# Document

# TaskGenieAI1.0

# 1. Problem

# 2. Solution

# 3. Key features

# 4. Technologies

# 5. Technical Documentation

# Backend

# Frontend

# 6. Conclusion

# Problem: Managing multiple project boards, such as Jira and Azure DevOps, and cross-collaborating between them is challenging due to manual task transfers that often lead to errors. A few of these challenges include a lack of detailed context in task creation and overlooked test scenarios by testers. Additionally, multiple stakeholders need to spend time creating detailed tasks for sprint planning and deliverables, a recurring effort before every sprint. We may have to prepare reports manually whenever needed.

# Solution:

# Our solution integrates multiple project management platforms, automating the entire lifecycle from task creation to test case generation using AI models. By preserving the project's context and mapping developers to their roles or skills, it simplifies sprint planning and enhances collaboration. Overtime, the model learns from project tasks, enabling it to generate increasingly accurate code and test cases. This not only reduces timelines but also improves team efficiency and productivity.

# Key features:

# Seamless integration of multiple boards

# automating task transfers

# generating detailed test case with scenario coverage

# Generating Pseudo code and sample code for developers,

# Automating story creation from emails.

# Technologies: Frontend: React JS Backend: Python Web framework: Flask LLM Models:

# Qwen/Qwen2.5-Coder-0.5B-Instruct

# facebook/bart-large-mnli

# Technical Documentation

# Backend

## 1. Project Overview

This Python Flask backend application is designed to support task management, email processing, and AI-driven automation. The key features include:

* Integration with Azure DevOps for managing work items, projects, and comments.
* Email analysis to map problems to relevant users based on their expertise.
* AI-powered generation of pseudocode and test cases using pre-trained NLP models.
* API endpoints for creating and updating Azure DevOps projects and work items.

## 2. Application Architecture

### 2.1 Frameworks and Tools

* **Flask**: A lightweight web framework for building RESTful APIs.
* **Flask-CORS**: For handling cross-origin resource sharing.
* **Azure DevOps API**: For managing work items and projects.
* **Transformers Library**: Pre-trained AI models for text classification and generation.

### 2.2 Key Components

1. **Flask Application**
   * Routes for handling requests and interacting with external APIs.
   * Configurations for headers, authentication, and environment settings.
2. **Email Processing**
   * Uses IMAP to connect to an email server, fetch emails, and analyze their content using an AI model.
3. **Azure DevOps Integration**
   * Automates project creation, task assignment, and status updates.
4. **AI Features**
   * Employs text-generation and zero-shot classification models to process inputs and generate outputs dynamically.

## 3. API Documentation

### 3.1 / (Home)

* **Method**: GET
* **Description**: Displays a welcome message to indicate that the backend is running.
* **Response**:

{

"message": "Welcome to the Flask backend!"

}

### 3.2 /email\_reciver

* **Method**: POST
* **Description**: Processes an email, maps its content to users, and creates a corresponding work item in Azure DevOps.
* **Input**:

{

"username": "email@example.com",

"password": "password",

"mail\_server": "imap.example.com"

}

* **Response**:
  + **Success**:

{

"AIoutput": { ... },

"message": "Work item created successfully",

"details": { ... }

}

* + **Error**:

{

"error": "Failed to create work item",

"status\_code": 400,

"details": "Error message"

}

### 3.3 /add\_comment

* **Method**: POST
* **Description**: Adds a comment to an Azure DevOps work item.
* **Input**:

{

"work\_item\_id": 12345,

"comment\_text": "This is a test comment",

"project": "MyProject"

}

* **Response**:

**Success**:

{

"message": "Comment added successfully",

}

**Error**:

{

"error": "Failed to add comment",

"status\_code": 400,

"details": "Error message"

}

### 3.4 /generate\_pseudocode

* **Method**: POST
* **Description**: Generates pseudocode and a code sample based on a project description.
* **Input**:

{

"project": "ProjectName",

"language": "Python",

"description": "Task description here"

}

* **Response**:

{

"response": "Generated pseudocode and code sample"

}

### 3.5 /generate\_test\_cases

* **Method**: POST
* **Description**: Creates test cases for a given task using the specified testing framework.
* **Input**:

{

"project": "ProjectName",

"frame\_work": "pytest",

"description": "Test task description here"

}

* **Response**:

{

"response": "Generated test cases"

}

## 4. Integration Points

### 4.1 Azure DevOps

* **Work Item Management**: Handles creation, updating, and commenting on work items using Azure DevOps REST APIs.
* **Project Management**: Automates the setup of new projects and user assignments.

### 4.2 AI Models

#### Zero-Shot Classification

* **Model**: facebook/bart-large-mnli
* **Functionality**: This model identifies the relevance of text content by comparing it against a set of predefined labels (e.g., user roles or problem categories). In this application:
  + It is used to analyze email content and map problems to the most suitable users based on their roles.
  + A confidence score is assigned to each label, and results above a threshold are considered relevant.

#### Pseudocode and Testcases

* **Model**: Qwen/Qwen2.5-Coder-0.5B-Instruct
* **Functionality**: This model generates coherent and contextually appropriate text outputs based on the given prompts. In this application:
  + It generates pseudocode and corresponding code samples based on task descriptions provided by the user.
  + It produces test cases dynamically, considering the provided task and testing framework.
  + The model ensures the outputs are modular, clear, and adhere to best practices.

## 5. Error Handling

* Validates required fields for all API inputs.
* Catches and returns detailed error messages for failed operations.
* Ensures HTTP status codes align with REST API best practices (e.g., 400 for client errors, 500 for server errors).

## 6. Dependencies and Configuration

### 6.1 Libraries

* Flask
* Flask-CORS
* Requests
* Transformers

### 6.2 Configuration

* Environment variables for sensitive data like PAT (Personal Access Token) and DevOpsURL.
* IMAP credentials for email fetching.

## 7. Workflow and Execution

1. **Email Analysis**:
   * Fetch the latest email using IMAP.
   * Use AI models to analyze and map problems to users.
2. **Work Item Creation**:
   * Extract relevant details from email content.
   * Create a work item in Azure DevOps with associated metadata (e.g., title, description, and assigned user).
3. **AI-Driven Generation**:
   * Accept user inputs for pseudocode or test case generation.
   * Generate results dynamically using AI models.
4. **Project and Comment Management**:
   * Automate the creation of Azure DevOps projects.
   * Enable adding comments to specific work items.

## 8. Future Enhancements

* Expand AI capabilities to handle additional programming languages and test frameworks.
* Real time sprint planning,
* Advance Project and Sprint health metrics
* Single click Report/Release notes generation,
* Integration with other PM tools)

# Frontend

## Overview

The frontend application is a React-based single-page application (SPA) designed to manage projects, generate pseudocode, and create test cases. It uses React Router for navigation, Axios for API communication, and Tailwind CSS for responsive and modern UI design.

This document details the implementation, structure, functionality, and key components of the frontend application.

## Project Structure

### ****File Organization****

1. **Navbar.js**:
   * Contains the navigation bar.
   * Provides links to key features like Create Project, Generate Pseudocode, and Generate Test Cases.
   * Implements dynamic text animations for the home page.
2. **CreateProject.js**:
   * A form-based component for project creation.
   * Manages user roles, technologies, and project metadata.
3. **GeneratePseudocode.js**:
   * Allows the user to generate pseudocode for a selected story in a chosen programming language.
4. **GenerateTestCase.js**:
   * Enables users to generate test cases for stories using specified frameworks.

## Key Components and Their Details

### 1. ****Navbar****

#### Purpose

Provides navigation between different parts of the application and displays dynamic text animations.

#### Key Features

* **Dynamic Typing Animation**:
  + Displays a welcoming message when on the home page.
  + Clears text when navigating to other pages.
* **Navigation Buttons**:
  + "Create Project": Navigates to the project creation form.
  + "Generate Test Case": Redirects to the test case generation page.
  + "Generate Pseudocode": Opens the pseudocode generation feature.

#### Code Example

useEffect(() => {

if (location.pathname === "/") {

let index = 0;

const typingInterval = setInterval(() => {

setTypedText((prev) => prev + textToType[index]);

index++;

if (index === textToType.length) {

clearInterval(typingInterval);

}

}, 100);

return () => clearInterval(typingInterval);

} else {

setTypedText("");

}

}, [location.pathname]);

### 2. ****Create Project****

#### Purpose

Allows users to create a project by providing details such as project name, description, tech stack, and user roles.

#### Functionality

1. **Form Inputs**:
   * Project Name: Input field for entering the project name.
   * Description: Text area for providing project details.
   * Tech Stack: Input field for specifying the technology stack.
2. **User Management**:
   * Add/Edit/Delete Users: Manages a list of users and their roles.
3. **Form Submission**:
   * Sends project details to the backend API.

#### Key Functions

* **handleAddUser**: Adds a new user or updates an existing user.

const handleAddUser = () => {

if (name && email && techstack) {

if (editingIndex !== null) {

const updatedUsers = [...usersAndRoles];

updatedUsers[editingIndex] = { name, email, techstack };

setUsersAndRoles(updatedUsers);

setEditingIndex(null);

} else {

setUsersAndRoles([...usersAndRoles, { name, email, techstack }]);

}

setName("");

setEmail("");

setTechstack("");

setShowPopup(false);

}

};

* **handleSubmit**: Submits the project data to the backend.

#### API Integration

* **Endpoint**: POST /create\_project
* **Payload**:

{

"project\_name": "Project Name",

"description": "Project Description",

"techstack": "Tech Stack",

"users": [

{"name": "User1", "email": "user1@example.com", "techstack": "React"}

]

}

### 3. ****Generate Pseudocode****

#### Purpose

Generates pseudocode for a user-selected story in a specified programming language.

#### Functionality

1. **Dropdown Menus**:
   * **Story Name**: Fetches and displays a list of stories dynamically.
   * **Language Selection**: Allows the user to choose a programming language.
2. **Result Display**:
   * Shows the generated pseudocode upon successful API response.

#### Key Functions

* **handleGenerate**: Sends the selected story and language to the backend for pseudocode generation.

const handleGenerate = async () => {

if (!storyName || !language) {

alert("Please select both a story name and a language.");

return;

}

try {

const response = await ApiUrl.post("generate\_pseudocode", {

storyName,

language,

description,

});

setResult(response.data.result);

} catch (error) {

setResult("Failed to generate pseudocode. Please try again.");

}

};

#### API Integration

* **Endpoint**: POST /generate\_pseudocode
* **Payload**:

{

"storyName": "Story A",

"language": "Python",

"description": "Detailed explanation"

}

### 4. ****Generate Test Case****

#### Purpose

Creates test cases for user stories using a specified testing framework.

#### Functionality

1. **Dropdown Menus**:
   * **Story Name**: Dynamically fetched from the backend.
   * **Framework Selection**: User can choose from available frameworks (JUnit, Jest, etc.).
2. **Result Display**:
   * Shows generated test cases in a modal upon success.

#### Key Functions

* **handleGenerate**: Submits the selected story and framework to the backend.

const handleGenerate = async () => {

try {

const response = await ApiUrl.post("generate\_test\_cases", {

storyName,

framework: selectedFramework,

description,

});

setResult(response.data.result);

} catch (error) {

setResult("Failed to generate test cases.");

}

};

#### API Integration

* **Endpoint**: POST /generate\_test\_cases
* **Payload**:

{

"storyName": "Story B",

"framework": "Jest",

"description": "Test details"

}

## Styling

The application uses **Tailwind CSS** for styling, with:

* Responsive layouts
* Gradient backgrounds
* Modern design elements

## Technologies Used

* **Frontend Framework**: React.js
* **Styling**: Tailwind CSS
* **Routing**: React Router
* **State Management**: React useState and useContext hooks
* **API Communication**: Axios

## Future Enhancements

* Add global state management using Redux or Context API.
* Implement user authentication and authorization.
* Improve error handling and display detailed API errors to users.
* Optimize API calls using caching strategies.

**Conclusion**

1.**Enhanced Collaboration**: The integration of multiple project management platforms significantly improves collaboration among teams by automating task transfers and preserving project context, which minimizes errors and miscommunication.

2.**Increased Efficiency**: By automating the entire lifecycle from task creation to test case generation, the solution reduces the time spent on manual processes, allowing teams to focus on delivering high-quality work more efficiently.

3.**AI-Driven Insights**: The utilization of AI models for generating pseudocode and test cases not only streamlines development but also enhances the accuracy and relevance of outputs, adapting over time to better meet project needs.

4.**User-Friendly Interface**: The React-based frontend provides an intuitive user experience, enabling users to easily navigate through features such as project creation and test case generation, thereby enhancing overall usability.

5**.Future Scalability**: The architecture is designed with scalability in mind, allowing for future enhancements such as support for additional programming languages and testing frameworks, real-time sprint planning, and integration with other project management tools.

6.**Strategic Advantage**: Implementing this solution positions organizations to respond more effectively to project demands, ultimately leading to improved productivity and a competitive edge in the fast-paced software development landscape.