Feature Development Plan Template

Feature Name: First 10 CRUD Operations

The requirement involves implementing CRUD (Create, Read, Update, Delete) operations for the first 10 fields within the Project Management System. These fields include Project Budget, Version History, Project Description, Scope, Project Stack (Tech), Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint wise detail, and Detailed timeline reference. The goal is to provide functionalities to manage and track information related to these aspects of the projects efficiently.

1. **Requirement Analysis**

This feature involves implementing CRUD operations for the first 10 fields of the Project Management System: Project Budget, Version History, Project Description, Scope, Project Stack (Tech), Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint wise detail, and Detailed timeline reference.

Key Requirements:

- **Project Budget:**
    - Create, read, update, and delete operations for managing the project budget.
- **Version History:**
    - CRUD operations to track and manage the version history of the project.
- **Project:**
    - Functionalities to add, view, update, and delete project descriptions and other details of project.
    - Enable users to manage project scopes through CRUD operations.
- **Project Stack (Tech):**
    - CRUD operations for managing the technological stack used in the project.
- **Escalation Matrix:**
    - Implement functionalities to add, view, update, and delete escalation matrices for project issues.
- **Stakeholders:**
    - CRUD operations for managing project stakeholders.
- **Risk Profiling:**
    - Enable users to profile and manage project risks through CRUD operations.
- **Phases/Milestones:**
    - Implement functionalities to add, view, update, and delete project phases or milestones.

- **Sprint wise detail:**
    - CRUD operations for managing details of each sprint within the project.
- **Detailed timeline reference:**
    - Enable users to manage detailed timeline references for the project.

2. Impact Analysis

Affected APIs:

- API endpoints for CRUD operations on each of the first 10 fields.

Analysis:

- Implementing CRUD operations will impact the backend API endpoints responsible for data manipulation.
- Creating database schemas may be necessary to accommodate new data structures or relationships.
- User interface components in the frontend will need to be updated to interact with the new CRUD functionalities.

3. Database Schema Changes

Current Schema Overview:

- Define database tables for the first 10 project fields.
- Define relationships between these entities and other relevant entities in the database schema.

4. API Endpoints Design

Endpoint Summary:

- **/api/projectbudget**: Endpoint for CRUD operations on Project Budget.

  HTTP Method: GET, POST, PUT

  Request Parameters: Project_id, Project_type, Durection, Budgeted_Hours

  Request Body Schema: budget **{**Project_id, Project_type, Durection, Budgeted_Hours }

  Response Body Schema: json

  Error Codes and Messages:

    - Successful responses: 200
    - Unauthorized: 401
    - Not found: 404
    - Request Timeout: 408

- **/api/versionhistory**: Endpoint for CRUD operations on Version History.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, type, change, change_reason, created_by, Revision_Date, approved_by

  Request Body Schema: Version **{** Project_id, type, change, change_reason, created_by, Revision_Date, approved_by }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/project**: Endpoint for CRUD operations on Project Description.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, Project_description, scope

  Request Body Schema: budget **{**Project_id, Project_description, scope }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/projectstack**: Endpoint for CRUD operations on Project Stack (Tech).

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, stack

  Request Body Schema: budget **{**Project_id, stack }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/escalationmatrix**: Endpoint for CRUD operations on Escalation Matrix.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id,operational_matrix, financial_matrix, tecnival_matrix

  Request Body Schema: budget **{** Project_id,operational_matrix, financial_matrix, tecnival_matrix }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/stakeholders**: Endpoint for CRUD operations on Stakeholders.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, Title, Name,Contact

  Request Body Schema: budget **{** Project_id, Title, Name,Contact }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/riskprofiling**: Endpoint for CRUD operations on Risk Profiling.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, Risk Type, Description,Severity, Impact , Remedial, Steps, Status ,Closure _Date

  Request Body Schema: budget **{** Project_id, Risk Type, Description,Severity, Impact , Remedial, Steps, Status ,Closure _Date

  }

  Response Body Schema: json

Error Codes and Messages:

- Successful responses: 200
- Unauthorized: 401
- Not found: 404
- Request Timeout: 408

- **/api/phasesmilestones**: Endpoint for CRUD operations on Phases/Milestones.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id,Title, Start_Date ,Completion_Date ,Approval_Date ,Status , Revised_Completion_Date ,Comments

  Request Body Schema: budget **{** Project_id,Title, Start_Date ,Completion_Date ,Approval_Date ,Status , Revised_Completion_Date ,Comments

  }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/sprintdetails**: Endpoint for CRUD operations on Sprint wise detail.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, Sprint, start_date, end_date, status, comments

  Request Body Schema: budget **{**: Project_id, Sprint, start_date, end_date, status, comments

  }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

- **/api/timelinereference**: Endpoint for CRUD operations on Detailed timeline reference.

  HTTP Method: : GET, POST, PUT

  Request Parameters: Project_id, timeline

  Request Body Schema: budget **{**Project_id, timeline }

  Response Body Schema: json

  Error Codes and Messages:

  - Successful responses: 200
  - Unauthorized: 401
  - Not found: 404
  - Request Timeout: 408

Design Details:

- Each endpoint will support HTTP methods (GET, POST, PUT, DELETE) for respective CRUD operations.
- Request and response bodies will be formatted as JSON objects containing relevant data.
- Error handling will be implemented to provide meaningful error messages in case of failures.

5. **Pseudo Code for Key Functionalities**

Functionality: CRUD Operations

pseudo code

```
// CRUD operations for Project Budget

FUNCTION CreateProjectBudget(budgetData,projectID)

// Implementation logic to create project budget

END FUNCTION


FUNCTION GetProjectBudget(budgetId, projectID)

// Implementation logic to retrieve project budget by ID

END FUNCTION


FUNCTION UpdateProjectBudget(budgetId, updatedData, projectID)

// Implementation logic to update project budget by ID with updated data
```

END FUNCTION

FUNCTION DeleteProjectBudget(budgetId, projectID)

// Implementation logic to delete project budget by ID

END FUNCTION

// Similar functions for Version History, Project, Scope, Project Stack (Tech), Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint wise detail, and Detailed timeline reference.

Feature Development Plan Template

Feature Name: **Export Project Details**

Implementing the functionality to export project details as a document in a predefined format like Project Budget, Version History, Project Description, Scope, Project Stack (Tech), Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint wise detail, Detailed timeline reference) from the provided template to understand the type of operations to be exported as PDF.

1. **Requirement Analysis:**

   Developing Export Feature:
   Developing a feature that allows stakeholders to export project details seamlessly in a predefined format.

   Collaboration:
   Collaborating with the team to ensure seamless integration of export functionality.

   Testing:
   Testing the export feature to ensure compatibility and accuracy.

2. **Impact Analysis:**

   Affected APIs:
   API for export project details as a document.
   /api/export-pdf: export project details as a pdf
   /api/export-word
   Analysis:
   API that take project id as a request and send word and pdf document as a response

3. **API Endpoints Design:**

   HTTP Method: POST

   Request Parameters: Project_id, section

   Request Body Schema: budget **{**Project_id, section: [section1, section2]}

   Response Body Schema: pdf

   Error Codes and Messages:

   - Successful responses: 200
   - Unauthorized: 401
   - Not found: 404
   - Request Timeout: 408

   /api/export-word: export project details as document

HTTP Method: POST

Request Parameters: Project_id, section

Request Body Schema: budget **{**Project_id, section: [section1, section2]}

Response Body Schema: word

Error Codes and Messages:

- Successful responses: 200
- Unauthorized: 401
- Not found: 404

Request Timeout: 408

4. **Pseudo Code for Key Functionalities:**

Export ad pdf

Functionality: Export as pdf

Function Name**: ExportPdf**

Parameters:

• project_id: Integer (The unique identifier for the project)

• section: Array of string(Section name which you want to download)

```
FUNCTION ExportPdf (project_id, section){
    IF project_id is in projectsArrays THEN
            IF section in allSections THEN
                    VAR pdf
                    VAR details = Find contant from database using project_id and section
                    Pdf.add(details)
                    RETURN Pdf
            ELSE RETURN "Error: Section id not valid"
    ELSE RETURN "Error: Project id not valid"
}
```

Export ad Doc

Functionality: Export as doc

Function Name**: ExportDoc**

Parameters:

- project_id: Integer (The unique identifier for the project)
- section: Array of string(Section name which you want to download)

```
FUNCTION ExportDoc (project_id, section){
    IF project_id is in projectsArrays THEN
        IF section in allSections THEN
            VAR doc
            VAR details = Find contant from database using project_id and section
            doc.add(details)
            RETURN doc
        ELSE RETURN "Error: Section id not valid"
    ELSE RETURN "Error: Project id not valid"
}
```