API BDD Test Code Generator

1. Introduction:

This document outlines the design of a generative AI-based system using AWS services like Bedrock to automate the generation of Gherkin scenarios and step definitions, with schema validation to ensure accurate input. The system utilizes state-of-the-art AI models hosted on AWS to support various HTTP methods (GET, POST, PUT, DELETE) and different programming languages, enabling seamless creation of test scenarios and step definitions for diverse testing frameworks. Generated assets are stored in a well-structured file system to ensure easy access and management, integrated with AWS services for scalable deployment and storage.

2. Design Overview:

2.1. Core Components

The API Scenario Generation system consists of several core components, each responsible for specific tasks to ensure proper generation of Gherkin scenarios and step definitions:

1. **Input Parser**: Takes API details as input, including endpoint URLs, HTTP methods, request headers, body, and expected response structure.
2. **Scenario Generator**: Processes API details and generates Gherkin scenarios based on the input. It handles different HTTP methods and builds appropriate scenarios for each.
3. **Step Definition Generator**: Creates step definitions for the scenarios in various programming languages such as Java, JavaScript, Python, and C#.
4. **Schema Validator**: Ensures that the input data conforms to the expected schema, preventing invalid API details from being processed.
5. **Output Formatter**: Formats the generated scenarios and step definitions into the desired structure and programming language syntax.
6. **Storage Module**: Saves the generated Gherkin scenarios and step definitions to a structured file system.

2.2. Schema Validation

The **Schema Validator** is a critical component in ensuring that the input data is correct. It validates the following aspects:

- **Request URL**: Checks the format and structure of the URL.
- **HTTP Method**: Validates that the method (GET, POST, PUT, DELETE) is supported.
- **Request Headers**: Ensures that mandatory headers such as Content-Type and Authorization are present.
- **Request Body**: For POST and PUT requests, ensures the JSON body or form data is structured correctly.
- **Response Structure**: Validates the expected response schema, including the status code and data format (JSON, XML, etc.).

2.3. File Structure

The generated Gherkin scenarios and step definitions are stored in a predefined file structure to ensure consistency and easy access. Below is an example of the file structure:

- **<endpoint_name>**: A directory for each API endpoint (e.g., /users, /products).
- **GET/POST/PUT/DELETE**: Subdirectories for each HTTP method.
- **scenario.feature**: Gherkin feature file with scenarios.
- **step_definitions.***: Step definition files in different programming languages.

2.4. Workflow

The workflow of the system involves the following steps:

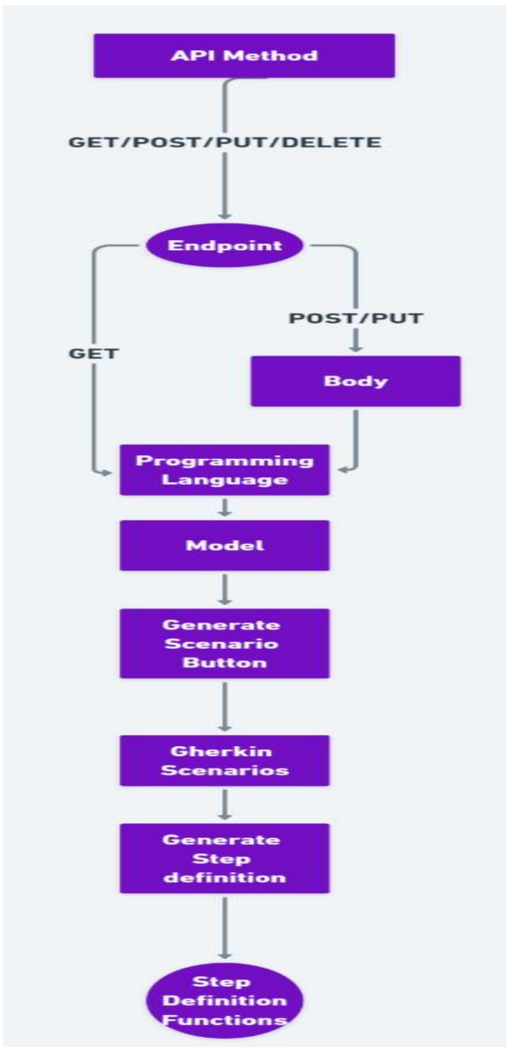
1. **Input Collection**: The user provides API details, including the endpoint, HTTP method, request body, headers, and expected response.
2. **Schema Validation**: The Schema Validator checks the input data for correctness.
3. **Scenario Generation**: Based on the API details, the Scenario Generator creates Gherkin scenarios.
4. **Step Definition Generation**: The Step Definition Generator generates corresponding step definitions in the desired programming language.

5. **File Storage**: The generated scenarios and step definitions are saved to a structured file system.

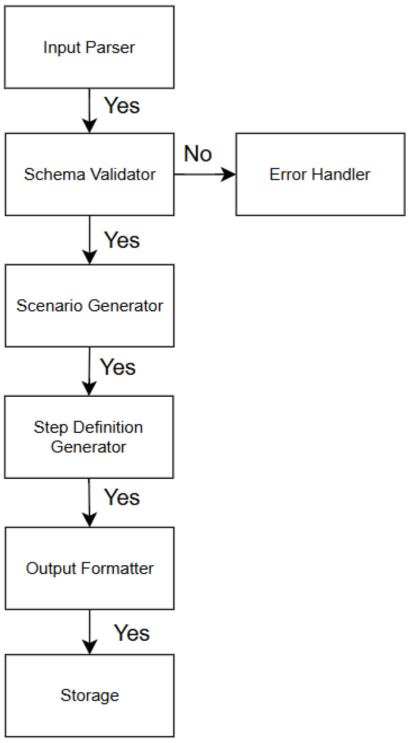
6. **Output Delivery**: The generated files are available for download or integration into a CI/CD pipeline.

3. Flow Diagram

Here's a flow diagram to represent the workflow:



.



4. Conclusion

The API Scenario Generation system automates the creation of Gherkin scenarios and step definitions based on API details. With components for schema validation, scenario generation, and file storage, the system simplifies the testing process for API-based applications. The addition of schema validation ensures that inputs are correct, while the organized file structure allows for easy management of generated assets.

5. Future Enhancement

The future enhancements will expand support for additional API testing frameworks and languages while enabling schema validation code generation with user-selected assertion libraries. The system will ensure reliable file generation, producing self-sufficient, executable code without manual modifications. Users will have the flexibility to customize folder structures for organizing generated assets. These improvements aim to streamline testing workflows and enhance adaptability across diverse development environments.

5. Project Requirements to run

1. Install python
   https://www.python.org/downloads/
2. Verify it is installed or not. Check in command prompt (CMD)
   - Python --version
   - pip --version
3. Install required Dependencies (Run it as **pip install -r requirements.txt)**
   - flask
   - boto3
   - dotenv
   - Zipfile
   - requests
4. To run the project, give below command

```
flask --app app.py run --debug --host=127.0.0.1 --port=5000
```