**Feature Development Plan Template**

**Feature Name**

*CRUD operations.*

# Requirement Analysis

**User Story Reference:** *[Link or reference to the User Story] (optinal)*

**Summary:** *This feature focuses on enhancing project management capabilities. Key requirements include implementing CRUD operations for Project Budget, Version History, Project Description, Scope, Project Tech Stack, Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint Wise Details, Detailed timeline reference, Approved Team, Resources, Client Feedback, and MoM of client meetings..*

# Impact Analysis (Optional)

**Affected APIs:** *CRUD APIs for Project Budget, Version History, Project Description, Scope, Project Tech Stack, Escalation Matrix, Stakeholders, Risk Profiling, Phases/Milestones, Sprint Wise Details, Detailed timeline reference, Approved Team, Resources, Client Feedback, MoM of client meetings.*

**Analysis:** *The introduction of these features impacts various CRUD operations, necessitating careful consideration of dependencies, potential conflicts, and areas requiring special attention during development and integration.*

# Database Schema Changes (Optional)

**Current Schema Overview:** *Tables for project-related entities.*

**Proposed Changes:** *Introduce necessary fields for each CRUD operation, establish relationships between tables, and accommodate additional data for enhanced project management..*

# API Endpoints Design

**Endpoint Summary:**

* **api/Project Budget (CRUD operations)**
* **api/Version History (CRUD operations)**
* **api/Project Description (CRUD operations)**
* **api/Scope (CRUD operations)**
* **api/Project Tech Stack (CRUD operations)**
* **api/Escalation Matrix (CRUD operations)**
* **api/Stakeholders (CRUD operations)**
* **api/** **Risk Profiling (CRUD operations)**
* **api/Phases/Milestones (CRUD operations)**
* **api/** **Sprint Wise Details (CRUD operations)**
* **api/** **Detailed timeline reference (CRUD operations)**
* **api/Approved-team (CRUD operations)**
* **api/Resources (CRUD operations)**
* **api/Client-feedback (CRUD operations)**
* **api/Project-updates (CRUD operations)**
* **api/Client-meetings (CRUD operations)**

*.*

**Design Details:** *For each endpoint, provide the following:*

* *Endpoint URL :* ***/api/Approved Team***
* *HTTP Method: POST, GET, PUT, DELETE*
* *Request Parameters : None*
* *Request Body Schema : None*
* *Response Body Schema :*

phase: {

type: String,

required: true,

},

name: {

type: String,

required: true,

},

role: {

type: String,

required: true,

},

availability: {

type: String,

required: true,

},

duration: {

type: String,

required: true,

},

* *Error Codes and Messages :*
* *200 : Successful retrieval*
* *404: Not Found: Project budget not found*
* *500 :Internal Server Error: Server failure*

# Pseudo Code for Key Functionalities

**Function Overview:** *Briefly describe the key functions to be implemented.*

**Pseudo Code:** *For each key function, provide pseudo code to illustrate the core logic and flow.*

Pseudo code is a way to describe the functionality of a program or algorithm in a format that’s more readable than actual code, but not as detailed as plain language. It helps in planning and understanding the logic before diving into the actual coding.

Let’s say the feature in question involves adding a new functionality to a user profile system, where the backend needs to handle a request to update a user’s email address. Here’s an example of how the pseudo code for this functionality might look:

**Pseudo Code for Update Email Function Functionality: Approved Team CRUD**

# Create Operation

FUNCTION CreateApprovedTeam(teamData)

# Validate teamData fields

IF teamData.name is empty OR teamData.members is empty THEN

RETURN { "status": "error", "message": "Invalid data provided for creating Approved Team" }

END IF

# Create a new team in the database

newTeam = CREATE teamData in approvedTeamTable

RETURN { "status": "success", "message": "Approved Team created successfully", "data": newTeam }

END FUNCTION

# Read Operation

FUNCTION GetApprovedTeam(teamId)

# Retrieve team from the database by teamId

team = GET team from approvedTeamTable WHERE id = teamId

# Check if team exists

IF team does not exist THEN

RETURN { "status": "error", "message": "Approved Team not found" }

END IF

RETURN { "status": "success", "data": team }

END FUNCTION

# Update Operation

FUNCTION UpdateApprovedTeam(teamId, updatedTeamData)

# Retrieve team from the database by teamId

team = GET team from approvedTeamTable WHERE id = teamId

# Check if team exists

IF team does not exist THEN

RETURN { "status": "error", "message": "Approved Team not found" }

END IF

# Update team data with updatedTeamData

UPDATE team in approvedTeamTable SET teamData = updatedTeamData

RETURN { "status": "success", "message": "Approved Team updated successfully" }

END FUNCTION

# Delete Operation

FUNCTION DeleteApprovedTeam(teamId)

# Delete team from the database by teamId

DELETE team from approvedTeamTable WHERE id = teamId

RETURN { "status": "success", "message": "Approved Team deleted successfully" }

END FUNCTION

This pseudo code outlines the basic logic for updating a user’s email. It includes input validation, checking if the user exists, and handling the database transaction, which are common steps in backend operations. Note that pseudo code is not meant to be executed, and the syntax is not language-specific. It serves as a blueprint for the actual code that will be written.

# Deployment and Configuration Changes (Optional)

**New App Settings:** *Environment variables for database connection.*

**Deployment Changes:** *Detail any changes required in the deployment process.*

**Testing and Verification:** *Includes unit tests, integration tests, and manual testing for the newly introduced functionalities..*