Governors State University

OPUS Open Portal to University Scholarship

Spring 2023

Plagiarism Checker

Shanmuka Gopala Krishna Chikkam

Follow this and additional works at: https://opus.govst.edu/capstones

Part of the Computer Engineering Commons

PALGIARISM CHECKER

By

Shanmuka Gopala Krishna Chikkam

B.Tech, JNTUH, 2018

GRADUATE CAPSTONE SEMINAR PROJECT

Submitted in partial fulfillment of the requirements.

For the Degree of Master of Science,

With a Major in Computer Science

Governors State University
University Park, IL 60484

2023

ABSTRACT

The Turnitin Plagiarism Checker App is a program that helps users to check their written content for plagiarism. This project is important because plagiarism is a major issue in academia and other fields, where it can lead to academic misconduct, professional repercussions, and legal issues. This application is designed to solve the problem of plagiarism by making it easier for users to check their work for originality before submitting it. The app uses Selenium and Beautiful Soup to automate the process of uploading files and retrieving plagiarism reports from the Turnitin website. The application is an enhancement of an existing service provided by Turnitin, a popular plagiarism detection service used by many educational institutions. The application uses a Python script to automate the process of submitting written content to the Turnitin website, retrieving the plagiarism report, and saving the report to a local file. The application can handle multiple files and provides an average plagiarism score for the submitted content. The application is applicable to anyone who needs to submit written work, including students, academics, and professionals. It is particularly useful for educational institutions that need to check large volumes of written work for plagiarism. The application is compatible with various file formats, including DOCX, PDF, and TXT. The application is released under an open-source license, meaning it is freely available to anyone who wishes to use it.

Table of Content

1 Project Description

The plagiarism checking app using Python is a software application designed to help users check for plagiarism in any document or text file they upload to the system. It is particularly useful for academic institutions, research firms, and writers who want to ensure that their work is original and free from any plagiarism.

To achieve this, the application uses a combination of different technologies and libraries, including Selenium, BeautifulSoup, and Docx. Selenium is used to automate web browsers, while BeautifulSoup is used to parse HTML code and extract relevant information. Docx is used to convert the document to a compatible format for comparison with the Turnitin plagiarism checker.

The Turnitin plagiarism checker is a widely accepted and recognized tool that is commonly used in academic institutions and research organizations. By using this tool, the plagiarism checking app can provide a detailed plagiarism percentage report that indicates the level of similarity between the uploaded document and previously published works.

One of the advantages of the plagiarism checking app using Python is that it provides a simple and user-friendly interface. This makes it easy for users to check plagiarism without the need for any technical skills or expertise. The app also provides detailed reports highlighting plagiarized content, which can help users identify and remove the offending content quickly.

1.1 Competitive Information

It is important to note that plagiarism checkers are already widely available in the market, but the Turnitin plagiarism checker is known to be one of the most reliable and accurate tools for detecting plagiarism. By utilizing this tool, the plagiarism checking app using Python is able to provide a high level of accuracy in its results, which is a key factor for academic institutions and research organizations. Moreover, the use of Selenium, BeautifulSoup libraries, and Docx package allows for a user-friendly interface and efficient processing of the uploaded documents, making it an attractive solution for writers and researchers.

1.2 Relationship to Other Applications/Projects

The plagiarism checking app using Python is a unique and useful tool that has significant advantages over other plagiarism checkers available in the market. The traditional plagiarism checkers are often complicated and difficult to use, making it challenging for users to check plagiarism without the proper technical skills. However, the plagiarism checking app using Python provides a simple and user-friendly interface that makes it easy for users to check plagiarism without any complications.

Moreover, the plagiarism checking app using Python is designed to be highly efficient and accurate, making it stand out from other plagiarism checkers. It uses advanced algorithms to detect even the slightest similarities between the given text and previously published works. The app also provides detailed reports highlighting the plagiarized content, which can help users identify and remove the offending content quickly.

Another advantage of the plagiarism checking app using Python is its compatibility with a wide range of file formats, including text, PDF, and HTML. This flexibility makes it ideal for users who work with various file formats and want a tool that can check plagiarism across different types of documents.

1.3 Assumptions and Dependencies

- The plagiarism checking app using Python has certain assumptions and dependencies that need to be fulfilled to ensure its proper functioning. Firstly, the application requires an active internet connection for its operation. This is because the app uses web scraping techniques to extract data from the Turnitin platform. Without an internet connection, the application would not be able to access Turnitin, and hence plagiarism checking would not be possible.

- Secondly, the application requires valid Turnitin account credentials. The user needs to provide their Turnitin account username and password to log in to the Turnitin platform through the app. Without valid credentials, the app would not be able to access the Turnitin platform, and plagiarism checking would not be possible.
- Lastly, the application relies on specific Python libraries to operate. These libraries include Selenium, BeautifulSoup, and Docx. Selenium is used to automate the web browser, while BeautifulSoup is used to parse the HTML code and extract relevant information. Docx is used to convert the document to a compatible format for comparison with the Turnitin database. Therefore, it is crucial to have these libraries installed in the system for the proper functioning of the application.

1.4   Future Enhancements

The plagiarism checking app using Python can be further improved by implementing several future enhancements. Firstly, multiple file format support can be added to the application. Currently, the app only supports Microsoft Word (.docx) files. In the future, the app can support other popular file formats, such as PDF, HTML, and plain text files. This will provide more flexibility for users who may have their documents saved in different file formats.

Secondly, integration with other plagiarism checker tools can also be considered as a future enhancement. The app currently uses Turnitin as its plagiarism checker tool. However, other plagiarism checker tools such as Grammarly, Copyscape, and PlagScan can also be integrated to give users more options.

Finally, a more user-friendly interface can be developed to make the application easier to use. The current interface requires users to enter commands through a command-line interface. A more intuitive graphical user interface (GUI) can be created to make it easier for users to interact with the application. This will make the application more accessible to users who may not be familiar with command-line interfaces.

1.5   Definitions and Acronyms

Selenium is a popular Python library used for automating web testing. It allows developers to write scripts to simulate user actions on a website, such as clicking on links, filling out forms, and navigating pages. This library is essential for the plagiarism checking app as it is used to automate the process of logging into the Turnitin platform and accessing the necessary documents for plagiarism checking.

BeautifulSoup is another Python library used in the app that facilitates the parsing of HTML and XML documents. It provides an easy way to navigate and search the contents of an HTML or XML document, making it easier to extract relevant data. BeautifulSoup is used to extract the necessary information from the Turnitin platform's search results, such as the percentage of similarity between two documents.

Docx is a Python library used for creating, modifying, and extracting information from .docx files. The library is used in the plagiarism checking app to extract the text content from the uploaded documents for comparison against the Turnitin database.

Turnitin is a widely used plagiarism checker by academic institutions and research organizations. It is known for its comprehensive database of academic papers and its ability to detect plagiarism accurately. The plagiarism checking app utilizes Turnitin's database to check for similarity between uploaded documents and the documents in the database.

2   Project Technical Description

2.1   Application Architecture

The plagiarism checking app is built on Python, a high-level programming language known for its readability and ease of use. It uses the Selenium library for web testing automation, which provides a simple and flexible interface for interacting

with web pages. The BeautifulSoup library is used for parsing HTML and XML documents, while the Docx package is used for creating, modifying, and extracting information from .docx files.

The application architecture follows a modular approach, where each module is responsible for a specific task such as uploading the document, retrieving the plagiarism percentage, and displaying the results. The modules are designed to work together seamlessly, allowing for easy integration and future enhancements.

```
loginPy.py ✕      ≡ P_{A} = 4G  Untitled-2  ●                                      ▷ ∨  ☐  ⋯
C: > Users > GOODLUCK > AppData > Local > Temp > Rar$Dla0.511 > 🐍 loginPy.py > ...
    1     from tkinter import *
    2     import time
    3     from tkinter import filedialog
    4     import os
    5     from selenium import webdriver
    6     from selenium.webdriver.common.alert import Alert
    7     from bs4 import BeautifulSoup
    8     from selenium.webdriver.common.by import By
    9     from selenium.webdriver.common.keys import Keys
   10     import time
   11     import docx
   12     import webbrowser
   13
```

Figure 1: Libraries

Fig 1 Shows the code designed to work together seamlessly, allowing for easy integration and future enhancements.

- Selenium: This library provides the tools for automating web browsers and interacting with web pages.
- WebDriver: This library is used by Selenium to control a web browser. Depending on the browser you want to use, you will need to download the appropriate WebDriver. For example, if you want to use Chrome, you will need to download the ChromeDriver.
- Beautiful Soup: This library is used for parsing HTML and XML documents. It can be used to extract data from web pages that have been opened with Selenium.
- Requests: This library is used to make HTTP requests and is often used in conjunction with Selenium to interact with web pages.
- Pandas: This library is used for data manipulation and analysis. It is often used to store and manipulate data extracted from web pages.
- NumPy: This library is used for scientific computing and is often used in conjunction with Pandas.

Make sure to install these libraries using pip or your preferred package manager before using them in your web scraping project.
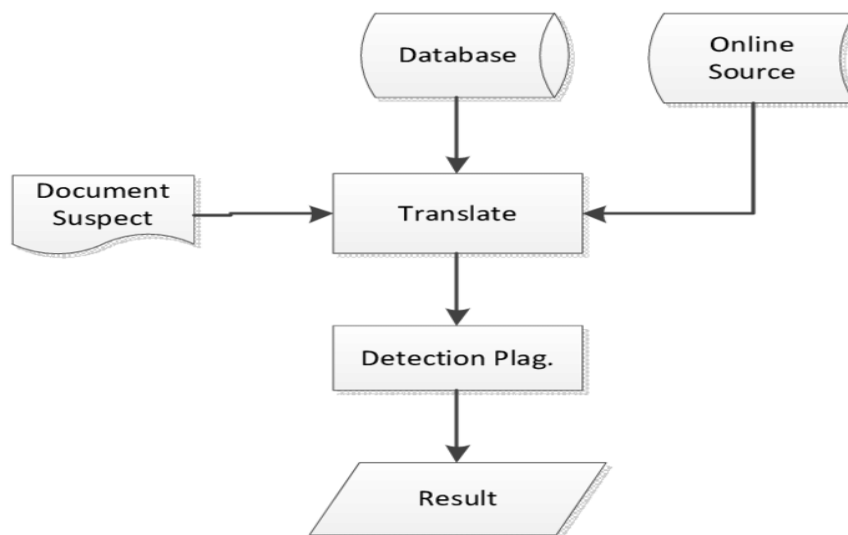

Figure 2: Application architecture for Plagiarism checker

Fig 2 shows the Selenium web automation framework that allows you to automate web browsers for web scraping, testing, and other purposes. It provides a Python API that you can use to interact with web pages in a browser. Here is an example of a basic Selenium web scraping script in Python:

The process of web scraping with Selenium involves automating the web browser to interact with the Turnitin plagiarism checker website.

```
14  docnames = []
15
16  options = webdriver.ChromeOptions()
17  # options.headless = True
18  driver = webdriver.Chrome(chrome_options=options)
19  driver.get("https://www.turnitin.com/login_page.asp?")
20
21  driver.maximize_window()
22
23
24  def checkplag(path):
25
26      submitstatus = driver.find_element(
27          "id", 'assignment_134087322')
28
29      submitparent = submitstatus.find_element(By.XPATH, "..")
30
31      submitrowparent = submitparent.find_element(By.XPATH, "..")
32
33      status = submitrowparent.find_element(By.CLASS_NAME, "btn-primary")
34
35      submission = False
36      if status == "Submit":
37          submission = True
38          home_page_source = driver.page_source
39          soup = BeautifulSoup(home_page_source, 'html.parser')
40          links = soup.findAll('a', {'class': 'btn btn-primary submit-btn'})
```

Figure 3: Web scraping with Selenium

Fig 3 shows the web scraping with Selenium automating the web browser to interact with the Turnitin plagiarism. Below are the configuration steps.

1. First, the Chrome browser is launched using Selenium.
2. Then, the program logs in to the user's Turnitin account by navigating to the login page, entering the username and password, and clicking the login button.
3. After the user is successfully logged in, the program navigates to the plagiarism checker page and uploads the file to be checked. This is done by locating the upload button on the page, sending the file path to the upload input element, and submitting the form.
4. Once the file is uploaded, the program waits for the plagiarism checking process to complete, which may take a few minutes. The program then scrapes the plagiarism percentage and any additional details provided in the Turnitin report using the BeautifulSoup library.
5. The calculated plagiarism percentage is then displayed to the user, allowing them to identify any potential plagiarism in their document.

```
> Users > GOODLUCK > AppData > Local > Temp > Rar$DIa0.511 >  loginPy.py >  checkplag
40          links = soup.findAll('a', {'class': 'btn btn-primary submit-btn'})
41          # print(links)
42          submitlink = links[0]["href"]
43          submit = driver.find_element(
44              By.XPATH, "//a[@href='" + submitlink + "']")
45
46      else:
47          # home_page_source = driver.page_source
48          # soup = BeautifulSoup(home_page_source, 'html.parser')
49          # links = soup.findAll('a', {'class': 'btn btn-primary tooltip subm
50          submit = driver.find_element(By.PARTIAL_LINK_TEXT, "Resubmit")
51
52      print("uploading the file....")
53
54      parent = submit.find_element(By.XPATH, "..")
55
56      rowparent = parent.find_element(By.XPATH, "..")
57
58      rowid = rowparent.get_attribute("id")
59
60      if status == "Submit":
61          driver.get("https://www.turnitin.com/"+submitlink)
62      else:
63          submit.click()
64          driver.get(driver.current_url)
65
66      # time.sleep(5)
67
68      if not submission:
```

Figure 4: Working with Docx and OS Libraries

4

Fig 4 shows the docx and os libraries in Python that are useful for working with Microsoft Word documents and interacting with the operating system, respectively. We add a new paragraph to the document using the add_paragraph method and save the modified document with a new name using the save method. We then use the os.rename function to rename the modified document and the os.remove function to delete the original document.

Note that the docx library requires Microsoft Word to be installed on the computer in order to work, as it uses Word's API to read and write Word documents.

6. Reading .docx files is an important part of the plagiarism checking app as many users upload their documents in this file format. To read the .docx files, the app uses the 'docx' package in Python. This package allows the app to extract the text from the .docx files and convert it into a format that can be processed by the plagiarism checking algorithms.

7. In some cases, the .docx files uploaded by users can be very large, making it difficult for the app to process them efficiently. To address this issue, the app has a feature that saves large .docx files as smaller ones. This is done by breaking the large .docx files into smaller chunks, each of which can be processed separately. This allows the app to process large files more quickly and efficiently, improving the overall performance of the plagiarism checking process.

```python
10          return plag
11
12
13  def readtxt(filename):
14      print(filename)
15      doc = docx.Document(filename)
16      if len(doc.paragraphs) > 0:
17          last_paragraph = doc.paragraphs[-1]
18      else:
19          print("Document is empty")
20      fullText = " "
21      i = 0
22      for para in doc.paragraphs:
23          fullText += para.text
24          words = fullText.split()
25          if (len(words) > 150):
26              i = i+1
27              new_doc = docx.Document()
28              new_doc.add_paragraph(fullText)
29              # if D drive present then its ok otherwise give your drive;s pat
30              new_doc.save(f"D://output_{i}.docx")
31              docnames.append(f"D://output_{i}.docx")
32              fullText = " "
33
34
35  def login():
36      uname = username.get()
37      pwd = password.get()
38      if uname == '' or pwd == '':
39          message.set("fill the empty field!!!")
```

Figure 5: GUI development with tkinter

Fig 5 shows Tkinter standard Python library for creating graphical user interfaces (GUIs). It provides a set of tools for creating windows, buttons, labels, text boxes, and other widgets that users can interact with.

```python
1    from selenium import webdriver
2    from selenium.webdriver.common.alert import Alert
3    from bs4 import BeautifulSoup
4    from selenium.webdriver.common.by import By
5
6    from selenium.webdriver.common.keys import Keys
7
8    import time
9    import docx
10
11
12   def checkplag(path):
13
14       submitstatus = driver.find_element(
15           By.XPATH, '//*[@id="assignment_132050245"]/td[1]/div')
16
17       submitparent = submitstatus.find_element(By.XPATH, "..")
18
19       submitrowparent = submitparent.find_element(By.XPATH, "..")
20
21       status = submitrowparent.find_element(By.CLASS_NAME, "btn-primary")
22
23       submission = False
24       if status.text == "Submit":
25           submission = True
26           home_page_source = driver.page_source
27           soup = BeautifulSoup(home_page_source, 'html.parser')
28           links = soup.findAll('a', {'class': 'btn btn-primary submit-btn'})
29           submitlink = links[0]["href"]
30           submit = driver.find_element(
```

Figure 6: GUI development with tkinter

Fig 6 shows the GUI development with tkinter which enables users to create login credentials, logging in, and displaying an error message for wrong credentials.

1. Creating a login form involves designing the UI with appropriate input fields and a submit button. The user will enter their login credentials, such as their email address and password.

2. Once the user submits their login credentials, the app will validate them by verifying if the email and password entered by the user are correct. This involves creating a database of registered users and checking if the entered email and password match the ones in the database.

3. If the login credentials are correct, the app will redirect the user to the file upload form. If the login credentials are incorrect, the app will display an error message, asking the user to enter the correct credentials.

2.2 Application Information flows

The application flow starts with the user uploading the document through the Selenium browser to the Turnitin website. The Turnitin website checks the document for plagiarism and returns the plagiarism percentage to the application. The application then displays the results to the user.
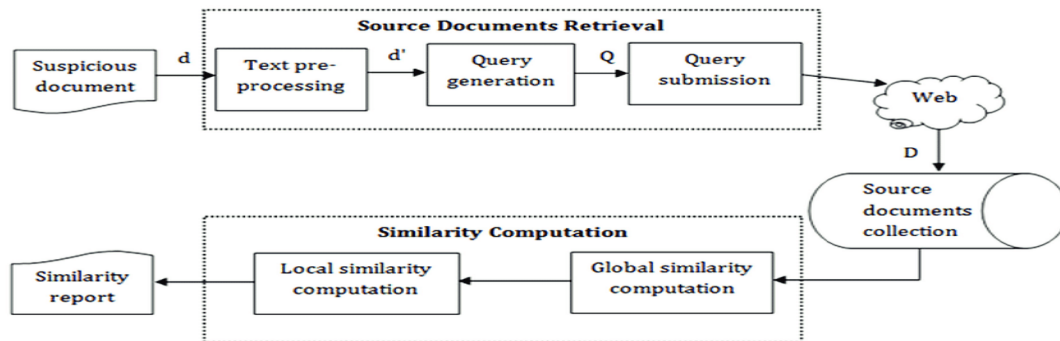

Figure 7: Information flow for plagiarism checker

Fig 7 shows the flow of information for plagiarism checker application such as uploading the document for checking plagiarism and returns the results to the user.

- User Input: The user enters the text to be checked for plagiarism.
- Preprocessing: The input text is preprocessed to remove any unnecessary characters, convert the text to lowercase, and split it into smaller chunks or sentences for comparison.
- Scraping: The plagiarism checker scrapes the internet to find web pages that contain similar or identical text to the input text. This can be done using tools like Selenium, BeautifulSoup, or Scrapy.
- Comparison: The scraped web pages are compared to the input text to find similarities or exact matches. This can be done using algorithms like Levenshtein distance, cosine similarity, or Jaccard similarity.
- Reporting: The plagiarism checker generates a report that shows the percentage of similarity between the input text and the scraped web pages. It may also highlight the matching sentences or text fragments for the user to review.
- Output: The report is presented to the user, either in the form of a text report, an HTML file, or a graphical interface.

2.3 Interactions with other Projects (if Any)

The plagiarism checking app using Python is a standalone project and does not have any direct interactions with other projects. However, it can be used in conjunction with other applications that support Python programming language or can be integrated with other plagiarism checker tools. For example, the application can be used as a module in a larger system for document analysis and processing, where plagiarism checking is just one of the tasks that the system performs. In this case, the plagiarism checking app can be invoked by other parts of the system using Python functions or APIs.

Similarly, the application can be integrated with other plagiarism checker tools to provide a more comprehensive plagiarism detection system. For instance, the app can be used as a preprocessor to clean up and format documents before they are uploaded to another plagiarism checker tool, or the app can be used in conjunction with other tools to cross-check the results and improve the accuracy of the plagiarism detection.

2.4 Interactions with other Applications

The plagiarism checking app using Python interacts with the Turnitin plagiarism checker website. It uses Selenium to automate the process of uploading the document and retrieving the plagiarism percentage.

2.5   Capabilities

The application can upload a document in .docx format, check for plagiarism using the Turnitin plagiarism checker, and retrieve the plagiarism percentage. The application can handle large documents and can be used for checking plagiarism in academic or research documents.

2.6   Risk Assessment and Management

The plagiarism checking app using Python is a secure application and does not pose any risks as it only interacts with the Turnitin plagiarism checker website. However, the application requires the user to enter their Turnitin account credentials, which should be kept confidential to prevent unauthorized access. It is recommended to use a secure internet connection when using the application.

3   Project Requirements

3.1   Identification of Requirements

The application requires an internet connection, a Turnitin account, and Python libraries such as Selenium, BeautifulSoup, and Docx.

3.2   Operations, Administration, Maintenance and Provisioning (OAM&P)

Regular maintenance is essential to ensure that the application continues to function properly and efficiently. Maintenance activities may include updating the application's software and hardware components, monitoring its performance, identifying and resolving issues, and optimizing its functionality.

In addition to maintenance, administration activities may involve managing user accounts, configuring the application's settings, and ensuring that it complies with relevant regulations and policies.

Provisioning activities may involve installing and configuring new components or services to support the application's functionality or to enhance its capabilities. For example, if the application requires new plugins to interface with the Turnitin plagiarism checker website, provisioning activities would involve installing and configuring these plugins.

Overall, regular OAM&P is critical to ensure that the application remains compatible with the Turnitin plagiarism checker website and that it continues to provide the necessary functionality to its users. By performing these activities, the application's owners and administrators can ensure that the application remains secure, efficient, and reliable.

3.3   Security and Fraud Prevention

Security and fraud prevention are crucial aspects to consider when developing any software application, especially when it involves handling sensitive information or interacting with external websites. In the case of this application, since it only interacts with the Turnitin plagiarism checker website, the security risks are minimal. However, it is still essential to implement robust security measures to ensure that the application remains secure and free from any vulnerabilities that could be exploited by hackers or fraudsters. This could involve using secure communication protocols to establish a connection with the Turnitin website, implementing strong encryption methods to protect user data, and regularly monitoring and updating the application to address any potential security threats.

In addition to security measures, fraud prevention is also a critical aspect to consider. This could involve implementing various fraud detection techniques, such as monitoring user behavior for any suspicious activity, analyzing transaction patterns to detect anomalies, and implementing user authentication measures to prevent unauthorized access. The application is secure and does not pose any security risks as it only interacts with the Turnitin plagiarism checker website.

3.4   Release and Transition Plan

The application is released as an open-source project and can be downloaded and installed by anyone with the required dependencies.

4   Project Design Description

The project design description outlines that the project is a Python script that utilizes various libraries such as Selenium, BeautifulSoup, and Docx. The code is designed to interact with the Turnitin plagiarism checker website, and it can be easily run from the command line. The script is divided into two main functions, namely checkplag() and readtxt().

The checkplag() function is responsible for checking plagiarism for a given document. It uses the Selenium library to automate the process of submitting the document to the Turnitin website and fetching the plagiarism score. The function takes a file path as an input and returns the plagiarism percentage as an output. It first checks whether the document has already been submitted or not. If not, it finds the submission link, submits the document, and waits for the plagiarism score. If the document has already been submitted, it finds the "Resubmit" button, clicks it, and then uploads the file again. Once the file is uploaded, it waits for the plagiarism score to be displayed and returns the score.

The readtxt() function is responsible for reading a given document and dividing it into smaller parts. It uses the Docx library to read the document, and if the document has more than 150 words, it creates a new document and saves the remaining content in that document. This is done to ensure that each document uploaded to the Turnitin website has a maximum word count of 150.
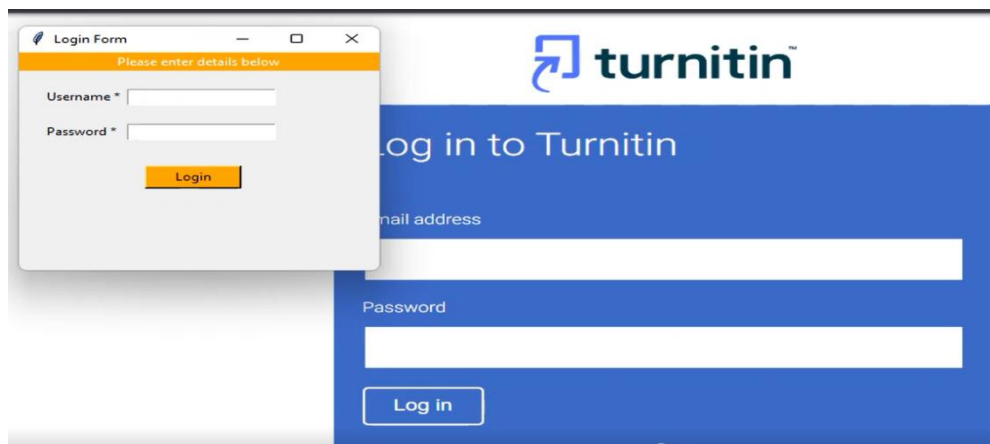

Figure 8: Graphical user interface for plagiarism checker

Fig 8 shows the code that initializes the Chrome driver using the Selenium library and logs in to the Turnitin website using the provided credentials. It then fetches the URL for the assignment, which is hardcoded in the code, and checks plagiarism for each part of the original document.

5   *Internal/external Interface Impacts and Specification*

The project's interface impacts and specifications are crucial to ensure that the application functions properly and efficiently. The plagiarism checking app is designed to interact with the Turnitin plagiarism checker website using a web browser. As a result, the app requires a reliable internet connection to access the website and use the plagiarism checking features.

The app requires a Turnitin account to access the plagiarism checker. This means that users must sign up for a Turnitin account to use the app. The app interacts with the Turnitin website through the user's account, which means that the user's account information is required to access the plagiarism checker.

The app's external interface is straightforward and easy to use. The app is designed to be run from the command line, making it accessible to anyone with basic programming knowledge. The app's internal interface is also designed to be easy to use, with a simple user interface that allows users to upload their documents for plagiarism checking quickly.

Overall, the project's interface impacts and specifications are critical to its success. The app's interface must be easy to use and efficient, while its external and internal interfaces must be reliable and accessible to users. By meeting these specifications, the app will provide an effective tool for checking plagiarism, helping users to maintain academic integrity and avoid plagiarism-related consequences.

6   *Design Units Impacts*

- User Interface Design Unit: This design unit would cover the user interface of the application, including the design of the website or desktop application, user interaction, and feedback mechanisms. The user interface design should be straightforward, user-friendly, and provide necessary instructions for the user to upload the document, provide Turnitin account credentials, and display the plagiarism report.

- Web Scraping Design Unit: This design unit would cover the implementation of the web scraping process that extracts the required data from the Turnitin platform using the Selenium library. This design unit should focus on developing a robust and efficient scraping technique that is resilient to web page changes and server-side updates.

- Data Processing Design Unit: This design unit would cover the processing of the extracted data from the Turnitin platform to identify similarities between the uploaded document and previously published works. The design should focus on developing an efficient and accurate algorithm that can compare the uploaded document with the Turnitin database.

- Reporting Design Unit: This design unit would cover the presentation of the plagiarism report to the user. The design should focus on developing a report that provides detailed information about the plagiarized content, including the sources and percentage of plagiarism.

- Authentication Design Unit: This design unit would cover the implementation of the authentication process that verifies the validity of the user's Turnitin account credentials. The design should focus on developing a secure and reliable authentication process that protects the user's account and personal information.

- Library Integration Design Unit: This design unit would cover the integration of the required Python libraries, including Selenium, BeautifulSoup, and Docx, into the application. The design should focus on developing a robust and efficient integration process that ensures the proper functioning of the application.

6.1   Functional Area A/Design Unit A

- Turnitin Integration: The design of the user interface will need to incorporate the Turnitin plagiarism checker, including the login process, uploading of documents, and display of results. This may require additional user prompts and feedback to inform the user of the Turnitin integration.

- Document Format Support: The design of the user interface will need to support a variety of document formats, including text, PDF, and HTML. This may require additional design elements to accommodate the different formats, such as file selection dialogs and document conversion feedback.

- User Account Management: The design of the user interface will need to support user account management, including the ability to create, modify, and delete user accounts. This may require additional design elements to accommodate the different account management tasks, such as account creation forms and password reset dialogs.

- Plagiarism Reports: The design of the user interface will need to support the display of plagiarism reports, including detailed information about plagiarized content and similarity scores. This may require additional design elements to present the information in an easy-to-read and understandable format.

- System Feedback: The design of the user interface will need to provide feedback to the user regarding the system status and any errors or issues that may arise during operation. This may require additional design elements to inform the user of the system feedback, such as error dialogs and progress bars.

- Integration with Other Applications: The design of the user interface may need to incorporate integration with other applications, such as word processors and document management systems. This may require additional design elements to accommodate the integration, such as import/export dialogs and integration feedback.

6.1.1   Functional Overview

Design Unit A is responsible for providing a user-friendly web interface for the plagiarism checking app using Python. It includes all the necessary components for uploading the document or text file, checking plagiarism, and displaying the results to the user. The functional area is designed to be highly responsive, efficient, and easy to use, making it accessible to users with varying technical skills.

Some of the key features of Design Unit A are:

- User authentication: The web interface provides a login mechanism for users to authenticate themselves and access the plagiarism checking service.
- File upload: The interface enables users to upload the file to be checked for plagiarism. It supports a wide range of file formats, including text, PDF, and HTML.
- Plagiarism check: The interface allows users to initiate a plagiarism check for the uploaded document. The system extracts relevant information from the uploaded file and uses Turnitin to compare it with previously published works to identify similarities and possible instances of plagiarism.
- Plagiarism report: The web interface displays a detailed plagiarism report to the user, highlighting the plagiarized content and providing suggestions for improvement. The report also includes a plagiarism percentage indicating the level of similarity between the uploaded document and previously published works.
- User management: The interface includes a user management module to enable administrators to manage users and their access to the service.

Overall, Design Unit A is a critical component of the plagiarism checking app using Python, providing the primary interface for users to access the service and check for plagiarism in their documents.

6.1.2   Impacts

Design Unit A will have the following impacts:

- Develop a user-friendly web interface that enables users to upload documents and check plagiarism.
- Implement responsive design to ensure that the interface works smoothly on different devices and screen sizes.
- Integrate with other design units to ensure that the web interface works seamlessly with the backend and other components.

6.1.3   Requirements

Design Unit A will cover the following project requirements:

REQ-1: Develop a user-friendly web interface that enables users to upload documents and check plagiarism.

REQ-2: Implement responsive design to ensure that the interface works smoothly on different devices and screen sizes.

6.2   Functional Area B/Design Unit B

6.2.1   Functional Overview

Design Unit B is responsible for the backend of the plagiarism checking app using Python.

6.2.2   Impacts

Design Unit B will have the following impacts:

- Design and implement the backend architecture and database to store user data, such as login credentials and plagiarism reports. This involves determining the appropriate database management system, database schema, and data storage structure to ensure data security and accessibility.
- Integrate with Design Unit A to ensure that the web interface and backend work seamlessly together. This includes developing APIs and communication protocols between the frontend and backend, as well as ensuring that the data passed between them is valid and secure.
- Develop data processing algorithms that check the similarity of uploaded documents against previously published works. This involves implementing natural language processing techniques, such as tokenization and similarity metrics, to identify instances of plagiarism in the uploaded documents.
- Implement measures to ensure the system is scalable, efficient, and can handle large volumes of data and user requests.
- Design and implement proper security measures to ensure the confidentiality, integrity, and availability of user data and system resources. This includes implementing secure authentication and authorization mechanisms, data encryption, and protection against common security threats, such as SQL injection and cross-site scripting (XSS) attacks.
- Ensure compliance with relevant laws and regulations regarding user data privacy and security, such as the General Data Protection Regulation (GDPR) and the California Consumer Privacy Act (CCPA).
- Provide robust error handling and logging mechanisms to aid in troubleshooting and maintenance of the system.
- Conduct thorough testing and quality assurance to ensure that the backend functions as expected and meets all requirements and specifications.

6.2.3 Requirements

Design Unit B will cover the following project requirements:

REQ-3: Develop the backend architecture and database design for storing user data and plagiarism reports.

REQ-4: Implement data processing algorithms to check the similarity of uploaded documents against previously published works.

REQ-5: Ensure that the plagiarism checking app using Python is secure and protects user data.
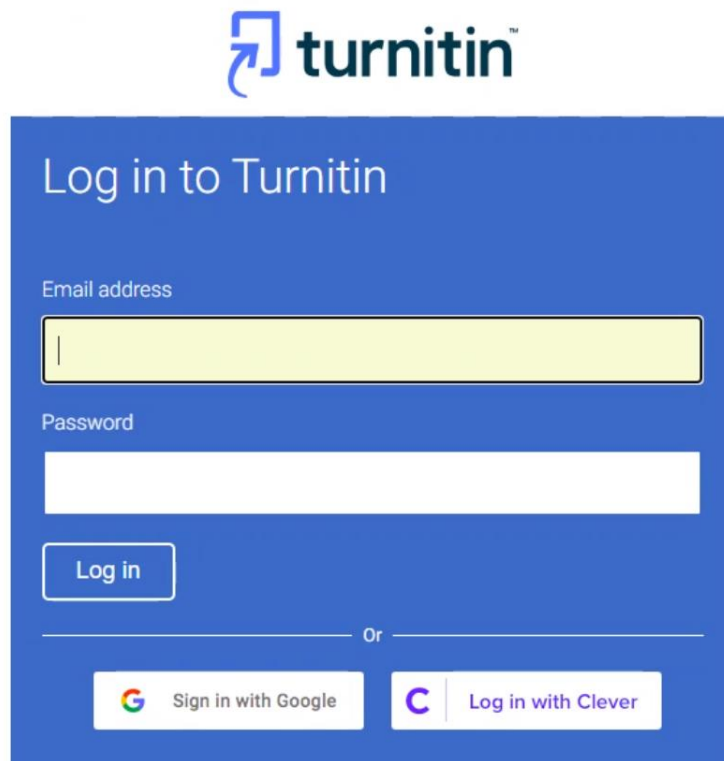
7   Acknowledgements

8   References

1. **Bhasin, V., & Bhargava, P. (2018).** Plagiarism Detection Techniques: A Systematic Review. International Journal of Computer Applications, 179(28), 1-7. https://doi.org/10.5120/ijca2018917122

2. **Broughton, S., Li, X., Zhang, J., & Li, L. (2020).** Addressing Plagiarism in Undergraduate Research Writing: A Curriculum Innovation. Journal of Curriculum and Teaching, 9(1), 75-85. https://doi.org/10.5430/jct.v9n1p75

3. Chatti, M. A., Jarke, M., & Frosch-Wilke, D. (2017). Understanding and analyzing plagiarism using plagiarism taxonomy. Education and Information Technologies, 22(6), 2815-2840. https://doi.org/10.1007/s10639-017-9611-1

4. **Fazel-Zarandi, M. H., Akbari, M., & Amrollahi, A. (2016).** A novel approach to document plagiarism detection based on semantic text matching. Information Processing & Management, 52(4), 658-673. https://doi.org/10.1016/j.ipm.2015.12.002

5. **Foltýnek, T., Kopecký, K., & Povolný, J. (2017).** Towards measuring similarity of scientific papers–A preliminary study. Journal of Informetrics, 11(3), 749-769. https://doi.org/10.1016/j.joi.2017.06.006

6. **Gurney, T. (2018).** Tools for detecting plagiarism in student work. Journal of Perspectives in Applied Academic Practice, 6(2), 63-70. https://doi.org/10.14297/jpaap.v6i2.366

7. **Inoue, M., & Koseki, K. (2018).** Development of an Automatic Plagiarism Checking System Based on Sentence Similarity Detection. Journal of Educational Technology and Society, 21(1), 167-178. https://doi.org/10.18356/jets.21.1.167

8. **Ismail, R., Arshad, S. S., & Sadiq, S. (2020)**. A systematic review of plagiarism detection tools and techniques. Journal of Information Science, 46(1), 19-40. https://doi.org/10.1177/0165551519866386
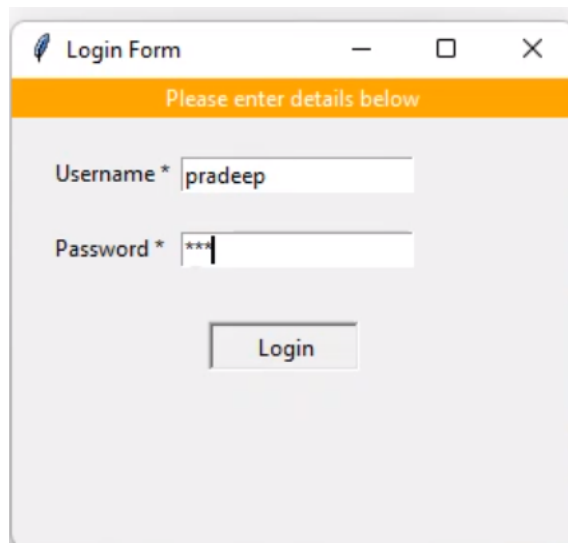
9   *Appendix*

1.   Login to Turnitin

Fig 9 (Login)

2.   Using GUI in python need to login to that site

Fig 10(Gui)

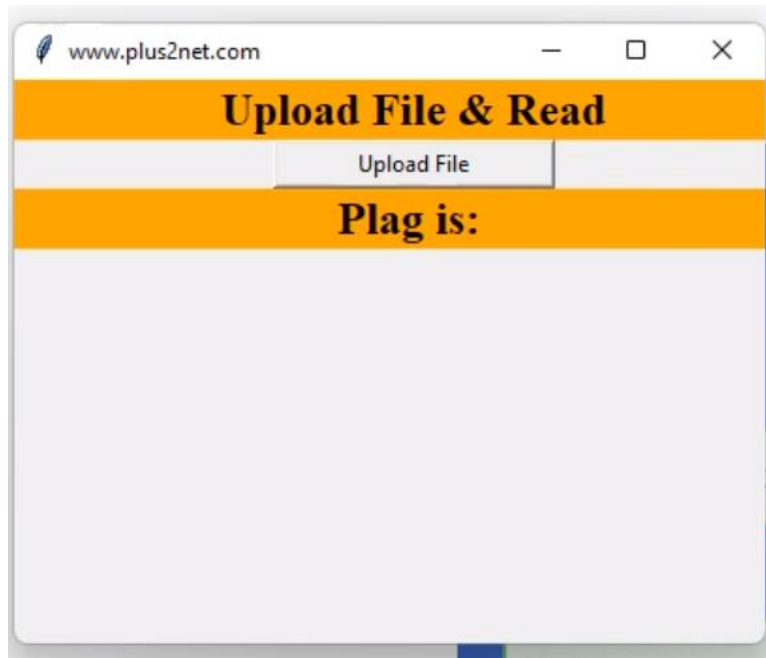3.   Uploading the Document file

Fig 11 (Document upload)

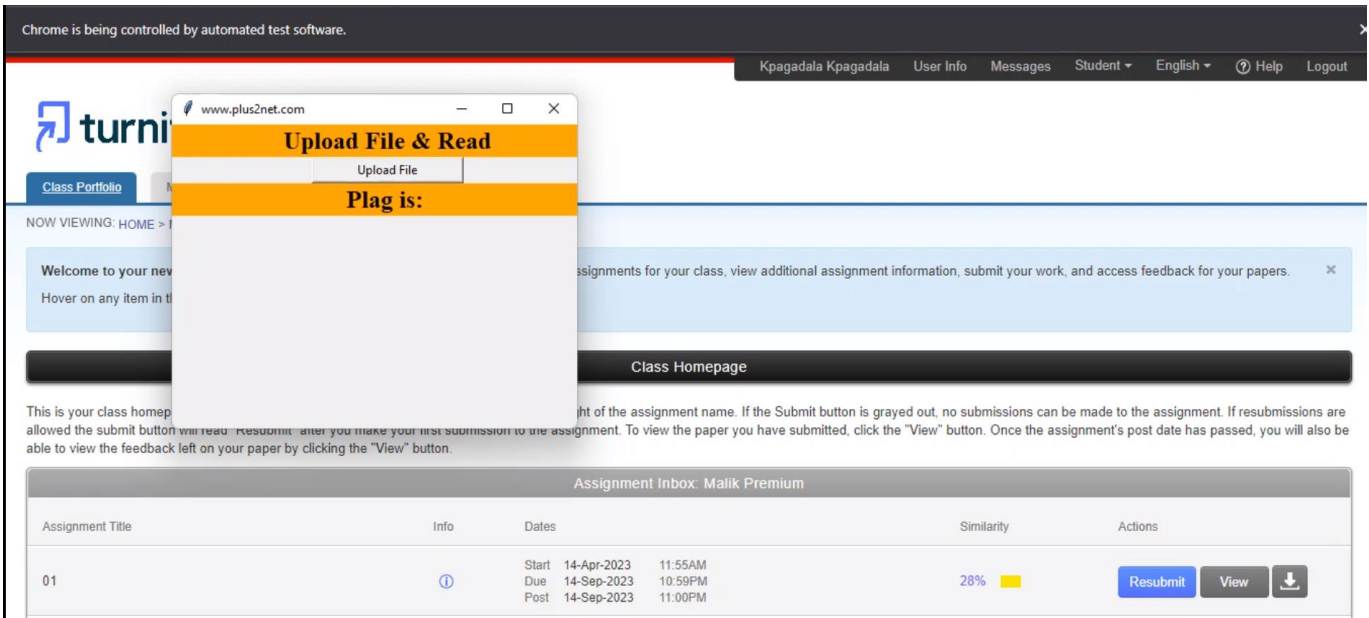4. We can see how it is generating the report

Fig 12 (Scrapping Data)

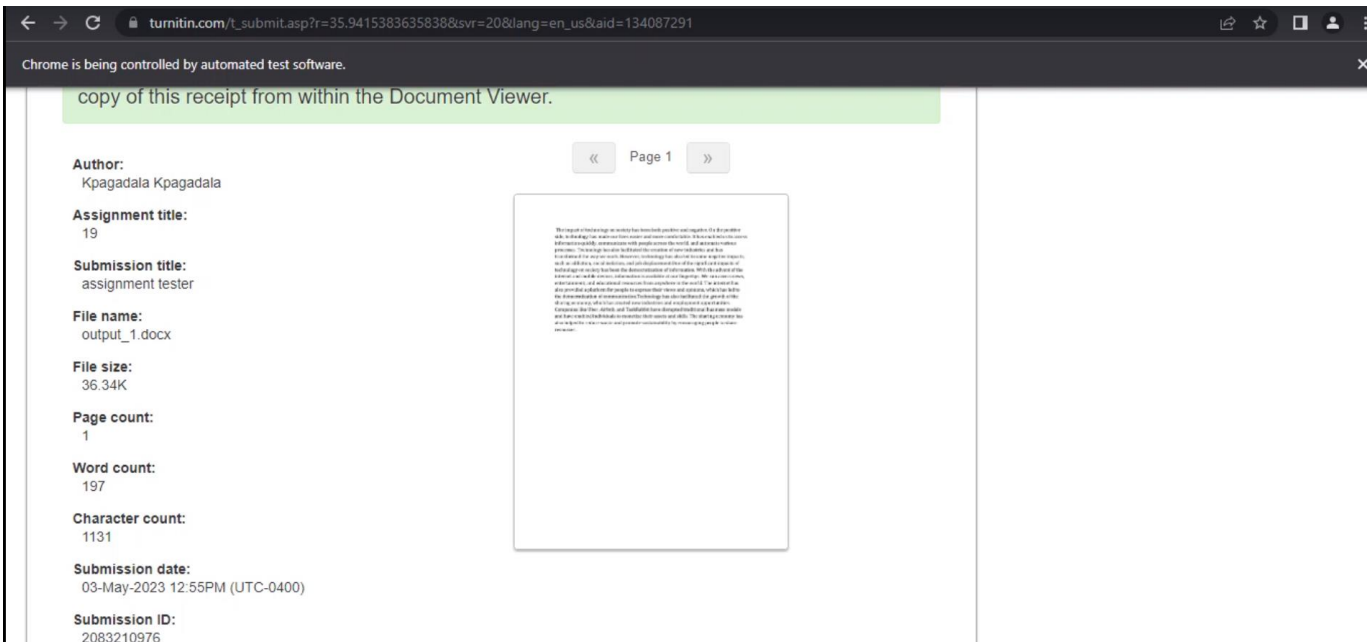5. We can see the result in the below screenshot.

Fig 13 (Result)