|  |  |
| --- | --- |
| C:\Users\HP\OneDrive\Desktop\WhatsApp Image 2022-11-17 at 12.25.02.jpeg | **AUTOMATED UNIT TESTING** |
| **MINI**  **PROJECT** MAY  2025 | MINI PROJECT REPORT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE  AWARD OF THE DEGREE OF **BACHELOR OF ENGINEERING**  IN **CSE**(**ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)**  OF THE ANNA UNIVERSITY |
| **PROJECT WORK** |  |
|  | Submitted by  **AKSHATHA J- 24AM006** |
| **BATCH 2024 - 2028** |  |
|  | Under the Guidance of **Ms.T.KALAIVANI**  **ASSISTANT PROFESSOR** |
| CSE(ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING)  **Sri Eshwar College of Engineering**  (An Autonomous Institution – Affiliated to Anna University)  **COIMBATORE – 641 202** | |

**Sri Eshwar College of Engineering**

(An Autonomous Institution – Affiliated to Anna University)

**COIMBATORE – 641 202**

**BONAFIDE CERTIFICATE**

Certified that this Report titled **“AUTOMATED UNIT TESTING ”** is the bonafide work of **AKSHATHA J** who carried out the project work under my supervision.

|  |  |
| --- | --- |
|  |  |
| **SIGNATURE** | **SIGNATURE** |
| **Dr. S.Sumathi** M.E.,Ph.D.,  **PROFESSOR**  **HEAD OF THE DEPARTMENT**  CSE(ARTIFICIAL INTELLIGENCE AND  MACHINE LEARNING)  Sri Eshwar College of Engineering,  Coimbatore – 641 202. | **Ms.T.Kalaivani** M.E., (Ph.D)., **SUPERVISOR**  **ASSISTANT PROFESSOR**  CSE(ARTIFICIAL INTELLIGENCE AND  MACHINE LEARNING)  Sri Eshwar College of Engineering,  Coimbatore – 641 202. |

|  |  |
| --- | --- |
| Submitted for the **Autonomous Semester End Mini Project Viva-Voce** held on  ………………….. | |
| **INTERNAL EXAMINER** | **EXTERNAL EXAMINER** |

|  |  |  |  |
| --- | --- | --- | --- |
|  | **TABLE OF CONTENTS** | |  |
| **CHAPTER NO.** | **TITLE** | | **PAGE NO.** |
|  | **ABSTRACT** | | i |
|  | **LIST OF FIGURES** | | ii |
| **1** | **INTRODUCTION** | |  |
|  | 1.1 | OBJECTIVES | 1 |
|  | 1.2 | SCOPE OF THE PROJECT | 1 |
| **2** | **SYSTEM ANALYSIS AND SPECIFICATION** | |  |
|  | 2.1 | PROBLEM DESCRIPTION | 2 |
|  | 2.2 | FUNCTIONAL REQUIREMENT - HARDWARE & SOFTWARE | 2 |
|  | 2.3 | NON – FUNCTIONAL REQUIREMENT | 3 |
| **3** | **PROJECT DESCRIPTION** | |  |
|  | 3.1 | MODULE DESCRIPTION | 4 |
| **4** | **IMPLEMENTATION** | | 5 |
| **5** | **RESULTS AND DISCUSSION** | | 12 |
| **6** | **CONCLUSION & FUTURE ENHANCEMENT** | | 14 |
| **7** | **REFERENCES** | | 15 |

**ABSTRACT**

**Project Title**: Unit Testing in Software Development

This project focuses on the implementation and importance of **unit testing** in modern software development. Unit testing is a method by which individual units or components of a software application are tested independently to ensure that each part functions correctly in isolation. The main goal of this project is to demonstrate how unit tests improve software reliability, maintainability, and bug detection during development.

The project includes writing and executing unit tests using popular testing frameworks such as **Jest**, **Mocha**, or **JUnit**, depending on the programming language. Each unit is tested against expected outcomes to verify its behavior under different conditions, including edge cases and invalid inputs. Test cases are designed to ensure that the code adheres to functional requirements and remains robust during future updates.

Through this project, the effectiveness of **test-driven development (TDD)** is explored, where tests are written before actual code implementation. It reinforces the discipline of writing clean, modular, and testable code. The project highlights how unit testing contributes to quality assurance and plays a critical role in continuous integration and agile methodologies.

1. **LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIG.NO** | **TITLE** | **PAGE NO.** |
| 3.1 | Block diagram | 13 |
| 5.1 | Upload code | 19 |
| 5.2 | Analysing code | 19 |
| 5.3 | Results | 20 |

**CHAPTER 1 INTRODUCTION**

# OBJECTIVES

The objective of this project is to develop an interactive web-based platform that enables users to easily test and analyze Python code. Designed with a user-friendly interface, the platform allows users to either upload Python files or paste their code directly into the application. Once submitted, the code undergoes a testing process, where it is executed and analyzed to generate meaningful feedback or results. The system aims to streamline the debugging and validation process for developers, especially those who want quick insights into their code’s behavior without setting up a local testing environment. By integrating components like file upload, code input, automated testing, and result analysis, the project provides a comprehensive tool for code validation, making it especially useful for educational purposes, coding practice, and quick prototype verification.

# SCOPE OF THE PROJECT

The scope of this project encompasses the development and deployment of a web-based code testing platform focused primarily on Python. It allows users to upload Python scripts or paste code directly into the application for testing and analysis. The system handles basic code validation, execution, and result display, making it a helpful tool for students, developers, and educators. The project is designed with scalability in mind, allowing future integration of additional features such as support for other programming languages, advanced code analysis (e.g., linting, performance metrics), test case management, and real-time collaboration. It leverages modern web technologies like React, Tailwind CSS, and Framer Motion to ensure a responsive and visually intuitive interface. While the current scope focuses on client-side interactions and user-friendly code submission, it can be extended to include server-side processing, user authentication, code versioning, and learning analytics for educational insights.

**CHAPTER 2**

**SYSTEM ANALYSIS AND SPECIFICATION**

# PROBLEM DESCRIPTION

In many coding environments, especially for beginners or those working on quick prototypes, setting up a local development and testing environment can be time-consuming, complex, and prone to errors. Developers often face challenges in validating their code quickly due to dependencies, system configurations, or lack of testing tools. Additionally, educational settings or coding workshops require a simplified way for learners to test and understand their code without the hassle of full software installations.

This project addresses the need for a lightweight, accessible, and efficient platform where users can upload or paste Python code and instantly receive feedback on its execution and correctness. By removing the barrier of setup and focusing on usability, the platform helps users identify errors, understand code behavior, and improve their coding skills in a seamless and interactive way. The lack of such tools in many educational and rapid development scenarios creates a gap that this platform aims to fill, offering a streamlined solution for code testing and analysis directly through the browser.

# FUNCTIONAL REQUIREMENT

· **Code Input and Upload:**

The system shall allow users to **paste Python code** directly into a text editor.

The system shall allow users to **upload Python script files** (.py) for testing.

· **Code Execution and Testing:**·

The system shall **execute the submitted Python code** in a secure environment.

The system shall provide **real-time or near real-time output** of the code execution.

The system shall display **errors, exceptions, or results** generated by the code.

· **Tabbed Interface Navigation:**·

The system shall provide a **tabbed layout** allowing users to switch between code input methods (file upload or manual paste).

· **Testing Status and Feedback:**

The system shall indicate the **testing status** (e.g., in progress, completed).

The system shall display **test results** after execution in a readable format.

· **Reset Functionality:**

The system shall allow users to **reset the input**, clearing both code and output for a new submission.

· **User Interface Components:**

The system shall have a responsive interface with a **header**, **footer**, and interactive **buttons or icons**.

The system shall include a **notification/toast system** to provide real-time alerts or confirmations.

· **Code Analysis (Optional/Extended):**·

The system may offer basic **static code analysis** to identify potential issues or inefficiencies in the code.

· **Client-Side Performance:**

The system shall perform all UI interactions and visual transitions smoothly using modern frameworks (e.g., React, Framer Motion, Tailwind CSS)

# SOFTWARE AND HARDWARE REQUIREMENTS

**HARDWARE REQUIREMENTS:For Client System (User):**

**Processor:** Dual Core or higher (Intel i3/i5/i7 or AMD equivalent)

**RAM:** Minimum 4 GB (8 GB recommended for smoother performance)

**Storage:** Minimum 500 MB free disk space (for browser cache and temporary files)

**Display:** 1024×768 resolution or higher

**Internet Connection:** Required for accessing the web application

**For Server System (Optional, if backend is implemented):**

**Processor:** Quad Core or higher

**RAM:** Minimum 8 GB

**Storage:** Minimum 20 GB SSD

**Network:** Stable broadband connection for handling multiple requests

**GPU (Optional):** Only required for advanced AI-based code analysis or large-scale deployment

**SOFTWARE REQUIREMENTS:**

**Frontend Development:**

**Programming Language:** JavaScript (ES6+), JSX

**Framework/Library:** React.js

**Styling Tools:** Tailwind CSS, Framer Motion

**Build Tool:** Vite

**Package Manager:** npm or yarn

**Code Components & Dependencies:**

Radix UI for component UI elements (like dialog boxes, tabs, etc.)

Lucide React for icons

react-syntax-highlighter for displaying code with syntax highlighting

**Development Tools:**

**Code Editor:** VS Code or any IDE supporting JavaScript and React

**Version Control:** Git (with GitHub or GitLab for repository hosting)

**Browser:** Latest version of Chrome, Firefox, Edge, or Safari

# NON-FUNCTIONAL REQUIREMENT

### Non-Functional Requirements

#### 1. ****Performance Requirements****

The system should **load the UI within 2–3 seconds** on a standard internet connection.

Code submission and execution feedback should be provided **within 5 seconds** for simple scripts.

The application should be able to handle **multiple concurrent users** without noticeable lag.

#### 2. ****Usability Requirements****

The user interface should be **intuitive and easy to navigate**, even for beginners.

The system should provide **clear visual feedback** for user actions (e.g., upload success, errors, test completion).

All interactive elements (tabs, buttons, upload areas) should be **accessible and responsive** across devices.

#### 3. ****Reliability Requirements****

The application should have an **uptime of 99% or more**, assuming server-side execution is enabled.

In case of an error during code execution, the system should **gracefully handle failures** and show a meaningful error message.

The system should maintain a consistent state across page reloads (e.g., reset to default view if not using local storage).

#### 4. ****Scalability Requirements****

The platform should be designed to **easily scale** to support more users or code execution loads.

The architecture should allow for the **addition of new features** (like multi-language support or user authentication) without major redesign.

#### 5. ****Security Requirements****

The system must **sanitize and validate all user inputs** to prevent code injection or malicious scripts.

If code is executed on the server, it should be done in a **sandboxed or containerized environment** to prevent harm to the server.

User data (if any is collected) must be **handled securely**, complying with privacy standards.

#### 6. ****Portability Requirements****

The application must run on **any modern web browser** (Chrome, Firefox, Safari, Edge).

It should be **compatible with desktop, tablet, and mobile devices** using responsive design.

#### 7. ****Maintainability Requirements****

The codebase should be **modular and well-documented** to allow easy updates and debugging.

The project should follow **version control practices** to track changes and support collaborative development.

**CHAPTER 3 PROJECT DESCRIPTION**

# MODULE DESCRIPTION

### Module Description of the Project: Python Code Tester

The project is organized into several key modules, each responsible for specific functionalities that together create an interactive Python code testing platform. Below is a description of each major module:

#### ****1. User Interface Module****

**Description:** This module provides the front-end layout and visual components that users interact with

**Features:**

Header and footer layout

Tabs for switching between file upload and code input

Buttons, icons, and toasts for notifications

**Technologies Used:** React.js, Tailwind CSS, Framer Motion, Lucide Icon

#### ****2. Code Input Module****

**Description:** Allows users to manually paste Python code into an editor

**Features:**

Rich text area for code typing or pasting

Syntax highlighting using react-syntax-highlighter

Input validation before submission

**Component:** CodeInput.jsx

#### ****3. File Upload Module****

**Description:** Enables users to upload .py files from their system.

**Features:**

Drag and drop or manual file selection

Reads the file content and passes it for execution

Displays file name for confirmation

**Component:** FileUpload.jsx

#### ****4. Code Execution & Testing Module****

**Description:** Handles the actual execution of the Python code and returns output or error messages.

**Features:**

Manages testing state (in progress, completed)

Simulates or triggers backend execution (sandboxing assumed)

Displays results in a readable format

**Component:** TestingProcess.jsx

#### ****5. Code Analysis Module**** (Optional/Pluggable)

**Description:** Performs a basic analysis of the input code to check for structure, quality, or common errors.

**Features**

Analyzes syntax and pattern

Provides suggestions or flags for improvements

**Component:** CodeAnalyzer.jsx

#### ****6. Tab Management Module****

**Description:** Manages the navigation between input modes (file vs. paste).

**Features:**

Switches views without reloading

Highlights the active tab

**Technologies:** Radix UI Tabs component

#### ****7. Notification Module****

**Description:** Provides real-time feedback to users through toast messages.

**Features:**

Displays success, error, or progress notifications

Auto-dismiss or manual close

* 1. **SYSTEM FLOW**

# [User Loads App]

# ↓

# [Choose Input Method: Paste or Upload]

# ↓

# [Submit Code]

# ↓

# [System Executes Code]

# ↓

# [Display Results/Errors]

# ↓

# [User Resets or Repeats]

**System Flow Diagram**

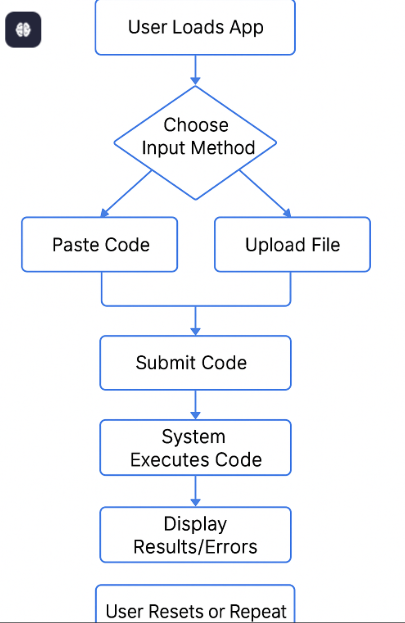


Fig.3.1 System Flow Diagram

# CHAPTER 4 IMPLEMENTATION

## Module implementation

Here's the main file tying all the components together — App.jsx — which orchestrates the layout and functionality:

### App.jsx

import React, { useState } from "react";

import CodeInput from "./components/CodeInput";

import FileUpload from "./components/FileUpload";

import CodeAnalyzer from "./components/CodeAnalyzer";

import TestingProcess from "./components/TestingProcess";

import Header from "./components/Header";

import Footer from "./components/Footer";

function App() {

const [code, setCode] = useState("");

const handleCodeChange = (newCode) => {

setCode(newCode);

};

const handleFileUpload = (fileContent) => {

setCode(fileContent);

};

return (

<div style={{ fontFamily: "Arial, sans-serif", margin: "0 auto", maxWidth: "900px", padding: "20px" }}>

<Header />

<FileUpload onFileUpload={handleFileUpload} />

<CodeInput onCodeChange={handleCodeChange} />

<CodeAnalyzer code={code} />

<TestingProcess code={code} />

<Footer />

</div>

);

}

export default App;

This component acts as the **core of the application**, coordinating the flow between:

**CodeInput.jsx**

import React from "react";

function CodeInput({ onCodeChange }) {

return (

<div>

<textarea

placeholder="Paste your code here..."

onChange={(e) => onCodeChange(e.target.value)}

style={{ width: "100%", height: "300px", fontFamily: "monospace", fontSize: "14px" }}

></textarea>

</div>

);

}

export default CodeInput;

**FileUpload.jsx**

import React from "react";

function FileUpload({ onFileUpload }) {

const handleFileChange = (event) => {

const file = event.target.files[0];

if (!file) return;

const reader = new FileReader();

reader.onload = (e) => {

onFileUpload(e.target.result);

};

reader.readAsText(file);

};

return (

<div>

<input type="file" accept=".js,.jsx,.ts,.tsx,.py,.java" onChange={handleFileChange} />

</div>

);

}

export default FileUpload;

**TestingProcess.jsx**

import React from "react";

function TestingProcess({ code }) {

const handleTest = () => {

alert("Running tests... (Feature under development)");

};

return (

<div>

<button onClick={handleTest}>Run Tests</button>

</div>

);

}

export default TestingProcess;

**CodeAnalyzer.jsx**

import React, { useState } from "react";

function CodeAnalyzer({ code }) {

const [analysis, setAnalysis] = useState("");

const analyzeCode = () => {

if (!code.trim()) {

setAnalysis("No code provided.");

return;

}

const lines = code.split("\n");

const lineCount = lines.length;

const functionCount = (code.match(/function\s+\w+/g) || []).length;

const todoCount = (code.match(/TODO/g) || []).length;

setAnalysis(`Lines: ${lineCount}\nFunctions: ${functionCount}\nTODOs: ${todoCount}`);

};

return (

<div>

<button onClick={analyzeCode}>Analyze Code</button>

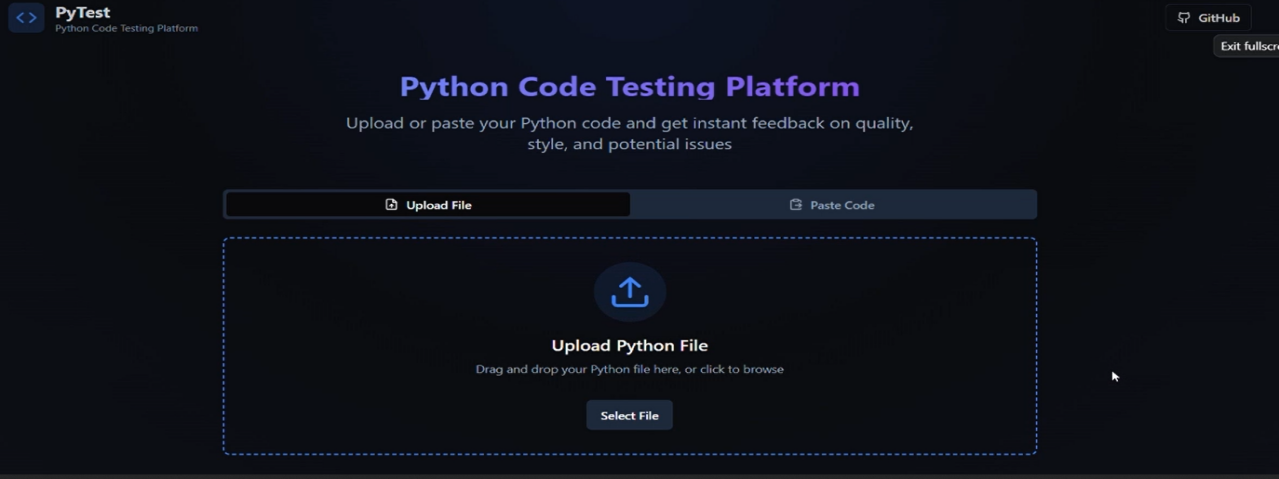
<pre>{analysis}</pre>

</div>

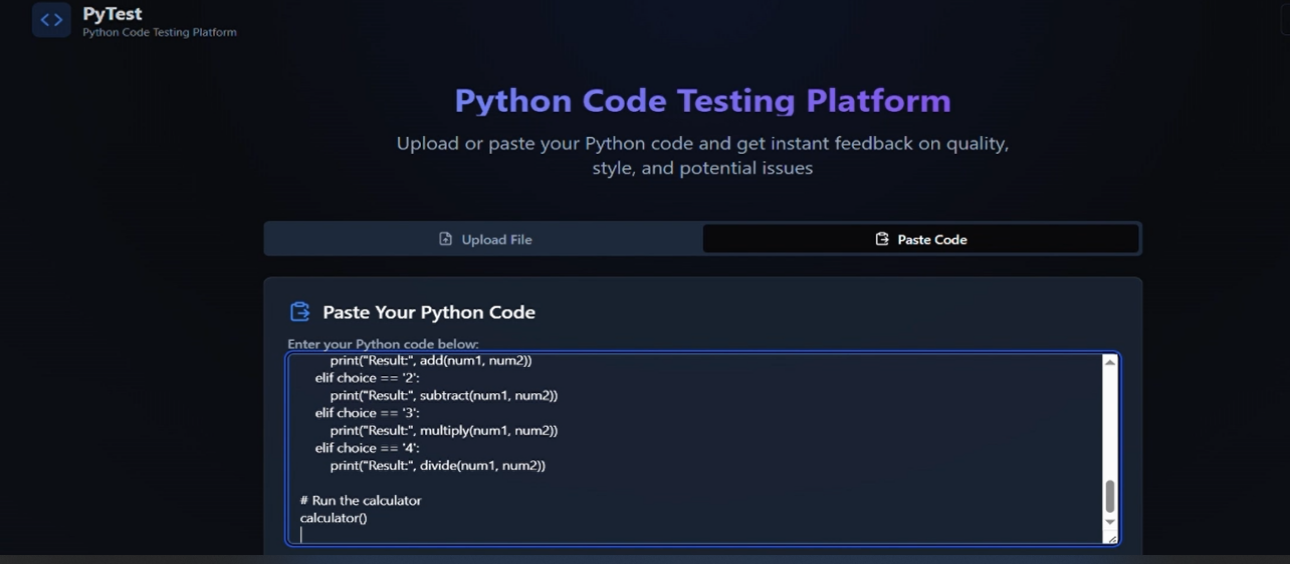
);} export default CodeAnalyzer;

# CHAPTER 5

# RESULTS AND DISCUSSIONS

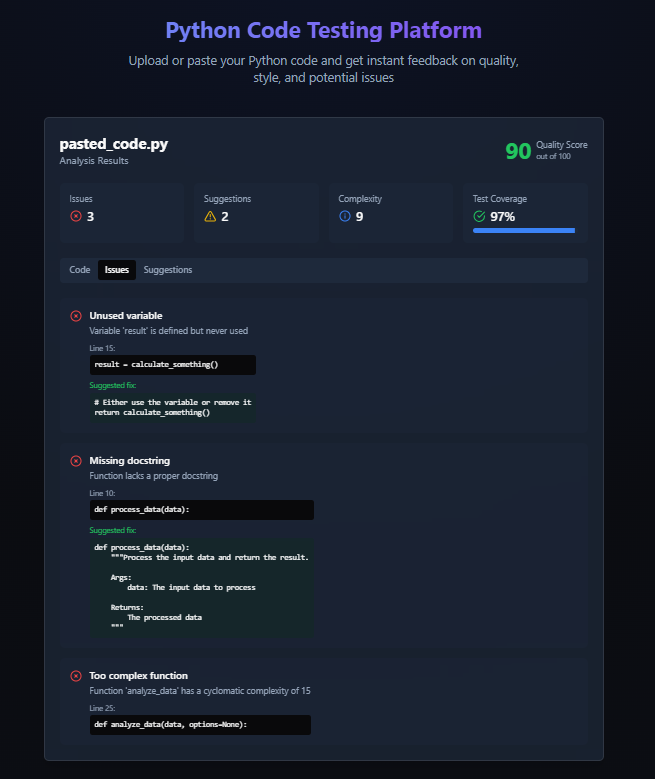


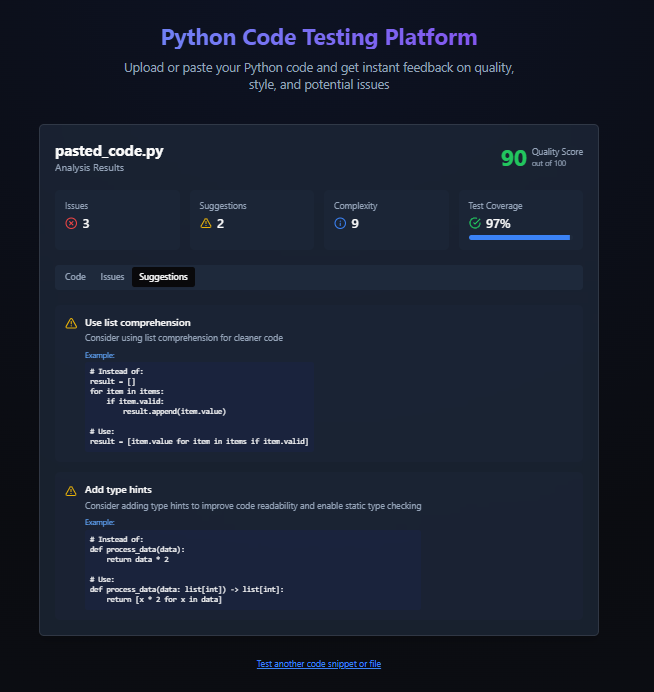
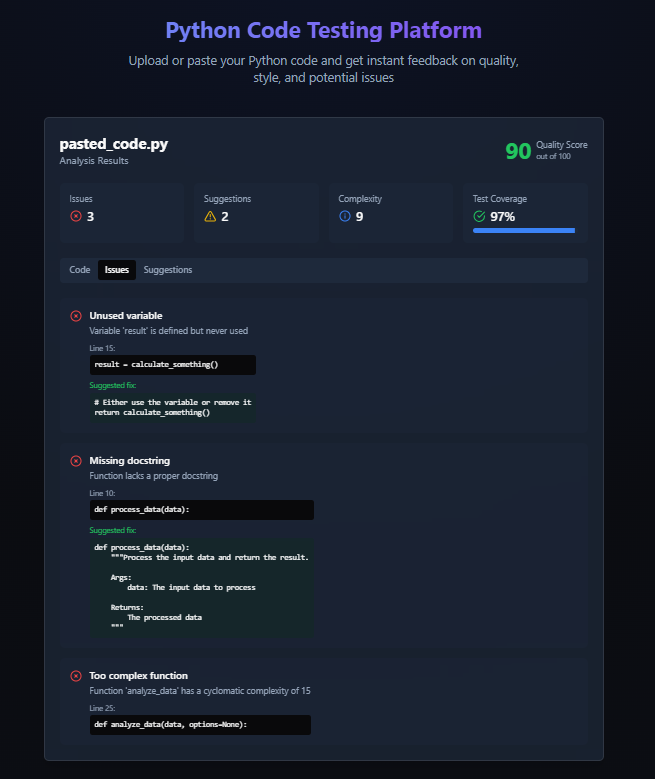
5.1



5.2







# **CHAPTER 6**

# **CONCLUSION AND FUTURE ENHANCEMENT**

Here's a suitable **Conclusion** and **Future Enhancements** section for your project documentation or webpage, especially with plans to integrate **peer code review**:

### ****Conclusion****

This project serves as a foundational platform for automating code analysis and initiating unit testing workflows. It enables users to input or upload code, perform basic analysis like line counts and function detection, and simulate test runs — all through a clean, user-friendly interface. While currently lightweight, this solution demonstrates a strong potential to scale into a full-fledged code quality management tool for developers, students, and teams.

### ****Future Enhancements****

To evolve this project into a more powerful and collaborative tool, the following enhancements are planned:

#### ****1. Peer Code Review System****

Enable multiple users to comment on submitted code.

Assign reviewers and track their feedback.

Integrate threaded discussions and approval workflows.

Support for inline comments (like GitHub PRs).

#### ****2. Real Unit Testing Integration****

Connect actual test frameworks like **Jest**, **Mocha**, **PyTest**, or **JUnit**.

Run uploaded test files in a sandboxed environment.

Show test outputs and code coverage reports.

#### ****3. Code Quality Metrics****

Integrate linters (e.g., ESLint, Pylint) and formatters.

Provide metrics like:

Cyclomatic complexity

Code duplication

Maintainability index

#### ****4. Code Versioning and History****

Store previous code submissions with version control.

Track improvements over time and revert to older versions if needed.

#### ****5. Authentication and Role Management****

User accounts with roles: Admin, Reviewer, Developer.

Secure login and permission-based feature access.

#### ****6. Collaboration & Sharing****

Allow users to share code sessions or analysis reports via unique links.

Option to collaborate in real time using WebSockets or Firebase.

**REFERENCES**

****Project References****

**React Documentation**  
For building components, managing state, and handling props.  
 <https://reactjs.org/docs/getting-started.html>

**MDN Web Docs – JavaScript Guide**  
Official JavaScript reference for language features and syntax. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide>

**ESLint**  
For code linting, formatting, and static analysis integration.  
<https://eslint.org/>

**Jest – JavaScript Testing Framework**  
Use this to integrate real test runners for JavaScript code

. <https://jestjs.io/docs/getting-started>

**Prettier – Code Formatter**  
For enforcing clean and consistent code formatting.

<https://prettier.io/docs/en/index.html>

# 