performance web framework for building APIs

Responsibilities: Handle HTTP requests and responses, perform CRUD operations, provide data validation and serialization.
Constraints: Requires knowledge of Python and asynchronous programming.
Composition: Endpoints, Models, Dependency Injection.
Uses/Interactions: Interacts with databases and frontend clients.
Resources: See bibliography for full documentation.
Processing: Processes HTTP requests and returns JSON responses, validates and parses input data.
Interface/Exports: API Endpoints.

hello-pangea/dnd
Classification: Drag and Drop Library
Definition: A modern drag and drop library for React
Responsibilities: Enable drag-and-drop functionality in React applications, manage drag-and-drop state and events.
Constraints: Drag-and-drop functionality specifically built for lists, so other layouts are not supported
Composition: DragDropContext, Draggable, Droppable components.
Uses/Interactions: nteracts with React components to provide drag-and-drop capabilities.
Resources: See bibliography for full documentation.
Processing: Handles drag-and-drop events and updates the UI accordingly.
Interface/Exports: DragDropContext, Draggable, Droppable components.

i18next
Classification: Internationalization Library
Definition: i18next is an internationalization-framwork for Javascript.
Responsibilities: Provide localization and translation.
Constraints: Requires understanding of reducers, actions, and middleware.
Composition: Translation files, Language Detector, Backend Connector.
Uses/Interactions: Interacts with React components to provide translations.
Resources: See bibliography for full documentation.
Processing: Loads and applies translations based on the current language config.
Interface/Exports: Translation functions, Language Switcher.

Pipenv
Classification: Dependency Management Tool
Definition: Pipenv is a Python virtualenv management tool
Responsibilities: Manage Python project dependencies, create and maintain virtual environments.
Constraints: Slow dependency resolution, limited compatibility with some tools, strict dependency management
Composition: Pipfile, Pipfile.lock, Virtual Environment.
Uses/Interactions: Interacts with Python projects to manage dependencies, works with virtual environments to isolate project dependencies.
Resources: See bibliography for full documentation.
Processing: Installs, updates, and removes project dependencies, generates and maintains Pipfile and Pipfile.lock for dependency management.
Interface/Exports: Pipfile, Pipfile.lock, Virtual Environment.

PostgreSQL
Classification: Object-relational Database Management System.
Definition: Please refer to bibliography for full definition and documentation.
Responsibilities: User Authentication, Persistent Data Storage
Constraints: Steep learning curve and operational complexity.
Composition: Please see Entity Relation Diagram for table composition.
Users/Interactions: Interact with backend for schema initialization and queries.
Resources: Please see bibliography for full documentation.
Processing: PostgreSQL supports SQL queries, thereby making it extremely powerful and versatile. Please refer to the SQL documentation in the bibliography for further details.
Interface/Exports: Please see bibliography for full documentation.

React + Vite
Classification: Frontend Framework
Definition: React is a JavaScript library for building user interfaces. Vite is a build tool that provides a faster and leaner development experience for modern web projects.
Responsibilities: Build and render UI components, manage component state and life cycle.
Constraints: Limits the application to a web app.

Composition: Components, Hooks, Context API, JSX.
Uses/Interactions: Interacts with backend APIs, integrates with state management libraries like Redux.
Resources: See bibliography for full documentation.
Processing: Renders UI based on state and props, handles user events and updates state.
Interface/Exports: Components, Hooks, Context Providers.

Redux
Classification: State Management Library
Definition: Redux is a predictable state container for JavaScript apps.
Responsibilities: Manage application state, enable state persistence and debugging.
Constraints: Requires understanding of reducers, actions, and middleware.
Composition: Store, Reducers, Actions, Middleware.
Uses/Interactions: Interacts with React components and backend APIs.
Resources: See bibliography for full documentation.
Processing: Dispatches actions to update state, selects state for use in components.
Interface/Exports: Store, Actions, Reducers.

3.8 Developing, Building, and Deploying

3.8.1 Prerequisites

3.8.1.1 Tools

This application requires the installation of the following tools:

- [LINK TO TOOL DOCUMENTATION] (vXX.XX.X+)
- PostgreSQL
- Node.js
- Npm
- Python
- Pipenv

3.8.1.2 Environment Variables

The application also requires a set of environment variables:

- [DATABASE VAR]

- [BACKEND VAR]
- [FRONTEND VAR]

3.8.1.3 Database

The application also requires a PostgreSQL database. The exact steps to set up the database may vary across operating systems, but below is a set of instructions one of our team members used. Please adjust as necessary for your environment.

You may also need to install psql on your command line, and ensure you have a PostgreSQL server running.

Configuration

1. Create a Postgres data directory.

```
sudo mkdir -p /var/lib/postgres/data
sudo chown postgres:postgres /var/lib/postgres/data
```

2. Initialize Postgres database.

```
sudo -u postgres initdb -D /var/lib/postgres/data
```

3. Start Postgres.

```
sudo systemctl start postgresql
```

Database and User Setup

4. Create a superuser for your system username.

```
sudo -u postgres createuser --superuser $(whoami)
```

5. Create databases.

```
sudo -u postgres createdb $(whoami)
sudo -u postgres createdb census
```

Verification

6. Verify the PostgreSQL service is running:

```
sudo systemctl status postgresql
```

3.8.2 Starting development environment

Assuming the project files are downloaded, in terminal (bash terminal):

1. Navigate to the project file directory (at the root directory **~/**).
2. Start Postgres.
3. Run `chmod u+x ./init.sh`. This grants the user permission to execute `./init.sh`.
4. Run `./init.sh`. This will automate running `./install.sh`, `./setup.sh`, then `./run.sh`. If there are any issues encountered, the three `.sh` scripts can be run manually for debugging purposes. Notably, `./setup.sh` depends on **Step 2** to be done properly.
    a. `./install.sh` installs project dependencies. More specifically, it will install npm packages (for the frontend) listed in **~/frontend/package.json**, set up the python virtual environment, then install python dependencies (for the backend) listed in **~/backend/Pipfile**.
    b. `./setup.sh` initializes and populates the Postgres database. More specifically, it runs `.sql` scripts in **~/database/**. See **3.9 File Structure Overview** for more details.
    c. `./run.sh` starts both the backend and frontend development servers. More specifically, it ensures that any processes using the designated ports are terminated before starting the servers, and it handles cleanup when the script is interrupted.

If the above steps are properly executed, this will start…

- a backend server on `localhost:8000`
- a frontend server on `localhost:5173`
- a database instance

3.8.3 Deploying the application

[we haven't gotten here, yet…]

3.9 File Structure Overview

Notation

- **name/** : File directory.
- **name.extension** : File.

- **~/** : The root directory for all source code.
    - **backend/** : Contains all backend logic.
        - **client/** : Contains API client wrappers.
        - **database/** : Contains modules that manage database interactions.
        - **routers/** : Contains API endpoint route definitions for various functionalities (authentication, project management, etc.).
        - **utils/** : Contains reusable utilities or helper functions used across the backend.
        - **.env** : Environment configuration for the backend.
        - **config.py** : Configuration settings for the backend application.
        - **main.py** : The entry point for the backend application.
        - **Pipfile** : Dependency management file.
    - **database/** : Contains SQL logic to set up the Postgres Database.
        - **data.sql** : Populates the database with sample data (mainly for testing and debugging purposes).
        - **schema.sql** : Dictates the structure of the data in the database. More specifically, this contains table, view, and trigger definitions.
        - **system.sql** : Creates the necessary setup (including extension and functions) for the 'admin' database.
    - **frontend/** : Contains all frontend logic.
        - **public/** : Contains static images.
        - **src/** : Store the main codebase of the application
            - **assets/** : Images like icons
            - **components/** : Reusable React components.
                - **GenericComponents/** : Contains generic reusable components.
                    - **AutoExpandingTextarea/** : Textarea input component with auto expanding capability
                    - **ConfirmPopup/** : Popup component to confirm or cancel actions.
                    - **Dropdowns/** : Contains different types of dropdown components.
                    - **EditingHeader/** : Contains header components with edit button functionality.
                    - **LanguageSelector/** : Component for selecting the language of the application.
                    - **NameEditor/** : Component for editing names.

        - **NumberEditor/** : Component for editing numerical values.
        - **Path/** : Contains components related to displaying and editing paths.
        - **Popup/** : General popup component.
        - **SimpleDatePicker/** : Date picker component.
        - **StatusSelector/** : Status selector component for projects and tasks.
    - **Navbar/** : Reusable Navigation Bar component.
    - **ProjectComponents/** : Contains reusable components related to projects.
        - **AddProjectButton/** : Button for adding a new project.
        - **ProjectPath/** : Displaying the path of a project.
        - **ProjectRow/** : Contains components related to displaying project rows.
        - **ProjectSidebar/** : Sidebar component for navigating project-related sections.
        - **ProjectView/** : Contains components related to viewing project details.
    - **TaskComponents/** : Contains reusable components related to tasks.
        - **AddTaskButton/** : Button for adding a new task.
        - **TaskAssignPicker/** : Assigning tasks to users
        - **TaskDeleteButton/** : Button for deleting tasks
        - **TaskDependsEditor/** : Editing task dependencies
        - **TaskDescription/** : Displaying and editing tasks descriptions.
        - **TaskFields/** : Displaying and editing task fields.
        - **TaskIcon/** : Displaying task icons
        - **TaskName/** : Displaying and editing task names.
        - **TaskPopup/** : Popup for task-related actions.
        - **TaskRow/** : Displaying task rows
        - **TaskStatusSelector/** : Selecting status of tasks.
- **hooks/** : Manages state and features
- **pages/** : Displays sections
- **redux/** : Helps manage shared application
- **types/** : A blueprint for an object

        - `utils/` : Reusable helper functions, classes, or modules that handle repetitive functionality
        - `i18n.tsx` : Configuration for internationalization.
    - `.env` : Stores environment variables
    - `package.json` : Dependency management file.
    - `README.md` : A markdown file used to provide documentation
    - [...]
- `init.sh` : Shell script that automates running `install.sh`, `setup.sh`, and `run.sh`.
- `install.sh` : Shell script that installs necessary dependencies and prerequisites.
- `README.md` : Documentation for the project.
- `run.sh` : Shell script that starts the backend and frontend processes.
- `setup.sh` : Shell script that sets up the environment and performs initial configuration.

3.10 API Documentation

Frontend: http://localhost:5173/
Backend: http://localhost:8000/ (for shorthand, `BACKEND/`)

All of the endpoints detailed in this section are hosted by the backend.

3.10.1 Authentication

✓ `POST` `/auth/register/`

Registers a new user account.

Request Body Parameter

Key	Type	Value
email	string	The email of the user being added.
password	string	The password of the user being added.

Sample Request

`http://localhost:8000/auth/register/`

Sample Body

```
{
  "email": "account",
  "password": "password"
}
```

Sample Response

```
{
    "message" : "Account created successfully",
}
```

✓ **POST /auth/login/**

Authenticates and logs in a user.

Request Body Parameter

Key	Type	Value
email	string	The email of the user's account.
password	string	The password of the user's account.

Sample Request

http://localhost:8000/auth/login/

Sample Body

```
{
  "email": "account",
  "password": "password"
}
```

Sample Response

```
{
   "message" : "Successfully logged in",
}
```

✓ **GET** **/auth/ping/**

Checks if a user is authenticated.

Sample Request

http://localhost:8000/auth/ping/

Sample Response

```
{
   "account" : "email",
}
```

3.10.2 Projects

✓ **GET** **/projects/**

Retrieves a list of all available projects.

Sample Request

http://localhost:8000/projects/

Sample Response

```
[
 {
   "id": 1,
   "parent": null,
   "name": "Awards Ceremony",
```

```
    "description": “An annual event”,
    "status": "to_do",
    "budget": 5000.00,
    "created_at": "2025-02-15T20:49:39.261610",
    "requested_by": “John Doe”,
    "date_requested": "2025-02-10",
    "actual_start_date": null,
    "actual_completion_date": null,
    "target_start_date": "2025-03-01",
    "target_completion_date": "2025-03-15",
    "archived": false
  }, …
]
```

✓ **DELETE** **/projects/delete**

Deletes multiple projects.

Sample Request

http://localhost:8000/projects/delete/

Sample Body

```
{
  "project_ids": [
    1,
    2
  ]
}
```

Sample Response

```
{
  "num_deleted": [
    {
      "count": 2
    }
  ]
}
```


3.10.3 Project

✓ **GET** **/project/{project_id}**

Retrieves details of a specific project.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

http://localhost:8000/project/1

Sample Response


```
{
  "id": 1,
  "parent": null,
  "name": "Awards Ceremony",
  "description": null,
  "status": "to_do",
  "budget": null,
  "created_at": "2025-02-15T20:49:39.261610",
  "requested_by": null,
  "date_requested": null,
  "actual_start_date": null,
```

```
"actual_completion_date": null,
"target_start_date": null,
"target_completion_date": null,
"archived": false,
"tasks": [
 {
  "id": 1,
  "project_id": 1,
  "name": "Create supply list",
  "description": null,
  "status": "done",
  "created_at": "2025-02-15T20:49:39.265541",
  "expected_cost": null,
  "actual_cost": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-14",
  "target_completion_date": "2025-03-15",
  "target_days_to_complete": 3,
  "depends_on": []
 }, …
],
"sub_projects": [
 {
  "id": 3,
  "parent": 1,
  "name": "Create Poster",
  "description": null,
  "status": "to_do",
  "budget": null,
  "created_at": "2025-02-15T20:49:39.26161",
  "requested_by": null,
  "date_requested": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": null,
  "target_completion_date": null,
```

```
    "archived": false
   }
  ],
  "path": [
   {
    "id": 1,
    "name": "Awards Ceremony"
   }
  ]
}
```


✓ **POST** **/project/create**

Creates a new project.

Sample Request

http://localhost:8000/project/create

Sample Body


```
{
   "name": "New Project",
}
```


Sample Response


```
{
  "id": 1,
  "parent": null,
  "name": "New Project",
  "description": null,
  "status": "to_do",
```

```
  "budget": null,
  "created_at": "2025-02-25T14:01:27.636867",
  "requested_by": null,
  "date_requested": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": null,
  "target_completion_date": null,
  "archived": false,
  "status_counts": {
    "done": 0,
    "to_do": 0,
    "on_hold": 0,
    "in_progress": 0
  },
  "expected_cost": 0,
  "actual_cost": 0,
  "budget_variance": 0
}
```


✓ GET /project/{project_id}/all-tasks

Fetches all tasks that belong to the project and any of its sub-projects.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

http://localhost:8000/project/1/all-tasks

Sample Response


```
[
```

```
{
  "id": 1,
  "project_id": 1,
  "name": "Create supply list",
  "description": null,
  "status": "done",
  "created_at": "2025-02-15T20:49:39.265541",
  "expected_cost": null,
  "actual_cost": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-14",
  "target_completion_date": "2025-03-15",
  "target_days_to_complete": 3,
  "depends_on": []
},
{
  "id": 2,
  "project_id": 1,
  "name": "Buy supplies",
  "description": null,
  "status": "in_progress",
  "created_at": "2025-02-15T20:49:39.265541",
  "expected_cost": null,
  "actual_cost": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-15",
  "target_completion_date": "2025-03-16",
  "target_days_to_complete": 4,
  "depends_on": [
    {
      "task_id": 1,
      "project_id": 1
    }
  ]
},
```

```
 {
   "id": 3,
   "project_id": 3,
   "name": "Design poster",
   "description": null,
   "status": "to_do",
   "created_at": "2025-02-15T20:49:39.265541",
   "expected_cost": null,
   "actual_cost": null,
   "actual_start_date": null,
   "actual_completion_date": null,
   "target_start_date": "2025-03-09",
   "target_completion_date": "2025-03-10",
   "target_days_to_complete": 5,
   "depends_on": []
 }
]
```


✓ **GET** **/project/{project_id}/summary**

Retrieves summary statistics of the project.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

http://localhost:8000/project/1

Sample Response


```
{
   "id": 1,
   "parent": null,
```

```
  "name": "Awards Ceremony",
  "description": “An annual event”,
  "status": "to_do",
  "budget": 5000.00,
  "created_at": "2025-02-15T20:49:39.261610",
  "requested_by": “John Doe”,
  "date_requested": "2025-02-10",
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-01",
  "target_completion_date": "2025-03-15",
  "archived": false
}
```


✓ **GET** **/project/{project_id}/cpm**

Fetches CPM statistics of tasks that belong to the project and any of its sub-projects.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

http://localhost:8000/project/1/cpm

Sample Response


```
{
 "id": 1,
 "cpm": [
  {
   "id": 1,
   "project_id": 1,
   "earliest_start": 0,
```

```
    "earliest_finish": 5,
    "latest_start": 0,
    "latest_finish": 5,
    "slack": 0,
    "is_critical": true,
    "dependencies": []
   },
   {
    "id": 2,
    "project_id": 1,
    "earliest_start": 0,
    "earliest_finish": 4,
    "latest_start": 8,
    "latest_finish": 12,
    "slack": 8,
    "is_critical": false,
    "dependencies": [0]
   },
   {
    "id": 3,
    "project_id": 1,
    "earliest_start": 4,
    "earliest_finish": 6,
    "latest_start": 12,
    "latest_finish": 14,
    "slack": 8,
    "is_critical": false,
    "dependencies": [2]
   }
  ],
  "cycleInfo": []
}
```


✓ **GET /project/{project_id}/evm**

Fetches aggregate EVM statistics of tasks that belong to the project and any of its sub-projects.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

`http://localhost:8000/project/1/evm`

Sample Response


```
{
  "id": 1,
  "evm": [
    {
      "earned_value": 0,
      "planned_value": 0,
      "actual_cost": 0,
      "schedule_variance": 0,
      "cost_variance": 0,
      "schedule_performance_index": null,
      "cost_performance_index": null,
      "estimate_at_completion": null
    }
  ]
}
```


✓ **GET** `/project/{project_id}/sensible_scheduling`

Generates a sensible schedule for a project based on the provided start date, end date

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Query Parameter

Key	Type	Value
wanted_start	string	The desired start date of the project in YYYY-MM-DD format.
wanted_end	string	The desired end date of the project in YYYY-MM-DD format.

Sample Request

http://localhost:8000/project/1/sensible_scheduling?wanted_start=2025-03-10&wanted_end=2025-03-17

Sample Response


```
{
 "id": 1,
 "suggested_schedule": [
  {
   "task_id": 1,
   "start_date": "2025-03-06",
   "end_date": "2025-03-10"
  },
  {
   "task_id": 3,
   "start_date": "2025-03-12",
   "end_date": "2025-03-17"
  },
  {
   "task_id": 2,
   "start_date": "2025-03-06",
   "end_date": "2025-03-11"
  }
 ],
```

```
  "givenDurationOverridden": "True",
  "projectStartDate": "2025-03-06",
  "projectEndDate": "2025-03-17",
  "projectDurationInDays": 7
}
```


3.10.4 Task

✓ **GET** **/project/{project_id}/task/{task_id}**

Retrieves details of a specific task.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.
task_id	integer	Unique identifier for the task.

Sample Request

http://localhost:8000/project/1/task/1

Sample Response


```
{
  "id": 1,
  "project_id": 1,
  "name": "Create supply list",
  "description": null,
  "status": "done",
  "created_at": "2025-02-15T20:49:39.265541",
  "expected_cost": null,
  "actual_cost": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-14",
```

```
  "target_completion_date": "2025-03-15",
  "target_days_to_complete": 3,
  "depends_on": []
}
```


✓ **PUT** **/project/{project_id}/task/{task_id}/update**

Updates details of a specific task.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.
task_id	integer	Unique identifier for the task.

Sample Request

http://localhost:8000/project/1/task/1/update

Sample Body


```
{
    "name": "Create supply list (updated)",
    "description": "Updated description: finalize supply details",
    "status": "in_progress",
    "expected_cost": 95,
    "actual_cost": null,
    "actual_start_date": "2025-03-14T08:00:00Z",
    "actual_completion_date": null,
    "target_start_date": "2025-03-14",
    "target_completion_date": "2025-03-15",
    "target_days_to_complete": 3,
    "depends_on": []
}
```

Sample Response

```
{
  "message": "task updated successfully"
}
```

✓ **POST** **/project/{project_id}/task/create**

Updates details of a specific task.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.

Sample Request

http://localhost:8000/project/1/task/create

Sample Body

```
{
    "name": "New Task",
    "description": "Detailed description of the new task",
    "status": "to_do",
    "target_start_date": "2025-03-20",
    "target_completion_date": "2025-03-22"
}
```

Sample Response

```
{
  "id": 8,
```

```
  "project_id": 1,
  "name": "New Task",
  "description": "Detailed description of the new task",
  "status": "to_do",
  "created_at": "2025-02-15T22:00:46.944756",
  "expected_cost": null,
  "actual_cost": null,
  "actual_start_date": null,
  "actual_completion_date": null,
  "target_start_date": "2025-03-20",
  "target_completion_date": "2025-03-22",
  "target_days_to_complete": null
}
```


✓ **DELETE** **/project/{project_id}/task/{task_id}/delete**

Deletes a specific task.

Path Parameter

Key	Type	Value
project_id	integer	Unique identifier for the project.
task_id	integer	Unique identifier for the task.

Sample Request

http://localhost:8000/project/1/task/1/delete

Sample Response


```
{
  "message": "task deleted successfully"
}
```

3.10.5 Tasks

✓ **DELETE** **/tasks/delete**

Deletes multiple tasks.

Sample Request

http://localhost:8000/tasks/delete

Sample Body

```
{
  "task_ids": [
    {
      "project_id": 1,
      "id": 1
    },
    {
      "project_id": 1,
      "id": 2
    }
  ]
}
```

Sample Response

```
{
  "num_deleted": [
    {
      "count": 2
    }
  ]
}
```

3.10.6 Accounts

✓ **GET** **/accounts**

Retrieves a list of all available accounts.

Sample Request

http://localhost:8000/accounts/

Sample Response


```
[
 {
  "id": 1,
  "email": "alexander.siracusa@gmail.com",
  "first_name": "alexander",
  "last_name": "siracusa"
 },
 {
  "id": 2,
  "email": "azgu@wpi.edu",
  "first_name": "alexander",
  "last_name": "gu"
 }
]
```


4. Glossary

Write here an ordered list of designed items used throughout the document.

5. Bibliography

Chart.js | chart.js. (n.d.). https://www.chartjs.org/docs/latest/

FastAPI. (n.d.). https://fastapi.tiangolo.com/

Franz, M. (n.d.). *Cytoscape.js*. https://js.cytoscape.org/#getting-started

Getting started. (n.d.). Vitejs. https://vite.dev/guide/

Getting Started with Redux | Redux. (2024, March 31). https://redux.js.org/introduction/getting-started

Hello-Pangea. (n.d.). *GitHub - hello-pangea/dnd.* https://github.com/hello-pangea/dnd?tab=readme-ov-file#documentation-

Introduction | i18next documentation. (n.d.). https://www.i18next.com/

Pipenv: Python Dev Workflow for Humans — pipenv 2024.4.0 documentation. (n.d.). https://pypi.org/project/pipenv/2024.4.0/

PostgreSQL: documentation. (n.d.). The PostgreSQL Global Development Group. https://www.postgresql.org/docs/

Quick start – react. (n.d.). https://reactjs.org/docs/getting-started.html

6. Appendix

[...]