Midterm Project Report

Advanced Computer Programming

Student Name : Ariun-Erdene Sodnombayar

Student ID : 113021196

Teacher : DINH-TRUNG VU

2025-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/Ariun-E
2) **Group Project Repository**: https://github.com/Jantsagdorj/ACP-AU-1132

1.2 Overview

In this midterm project, I will demonstrate the use of the Scrapy web scraping framework, along with features such as custom settings, XPath/CSS selectors, and feed exporters, to extract structured data from my personal GitHub profile.

The program extracts the following information from each repository:

- URL
- Description (About)
- Last Updated timestamp
- Programming languages used
- Number of commits

Advanced Features and Libraries Used:

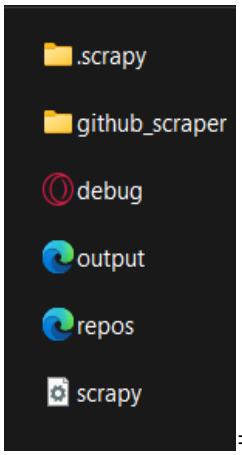
- **Scrapy**: A powerful web scraping framework for handling requests and parsing HTML.
- **Playwright Integration**: To handle JavaScript-rendered content on GitHub pages.
- **Regex**: Used for extracting specific data patterns like commit counts.
- **Feed Exporter**: To export the scraped data in XML format.

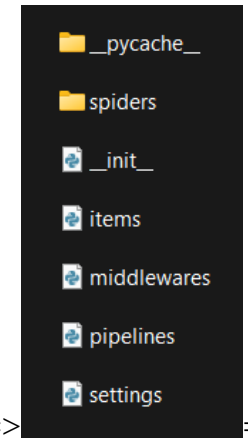
The scraper targeted the personal GitHub account https://github.com/Ariun-E and successfully extracted data into an XML file `repos.xml`.

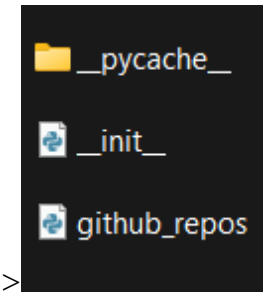
Scrapy version:

```
PS C:\Users\User> pip show scrapy
Name: Scrapy
Version: 2.12.0
Summary: A high-level Web Crawling and Web Scraping framework
Home-page: https://scrapy.org
Author: Scrapy developers
Author-email: pablo@pablohoffman.com
License: BSD
Location: C:\Users\User\AppData\Local\Programs\Python\Python313\Lib\site
-packages
Requires: cryptography, cssselect, defusedxml, itemadapter, itemloaders,
 lxml, packaging, parsel, protego, PyDispatcher, pyOpenSSL, queuelib, se
rvice-identity, tldextract, Twisted, w3lib, zope.interface
Required-by: scrapy-playwright
PS C:\Users\User>
```

Folders:



=> github_scraper=>


=>spiders=>


Chapter 2 Implementation

2.1 Class 1

GithubSpider

This is the **main spider class** that defines the behavior of the GitHub scraper. It extends the *scrapy.Spider* class and implements all logic required to visit a GitHub profile page, extract repository data, and follow links to each repository for deeper scraping.

```
import scrapy

class GithubSpider(scrapy.Spider):
    name = "github"
    start_urls = ["https://github.com/Ariun-E?tab=repositories"]

    # Feed Exporter configuration for XML output
    custom_settings = {
        "FEEDS": {
            "repos.xml": {
                "format": "xml",
                "encoding": "utf-8",
                "fields": ["url", "about", "last_updated", "languages", "commits"],
                "store_empty": False,
                "indent": 4
            }
        },
    }

    def parse(self, response):
        for repo in response.css('li[itemprop="owns"]'):
            relative_url = repo.css('a::attr(href)').get()
            full_url = response.urljoin(relative_url)
            about = repo.css('p::text').get(default='').strip()
            last_updated = repo.css('relative-time::attr(datetime)').get()

            repo_data = {
                "url": full_url,
                "about": about if about else relative_url.strip('/').split('/')[-1],
                "last_updated": last_updated,
                "languages": None,
                "commits": None
            }

            yield scrapy.Request(full_url, meta={"repo_data": repo_data},
callback=self.parse_repo)

        # Handle pagination
        next_page = response.css('a.next_page::attr(href)').get()
        if next_page:
            yield scrapy.Request(response.urljoin(next_page), callback=self.parse)

    def parse_repo(self, response):
        repo_data = response.meta["repo_data"]

        # Check if the repository is empty
        is_empty = response.css('div.Blankslate').get() is not None

        if is_empty:
```

```
            repo_data["languages"] = "None"
            repo_data["commits"] = "None"
        else:
            # Extract languages
            language_names = response.css('a[href$="languages"] span::text').getall()
            repo_data["languages"] = ', '.join([lang.strip() for lang in language_names
if lang.strip()]) or "None"

            # Extract number of commits
            commit_count = response.css('li a[href$="commits"]
span::text').re_first(r'\d[\d,]*')
            repo_data["commits"] = commit_count.replace(',', '') if commit_count else
"None"

        yield repo_data
```

2.1.1 Fields

- ***name**: "github"* – The unique name used to run this spider.
- ***start_urls*** A list containing the GitHub profile page URL to begin scraping.
- ***custom_settings***: Dictionary that overrides the default Scrapy settings to export results into an XML file (*repos.xml*).

Includes *format: "xml", encoding: "utf-8", fields* to be exported, and output formatting.

2.1.2 Methods/Functions/

- ***parse(self, response)**:*
  This is the default callback for the start URL. It extracts a list of repositories, and for each one:
  - Gets the repository link.
  - Extracts description (About).
  - Extracts last updated date.
  - Sends a request to the repository's page and passes the extracted data using the *meta* parameter.

- ***parse_repo(self, response)**:*
  This method handles the response from each individual repository page. It extracts:
  - The programming languages used.
  - The number of commits.

- If the repository is empty, it sets these values to *"None"*.

Chapter 3 Results

3.1 Result

After running the spider with *scrapy crawl github_repos -o output.xml* in powershell, the program generates an XML file called *repos.xml*. It was able to successfully scrape important details from a GitHub repository. It collected the repository URL, its description (or "About" section), the last updated time, the programming languages used, and the number of commits.

The repository seems to be either new or still empty, which is why the values for languages and commits are both listed as "None". This confirms that the spider works as expected—even when a repository doesn't have much data, it still processes it smoothly without errors.

.

Chapter 4 Conclusions

This midterm project gave me hands-on experience with web scraping using Python and Scrapy. I built a spider that collects information from GitHub repositories and exports it into an XML file. Going through the process of building this scraper helped me learn and apply a lot of different concepts related to web automation and data processing.

One of the main things I learned was how to set up a Scrapy spider to handle nested content by making multiple requests between different pages. I also got better at using CSS selectors to pull specific elements from a webpage and cleaning the data with Python to make it usable.

Another key takeaway was learning how to export the data in a structured format like XML. This made it much easier to organize the data and use it for other tasks or reports.

I also got to understand how important it is to simulate browser headers to avoid being blocked by anti-scraping measures. On top of that, I added features to handle pagination and detect empty repositories, which made the spider more reliable.

Overall, this project helped me get a better grasp of web scraping and improved my skills in Python, especially when dealing with real-world problems.