Python Web Scraper Internship - Practical Assessment Company: International Credit Score

Introduction

Thank you for your interest in the Python Web Scraper internship at International Credit Score. This practical assessment is designed to evaluate your skills in web scraping, data handling, and automation. The task simulates a real-world project where we need to gather job market data to analyze employment trends.

Scenario

Your goal is to build a Python script that scrapes UK-based "Data Analyst" job postings from two different (simulated) web pages, processes the information, and stores it in a PostgreSQL database.

Technical Requirements

- **Language:** Python
- **Libraries:**
  - `requests` & `BeautifulSoup` for static scraping.
  - `Selenium` for dynamic scraping.
  - A PostgreSQL adapter like `psycopg2-binary` or `asyncpg`.
- **Database:** PostgreSQL

Part 1: Database Setup

Before you begin scraping, set up a local PostgreSQL database.

1. Create a new database (e.g., `internship_assessment`).
2. Within that database, create a table named `job_listings`.
3. Use the following DDL (Data Definition Language) to define the table structure.

SQL
```
CREATE TABLE job_listings (
    id SERIAL PRIMARY KEY,
    job_title VARCHAR(255),
    company_name VARCHAR(255),
    location VARCHAR(255),
    job_url VARCHAR(512) UNIQUE,
    salary_info TEXT,
    job_description TEXT,
    source_site VARCHAR(100),
    scraped_at TIMESTAMPTZ DEFAULT CURRENT_TIMESTAMP
);
```

Part 2: Scraping a Job Listing Page (`requests`)

For this part, you will scrape a job board that has a simple, static list of jobs. A good example to practice on is the https://wellfound.com/jobs programming jobs section.

Task:

1. Write a script using `requests` and `BeautifulSoup` to scrape the first two pages of job listings for a search term like "Python" or "Data".
2. For each job posting on the list, extract the following summary data:
    - Job Title
    - Company Name
    - Location
    - Salary Information (if available)
    - A direct URL to the full job details page.
3. Insert each job's summary data as a new row into your `job_listings` PostgreSQL table. The `job_description` field should be left `NULL` for now.
4. Set the `source_site` field to the name of the website you scraped (e.g., 'WeWorkRemotely').

Part 3: Scraping a Dynamic Job Details Page (`Selenium`)

For this part, you will use the job URLs you collected in Part 2. We will simulate a scenario where the full job description is loaded dynamically with JavaScript after the page loads.

Task:

1. Iterate through the jobs you just inserted into your database.
2. For each job, use `Selenium` to navigate to its `job_url`.
3. Implement a wait strategy (e.g., `WebDriverWait`) to ensure the description element is present before you attempt to scrape it.
4. Once the description is visible, extract the full text content.
5. Write an `UPDATE` statement to add this `job_description` to the correct row in your `job_listings` table (identifying the row by its `job_url`).

Bonus Challenges (Optional, but highly recommended)

- **Error Handling:** Implement `try...except` blocks to handle potential issues gracefully (e.g., a URL that fails to load, a missing data element).
- **Code Structure:** Organize your code into logical functions or classes (e.g., `setup_database()`, `scrape_list_page()`, `scrape_detail_page()`).
- **Configuration:** Use a separate configuration file (e.g., `.env` or `config.ini`) to store database connection details instead of hardcoding them and the website is required to be scraped without any login.

Submission Guidelines

Please submit your work by providing a link to a public **GitHub repository** containing the following:

1. Your Python script(s).
2. A `requirements.txt` file listing all necessary libraries.
3. A `README.md` file that includes:
    - A brief overview of your approach.
    - Clear instructions on how to set up the database and run your script.

Evaluation Criteria

Your submission will be evaluated based on the following:

- **Correctness:** Does the script run successfully and populate the database as requested?
- **Code Quality:** Is the code clean, readable, well-commented, and logically structured?
- **Data Integrity:** Is the data clean and accurately stored in the correct database columns?
- **Best Practices:** Have you followed the submission guidelines and demonstrated good development habits?

Good luck! We look forward to reviewing your work.