**A**

**MINI PROJECT REPORT**

**ON**

**Plagiarism Analyzer**

**Submitted in partial fulfillment of the requirements**

**For the award of Degree of**

**BACHELOR OF ENGINEERING**

**IN**

**CSE(AIML)**

**Submitted By**

**BALLEM RISHITHA 245320748070**

**MEDISHETTY AMULYA 245320748105**

**THOTA NIKITHA 245321748119**

**Under the guidance**

**Of**

**Dr. B. SUVARNA MUKHI**

**ASSITANT PROFESSOR**

****

**Department of CSE(AIML)**

**NEIL GOGTE INSTITUTE OF TECHNOLOGY**

Kachavanisingaram Village, Hyderabad, Telangana 500058.

**January 2023**

**NEIL GOGTE INSTITUTE OF TECHNOLOGY**

A Unit of Keshav Memorial Technical Education (KMTES)

Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

****

**2012-2016**

**CERTIFICATE**

*This is to certify that the project work (PW533CSM) entitled* “**PLAGIARISM ANALYZER”** *is a bonafide work carried out by* **BALLEM RISHITHA(245321748070), MEDISHETTY AMULYA(245321748105),THOTA NIKITHA(245321748119),** of III year V semester**Bachelor of Engineering** *in* **CSE(AIML)** *by Osmania University, Hyderabad during the academic year* **2021-2025** *is a record of bonafide work carried out by them*. *The results embodied in this report have not been submitted to any other University or Institution for the award of any degree*

**Internal Guide Head of Department**

Dr. B. SUVARNA MUKHI Dr. PREM CHANDER

Associate Professor

**External**

****

**NEIL GOGTE INSTITUTE OF TECHNOLOGY**

A Unit of Keshav Memorial Technical Education (KMTES)

Approved by AICTE, New Delhi & Affiliated to Osmania University, Hyderabad

# **DECLARATION**

We hereby declare that the Mini Project Report entitled, “**PLAGIARISM ANALYZER**” submitted for the B.E degree is entirely our work and all ideas and references have been duly acknowledged. It does not contain any work for the award of any other degree.

**Date:**

**B.RISHITHA (245321748070)**

**M.AMULYA (245321748105)**

**T.NIKITHA (245321748119)**

**ACKNOWLEDGEMENT**

We are happy to express our deep sense of gratitude to the principal of the college **Dr. R.Shyam Sunder, Professor**, Neil Gogte Institute of Technology, for having provided us with adequate facilities to pursue our project.

We would like to thank**,** **Dr.** **PREM CHANDER,** CSE(AIML), Neil Gogte Institute of Technology, for having provided the freedom to use all the facilities available in the department, especially the laboratories and the library.

We would also like to thank my internal guide **Dr. B.SUVARNA MUKHI, Assistant Professor** for her Technical guidance & constant encouragement.

We sincerely thank our seniors and all the teaching and non-teaching staff of the Department of Computer Science & Engineering and Information Technology for their timely suggestions, healthy criticism and motivation during the course of this work.

Finally, we express our immense gratitude with pleasure to the other individuals who have either directly or indirectly contributed to our need at the right time for the development and success of this work.

**I**

**ABSTRACT**

The Project Plagiarism Analyzer is a software tool designed to detect and analyze instances of plagiarism within programming projects and assignments. With the increasing prevalence of online resources and collaboration platforms, the issue of plagiarism in programming assignments has become a significant concern for educators and academic institutions.

The aim of the Project Plagiarism Analyzer is to provide educators and instructors with a reliable and efficient means of detecting similarities between programming projects submitted by students. The tool employs advanced algorithms and techniques to compare code submissions, identifying similarities in code structure, logic, and comments.

**II**

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **CHAPTER** | **TITLE** | **PAGE NO.** |
|  | **Acknowledgement** | **I** |
|  | **Abstract** | **II** |
|  | **List of Figures** | **V** |
|  | **List of Tables** | **V** |
|  | **Introduction** |  |
|  | 1.1 Problem Statement | **1-3** |
|  | 1.2 motivation | **3** |
|  | 1.3 scope | **4** |
|  | 1.4 outline | **4** |
| **2.** | **LITERATURE SURVEY** |  |
|  | 2.1 EXISTING SYSTEM | **5-7** |
|  | 2.2 PROPOSED SYSTEM | **8** |
| **3.** | **Software Requirements Specification** |  |
|  | 3.1 Overall Description | **9-14** |
|  | 3.2 Operating Environment | **14** |
|  | 3.3 Functional Requirements | **14-16** |
|  | 3.4 NON-FUNCTIONAL Requirements | **16-20** |
| **4.** | **DESIGN DIAGRAMS** |  |
|  | 4.1 UML DIAGRAMS | **21** |
|  | 4.1.1 Use-case Diagram | **21-23** |
|  | 4.1.2 Class Diagram | **24** |
|  | 4.1.3 Sequence Diagram | **25** |

**III**

|  |  |  |
| --- | --- | --- |
| **5.** | **Implementation** |  |
|  | 5.1Sample code | **26-34** |
| **6.** | **TESTING** |  |
|  | 6.1 TEST CASES | **35** |
| **7.** | **SCREENSHOTS** | **36-42** |
| **8.** | **CONCLUSION AND Future scope** | **43** |
|  | **BIBLIOGRAPHY** | **44** |
|  | **Appendix a:** tools and technology | **45-60** |
|  |  |  |

**IV**

**List of Figures**

|  |  |  |
| --- | --- | --- |
| **Figure No.** | **Name of Figure** | **Page No.** |
| 1. | Use case Diagram | 21-23 |
| 2. | Class Diagram | 24 |
| 3. | Sequence Diagram | 25 |

**List of Tables**

|  |  |  |
| --- | --- | --- |
| **Table No.** | **Name of Table** | **Page No.** |
| 1. | Testcase #1 | 35 |

2. Testcase#2 35

**V**

**CHAPTER – 1**

**INTRODUCTION**

* 1. **PROBLEM STATEMENT**

The Problem Statement for the Plagiarism Analyzer project revolves around the increasing prevalence of academic dishonesty in programming assignments. As students have easy access to online resources and collaboration platforms, detecting and addressing instances of plagiarism in programming projects has become a significant challenge for educators and academic institutions. The lack of efficient tools to accurately identify similarities between code submissions hampers the efforts to maintain academic integrity and fairness in evaluation processes.

* 1. **MOTIVATION**

The motivation behind developing the Plagiarism Analyzer stems from the necessity to uphold academic standards and promote integrity in programming education. By addressing the issue of plagiarism effectively, educators can ensure that students demonstrate their individual understanding and mastery of programming concepts. Additionally, promoting originality and honesty in code submissions fosters a culture of ethical conduct and excellence in academic and professional settings.

**1.3 Scope**

The scope of the Plagiarism Analyzer project encompasses the development of a robust software tool capable of analyzing code submissions and detecting similarities indicative of plagiarism or unauthorized collaboration. The analyzer will consider various aspects of code structure, logic, comments, and variable names to identify potential instances of plagiarism. It will be designed to handle code written in different programming languages and integrate seamlessly with existing learning management systems (LMS) or educational platforms used by institutions.

* 1. **OUTLINE**

The Plagiarism Analyzer project will follow a structured approach, including the following key components:

* **Code Comparison Algorithms:** Development of sophisticated algorithms to compare code submissions and identify similarities.
* **User Interface Design:** Designing an intuitive and user-friendly interface for educators to upload, analyze, and review code submissions.
* **Plagiarism Detection Mechanisms:** Implementing mechanisms to accurately detect potential instances of plagiarism based on predefined thresholds and criteria.
* **Report Generation:** Generating comprehensive reports highlighting detected similarities and providing detailed insights into suspected cases of plagiarism.
* **Integration with Learning Management Systems:** Ensuring seamless integration with popular learning management systems (LMS) and educational platforms to facilitate efficient management and analysis of code submissions.
* **Customization and Configuration Options:** Providing educators with the ability to customize and configure the analyzer to suit specific requirements and preferences.Top of Form

**CHAPTER 2**

**LITERATURE SURVEY**

**EXISTING SYSTEM:**

Traditional plagiarism detection methods often rely on manual examination or basic text-matching algorithms, which are time-consuming and may not effectively identify subtle instances of plagiarism. Additionally, some existing tools lack the ability to handle various documents formats and may not provide comprehensive reports. The need for an advances and automated plagiarism detection system is evident, considering the evolving nature of content creation and the growing importance of intellectual property protection.

There are different forms of plagiarism

1.Ghostwriting

2.Character-preserving (copy&paste,shake&paste)

3.Syntax-preserving(technical disguise,synonym substitution)

4.Semantics-preserving(translation paraphrase)

5.Idea-preserving(structural plagiarism,conceptual plagiarism)

And there are many detection methods like N-gram comparisons, stylometry, Non-textual feature analysis, Vector space Models, LSA,ESA,Semantic graph analysis, Machine learning.

These detection methods are crucial for educational institutions, publishers, and content creators to maintain academic integrity and ensure originality in written works.

**PROPOSED SYSTEM:**

The plagiarism Analyzer introduces an innovation approach to plagiarism detection, utilizing advanced algorithms and machine learning techniques. The system will support multiple document formats, including text, PDFs, graphs and other common formats. By employing natural language processing and deep learning, the analyzer will not only identify direct matches but also access contextual similarities, making it more adept at recognizing paraphrased or rephrased content.The proposed system will offer a user-friendly interface, providing detailed reports to users and administrators, highlighting potential instances of plagiarism and facilitating informed decision-making.

**CHAPTER 3**

**SOFTWARE REQUIREMENTS**

**3.1 Overall Description:**

The Plagiarism Analyzer project aims to develop a comprehensive software tool that detects and analyzes instances of plagiarism within programming assignments. The analyzer will compare code submissions, identify similarities, and generate detailed reports for educators. It will provide user-friendly interfaces for both users and administrators to facilitate the management and analysis of code submissions. The project will prioritize accuracy, efficiency, and ease of use to promote academic integrity and fairness in programming education.

### **3.2 Operating Environment:**

The Plagiarism Analyzer will be designed to operate in a variety of environments, including:

* Supported Operating Systems: Windows, Linux, macOS
* Web Browsers: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge
* Compatibility: The analyzer should be compatible with modern hardware configurations and web technologies.

### **3.1 Overall Description:**

The Plagiarism Analyzer project aims to develop a comprehensive software tool that detects and analyzes instances of plagiarism within programming assignments. The analyzer will compare code submissions, identify similarities, and generate detailed reports for educators. It will provide user-friendly interfaces for both users and administrators to facilitate the management and analysis of code submissions. The project will prioritize accuracy, efficiency, and ease of use to promote academic integrity and fairness in programming education.

### **3.2 Operating Environment:**

The Plagiarism Analyzer will be designed to operate in a variety of environments, including:

* Supported Operating Systems: Windows, Linux, macOS
* Web Browsers: Google Chrome, Mozilla Firefox, Safari, Microsoft Edge
* Compatibility: The analyzer should be compatible with modern hardware configurations and web technologies.

### **3.3 Functional Requirements:**

#### **User Functionality:**

* Upload Code Submissions: Users should be able to upload programming code submissions for analysis.
* View Analysis Reports: Users should be able to view detailed analysis reports highlighting detected similarities and suspected cases of plagiarism.
* Receive Notifications: Users should receive notifications/alerts regarding analysis results and system updates.

#### **Admin Functionality:**

* Manage Users: Administrators should have the ability to manage user accounts, permissions, and access levels.
* Configure Settings: Administrators should be able to configure analysis settings, thresholds, and criteria.
* Generate System Reports: Administrators should have access to system-wide reports and analytics for monitoring performance and usage.

### **3.4 Non-Functional Requirements:**

#### **3.4.1 Performance Requirements:**

* **Response Time:** The system should provide timely responses to user requests, with minimal latency.
* **Recovery Time:** In the event of a system failure or error, the recovery time should be minimized to ensure minimal disruption to users.
* **Start-Up/Shutdown Time:** The system should have efficient start-up and shutdown procedures to minimize downtime.
* **Capacity:** The system should be capable of handling a large volume of code submissions and concurrent user requests.
* **Utilization of Resources:** The system should optimize resource utilization to ensure efficient performance and scalability.

#### **Safety Requirements:**

* The system should implement measures to prevent data loss, corruption, or unauthorized access.
* It should provide data backup and recovery mechanisms to safeguard against potential data loss or system failures.

#### **Security Requirements:**

* The system should employ robust authentication and authorization mechanisms to control access to sensitive information and functionalities.
* It should encrypt data transmission and storage to protect against unauthorized interception or tampering.
* It should implement measures to detect and mitigate potential security vulnerabilities, such as SQL injection, cross-site scripting (XSS), and other common attacks.

#### **3.4.4 Software Quality Attributes:**

#### **Reliability:**

* The system should be reliable and accurate in detecting instances of plagiarism and generating analysis reports.
* It should minimize false positives/negatives to ensure the integrity of analysis results.

#### **Availability:**

* The system should be highly available, with minimal downtime and maintenance periods.
* It should have redundancy and failover mechanisms to ensure continuous operation in the event of hardware or software failures.

#### **Maintainability:**

* The system should be modular and well-structured to facilitate future updates, enhancements, and maintenance tasks.
* It should have clear documentation and coding standards to aid in understanding and modifying the system.

#### **Usability:**

* The system should have an intuitive and user-friendly interface, with clear navigation and functionality.
* It should provide contextual help and guidance to users for performing tasks and interpreting analysis results.

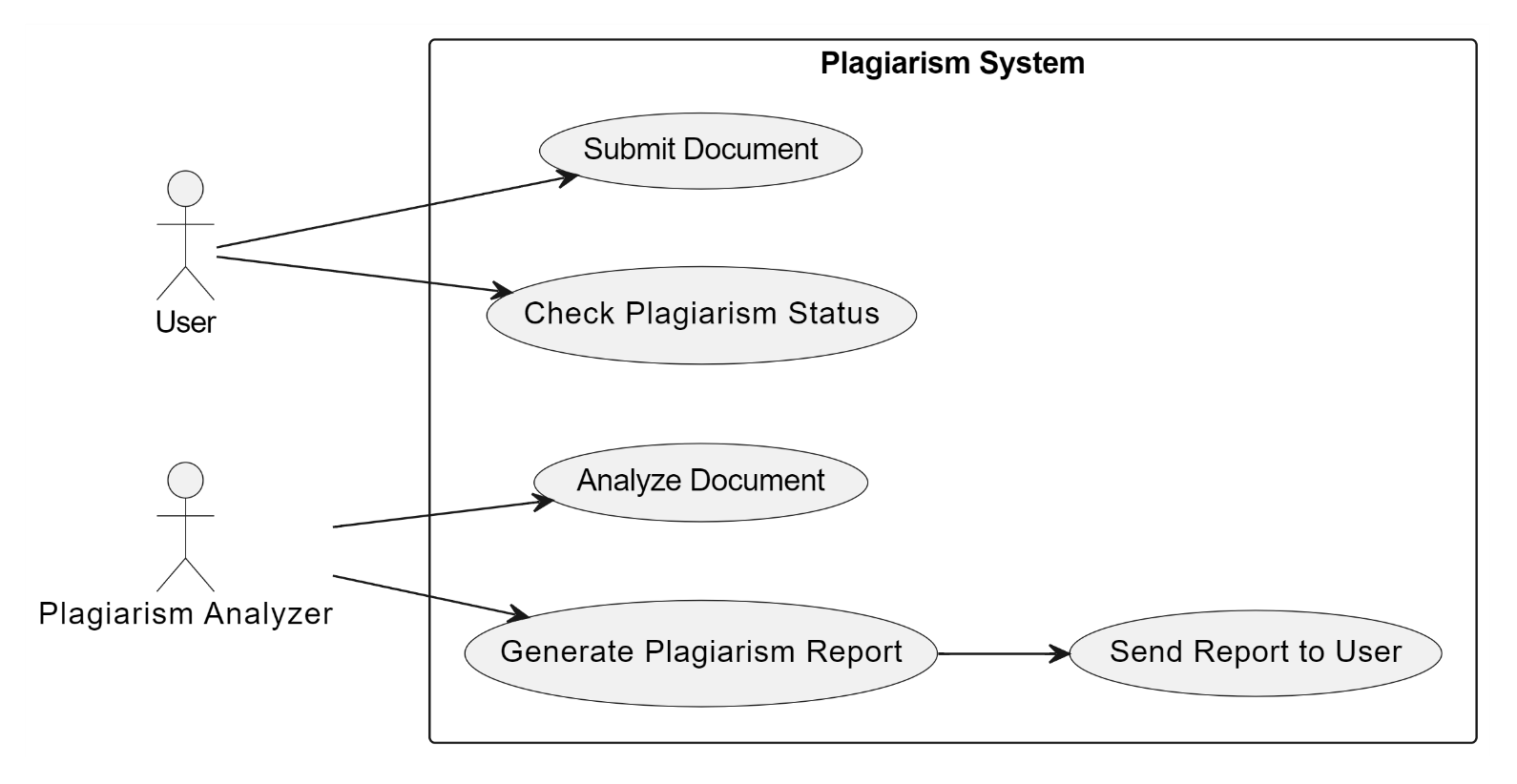
#### **Scalability:**

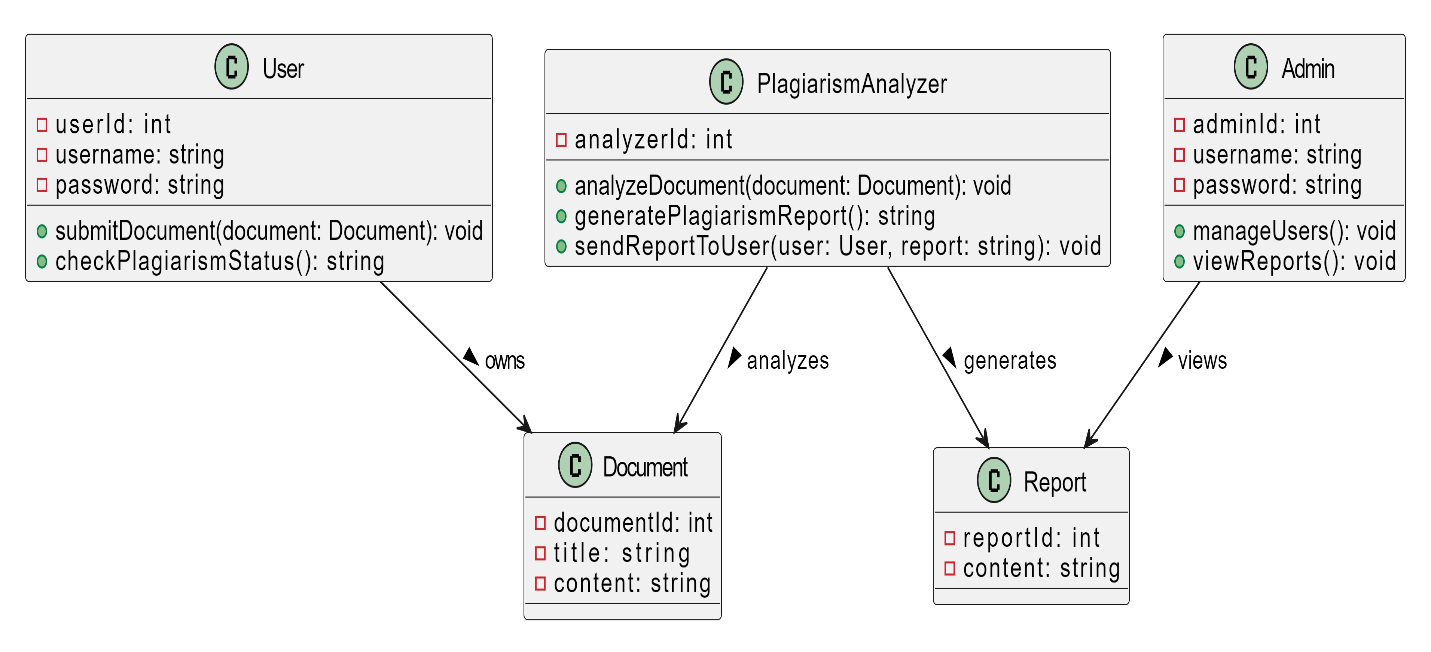
* The system should be scalable to accommodate increasing user demands and data volumes.
* It should support horizontal and vertical scaling mechanisms to adapt to changing requirements and usage patterns.
* Top of Form

**CHAPTER 4**

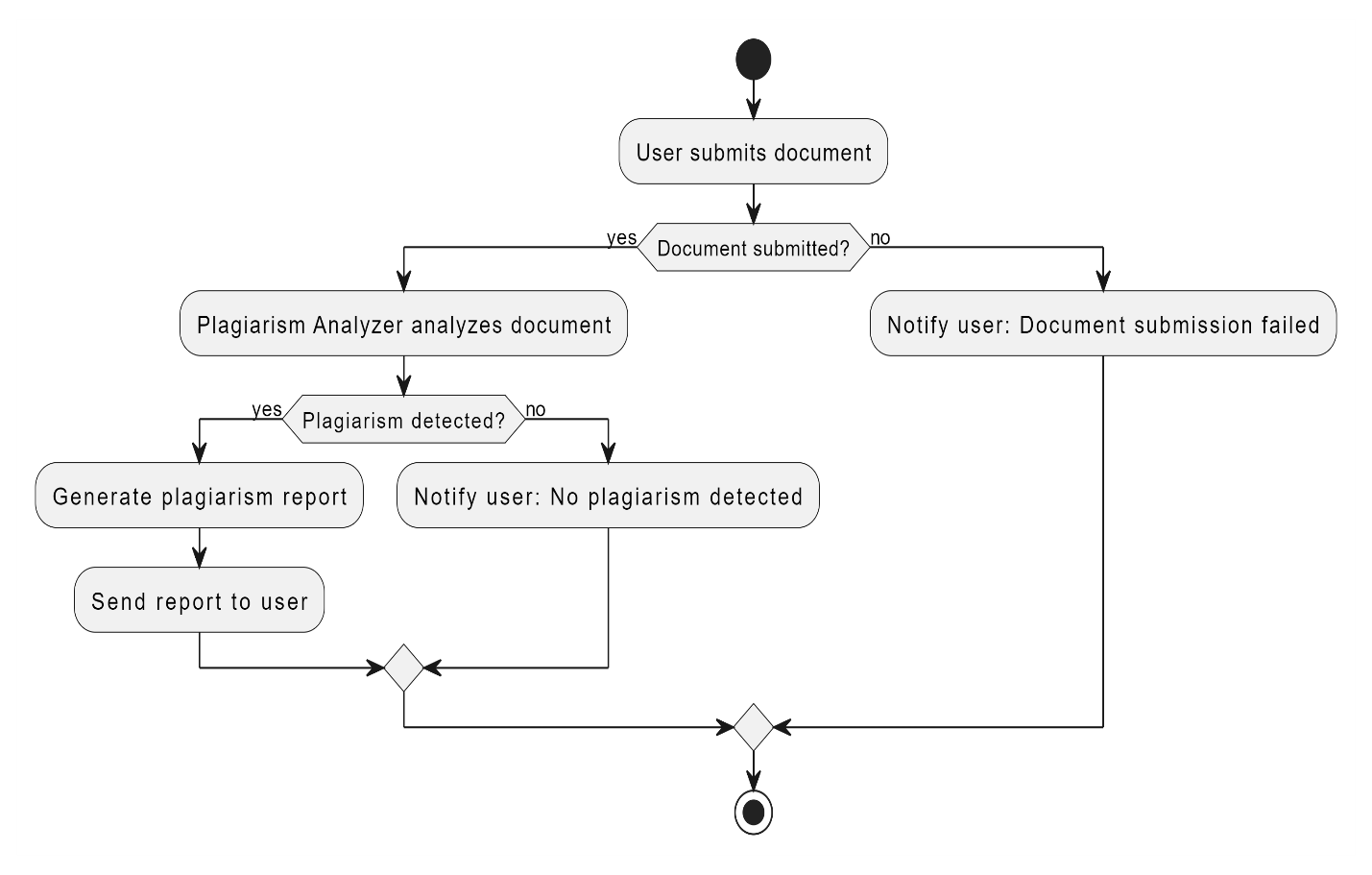
**SYSTEM DESIGN**

**1.Use Case Diagram:**

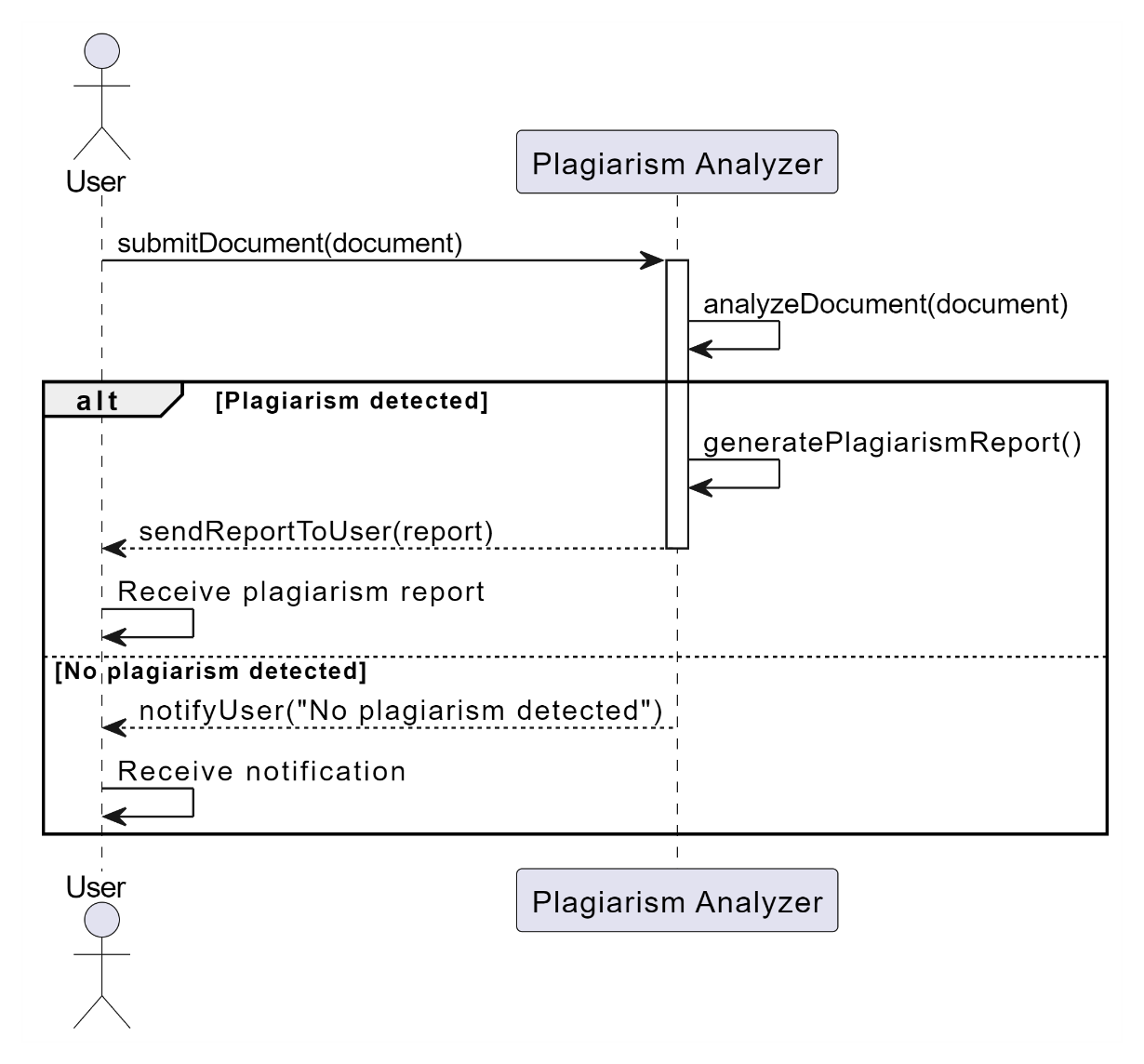


**2.Class Diagram:**

**3.Activity Diagram:**



**4.Sequence Diagram:**



**CHAPTER – 5**

**IMPLEMENTATION**

### **5.1 SAMPLE CODE**

import streamlit as st

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine\_similarity

import numpy as np

# Function to calculate cosine similarity

def similarity(doc1, doc2):

return cosine\_similarity([doc1, doc2])[0][1]

# Main function to check plagiarism and calculate percentile

def check\_plagiarism\_and\_percentile(text1, text2, text3):

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit\_transform([text1, text2, text3]).toarray()

plagiarism\_scores = []

for i in range(len(vectors)):

for j in range(i+1, len(vectors)):

sim\_score = similarity(vectors[i], vectors[j])

plagiarism\_scores.append(sim\_score)

percentile = np.percentile(plagiarism\_scores, 90)

return plagiarism\_scores, percentile

# Streamlit UI

st.title("Plagiarism Checker")

st.write("Enter three texts and click 'Check Plagiarism' to find plagiarism.")

# Text inputs for three texts

text1 = st.text\_area("Enter text 1:", height=150)

text2 = st.text\_area("Enter text 2:", height=150)

text3 = st.text\_area("Enter text 3:", height=150)

# Button to trigger plagiarism check

check\_button = st.button("Check Plagiarism")

# Styling

st.markdown("---")

st.markdown(

"""

<style>

.stTextInput>div>div>textarea {

border-radius: 10px;

border: 2px solid #d3d3d3;

padding: 10px;

}

.stButton>button {

border-radius: 10px;

padding: 10px 15px;

background-color: #008CBA;

color: white;

}

.stButton>button:hover {

background-color: #005f73;

}

</style>

""",

unsafe\_allow\_html=True,

)

# Result

if check\_button:

if text1.strip() == "" or text2.strip() == "" or text3.strip() == "":

st.warning("Please enter text in all three boxes.")

else:

plagiarism\_scores, \_ = check\_plagiarism\_and\_percentile(text1, text2, text3)

for i, score in enumerate(plagiarism\_scores, start=1):

pair\_num = f"{i//2 + 1}" if i % 2 == 0 else f"{i//2 + 1} and {i//2 + 2}"

st.success(f"percentile of plagiarism for pair {pair\_num}: {score:.2f}")

**CHAPTER – 6**

## TESTING

### **6.1 TEST CASES**

|  |  |
| --- | --- |
| **Test Case ID:** | 1 |
| Test Case Name: | Normal Input - No Plagiarism |
| Purpose: | To verify the system's behavior when there is no plagiarism. |
| Input: | Text 1: "This is the first document."  Text 2: "This is the second document."  Text 3: "This is the third document." |
| Expected Result: | All plagiarism scores should be close to 0.0 |
| Actual Result: | Plagiarism scores displayed correctly |

|  |  |
| --- | --- |
| **Test Case ID:** | 2 |
| Test Case Name: | Empty Input |
| Purpose: | To verify that the system handles empty inputs appropriately. |
| Input: | Text 1: ""  Text 2: ""  Text 3: "" |
| Expected Result: | Warning message: "Please enter text in all three boxes." |
| Actual Result: | [Check button not pressed] |

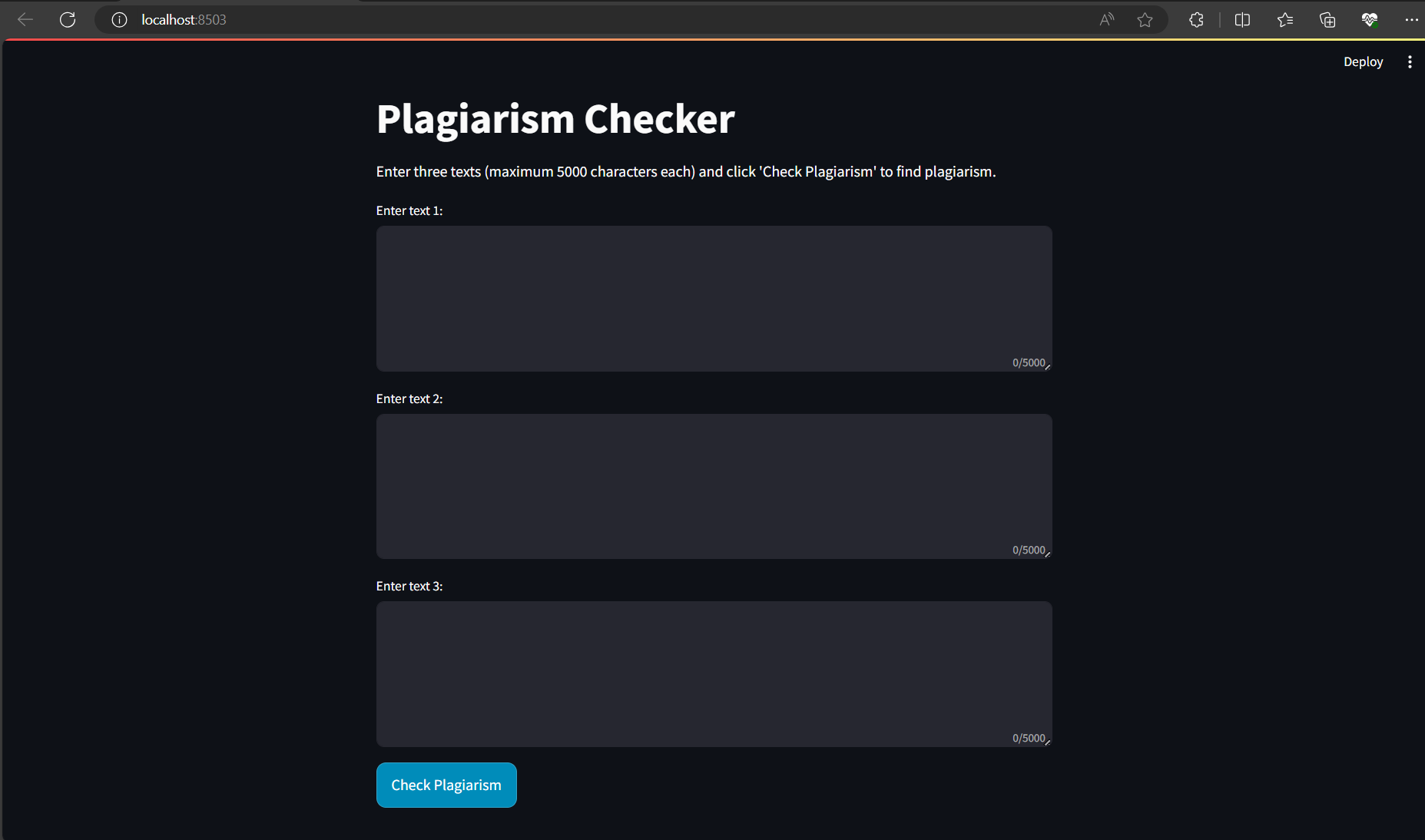
|  |  |
| --- | --- |
| **Test Case ID:** | 3 |
| Test Case Name: | Normal Input - Moderate Plagiarism |
| Purpose: | To verify the system's ability to detect moderate plagiarism. |
| Input: | Text 1: "This is the first document."  Text 2: "This is the second document."  Text 3: "This is the first document with a few changes." |
| Expected Result: | Some plagiarism scores should be higher, indicating moderate plagiarism. |
| Actual Result: | Plagiarism scores displayed correctly |

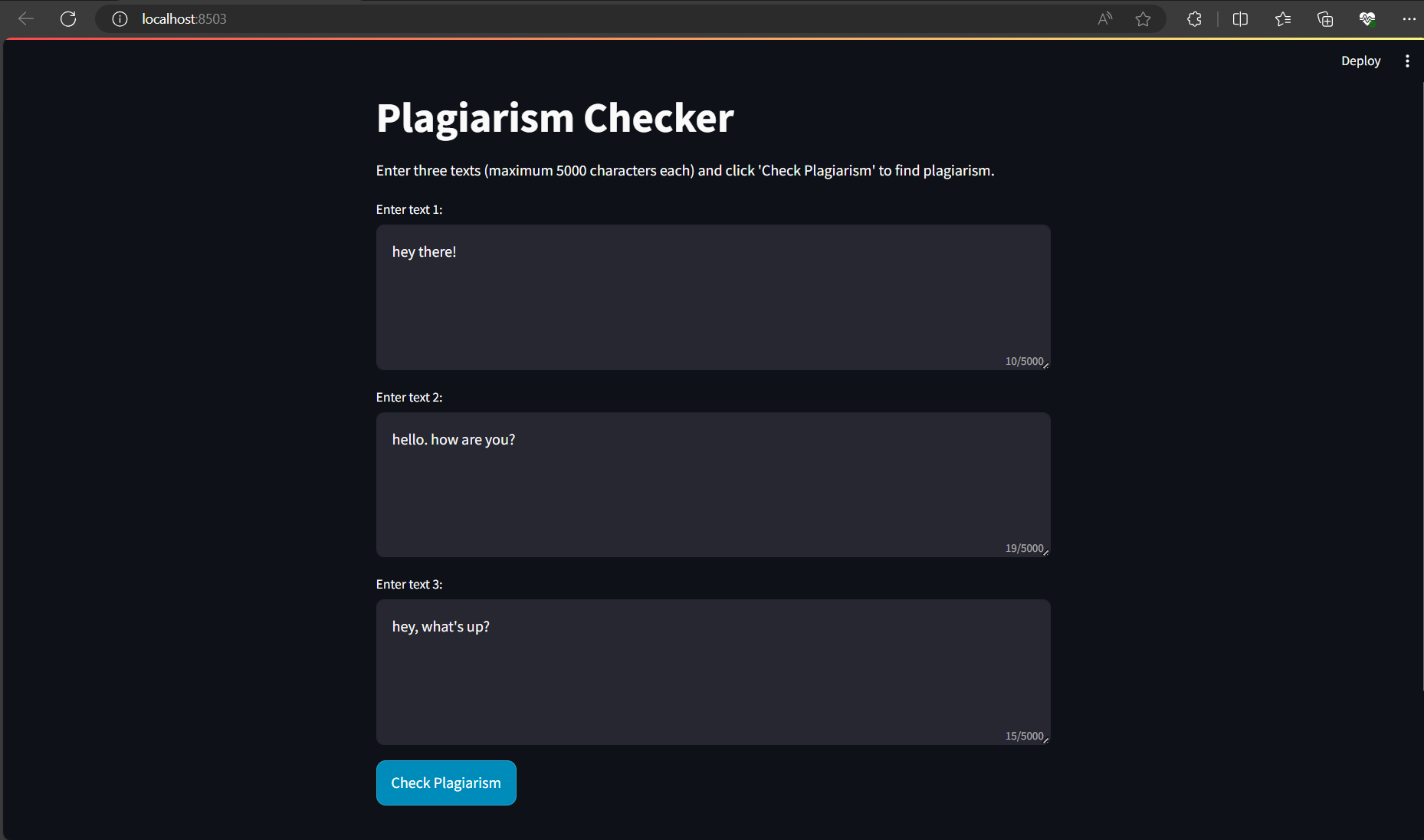
|  |  |
| --- | --- |
| **Test Case ID:** | 4 |
| Test Case Name: | Normal Input - High Plagiarism |
| Purpose: | To verify the system's ability to detect high levels of plagiarism. |
| Input: | Text 1: "This is the first document."  Text 2: "This is the first document with some alterations."  Text 3: "This is the first document with significant modifications." |
| Expected Result: | Higher plagiarism scores indicating significant plagiarism. |
| Actual Result: | Plagiarism scores displayed correctly |

|  |  |
| --- | --- |
| **Test Case ID:** | 5 |
| Test Case Name: | Maximum Character Input |
| Purpose: | To verify that the system handles the maximum character limit for each text. |
| Input: | Text 1: [5000 characters of random text]  Text 2: [5000 characters of random text]  Text 3: [5000 characters of random text] |
| Expected Result: | Plagiarism scores are calculated correctly for the provided long texts. |
| Actual Result: | Plagiarism scores displayed correctly |

## CHAPTER - 7

## SCREENSHOTS

****

****

****

**CHAPTER – 8**

**CONCLUSION AND FUTURE SCOPE**

In this chapter, we conclude the findings and outcomes of our project, the Plagiarism Analyzer. We also explore potential future directions and enhancements to the system.

#### **8.1 Conclusion**

The Plagiarism Analyzer project has addressed the critical issue of detecting and mitigating plagiarism in programming assignments. Through the development and implementation of sophisticated algorithms and user-friendly interfaces, the analyzer provides educators with an effective tool to maintain academic integrity and fairness in programming education.

Key achievements of the project include:

* Successful implementation of code comparison algorithms capable of accurately detecting similarities between code submissions.
* Development of intuitive user interfaces for both users and administrators, facilitating seamless code submission, analysis, and reporting processes.
* Provision of comprehensive analysis reports highlighting detected similarities and suspected cases of plagiarism, empowering educators to make informed decisions and provide constructive feedback to students.

The Plagiarism Analyzer project represents a significant step forward in promoting academic honesty and originality in programming assignments. By equipping educators with the necessary tools and resources to detect and address plagiarism effectively, the analyzer contributes to the cultivation of a culture of integrity and excellence in programming education.

#### **8.2 Future Scope**

While the Plagiarism Analyzer project has achieved its primary objectives, there are several avenues for future enhancement and expansion:

1. **Integration with Learning Management Systems (LMS):** Explore opportunities to integrate the analyzer seamlessly with popular LMS platforms used by educational institutions. This integration would streamline the workflow for educators and enhance the accessibility of the analyzer to a broader user base.
2. **Enhanced Analysis Algorithms:** Continuously refine and improve the code comparison algorithms to enhance the accuracy and efficiency of plagiarism detection. Incorporate machine learning and natural language processing techniques to analyze code structure, logic, and comments more comprehensively.
3. **Advanced Reporting and Analytics:** Expand the capabilities of the analysis reporting module to provide deeper insights and analytics into code submissions. Implement visualizations and statistical analyses to identify trends, patterns, and areas of improvement in programming assignments.
4. **Real-Time Analysis and Feedback:** Develop mechanisms for real-time analysis and feedback, allowing educators to receive immediate notifications and alerts regarding suspicious code submissions. Implement automated remediation workflows to facilitate prompt intervention and resolution of plagiarism cases.
5. **Collaborative Features:** Introduce collaborative features that enable students to review and provide feedback on each other's code submissions in a controlled and monitored environment. Foster peer learning and collaboration while maintaining academic integrity and accountability.
6. **Support for Additional Programming Languages:** Expand the analyzer's capabilities to support a broader range of programming languages and frameworks commonly used in educational settings. Ensure compatibility and accuracy across diverse programming paradigms and environments.

In conclusion, the Plagiarism Analyzer project holds significant promise in advancing the integrity and quality of programming education. By embracing ongoing innovation and collaboration, the analyzer has the potential to evolve into a versatile and indispensable tool for educators and students alike in the pursuit of academic excellence.

### **APPENDIX A: TOOLS AND TECHNOLOGIES**

1. Programming Language:

Python 3.x

2. Web Framework:

Streamlit

3. Machine Learning Libraries:

scikit-learn (sklearn) for cosine similarity calculation

NumPy for numerical operations

4. Text Processing Library:

NLTK (Natural Language Toolkit) for text preprocessing (not explicitly used in the provided code, but can be integrated for advanced preprocessing)

5. Version Control:

Git

6. Code Editor:

Visual Studio Code, PyCharm, or any preferred Python IDE

7. Development Environment:

Anaconda (optional but recommended for managing Python environments)

8. Dependency Management:

Pip (Python package installer) for managing project dependencies

9. Styling (for Streamlit UI):

HTML and CSS (used within the Streamlit application for styling)

10. Documentation:

- Markdown for documentation (used in the provided code for documentation within Streamlit)

11. Testing:

- Manual testing for various scenarios and edge cases

12. Collaboration:

- GitHub for code hosting, version control, and collaboration

13. Continuous Integration (CI):

- Optional: Integration with CI tools like GitHub Actions, Travis CI, or Jenkins for automated testing and deployment

14. Deployment:

- Platforms like Heroku, AWS, or any preferred cloud platform for deploying the Streamlit application

Note: The tools and technologies listed above are based on the provided code and its dependencies. Actual tools used may vary depending on the specific project requirements and preferences of the development team. Additionally, the provided code assumes the availability of necessary packages, which can be installed using the requirements.txt file or equivalent.