Web Scraper framework in Python

Class: ITF20119-1 23V Rammeverk

Delivery Date: 30.04.2024

Written by: Kristoffer Snopestad Søderkvist

Table of Contents

Introduction

Introduction of Web Scraping

Web scraping tools have become a tool often used in this modern era for collecting data and analysis. It often allows collecting data and relevant information automatically from the internet. When I started this I wanted to create a web scraper framework for Python, making it more efficient to get data when the developer needs it.

The point is to create a sturdy Python framework for web scraping, aimed at simplifying data extraction, processing and storage. The framework will feature user-friendly interfaces, easy to read documentation, and adaptability to customize and scale, supported by multithreading capabilities. It’s designed to be an invaluable tool for researchers, analysts, and businesses alike.

Key features of the framework include:

User-friendly Interface

It offers an intuitive interface that simplifies the configuration of scraping tasks, allowing users to specify the target website and define the data fields they want to extract.

Website Crawling

The framework includes a web crawler that navigates websites efficiently, traverses links and uncovers new pages to scrape, utilizing smart strategies to optimize efficiency and minimize unnecessary requests.

Data Extraction and Parsing

Equipped with robust parsing capabilities, the web scraper framework supports various techniques like HTML parsing and XPath, enabling efficient extraction of specific data elements.

Data Cleaning and Transformation

Understanding the importance of data quality, it incorporates tools for data cleaning and transformation, such as handling missing values, removing duplicates and standardizing formats to ensure high quality data output.

Robust Error Handling

The framework is designed to reduce the frustration associated with troubleshooting by offering comprehensive and well-documented error handling mechanisms.

Data Storage and Export

It supports multiple data storage options and provides the flexibility to store scraped data in formats like CSV, JSON or XML. It also includes features to export data to different types of file formats, adjusted to different needs from the user.

Background

Existing Solutions

Web scraping and crawling has become a vital technique for extracting data from websites, empowering researchers, businesses and analysts to leverage the extensive information available on the internet. Over time many tools and libraries have been created to support web scraping activities. To develop a user-friendly and straightforward framework, I read up and tried to do an analysis of the existing frameworks and their functionalities. This section provides a summary of pros and cons of the three widely-used web scraping frameworks for Python: Beautiful Soup, Scrapy and Selenium.

Beautiful Soup

- **Advantages**: Beautiful Soup is a Python library acclaimed for its ease of use, making it ideal for beginners and projects with tight deadlines. It is great at parsing HTML and XML pages and can handle poorly formatted websites.
- **Disadvantages**: Its main disadvantage is that it can not process JavaScript since it only works with static web content. .
  (“Beautiful Soup Documentation”)

Scrapy

- **Advantages**:It is a versatile and powerful framework. SCrapy framework is designed for large-scale web scraping. It supports asynchronous requests, which will make the work go faster. This framework is also open, so developers can make custom additional features if there are something missing.
- **Disadvantages**: Scrapy’s complexity and steep learning curve can make it difficult for developers to start learning it. Scrapy also can not work with JavaScript for the start, for this a developer must use additional tools to get this to work.
  (“Scrapy 2.11 documentation”)

Selenium

- **Advantages:** Selenium is a framework that has the ability to automate browser operations, acting as a user to interact with webpage elements, such as clicks and keystrokes. This is very useful for pages that are using content generated by JavaScript, which is common on modern websites.
- **Disadvantages:** Selenium has a major downside including its slow performance and that it needs to be adjusted for every script, since other web pages often get updated.
  (“The Selenium Browser Automation Project”)

Methodology

The development of this framework I tried to do it with a systematic approach that included a few stages: API Design Specification, API Design, Implementation, and a little bit of User Testing. In this section below I provide a description of methods and practice that was used in each stage.

API Design Specification

In the first phase of creating my web scraping framework, I focused on defining its requirements and scope. Researching existing tools like Beautiful Soup, Scrapy and Selenium was an important step to understand the functionalities that users value and the gap in other web scraping frameworks. Insights from this research guided the features and architecture outlined in my API Design Specification. This document, forming the project's blueprint, articulated the envisioned functionality and system design, laying the groundwork for all development stages to follow.

API Design

Using the API Design Specification, I designed the framework's APIs with a focus on the end-user developers needs. Emphasizing simplicity and intuitive interaction, the design ensures ease of use and seamless integration into diverse applications. The aim was a straightforward, consistent API that developers can quickly adopt and implement, so it can be used in various applications.

Implementation

During the implementation phase, I chose python due to its popularity and the rich set of libraries available. To make it well-organized, modular and reusable I chose to make it OOP (object-oriented programming) principles to ensure the code was that. This established a strong foundation for the framework's functionality.

User Testing

Due to the issue that I didn't know who to ask for external user testing that was not conducted. However, a lot of self-testing was carried out to ensure the functionality and reliability of this web scraper framework. This testing was performed on multiple computers with different configurations to simulate different user environments and ensure broad compatibility and performance. More about this in the Discussion section at almost the end of this document.

Design process and results

Scenario driven development

Scenario-driven development is a methodology that places user scenarios at the front of software development. This approach begins by defining and prioritizing specific use cases or scenarios that illustrate how the end-users will interact with the software under different conditions. Each Scenario details the user's objectives, actions and the expected outcome, providing a clear picture of the necessary features and functionalities.

By focusing on these detailed scenarios, developers can better understand the diverse needs and expectations of their users. This ensures the design and development processes are aligned with actual user requirements, facilitating the creation of software that effectively meets those needs and enhances user satisfaction. Through scenario-driven development, the project aims to deliver a product that is not only functional but also intuitive and responsive to user interactions.

Scenario-Driven Design

Design the web scraper framework to manage different website structures, utilize scraping techniques like CSS selectors or XPath, handle dynamic content with JavaScript or AJAX, incorporate error handling and ensure user-friendly outputs.

Scenario-Based Testing

Create and execute test cases for different website types based on identified scenarios, ensuring the web scraper accurately retrieves data and handles different situations effectively.

Scenario Analysis and Prioritization

This section delves into identifying potential challenges and prioritizing scenarios for effective web scraping development.
Challenge Identification: Key challenges include handling diverse website layouts, dynamic content, login requirements and error management. It’s crucial to consider website-specific rules and limitations, such as following the rules to robots.txt to ensure compliance and functional robustness.
Scenario Prioritization: Scenarios are ranked based on their importance to the core functionality of the framework. The main scenario, offering essential features is prioritized first. Other scenarios are ordered according to their relevance and the demands of the users, ensuring that the most impactful features are developed and refined early in the process.

Scenario-Based Pseudo Code

Under this there are some pseudo code examples on different scenarios for scraping data. These are scenarios used to make the file scrape.py, a lot of these are copy paste with small changes this is how I was thinking the code should look like.

In file Scrape.py -> function: scrape()

These are the main scenarios, that was focused on first

Scenario 1: Scraping data from normal HTML data

```
Url = "https://itavisen.no"
DataType = "html"
ScrapedData = Scraper.scraper(Url, DataType)
```

Scenario 2: Unsupported data types that is not made or does not fit the script

```
Url = "https://itavisen.no"
DataType = "csv"
Try:
Scraped_data=Scraper.scrape(Url, dataType)
Except ValueError as error:
Print(error)
```

This will at least print the error in the console

Scenario 3: Scrape Json data

```
Url = "https://itavisen.no"
DataType = "html"
Scraped_data=Scraper.scrape(Url, dataType)
```

In file Scrape.py -> function: exportTo_file()

Scenario 1: Exporting all data from the website to a file.

```
Html_content = "<html><body><h1> A title </h1></body></html>"
File_path = "C:\Users\user_name\Documents" Where it should save
File_name = "testing.html" Its name and filetype
Scraper.exportTo_file(html_content, file_path, file_name)
```

And have created some similar function that also works with JSON, CSV etc.

Scenario 2: Read JSON data from API and save it locally

```
Json_data = {"name": "Kristoffer", "age" :26}
```

```
File_path = "C:\Users\user_name\Documents"
File_name = "testing.json"
Scraper.exportTo_file(json_data, file_path, file_name)
```

Could add more for these functions but felt like it would be too many duplications. Next is a few with the pseudo code for web crawler.

In file Scrape.py ->Class Crawler:()

This was second on the list to be created.

Scenario 1: Make the webcrawler multithread for higher speed and be able to have more web pages in que.

```
 Urls = ['https://example.com',
'https://test.no/1,
'https://test.no/2
'https://test.no/3,
'https://test.no/4
]
Num_of_threads = 3 adjust the speed with multithreading
crawl_depth=1
Crawler = WebCrawler()
Crawler.set_max_depth(crawl_depth, num_of_threads)
Crawler.crawl(urls)
```

In the framework I created functions also for getting product information and extracting headlines, but adding this will clutter up the document. These are also adjustable for getting more than what is written, making them great for developer's projects.

Resulting Framework

In this section, I present the completed framework, tailored to the specific scenarios I targeted. By integrating design patterns, establishing principles and using what I found that should be fixed, I have tried to develop a universal and robust web scraping and crawling framework.

The framework is structured around a core set of classes and methods designed to make the data extraction and organization processes from web pages as easy as possible. It's built to be modular and extensible, allowing for customization based on specific scraping needs.

Key design principles are embedded throughout the framework. For instance, I use inheritance to create a class hierarchy that enhances code reusability and provides a clear organizational structure. The "WebPage" class acts as the foundational base class with specialized subclasses like "NewsSite" and "Webstore" for specific types of web pages. This structure allows the framework to efficiently manage and extract data from diverse web environments. Additionally specific classes such as "Article" for news sites and "Product" for pages like e-stores/e-commerce sites facilitate targeted scraping, making it easier to gather relevant information from sites like itavisen.no, komplett.no and power.no.

Furthermore, the framework employs encapsulation by confining related functionality within each class. Each class has attributes and methods specific to its role, ensuring that data and operations are well-contained and isolated. This encapsulation significantly enhances code organization, readability and maintainability.

In conclusion the resulting framework provides a comprehensive solution for web scraping and crawling. By incorporating design patterns, principles and self user testing, I have created a modular and adaptable framework that efficiently extracts data from web pages. The code presented in the selected scenarios demonstrates the functionality and versatility of the framework, showcasing its ability to handle different types of web pages and extract specific information. With its well organized structure and hopefully user-friendly approach, the web scraper framework can serve as a powerful tool for various web scraping tasks.

Discussion

I could have begun earlier with this framework and that would have given me more time to maybe develop an even better framework, but I had work on weekdays and was sometimes too tired to work on weekends. But when I got started it went smoothly, and I got more into a routine. An improvement for this could be to start the routine earlier than what I did. Another problem was that I did not know that there was a discord group that I could have used to ask people for user testing, and when I thought about it it was too late to make any major changes to the code.

I had some challenges when I started to code this framework. I have coded in Python a lot awhile ago, and since I had not tried to create something that should be useful for other developers like a framework it was a bit difficult. But after googling a lot I found some different solutions to the problem. I actually had most problems setting up a virtual environment in Python. Otherwise mostly everything went fine, the Pytest created some problems but after a bit of reading and some videos online they started to run better.

Sources

“Beautiful Soup Documentation.” *Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation*, https://beautiful-soup-4.readthedocs.io/en/latest/. Accessed 28 April 2024.

"Scrapy 2.11 documentation." *Scrapy 2.11 documentation — Scrapy 2.11.1 documentation*, https://docs.scrapy.org/en/latest/. Accessed 28 April 2024.

"The Selenium Browser Automation Project." *Selenium*, 17 November 2023, https://www.selenium.dev/documentation/. Accessed 28 April 2024.