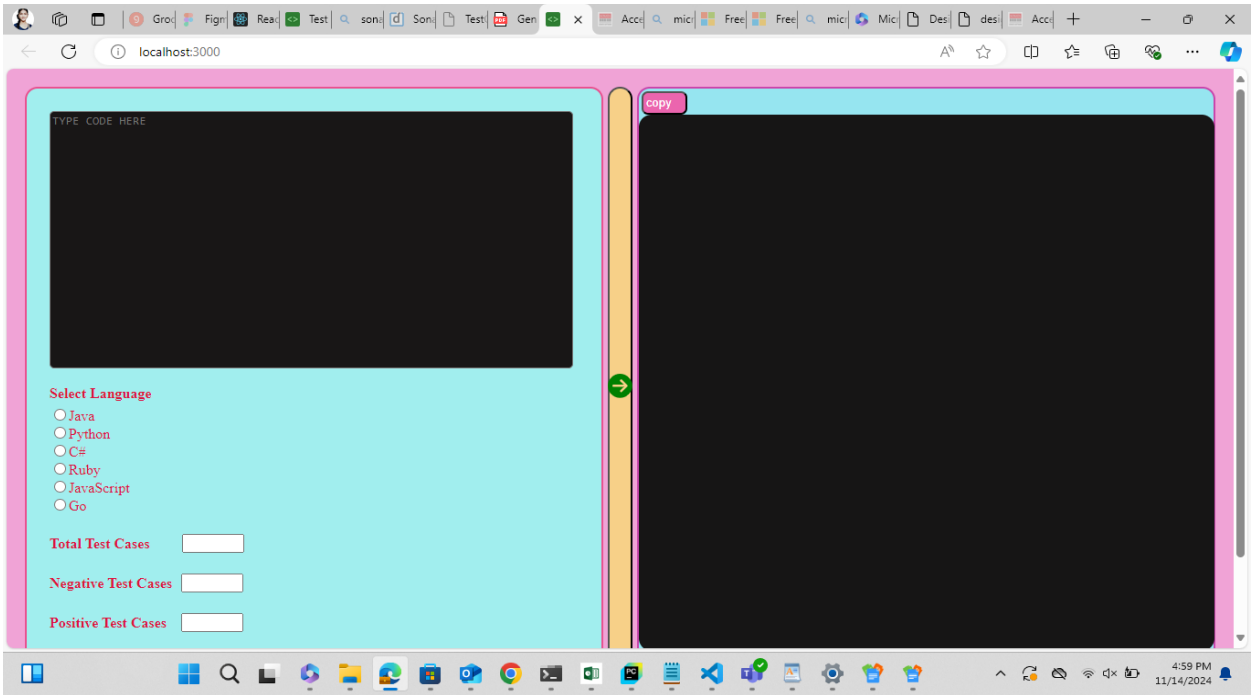
DESIGN AND APPROACH DOCUMENT

1.Introduction

This document outlines the design and approach used for the Unit Test Case Generator application, which automates the creation of unit test cases based on user-provided code. The application accepts the code, the programming language, and test case inputs (positive and negative) and generates appropriate unit test cases.



2. System Overview

The system is divided into two main components

Frontend: A React-based application that allows users to input the code, language, and test case information.

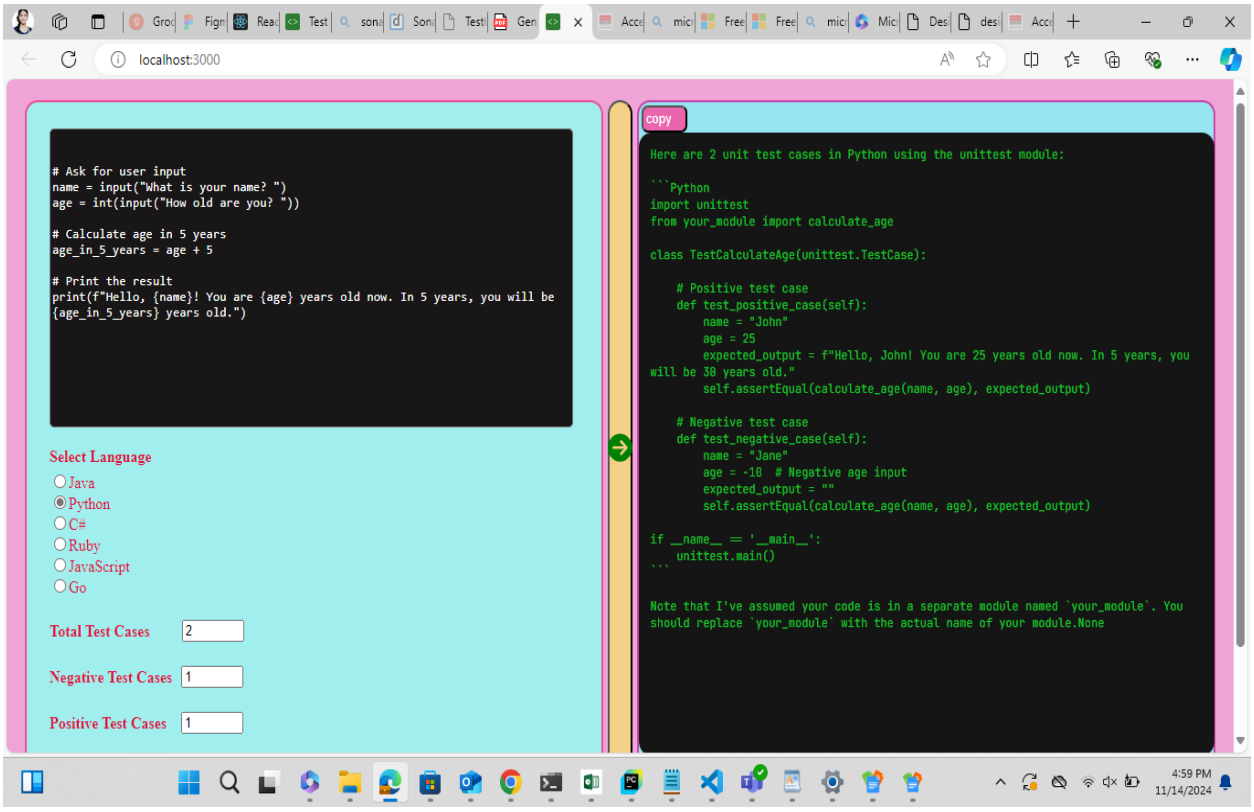
Backend: A Flask server running with Groq's model to process the user's code and generate appropriate unit tests.

3. Architecture Overview

The architecture is a simple client-server model:

Frontend (React.js): Responsible for accepting user input (code, language,

total test cases, positive test cases, and negative test cases) and displaying the output.
It sends a request to the backend to generate unit test cases and displays the generated result.
Backend (Flask + Groq): A Flask application that processes the user's input. It uses Groq s API to communicate with a model (Llama3-8b-8192) for generating unit test cases.
The model takes the user inputs and generates unit test cases based on the programming language and input code.



4. Data Flow

Input Phase: The user inputs code, selects a programming language, and specifies the number of positive and negative test cases.
Validation: The input is validated on the frontend to ensure the total test cases are correct, and there are no negative values for test cases.
Backend Communication: The frontend sends the validated input data to the backend (Flask server).
Model Interaction: The backend uses Groq s API to communicate with the language model (Llama3-8b-8192).

Output Phase: The model returns generated unit test cases, which are displayed to the user on the frontend.

5. Technologies Used

Frontend: React.js, JavaScript, HTML, CSS
Backend: Flask, Python, Groq API, Llama3-8b-8192 Model
API: HTTP/RESTful API communication between frontend and backend
UI/UX: Simple and interactive form-based design with input fields and validation.

6. Key Features

Dynamic Language Selection: Users can choose from multiple languages (Java, Python, C#, Ruby, JavaScript, Go).
Validation: Validates input data to ensure test cases are correctly entered.
Automated Test Case Generation: Generates positive and negative test cases based on the provided code and inputs.
Clipboard Copying: Allows users to copy the generated code directly to the clipboard for easy integration into their projects.

7. Limitations and Future Enhancements

Limitations:
Currently supports a predefined set of languages.
Limited error handling for unexpected input cases.
Future Enhancements:
Support for additional languages.
More robust error handling and edge case management.
Extend model capabilities to support more complex code analysis and test generation.