A Mini Project report submitted on

PLAGIARISM TEXT CHECKER

Partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

SUBMITTED

By

D.ANURADHA

21671A0583

Under the esteemed guidance of

Mr. D. HIMAGIRI

ASSOCIATE PROFESSOR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

J.B. INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC AUTONOMOUS

(Accredited by NAAC & NBA, Approved by AICTE & Permanently affiliated by JNTUH)

Yenkapally, Moinabad mandal, R.R. Dist-75 (TG)

2021 – 2025

J.B. INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC AUTONOMOUS

(Accredited by NAAC & NBA, Approved by AICTE & Permanently affiliated by JNTUH)

Yenkapally, Moinabad mandal, R.R. Dist-75 (TG)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Mini Project report entitled "PLAGIARISM TEXT CHECKER " submitted to the Department of Computer Science and Engineering, J.B Institute of Engineering & Technology, in accordance with Jawaharlal Nehru Technological University regulations as partial fulfilment required for successful completion of Bachelor of Technology is a record of bonafide work carried out during the academic year 2024-2025 by,

D. ANURADHA

21671A0583

Internal Guide

Mr. D. HIMAGIRI

ASSOCIATE PROFESSOR

Head of the Department

Dr. G SREENIVASULU

ASSOCIATE PROFESSOR

J.B. INSTITUTE OF ENGINEERING & TECHNOLOGY

UGC AUTONOMOUS

(Accredited by NAAC & NBA, Approved by AICTE & Permanently affiliated by JNTUH)

Yenkapally, Moinabad mandal, R.R. Dist-75 (TG)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

I hereby declare that the Mini Project report entitled "**PLAGIARISM TEXT CHECKER**" carried out under the guidance of, **Mr. D. HIMAGIRI**, **Associate Professor** is submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology** in **Computer Science and Engineering.** This is a record of bonafide work carried out by us and the results embodied in this project report have not been reproduced or copied from any source. The results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

Date: / /

D.ANURADHA
21671A0583

J.B. INSTITUTE OF ENGINEERING & TECHNOLOGY

(UGC AUTONOMOUS)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this Industry Oriented Mini project report entitled "**PLAGIARISM TEXT CHECKER"** submitted to the Department of Computer Science and Engineering, JB Institute of Engineering & Technology, is a bonafide work done by **D.Anuradha** bearing hall ticket no. **21671A0583** under my supervision from **13-May-2024 to 19-June-2024**.

Dr . G SREENIVASULU

Associate Professor of CSE & HOD

ACKNOWLEDGEMENT

At outset I express my gratitude to almighty lord for showering his grace and blessings upon me to complete this Mini Project. Although my name appears on the cover of this book, many people have contributed in some form or the other to this project development. I could not have done this Project without the assistance or support of each of the following.

First of all I am highly indebted to Dr. P. C. KRISHNAMACHARY, Principal for giving us the permission to carry out this Mini Project.

I would like to thank Dr. G. SREENIVASULU, Associate Professor & Head of the Department of COMPUTER SCIENCE AND ENGINEERING, for being moral support throughout the period of the study in the Department.

I am grateful to Mr. D. HIMAGIRI, Associate Professor COMPUTER SCIENCE AND ENGINEERING, for his valuable suggestions and guidance given by him during the execution of this Project work.

I would like to thank the Teaching and Non-Teaching Staff of the Department of Computer Science & Engineering for sharing their knowledge with me.

D.ANURADHA
21671A0583

TABLE OF CONTENTS

ABSTRACT

Plagiarism is a act of using someone else ‘s work without proper attribution plagiarism analyzer ensures the originality and integrity of written content .It is included in various domains such as education, publishing, and reseach .The analyzer compares the input text with existing databases or reference materials . Identifies the similarities and highlights . Plagiarism analyzer is bulit using industry-standard tools and frame works such as HTML, CSS, Javascript for algorithm implementation .Our system employs a database to store reference materials and preprocessed data, ensuring efficient comparision and analysis. The analyzer prioritized data privacy security and responsibility for misuse or false accusations of plagiarism .

LIST OF FIGURES

1.INTRODUCTION

In the current digital era, plagiarism detection techniques are crucial for maintaining the originality and integrity of written work. These programs use a large database of previously published works, such as scholarly articles, websites, and publications, to compare texts for similarities. Plagiarism analyzers assist writers, students, and professionals in upholding ethical standards and avoiding the hazards of inadvertent or intentional plagiarism by pointing out possible instances of copied content .Using a large database of previously published works as a comparison, a plagiarism analyzer is a technology that finds instances of duplicated or unoriginal content. It is crucial for students, teachers, and professionals to avoid plagiarism and follow ethical standards in writing since it contributes to ensuring the integrity and originality of written work. Plagiarism is a serious academic and professional offense that can have severe consequences ,including loss of credibility ,academic penalties, and even legal repercussions. With the increasing availability of online resources and digital content, the risk of unintentional plagiarism has also risen.

2.LITERATURE SURVEY

1.Title: Duplicate and Plagiarism Search in Program Code Using Suffix Trees over Compiled Code

Authors: Igor Andrianov, Svetlana Rzheutskaya, Alexey Sukonschikov, Dmitry Kochkin, Anatoly Shvetsov

Year: May 24,2020

Observations:

Efficiency of Suffix Trees: The longest common substring identification, pattern matching, and substring searches can all be effectively performed with suffix trees, making them an efficient data structure. To some extent, the approach is scalable, but as the size and number of codebases grow, there may be issues. Partial processing and optimizations can help with some scaling problems.

Shortcomings: It is not easy to implement suffix trees and modify them for compiled code analysis. It necessitates a thorough comprehension of the produced code format's subtleties as well as the data structure. Suffix tree generation and traversal algorithms can be intricate, requiring careful optimization to be effective in real-world scenarios.

2.Title: Investigating the Understanding of Plagiarism: ACase Study of Code .

Authors: Lujiang Yu, Hui Jiang, Hongming Zhu, Qinpei Zhao Tongji University

Year: August 18-20, 2020

Observations:

Plagiarism Prevalence: Code plagiarism is very common among Chinese professionals and students frequent reusing of code without giving due credit .The curriculum does not provide a thorough instruction on plagiarism. Academic integrity and ethical coding standards are not given enough attention. Shortcomings: Limited Scope: The results may not be as applicable to other places due to the focus.

3.Title: Revisiting the Challenges and Opportunities in Software Plagiarism Detection

Authors: Xi Xu, Ming Fan, Ang Jia , Yin Wang, Zheng Yan, Qinghua Zheng, and Ting liu .

Year: May 31,2020

Observations:

Growing Prevalence of Software Plagiarism : Professional software development and academic integrity are facing serious obstacles as a result of the increasing accessibility of online resources and the simplicity with which code can be copied. Because code can be changed or obscured in a number of ways, including renaming variables, removing comments, or rearranging code segments without affecting functionality, software plagiarism detection is intrinsically complex.

Shortcomings:

High Computational Cost: A lot of sophisticated plagiarism detection approaches demand a lot of processing power, especially those that use machine learning and graph-based algorithms. Institutions or groups with minimal funding may find this to be an obstacle.

4.Title: Closed Domain Bangla Extrinsic Monolingual Plagiarism Detection and Corpus Creation Approach

Authors: Adil Ahnaf, Shadhin Saha, Nahid Hossain Department of Computer Science and Engineering United International University

Year: June 16,2021

Observations:

Comprehensive Dataset: The article most likely describes the process of building a comprehensive corpus that is unique to the Bangla language. Diversity of Sources: To guarantee the stability of the detection system, the dataset may contain a wide variety of texts, including news stories, blog entries, and scholarly publications.

Shortcomings:

Diversity and Size of the Corpus: A possible drawback would be the small size of the corpus that was produced. A short dataset might not fully represent the linguistic diversity, which could result in overfitting or poor performance in practical setting.

5.**Title:** Code Plagiarism Detection Method Based on Code Similarity and Student **Authors:** Qiubo Huang,Guozheng

Year: September 07,2020

Observations:

Integration of Code Similarity and Behavioral Analysis: This method offers a comprehensive approach to detecting plagiarism by combining the detection of code similarity with the analysis of student behavior characteristics. By taking into account trends in student behavior in addition to code similarity, the dual-layered technique has the potential to improve the accuracy and reliability of plagiarism detection.

Shortcomings:

Gathering and examining data on student behavior presents serious privacy concerns. To address these issues, a clear policy on data usage, storage, and protection is required. Data Collection Overhead: Gathering comprehensive behavioral data is an extra need that may result in costs for computing power and data storage.

6.Title: Plagiarism Detection:A Data Search Challenge Authors: Preeti Chauhan,NoritaAhmad Published

Year: 19 November 2020

Observations:

Technological Advancements: It is possible that the article examines the latest developments in plagiarism detection technology, emphasizing the use of machine learning models and algorithms to detect copied content.

Data Sources: It looks at a range of data sources, such as scholarly articles, websites, code repositories, and more, that are used to identify plagiarism. The paper addresses many algorithms and methods for identifying plagiarism, including string matching, fingerprinting, stylometry, and semantic analysis.

Shortcomings:

Data Quality and Diversity: One typical flaw in plagiarism detection systems may be the insufficient diversity and quality of the data sets used for testing them, which may not be sufficient to cover all possible plagiarism instances. Complexity and Overhead: Some sophisticated algorithms may be computationally costly, resulting in lengthy processing times.

3. SYSTEM ANALYSIS

3.1 Existing System

A Synopsis of the Current Code Plagiarism Detection System Several significant flaws in the existing code plagiarism detection technique impair its effectiveness:

Error in the Obfuscated Code:

The system finds it difficult to correctly discover code similarities, particularly when the code is purposefully changed or obfuscated to avoid detection. Plagiarism that goes unnoticed may arise from this.

3.2 Absence of Advanced Methods of Detection:

The system doesn't employ sophisticated methods to spot minute alterations in code or odd activity patterns. This implies that complex instances of plagiarism where the code has been marginally altered or rearranged may go unnoticed. Absence of Contextual Analysis Important contextual data, such assignment instructions and code comments, is ignored by the system. This error may cause a lag in the detecting process and a delayed reaction time. Concerns about Fairness and Transparency: The method by which the system determines plagiarism is opaque. Due to a lack of clarity on the criteria utilized, students may be unfairly accused of plagiarism, which could compromise fairness.

Dependence on Detection Based on Syntax:

Syntax-based algorithms are the primary means by which the system compares submissions of code. This approach has limitations since it might miss instances of plagiarism where the syntax is altered but the fundamental logic is replicated. All things considered, the current system lacks sophisticated analytical tools, has trouble identifying complex plagiarism.

3.3 Proposed system:

An Overview of the Suggested Code Plagiarism Detection System The suggested approach uses cutting-edge deep learning models to greatly increase the efficacy and accuracy of code plagiarism detection. Here's a quick rundown:

More Complex Deep Learning Methods:

Tokenization is the technique of dividing code into manageable chunks (tokens) so they may be examined in more depth. Semantic analysis looks beyond simple syntax to understand the functioning and meaning of the code in order to find logical parallels. Neural embedding is the process of using neural networks to build complex code representations that capture the deeper traits and connections inside the code. How Convolutional Neural Networks (CNN) Are Used:

- **Code Representation**: To represent and analyze code structures, CNNs will be employed. These networks are good at finding intricate connections and patterns in the code. Similarity Detection: The system uses CNNs to identify complex types of plagiarism, such as those in which the code has been changed or obscured. Combining with the Current System:
- **Improved Capabilities**: By integrating the new deep learning components into the existing system, its general robustness and functionality will be improved.
- **Increased Accuracy:** As a result of the integration, plagiarism detection will be more accurate, resulting in a decrease in false positives and negatives.
- **Encouraging Academic Fairness and Integrity:**
- **Academic Integrity:** By successfully identifying instances of plagiarism, the improved detection capabilities will assist uphold high standards of academic honesty.
- **Fairness and Transparency**: The system's operations will be visible, guaranteeing that the procedures used for plagiarism detection are equitable and intelligible to all users. In summary, the suggested approach improves upon the current code plagiarism detection system by utilizing cutting-edge deep learning methods, especially CNNs.

3.3.1 Proposed Model

We are implementing deep learning models for code similarity detection and to enhance the accuracy detection. Deep learning model includes techniques such as tokenization ,semantic analysis and neural embending to capture intricate code relations.Here we are integrating the deep learning models such as Convolutional Neural Networks (CNN) for code representation and similarity detection. These models can learn complex patterns and relationships with the code and improves accuracy of similarities detection. By implementing these components into the existing system,we can create a more robust.effective plagiarism detection system. system promotes academic integrity ,and ensures fairness and transparency in assessment processes.

3.4 Software Requirements

- **Compatible Web Browser:**

  Users need a modern web browser that supports HTML5, CSS3, and JavaScript. Popular options include Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari. JavaScript Enabled: The browser must have JavaScript enabled since the functionality of the program relies on JavaScript for handling user interactions and performing the plagiarism analysis. Support for CSS: The browser should support CSS for styling the HTML elements.

  This ensures that the webpage appears visually appealing and is easy to use. Text Editor or Integrated Development Environment (IDE): You'll need a text editor or an IDE to write, edit, and save the HTML, CSS, and JavaScript code. Some common options include:

- **Visual Studio Code:**

  A lightweight but powerful code editor with support for HTML, CSS, JavaScript, and many other programming languages. Sublime Text is another popular code editor known for its speed and ease of use. Atom is a customizable text editor developed by GitHub. Notepad++ is a free source code editor and Notepad replacement that supports several programming languages.

- **Text Editor or Integrated Development Environment (IDE):**

  Purpose: A text editor or IDE is necessary for writing, editing, and managing the HTML, CSS, and JavaScript code.

  Examples: Popular text editors and IDEs include Visual Studio Code, Sublime Text, Atom, Notepad++, and more. These tools offer features like syntax highlighting, code completion, and easy project management.

- **Operating System (OS):**

  Purpose: The software should be compatible with the operating system used by developers and end-users. The software should support major operating systems like Windows, macOS, and Linux. Examples: Windows 10, macOS Catalina, Ubuntu 20.04, etc.

- **Dependencies and Libraries:**

  Purpose**:** Additional libraries or frameworks may be used to enhance the functionality or appearance of the web application. Installation and management of any required libraries or frameworks. Examples: jQuery, Bootstrap, React, Vue.js, etc.
- **Testing Environment:**

  Purpose: Developers might need a testing environment to verify the functionality and behavior of the JavaScript code. A testing environment could be a local development server or an online code editor with live preview features. Examples: Localhost server setup using tools like XAMPP, MAMP, or running a simple HTTP server with Python's

  .

3.5 Hardware Requirements

- **Processor: Intel** Core i3 and above
- **Memory (RAM):** A minimum of 4 GB of RAM is recommended
- **Network Connectivity**: A stable and reliable internet connection through wired Ethernet or Wi-Fi

Datasets

Large text datasets are usually required in order to compare the text being analyzed with the text being built as a plagiarism analyzer. The following resources and datasets are available for use: assemblage of texts collection of materials that have been published, including scholarly papers, books, and articles. They act as a point of reference when submitting works to be compared to previously published works. A lot of plagiarism detectors search webpages and articles found online to look for patterns in submitted writings. Common sources include search engines, scholarly databases, and online repositories.

Making Use of the Datasets

- **Preprocessing:** Simplify the datasets by breaking them into digestible segments, standardizing the language, and eliminating superfluous characters.
- **Algorithms for Text Comparison** : For semantic similarity, use algorithms like fingerprinting, string matching (like Rabin-Karp), and more sophisticated methods like cosine similarity, Jaccard similarity, or machine learning models like BERT.

- **String Tokenization**: This technique divides a text string into smaller pieces known as tokens. Usually, tokens consist of words, phrases, or symbols that are divided by commas, spaces, or special characters. In natural language processing (NLP) applications including text analysis, sentiment analysis, and machine translation, tokenization is an essential first step.
- **Stop Word Removal:** When preparing text for tasks involving natural language processing, stop word removal is an essential step. Stop words are frequently used terms with little semantic value that are eliminated to cut down on noise and boost algorithm performance. Stop words include, for example, "the," "is," "and," "in," "of," etc. H

4. SYSTEM DESIGN

4.1 ARCHITECTURE

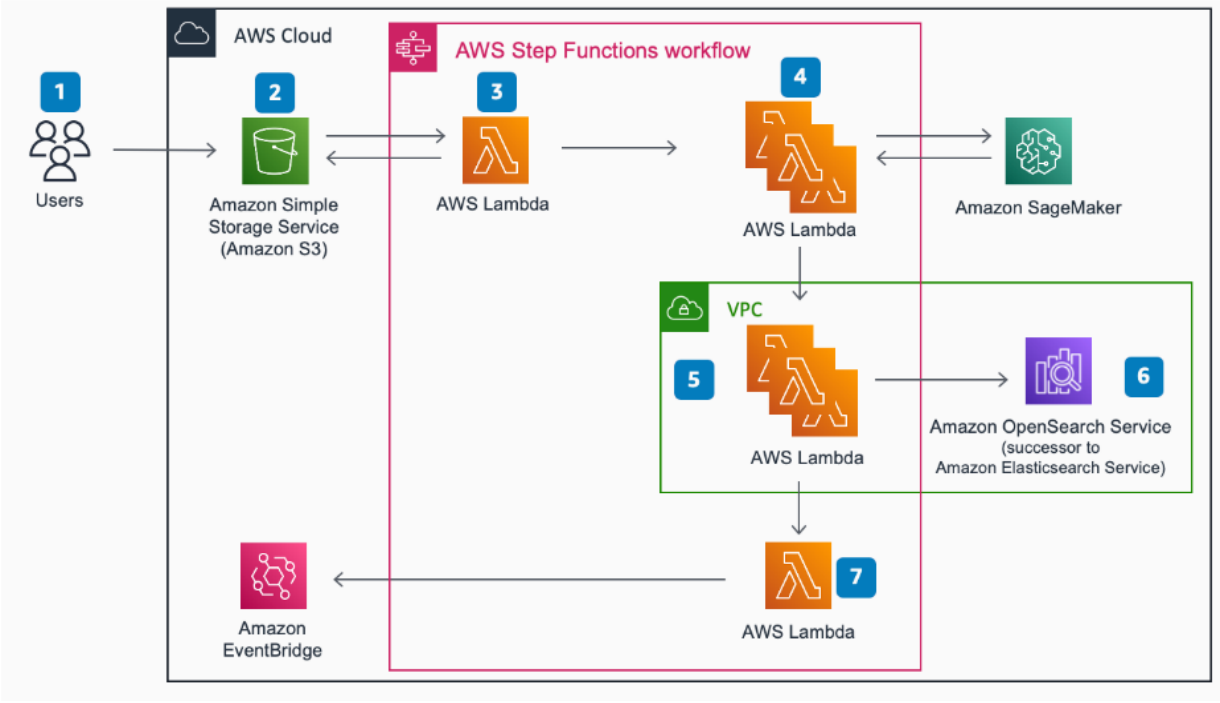

fig1. Plagiarism Detection Architecture Diagram

Module Description

1. Copy the document you'd like to run plagiarism detection on to **Amazon Simple Storage Service** (Amazon S3).
2. **Amazon S3** event triggers start of **AWS Step Functions** workflow.
3. **AWS Lambda** function extracts text from document using Tika (a content analysis toolkit that detects and extracts metadata and text from over a thousand different file types.
4. For each paragraph in the document, text is passed to a pre-trained Bidirectional Encoder Representations from Transformers (BERT)-based model to extract word embedding vectors.
5. For each word embedding vector, a K-Nearest Neighbor (KNN) search is run using a cosine-similarity algorithm.
6. **Amazon OpenSearch Service** (OpenSearch Service) domain stores an index of pre-processed works that have been converted into word embedding vectors and indexed.

7. Based on the configured similarity threshold that is compared against the **OpenSearch Service** query result score, an event bridge event is raised, specifying source

Detailed input module workflow user access:

- Users log in through a secure login interface. If you are a new user, you can register and create an account. Transferring data: Users have the option of uploading a file or directly pasting text/code. To download a file, the user selects the file on their device and downloads it. To enter text directly, users paste the text/code into the text field provided. Inspection and storage: The system verifies the correct format and size of input data. Once verification is completed successfully, your data will be temporarily stored in a secure location.
- Shipping Confirmation: The user receives a confirmation message confirming that the submission was successful. If there are any errors, an appropriate error message is displayed and the user is prompted to resolve the issue. Your data is ready for analysis. Now the preprocessed data is ready to be passed on to the next module (comparison module) for plagiarism detection. This detailed analysis allows the input module to efficiently collect and prepare data for plagiarism analysis, providing a seamless user experience while maintaining data integrity and security users paste the text/code into the text field provided.

Detailed processing module workflow Receive input:

- Receive preprocessed data from the input module. Load reference material from the database. Similarity detection: Runs the Greedy String-Tiling (GST) algorithm to find the best matching substring. Perform N-gram analysis to detect sequence-based similarities. Use a hybrid platform for different types of data (text and code). Deep Learning Analysis: Tokenize input data for compatibility with deep learning models. Use semantic analysis to understand the context and meaning of your data. Use neural embeddings to represent data in multidimensional space.
- Evaluate assignment statements and code comments for contextual information. Analyze transmission behavior for unusual patterns. Database comparison: Compare your input data to a database of reference materials. Run detection algorithms and models using the comparison engine. Aggregating results: Calculate similarity scores and confidence.

Detailed database module workflow Circuit definition and design:

- Develop a database schema that defines tables, fields, and relationships. Normalize schemas to minimize redundancy and ensure data integrity. To improve query performance, create indexes on key fields. Receive and store data: Store reference material, including documents, articles, and code snippets. Stores preprocessed data, including sanitized and tokenized versions of input data.
- Stores metadata such as upload timestamp, file type, word count, and attribute information. Efficient data retrieval: Implement optimized SQL queries to retrieve data. We use search algorithms to find and retrieve relevant references based on keywords, patterns, and other criteria. Database Management: Use MySQL or a similar DBMS to store and manage your data. Use database management tools for schema management, performance tuning, backup, and recovery.

Feedback and Reporting module:

- Collect user feedback: Feedback Form: Provides a structured feedback form that allows users to evaluate the accuracy and effectiveness of the plagiarism analysis results. Survey questions: Include survey questions to gather qualitative feedback about user experience, usability, and suggestions for improvements. Anonymous Feedback: To encourage honest opinions and suggestions, we provide users with the option to provide anonymous feedback. Feedback analysis and processing: Data Collection: Collect feedback data from users and securely store it in a database for analysis.
- Sentiment Analysis: For quantitative analysis, we use sentiment analysis techniques to classify reviews as positive, neutral, or negative. Keyword Analysis: Analyze feedback text for recurring keywords or topics to identify common issues or areas for improvement. Reporting Panel: Review Summary: Displays a summary of review data, including overall satisfaction ratings, sentiment analysis results, and key insights. Trend Analysis. Provides trend analysis charts or graphs to visualize feedback trends over time and highlight areas for improvement or attention.

4.2 UML DIAGRAMS

Use Case Diagram

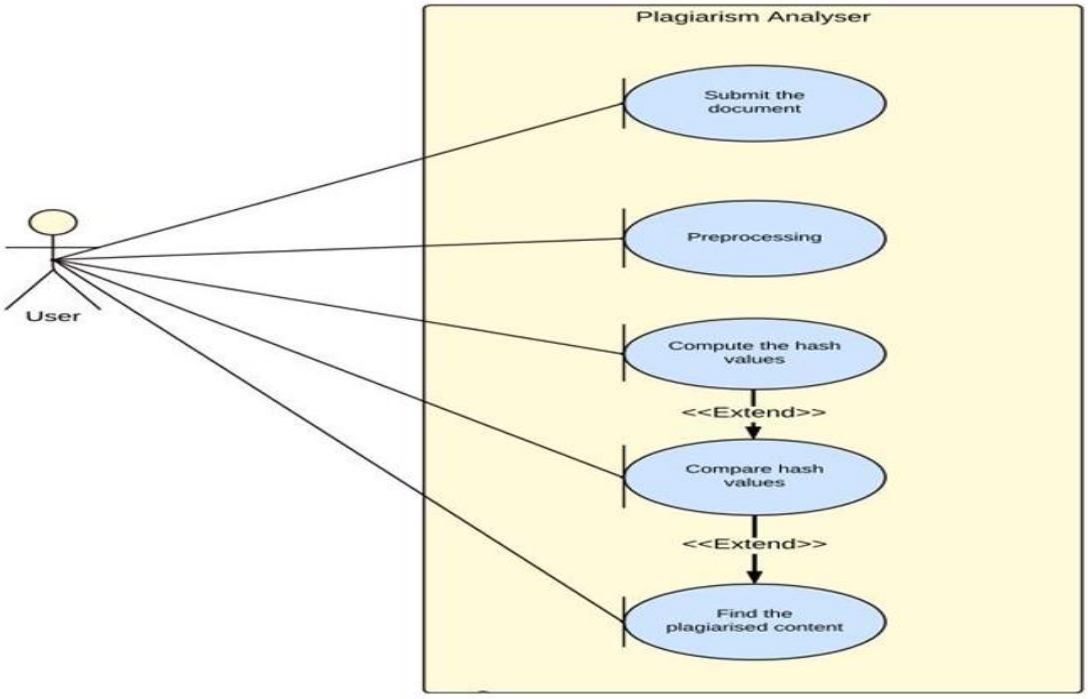

Fig2.Use case diagram

The user's interactions with the Plagiarism Analyzer system are depicted in the diagram. It draws attention to the different features (use cases) that the system provides as well as how the user interacts with these use cases.The main person who uses the Plagiarism Analyzer is the actor. This could be a researcher, instructor, student, or anybody else wishing to ensure that a material is free of plagiarism.

Summary of Diagram: The procedure is started by the user when they submit a document.In order to get the document ready for analysis, the system preprocesses it.The content of the document is calculated as hash values. After then, these hash values are contrasted with current ones to look for patterns.The technology notifies the user of the plagiarized content and identifies it if similarities are detected.

Relevance : The first step in the entire process is to submit the document. Preprocessing guarantees that the document is formatted appropriately for analysis. A critical first step in generating a digital fingerprint of the information is computing the hash values. Finding the Plagiarized Content and Comparing Hash Values are essential tools for identifying.

4.3 SEQUENCE DIAGRAM

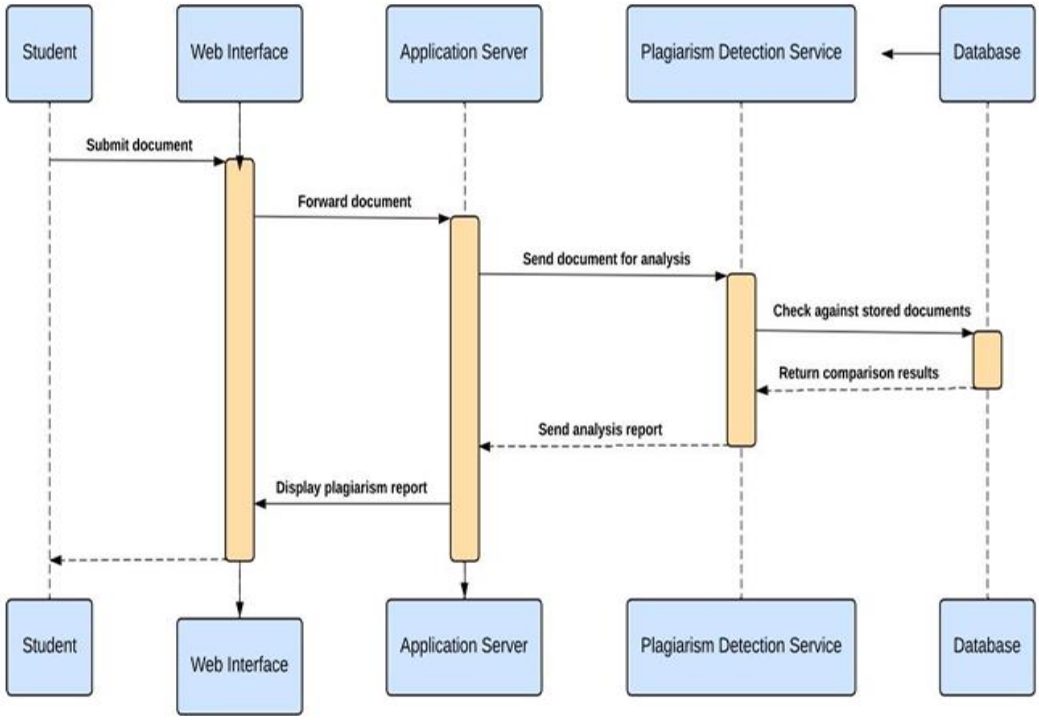

Fig3. Representing the sequence diagram

Actors and Objects:

Pupil: The individual who sends in a document to be checked for plagiarism. The front end that students use to engage with the system is called the web interface. Server that handles requests is known as the application server. Services that verify submitted documents for plagiarism are known as plagiarism detection services. Documents stored for comparison are kept in a database.

Process Overview:

After a student submits a document, the application server and web interface handle it. A service that detects plagiarism receives the document from the application server. A database of previously stored documents is consulted by the plagiarism detection service to verify the document. The student is provided with the outcomes of this comparison via the identical pathway. This flowchart explains how a student's paper is submitted, processed, examined.

4.4 COMPONENT DIAGRAM

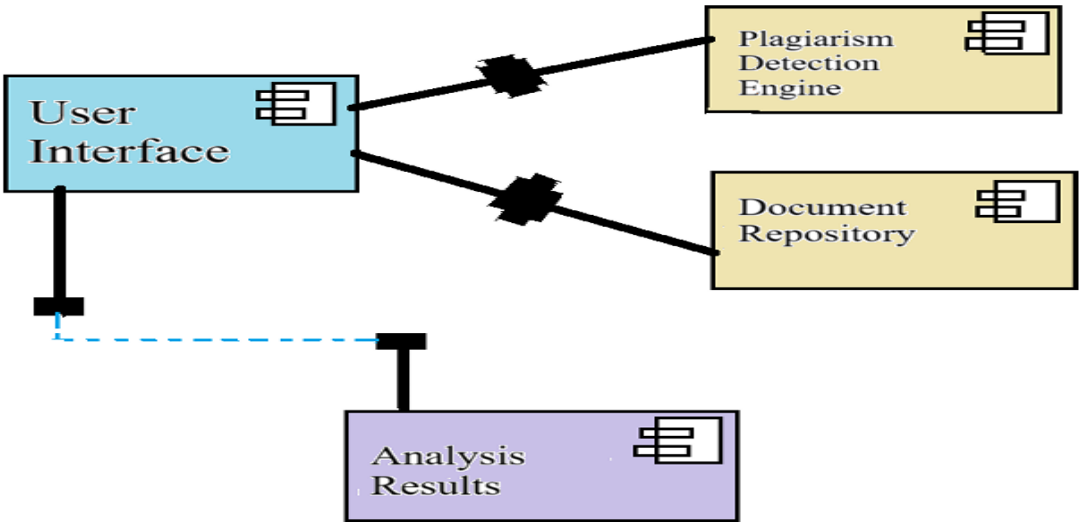

fig4.component diagram

Explaining UML Component Diagrams The Plagiarism Detection System's high-level architecture is represented by the UML Component Diagram, which also illustrates the components' interconnections.

Diagram Synopsis The user interface displays analytical results and serves as the entry point for user interactions. It also permits the input of documents. The primary task of identifying plagiarism in the supplied documents is handled by the plagiarism detection engine. All of the papers that users submit for processing are kept in the document repository. Analysis Results: This file contains the results of the plagiarism detection procedure and offers comprehensive reports and insights. Process Workflow The User Interface is used by the user to submit a document. In the Document Repository, the document is stored by the User Interface. The document is sent to the Plagiarism Detection Engine for analysis via the User Interface. The document is processed by the Plagiarism Detection Engine, and the results are kept in the Analysis Results component. The results are retrieved and shown to the user by the User Interface from the Analysis Results component.

The Plagiarism Detection System's high-level structure is clearly shown in this component diagram, which also shows how the system's various components work together to detect plagiarism.

5. IMPLEMENTATION

SOURCE CODE

```
from kivy.lang import Builder
from kivymd.app import MDApp
from kivy.core.window import Window
from kivy.uix.screenmanager import ScreenManager, Screen
from difflib import SequenceMatcher

class Login(Screen):
    pass

class Signup(Screen):
    pass

class Dashboard(Screen):
    pass

class Result(Screen):
    pass

string="""
<Login>:
    MDLabel:
        text: "Plagiarism Detection"
        font_size: 50
        pos_hint: {"center_x": 0.5, "center_y": 0.95}
        halign: 'center'
        size_hint_y: None
        padding_y: 15

    MDLabel:
        text: "LOGIN"
        font_size: 35
        pos_hint: {"center_x": 0.5, "center_y": 0.70}
        halign: 'center'
        size_hint_y: None
        padding_y: 15


    MDRoundFlatButton:
        text: "Login"
        font_size: 25
        pos_hint: {"center_x": 0.5, "center_y": 0.15}
        size_hint_x: 0.7
        on_press: app.login(email.text,password.text)

    <Signup>:
```

```
MDLabel:
    text: "SIGN UP"
    font_size: 35
    pos_hint: {"center_x": 0.5, "center_y": 0.95}
    halign: 'center'
    size_hint_y: None
    padding_y: 15


MDTextField:
    id: email
    hint_text: "Email"
    size_hint_x: 0.9
    font_size: 25
    pos_hint: {"center_x": 0.5, "center_y": 0.8}
    required: True


MDTextField:
    id: password
    hint_text: "Password"
    password: True
    size_hint_x: 0.9
    required: True
    font_size: 25
    pos_hint: {"center_x": 0.5, "center_y": 0.65}

MDTextField:
    id: password1
    hint_text: "Re-Enter password"
    password: True
    size_hint_x: 0.9
    required: True
    font_size: 25
    pos_hint: {"center_x": 0.5, "center_y": 0.5}

MDLabel:
    id: err
    text: ""
    size_hint_x: 0.9
    required: True
    font_size: 25
    color: 1,0,0,1
    pos_hint: {"center_x": 0.83, "center_y": 0.4}

MDRoundFlatButton:
    text: "Sign up"
    font_size: 25
    pos_hint: {"center_x": 0.5, "center_y": 0.3}
    size_hint_x: 0.7
```

```
        on_press: app.createaccount(email.text,password.text,password1.text)

    MDRoundFlatButton:
        text: "Go Back"
        font_size: 25
        pos_hint: {"center_x": 0.5, "center_y": 0.08}
        size_hint_x: 0.8
        on_press: app.goback()

<Result>:
    MDLabel:
        text: "Result"
        font_size: 40
        pos_hint: {"center_x": 0.5, "center_y": 0.95}
        halign: 'center'
        size_hint_y: None
        padding_y: 15


    MDLabel:
        id: result
        text: "Plagiarism is :"
        font_size: 40
        pos_hint: {"center_x": 0.5, "center_y": 0.7}
        halign: 'center'
        size_hint_y: None

class MainApp(MDApp):
    def build(self):
        self.theme_cls.theme_style = "Light"
        self.theme_cls.primary_palette = "BlueGray"


        sm.add_widget(Login(name='Login'))
        sm.add_widget(Signup(name='Signup'))
        sm.add_widget(Dashboard(name='Dashboard'))
        sm.add_widget(Result(name='Result'))
        sm.current = 'Login'
        return sm

    def signup(self):
        sm.current='Signup'

    def login(self,email,password):
        s = f"Email: {email}\nPassword: {password}\n"

        if email=="admin@gmail.com" and password=="admin":
            print("successful\n"+s)
            sm.current='Dashboard'
            self.file_manager_open()
```

```
        else:
            print("failed\n"+s)
            sm.get_screen('Login').ids.err.text="Wrong email / password"

         def file_manager_open(self):
        Window.bind(on_dropfile=self._on_file_drop)

    def _on_file_drop(self, window, file_path):
        global c
        global l
        print(file_path)
        if c<2:
            c+=1
            l.append(str(file_path))
        if c==2:
            x=l[0][2:-1].split('\\')
            x=x[::2]
            file_path1="/".join(x)
            with open(file_path1) as f1:
                data1 = f1.read()

            y=l[1][2:-1].split('\\')
            y=y[::2]
            file_path2="/".join(y)
            with open(file_path2) as f2:
                data2 = f2.read()


            similarity_ratio = SequenceMatcher(None,data1,data2).ratio()
            print(similarity_ratio*100 )
            c=0
            l=[]
            sm.current='Result'




  sm.get_screen('Result').ids.result.text="Plagiarism is : "+ str(similarity_ratio*100) +" %"
            print("success")
            sm.current='Result'
        return

MainApp().run()
```

6. TESTING

6.1 Testing Methodologies

Testing is a vital part of the development process to ensure that the driver drowsiness detection system performs accurately and reliably under various conditions. The system will undergo rigorous testing to validate its core functionalities, performance, and overall stability.

Test Types:

- Precision Testing: Verify the accuracy of plagiarism detection.
- Recall Testing: Measure the ability to detect all instances of plagiarism.
- False Positive Testing: Identify incorrect plagiarism flags.
- False Negative Testing: Detect missed plagiarism instances.
- Robustness Testing: Evaluate performance with varying input formats.

Test Data:

- Text Corpora: Large collections of text data (e.g., articles, books).
- Plagiarized Text: Intentionally plagiarized text samples.
- Original Text: Unique, unpublished text samples.
- Mixed Text: Combination of original and plagiarized text.

Test Scenarios:

- Exact Matching: Test detection of identical text.
- Paraphrasing: Evaluate detection of rewritten text.
- Word Replacement: Test detection of text with replaced words.
- Sentence Rearrangement: Evaluate detection of rearranged sentences.
- Multi-Language Support: Test detection across languages.

Evaluation Metrics:

- Precision: True Positives / (True Positives + False Positives)
- Recall: True Positives / (True Positives + False Negatives)
- F1-Score: Harmonic mean of Precision and Recall
- Accuracy: (True Positives + True Negatives) / Total Samples
- Mean Average Precision (MAP): Average precision at different recall levels

6.2 TEST CASES

S. No	Test Case	Test scenario	Test steps	Test Data	Expected Results	Actual Results	Status
1	Numbers	Giving Numbers	1.Enter number in text box1 2.Enter Number in text box 2 3.Analyse	200433222	Similarity: 0.00%	Similarity: 0.00%	Pass
2.	Characters	Giving Charecters	1.Enter Character in text box1 2.Enter Characters in text box 2 3.Analyse	%^$#(*	Similarity: 0.00%	Similarity: 0.00%	Pass
3.	Text	Giving Text	1.Enter Text in text box1 2.Enter Text in text box 2 3.Analyse	Text1: This is team E4 Text2: We are from team E4	Similarity: 45.00%	Similarity: 40.00%	Fail
4.	Text	Giving Text	1.Enter Text in text box1 2.Enter Text in text box 2 3.Analyse	Text1: Learn Concpet Text2: Concpet should be Learned	Similarity: 20.00%	Similarity: 25.00%	Pass
5.	Text	Giving Text	1.Enter Text in text box1 2.Enter Text in text box 2 3.Analyse	Text1: National Earth Day is celebrated on April 22 Text2:Every year on 22nd April National Earth day is celebrated	Similarity: 75.00%	Similarity: 0.00%	Fail

7. EXPERIMENTAL RESUITS

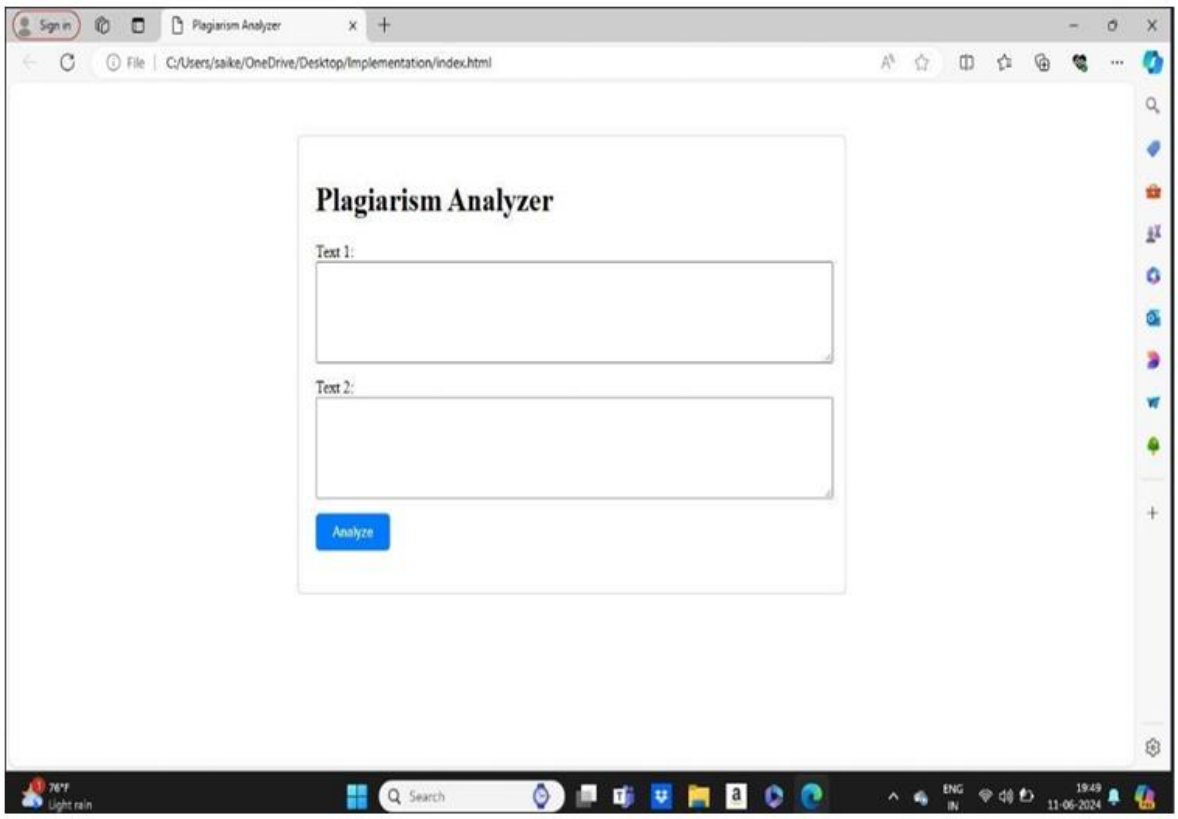

Fig1.output screen

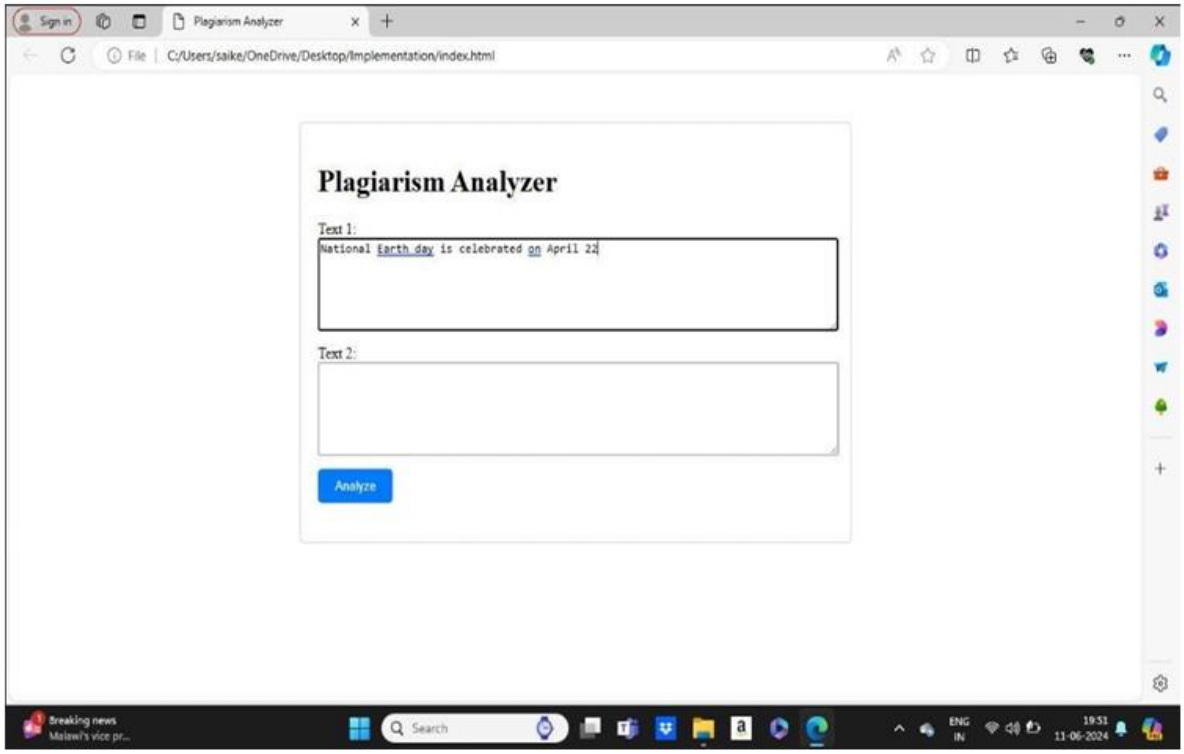

Fig2. Entering text 1

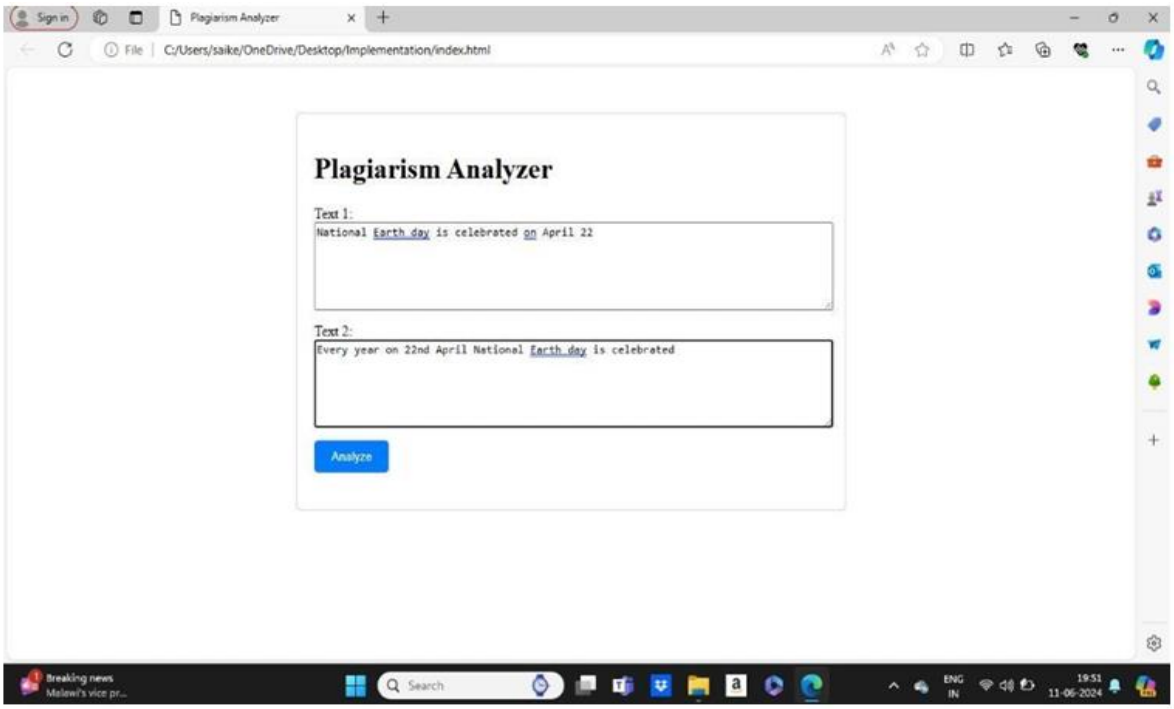

Fig3. Entering text 2

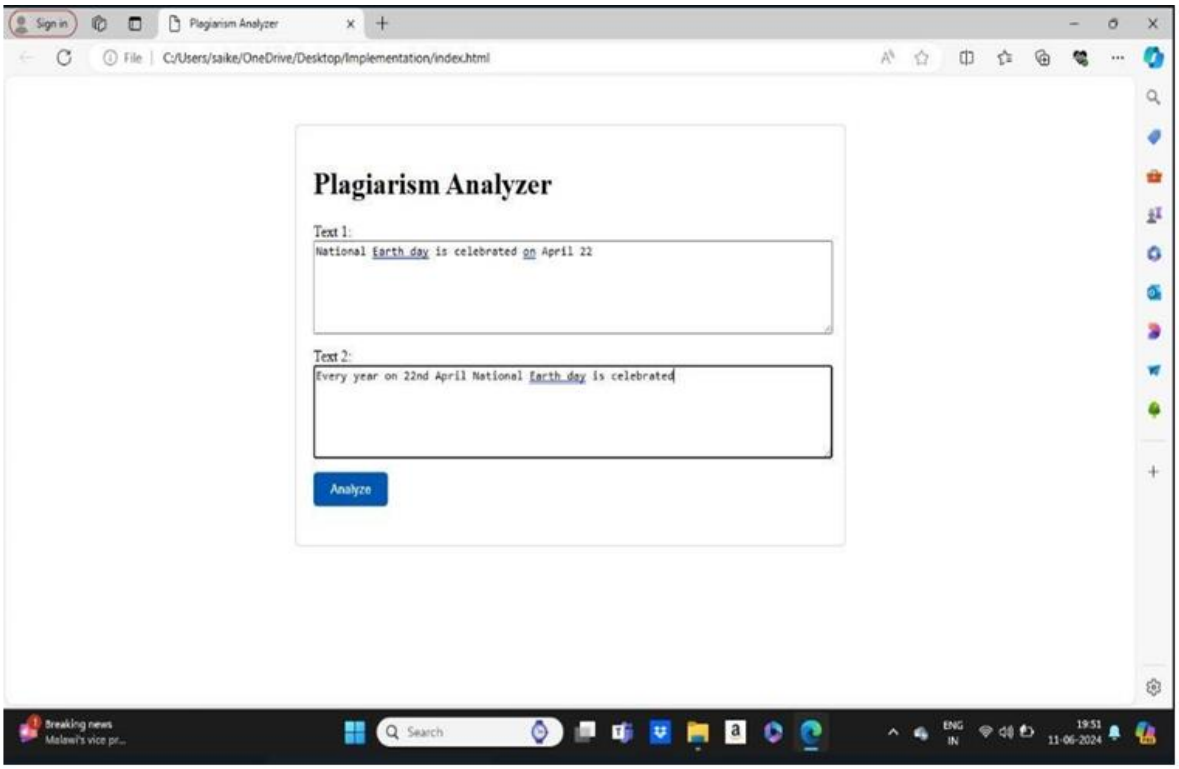

Fig4. Comparision and analysing

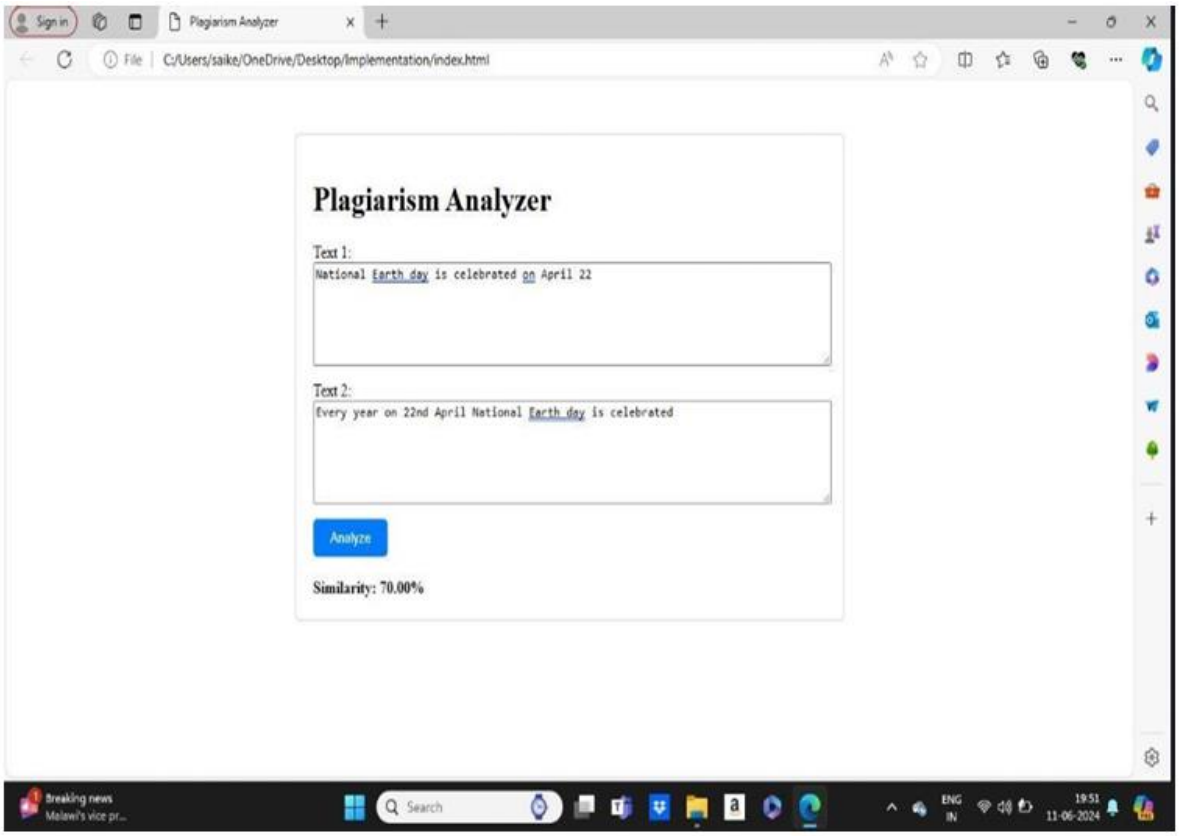

fig5.Displaying the Result

8. CONCLUSION

The purpose of the project is to analyze the similarity between two input texts and determine the percentage of overlap or similarity. It achieves this by tokenizing the input texts into individual words and comparing them to identify common words. The similarity percentage is then calculated based on the number of common words relative to the total number of words in the longer text. The user interface is straightforward and minimalistic, consisting of two text areas for inputting texts, an "Analyze" button to trigger the analysis, and a result area to display the calculated similarity percentage.

The interface design is responsive, with the container having a maximum width of 600 pixels and being centered horizontally on the page for better presentation. The project serves as a valuable educational resource for understanding fundamental concepts of web development, including HTML, CSS, and JavaScript integration. It also introduces basic principles of text analysis and similarity calculation, laying the groundwork for further exploration in natural language processing (NLP) and related fields.

In conclusion, while the Plagiarism Analyzer project represents a simple implementation of plagiarism detection, it provides a solid foundation for future development and innovation in the realm of text analysis and plagiarism detection tools. By addressing its limitations and exploring future scope areas, the project can evolve into a more comprehensive and powerful tool for assisting users in identifying and preventing plagiarism.

9. FUTURE SCOPE

Semantic Analysis: Implement algorithms for semantic analysis to better understand the meaning of text beyond simple word matching. Techniques like word embeddings (Word2Vec, GloVe) or deep learning models (BERT, GPT) can capture the semantic similarity between texts more accurately.

There are numerous avenues for future enhancement and expansion of the project, including implementing advanced algorithms for semantic analysis, enhancing tokenization techniques, integrating with databases for user authentication and result storage, optimizing performance, and providing interactive visualizations and customization options.

Advanced Tokenization: Enhance tokenization to handle punctuation, special characters, and different languages effectively. Utilize libraries like NLTK (Natural Language Toolkit) or spaCy for robust tokenization.

Detection of Paraphrasing: Develop algorithms specifically designed to detect paraphrasing and rephrasing of text, as these can often evade simple word-matching approaches.

Performance Optimization: Optimize the code and algorithms for better performance, especially when dealing with large volumes of text. Consider parallel processing or distributed computing techniques to handle scalability.

10. BIBLIOGRAPHY

[1] S. Grier, “Tool that Detects Plagiarism in PASCAL Programs”, SIGSCE Bulletin, Vol. 13, 1981.

[2] B. Baker, ”A theory of parameterized pattern matching: algorithms and applications”, in Proc. of the 25th ACM Symp. on the Theory of Computing, 1993, pp. 71-80.

[3] Paul Clough, “Plagiarism in natural and programming languages: an overview of current tools and technologies”, Department of Computer Science, University of Sheffield, 2000. Web: http://www.dcs.shef.ac.uk/~cloughie/papers/Plagiarism.pdf

[4] T. Cholakov, D. Birov, “Duplicate code detection algorithm”, CompSysTech '15: Proc. of the 16th Int. Conf. on Computer Systems and Technologies, June 2015, pp. 104–111.

[5] Hyo-Sub Lee, Kyung-Goo Doh, “Tree-pattern-based duplicate code detection”, DSMM '09: Proc. of the ACM first international workshop on Data-intensive software management and mining, November, 2009, pp. 7–12.

[6] Jeong-Hoon Ji, Gyun Woo, Hwan-Gue Cho, “A source code linearization technique for detecting plagiarized programs”, ITiCSE '07: Proc. of the 12th annual SIGCSE conference on Innovation and technology in computer science education”, June 2007, pp. 73–77.

[7] I. Andrianov, A. Grigorieva, “Effective search for plagiarism in program code for a remote programming workshop system”, in Proc. of the international scientific and practical conference "Informatization of engineering education (INFORINO-2016)”, Moscow: MEI Publishing House, 2016, pp. 485–488.

[8] S.F. Svinyin, I.A. Andrianov, “Application of evenly sparse suffix tree for string processing tasks”, SPIIRAS Proceedings, No. 3 (34), 2014. pp. 247-260.

[9] Dan Gusfield, “Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology”, Cambridge University Press, 1997, 534 p.

[10] A. Anderson, S. Nilsson, “Efficient implementations of suffix trees”, Software – Practice and Experience, 1995, Vol. 25, pp. 129-141.

[11] Juha Kärkkäinen, “Sparse suffix trees”, Proc. of the 2nd Annual International Conf. on Comput. and Combinatorics. Lecture Notes in Computer Science, 1996, Vol. 1090, pp. 219-230.

[12] Thanh Tri Le Nguyen, Angela Carbone, Judy Sheard, Margot Schuhmacher, "Integrating source code plagiarism into a virtual learning environment: benefits for students and staff", ACE '13: Proceedings of the Fifteenth Australasian Computing Education Conference, Volume 136, January 2013, pp. 155–164

[13] Lutz Prechelt, Guido Malpohl, Michael Philippsen, “Finding Plagiarisms among a Set of Programs with JPlag”, Journal of Universal Computer Science, vol. 8, no. 11 (2002), pp. 1016 1038.

[14] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken, “Winnowing: Local Algorithms for Document Fingerprinting”, SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 2003, pp. 76–85.

[15] I.A. Andrianov, N.O. Menuhova, “Refinement of a remote workshop of Vologda SU for checking problems on informatics according to the rules of school olympiads”, Proc. of Int. scientific and practical conference “Modern society, education and science”, Tambov, 2014, pp. 10-13.

[16] I.A. Andrianov, “Automation of checking of logic circuits”, Proc. of tenth international scientific and technical conference “Intellectual Information Technologies and Intellectual Business (INFOS-2019)”, Vologda: Vologda State University, 2019, pp. 114-117.