**Design & Approach Document: Automated Unit Test Generation using AI**

# 1. Introduction:

The purpose of this application is to automate the generation of unit tests for code files, thereby improving code coverage and the overall quality of software projects. The application takes a code file as input and produces a set of unit tests that exercise various parts of the code.

# 2. Objectives:

Automated Test Generation: Develop a system that can analyze code files and generate relevant unit tests automatically.

Code Coverage Improvement: Increase the code coverage by targeting different components and functionalities within the code.

Compatibility: Support multiple programming languages.

Usecase

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2 | Unit Test Case Generation | Create an application that takes a code file written in C# or Java or Python | Code snippet or file of C#, Java, Typescript | Should have design & approach of the solution The application should have a UI interface to upload code file or code snippet. Be able to generate unit test cases for a given code snippet or the code file Have test cases created for validation | Generate unit test cases for the code supplied with required UI interface in React or Angular  Design & approach document Test Cases document Test Report Code repo branch Demo Recording |

# 3. Architecture:

## 3.1 Frontend:

The frontend will be CLI – Command line interface allowing users to:

* Upload code files.
* Select the unit test framework.
* View and download the generated unit tests.

## 3.2 Backend:

The backend will consist of the following components:

File Parser: Responsible for parsing the uploaded code file to understand its structure and dependencies.

Code Analyzer: Examines the code to identify testable components and generate relevant test cases.

AI Test Generator: Produces unit test code based on the analysis and identified testable components using openAI APIs

Compatibility Modules: Language-specific modules ensuring compatibility with different programming languages – Javascript, C# and Java.

Storage: A secure storage system for storing both input and output files temporarily.

# 4. Workflow:

User Uploads Code File:

Users will specify the full path of their code files through the CLI.

File Parsing:

The backend parses the uploaded file to understand its structure, dependencies, and programming language.

Code Analysis:

The code analyzer examines the parsed file to identify testable components and potential test scenarios.

Test Generation:

User specifies the Unit Test framework in CLI.

Based on the analysis, the test generator creates unit test code for the identified components.

Compatibility Checks:

Language-specific modules ensure that the generated tests are compatible with the selected programming language.

Test Storage and Retrieval:

The generated tests are temporarily stored securely and made available for download.

# 6. Approach

* Clone or download the node project - **test-case-generator**
* **npm i** to install packages
* To run - **npm run writeTest**
* Prompt asks for File path and asks to choose the test framework
* Prompt spinner loads, displays Unit tests in the terminal
* Unit test file for relevant programming language also generated

# 5. Technologies:

Frontend: CLI – command line interface

Backend: Node.js

Test Generation: openAI model **gpt-3.5-turbo and openai.chat.completions**

# 8. Conclusion:

This automated unit test generation application aims to streamline the testing process for developers, fostering code quality and maintainability. This engine ensures efficient and effective test generation for a variety of codebases.