Python-Powered Cross-Platform Web Crawler

Members:

- Arham Hussain [22k-4080]
- Aarib Ahmed [22K-4004]

Project Description

Our crawler will be a **Python-only** solution combining four libraries:

- **Scrapy:** a fast, high-level web crawling framework built around Twisted for asynchronous, event-driven crawling.It offers built-in support for cookies, session handling, user-agent spoofing, robots.txt compliance, and crawl depth restriction.
- **Requests:** the de facto standard HTTP library for Python, providing a simple API for GET/POST/PUT/DELETE methods, automatic connection pooling, keep-alive support, and header customization.
- **BeautifulSoup:** a parsing library that builds parse trees from HTML/XML, enabling idiomatic navigation, searching, and modification of the document, even when the markup is poorly formed.
- **Selenium:** a browser automation tool ideal for scraping dynamic, JavaScript-heavy sites (SPAs), allowing us to simulate user actions such as scrolling and clicking to load additional content.

Functional Features

1. **URL Discovery:** Start from seed URLs, extract links, and enqueue new targets using Scrapy's scheduler.
2. **Content Extraction:** Fetch pages with Requests, parse static parts with BeautifulSoup, and handle dynamic elements via Selenium.
3. **Data Storage:** Output structured data (JSON/CSV) through Scrapy pipelines for easy analysis.
4. **Dynamic Handling:** Automate infinite scroll and clicking to reveal hidden content with Selenium.
5. **Error Handling & Logging:** Implement retry logic, HTTP status checks, and detailed logs within Scrapy middlewares.

Plan of Work

Phase	Tasks
1	• Set up Python virtual environments and install Scrapy, Requests, BeautifulSoup, Selenium. • Define crawl scope & seed URLs.
2	• Implement Scrapy spiders for basic crawling and URL extraction. • Build initial pipeline for Requests + BeautifulSoup parsing.
3	• Integrate Selenium for JavaScript-rendered page handling (infinite scroll, button clicks). • Develop data storage schemas (JSON/CSV).
4	• Enhance error-handling middlewares: retries, timeouts, HTTP checks. • Add logging and progress reporting.
5	• Conduct end-to-end tests, debug anomalies, and optimize performance. • Prepare project documentation and presentation.

Individual Contributions that can be made:

Arham:

- Develop Scrapy spiders and scheduler logic.
- Integrate Selenium workflows for dynamic content.
- Design data storage structures and pipelines (May be using CSV/JSON or SQLite).

Aarib:

- Implement Requests + BeautifulSoup parsing modules.
- Build error-handling and logging middlewares.
- Conduct unit tests and performance tuning.

References

1. Scrapy 2.12 Documentation – “Scrapy is a fast high-level web crawling and web scraping framework, used to crawl websites and extract structured data from their pages.” || https://docs.scrapy.org/
2. Scrapy & Twisted – https://scrapfly.io/blog/web-scraping-with-scrapy

3. Requests Library – “The Requests library is the de facto standard for making HTTP requests in Python.” || https://realpython.com/python-requests/
4. BeautifulSoup – “Beautiful Soup is a Python library designed for parsing HTML and XML documents. It creates parse trees that make it straightforward to extract data.” || https://realpython.com/beautiful-soup-web-scraper-python/
5. BeautifulSoup Documentation – “Works with your favorite parser to provide idiomatic ways of navigating, searching, and modifying the parse tree.” || https://realpython.com/beautiful-soup-web-scraper-python/
6. Selenium & Python – “This makes Selenium an excellent choice for scraping data from dynamic, JavaScript-heavy websites (SPAs).” || https://www.scrapingbee.com/blog/selenium-python/
7. Dynamic Scraping – Techniques for scraping dynamic content including Selenium automation for infinite scroll. || https://oxylabs.io/blog/dynamic-web-scraping-python