Web Scraping Project Report

Objective

The objective of this project is to develop a Python script using Selenium to automate the login process on GitHub, scrape specific user profile data, and handle potential issues like incorrect login details and data security.

1. Website Selection

Website: GitHub

GitHub is a widely-used platform for version control and collaboration. The user profiles on GitHub contain valuable information such as name, bio, and follower count, making it an ideal target for data scraping.

2. Login Automation

The script automates the login process on GitHub. Below are the key steps involved:

- Open Login Page: The script navigates to the GitHub login page.
- Input Credentials: It locates the username and password fields and inputs the credentials stored in environment variables.
- Submit Login Form: The script clicks the login button to authenticate the user.

Handling Login Issues

- The script checks for any error messages that indicate incorrect login details and prints an appropriate message.
- The script does not attempt to bypass CAPTCHAs, adhering to ethical guidelines and terms of service of the website.

3. Data Extraction

The script targets and extracts the following data from the user's GitHub profile:

- Name: The name of the user.
- Bio: The biography or description of the user.
- Followers: The number of followers the user has.

This data is then saved into a CSV file for further analysis or use.

4. Error Handling and Data Security

Error Handling

The script includes robust error handling to manage issues during the login or data extraction process. For example:

- It catches exceptions during the login process and prints error messages.
- If an element is not found or the page does not load correctly, the script will log the error and take a screenshot for debugging purposes.

Data Security

- The script uses environment variables to store and access login credentials securely. This approach ensures that sensitive information is not hard-coded in the script.
- The extracted data is saved in a CSV file in append mode, ensuring that each run of the script adds new data without overwriting existing data.

5. Documentation

The Python script is thoroughly documented with comments. Below is the complete code with comments:

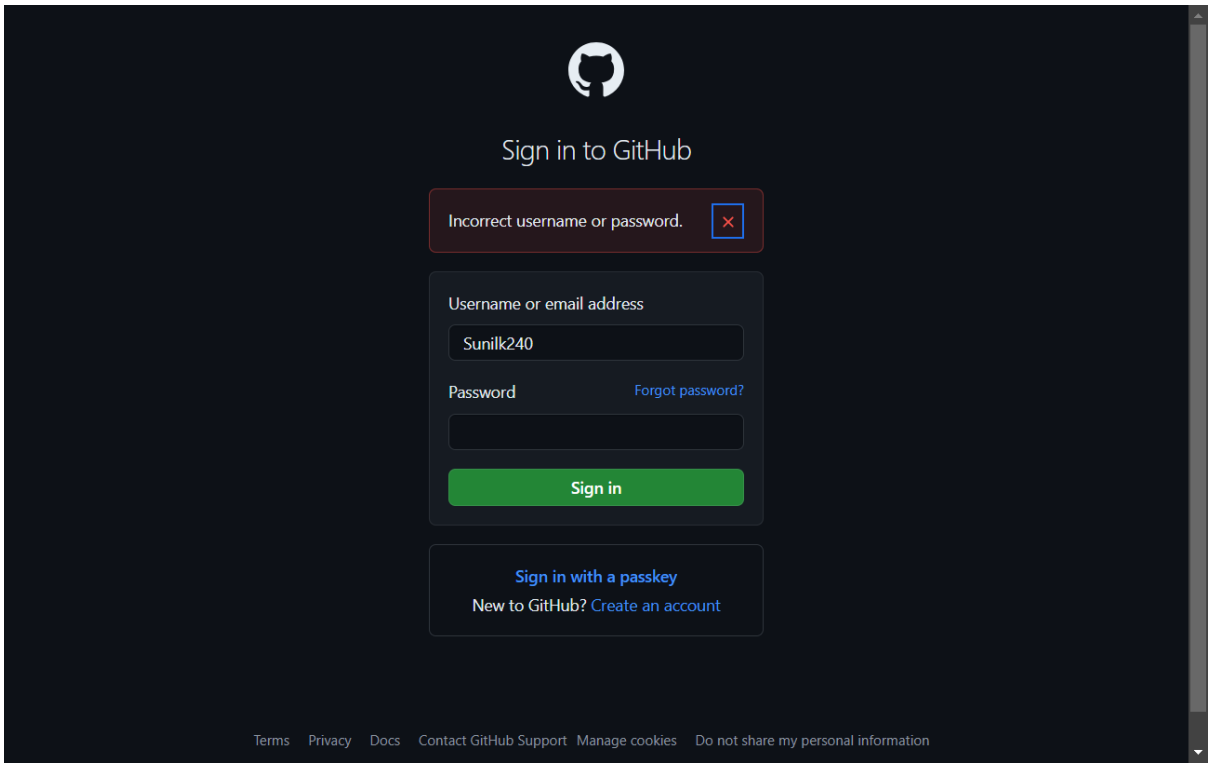
Code

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from webdriver_manager.chrome import ChromeDriverManager
import os
import csv

def login_and_scrape():
    # Setup Chrome WebDriver
    driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

    try:
        # Open GitHub login page
        driver.get('https://github.com/login')
        print("Opened GitHub login page")

        # Locate the username, password fields, and login button
        username = driver.find_element(By.ID, 'login_field')
        password = driver.find_element(By.ID, 'password')
        login_button = driver.find_element(By.NAME, 'commit')

        # Enter login credentials
        username.send_keys(os.getenv('GITHUB_USERNAME'))
        password.send_keys(os.getenv('GITHUB_PASSWORD'))

        # Click the login button
        login_button.click()
```

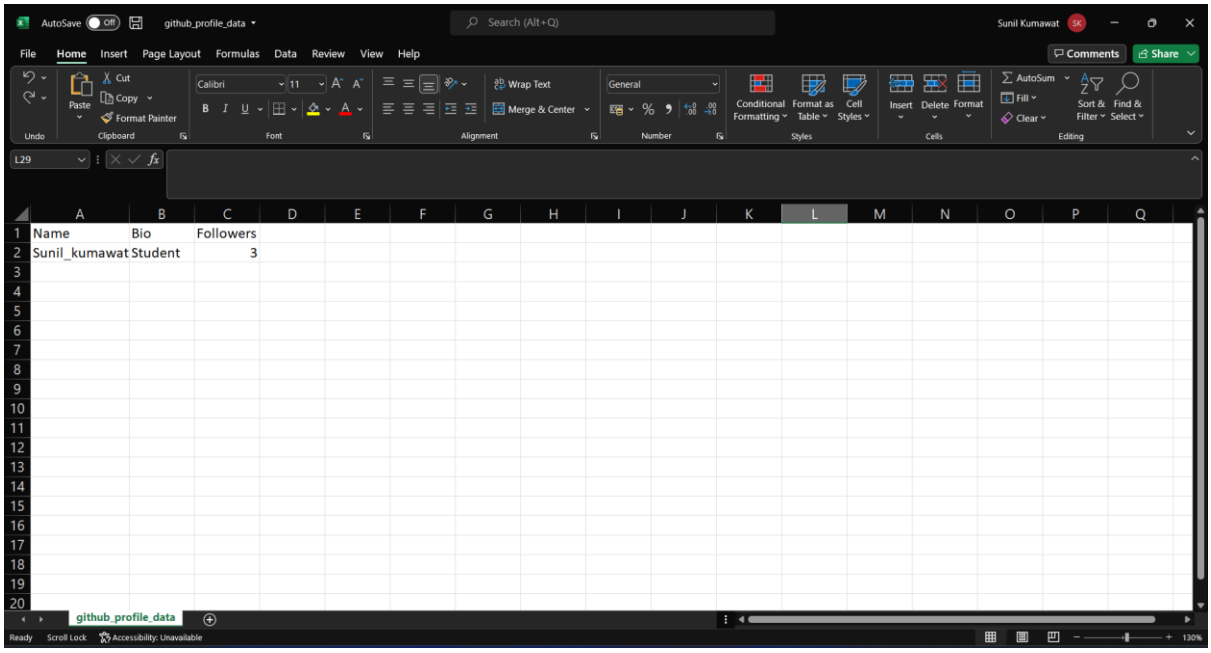
```
        # Wait for page to load and check current URL to verify login
        WebDriverWait(driver, 10).until(EC.url_changes('https://github.com/login'))
        current_url = driver.current_url
        if 'login' in current_url:
            print("Login failed: still on login page")
            return
        else:
            print("Login successful")

        # Navigate to the user's profile page
        driver.get(f'https://github.com/{os.getenv("GITHUB_USERNAME")}')
        print("Navigated to user profile page")

        # Extract profile data
        name = WebDriverWait(driver, 10).until(
            EC.presence_of_element_located((By.CLASS_NAME, 'p-name'))
        ).text
        bio = driver.find_element(By.CLASS_NAME, 'p-note').text
        followers = driver.find_element(By.XPATH, "//a[contains(@href, 'followers')]/span").text

        # Print extracted data
        print(f"Name: {name}, Bio: {bio}, Followers: {followers}")

        # Save data to a CSV file
        with open('github_profile_data.csv', 'a', newline='') as file:
            writer = csv.writer(file)
            writer.writerow(['Name', 'Bio', 'Followers'])
            writer.writerow([name, bio, followers])
        print("Data saved to github_profile_data.csv")

    except Exception as e:
        print(f"An error occurred: {e}")
        driver.save_screenshot("error_screenshot.png")

    finally:
        # Close the browser
        driver.quit()

if __name__ == "__main__":
    # Load environment variables
    from dotenv import load_dotenv
```

```
load_dotenv()

# Run the scraping function
login_and_scrape()
```

Error Screenshot (if any)



Output csv (in case of no error)



Challenges and Solutions

Challenge 1: Dynamic Content Loading

Solution: Used explicit waits to ensure that the elements are loaded before attempting to interact with them.

Challenge 2: Handling Incorrect Login

Solution: Added checks for error messages indicating incorrect login and printed appropriate feedback.

Challenge 3: Ensuring Data Security

Solution: Used environment variables to store login credentials securely, and ensured the script does not store sensitive information in plain text.

Insights and Applications

The data scraped from GitHub user profiles can be used for various purposes such as:

Analyzing Developer Profiles: Understanding the backgrounds and interests of developers on GitHub.

Recruitment Process: Identifying potential candidates for job opportunities based on their GitHub activity and profile information.

Market Research: Gathering insights into the user base of GitHub for developing targeted marketing strategies.

Conclusion

This project demonstrates the ability to automate web interactions using Selenium, handle login processes securely, and scrape and store data efficiently. The solutions implemented address common challenges in web scraping, ensuring robustness and data security.