**Feature Development Plan Template**

**Feature Name**

*Email Notification System.*

# Requirement Analysis

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

**Summary:** *The Email Notification System aims to enhance project communication by sending real-time email notifications to stakeholders whenever there is an update or change in the system. Key requirements include seamless integration with the existing project features, customizable email triggers, and robust error handling.*

# Impact Analysis (Optional)

**Affected APIs:** *Email notification API.*

**Analysis:** *The introduction of the Email Notification System will impact the existing communication flow. Dependencies on the email service provider and potential conflicts with other notification mechanisms need to be considered. Special attention should be given to ensure reliable and timely delivery of notifications..*

# Database Schema Changes (Optional)

**Current Schema Overview:** *A brief overview of the relevant part of the current database schema.*

**Proposed Changes:** *No direct changes to the database schema are required for the Email Notification System..*

# API Endpoints Design

**Endpoint Summary: *POST /api/Email-Notification/Trigger***

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

* *Endpoint URL : /api/Email-Notification/Trigger*
* *HTTP Method: POST*
* *Request Parameters:*
* *Request Body Schema : "stakeholderId": "string", "projectId": "string", "updateType": "string"*
* *Response Body Schema : "status": "string", "message": "string"*
* *Error Codes and Messages :*

*400: Bad Request - Invalid request parameters*

*404: Not Found - Stakeholder or project not found*

*500: Internal Server Error - Failed to send email*

# Pseudo Code for Key Functionalities

**Function Overview:** *Trigger Email Notification.*

**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 Function to Trigger Email Notification**

**FUNCTION TriggerEmailNotification(stakeholderId, projectId, updateType)**

**stakeholder = FIND stakeholder in 'stakeholders' collection WHERE \_id = stakeholderId**

**IF stakeholder is NULL THEN**

**RETURN { "status": "error", "message": "Stakeholder not found" }**

**END IF**

**emailContent = GenerateEmailContent(updateType, projectId)**

**# Assuming 'emailQueue' collection has fields like 'recipient', 'subject', 'content', etc.**

**emailQueueItem = {**

**"recipient": stakeholder.email,**

**"subject": "Project Update Notification",**

**"content": emailContent,**

**"status": "pending", # Status can be 'pending', 'sent', 'failed', etc.**

**"createdAt": CURRENT\_TIMESTAMP**

**}**

**INSERT emailQueueItem INTO 'emailQueue' collection**

**RETURN { "status": "success", "message": "Email added to the queue for sending" }**

**END FUNCTION**

**# Function to Generate Email Content**

**FUNCTION GenerateEmailContent(updateType, projectId)**

**# Logic to generate email content based on the updateType and projectId**

**# This could include fetching relevant project details and creating a summary**

**RETURN "Email content based on the updateType and projectId"**

**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:** *Email service provider settings (SMTP server, credentials, etc.).*

**Deployment Changes:** *Email service provider settings (SMTP server, credentials, etc.)*

* **Testing and Verification:** *Perform unit tests for the email notification trigger functionality.*
* *Conduct integration tests for email service provider integration.*
* *Manual testing for email content generation and delivery.*