亞洲大學
ASIA UNIVERSITY

Midterm Project Report

Advanced Computer Programming

Student Name : Chinzorig Battulga

Student ID : 113021191

Teacher : DINH-TRUNG VU

2025-04

Chapter 1 Introduction

1.1 Github

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

The objective of this project was to create a web scraper using Python's Scrapy framework to extract structured information from a GitHub profile. This tool is particularly useful for developers and researchers who wish to analyze repositories at scale without manually copying data. The main goal was to collect information from each public repository under a specific user, including its URL, description (About), last updated timestamp, programming languages used, and number of commits.

1.2 Overview

To complete this project, the following advanced Python programming tools and libraries were used:

- **Scrapy**: The core framework used for web scraping. It allowed for easy spider creation, request handling, and HTML data extraction.
- **cssselect**: Used in conjunction with Scrapy to define CSS selectors and extract specific elements from the GitHub HTML structure.
- **Feed exporter**: A Scrapy built-in tool that outputs scraped data in structured formats, such as XML, JSON, and CSV. For this project, XML was chosen.

The scraper navigates from the GitHub profile page (https://github.com/Mimikooo?tab=repositories) to each listed repository, gathering relevant data. The collected data is exported to an XML file called repos.xml, which contains detailed information for further use.

Implementation

1.1 Setup and Environment

1.1.1 Environment Information

- Operating System: Windows 11
- Python Version: 3.12
- Scrapy Version: 2.12.0
- IDE/Text Editor: VS Code and PowerShell Terminal

1.1.2 Installing ScrapyFunctions and creating the project

To install Scrapy, I used pip in the PowerShell terminal:

```
PS C:\Users\chinz> pip install scrapy
Defaulting to user installation because no
```

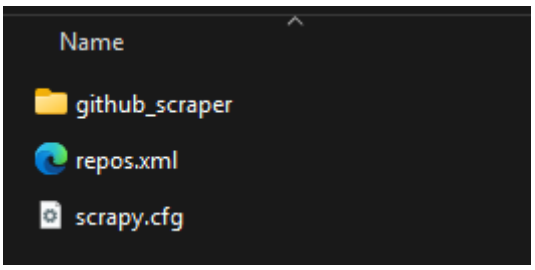
After installation, I verified that Scrapy was working:

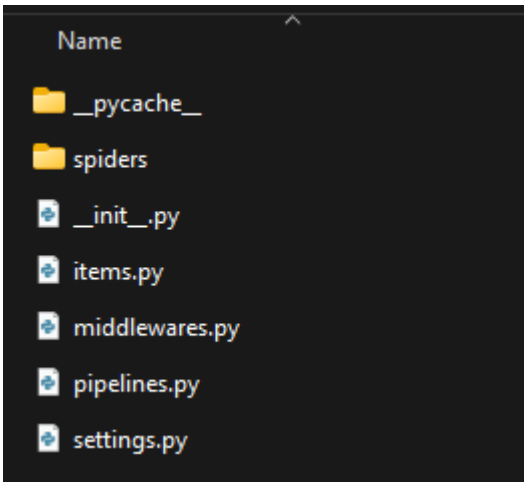
```
PS C:\Users\chinz\OneDrive\Documents\Advanced Computer - Midterm\github_scraper> python -m scrapy -v
Scrapy 2.12.0 - active project: github_scraper
```

I created a new Scrapy project named github_scraper with the following command:

```
PS C:\Users\chinz\OneDrive\Documents\Advanced Computer - Midterm> python -m scrapy startproject github_scraper
New Scrapy project 'github_scraper', using template directory 'C:\Users\chinz\AppData\Local\Packages\PythonSoftwareFound
ation.Python.3.12_qbz5n2kfra8p0\LocalCache\local-packages\Python312\site-packages\scrapy\templates\project', created in:
    C:\Users\chinz\OneDrive\Documents\Advanced Computer - Midterm\github_scraper
```

This generated the necessary folder structure:





1.2 Class

Class: GithubScraperItem (items.py)

Description: Defines the fields that will be scraped and stored for each repository. This item acts like a data structure.

Fields:

- url: Complete URL to the GitHub repository
- about: Description or purpose of the repository. If not provided and the repository is not empty, defaults to the repository name.
- last_updated: Datetime string showing the latest update timestamp
- languages: List of programming languages used in the repository (if any)
- commits: Total number of commits in the repository (if available)

items.py 1

C: > Users > chinz > OneDrive > Documents > Advanced Computer - Midterm >

```python
import scrapy

class GithubScraperItem(scrapy.Item):
    url = scrapy.Field()
    about = scrapy.Field()
    last_updated = scrapy.Field()
    languages = scrapy.Field()
    commits = scrapy.Field()

```

1.3 Spider

Spider: github_spider.py

Description: Scrapy spider that starts from the repositories page and navigates into each repo to extract information.

Method: parse()

Navigates through the repositories listed on the user's GitHub page and follows each repository link.

Method: parse_repo(response)

Scrapes details of each repository:

- Checks if the repo is empty
- Extracts about, last_updated, languages, and commits (if available)
- Yields structured GithubScraperItem

items.py 1 github_spider.py 2

C: > Users > chinz > OneDrive > Documents > Advanced Computer - Midterm > github_scraper > github_scraper > spiders > git

```python
import scrapy
from github_scraper.items import GithubScraperItem

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

    Tabnine | Edit | Test | Explain | Document
    def parse(self, response):
        repo_links = response.css('h3 a::attr(href)').getall()
        for link in repo_links:
            yield response.follow(link, self.parse_repo)

    Tabnine | Edit | Test | Explain | Document
    def parse_repo(self, response):
        item = GithubScraperItem()
        item['url'] = response.url

        about = response.css('p.f4::text, p.f4 span::text').get()
        about = about.strip() if about else None
        is_empty = response.css('.blankslate').get() is not None

        repo_name = response.url.split('/')[-1]
        item['about'] = about if about else (None if is_empty else repo_name)

        last_updated = response.css('relative-time::attr(datetime)').get()
        item['last_updated'] = last_updated

        if not is_empty:
            languages = response.css('.repository-language-color + span::text').getall()
            item['languages'] = languages if languages else None

            commits_text = response.css('li.commits a span::text').get()
            if commits_text:
                item['commits'] = commits_text.strip().replace(',', '')
            else:
                item['commits'] = None
        else:
            item['languages'] = None
            item['commits'] = None

        yield item
```

Chapter 2 Results

1.1 Results

Result 1: The spider successfully scraped the public repository Test. Since this repository is likely empty or minimal, some fields returned None.

Output (repos.xml):

```
<item>
  <url>https://github.com/Mimikooo/Test</url>
  <about>Test</about>
  <last_updated>None</last_updated>
  <languages>None</languages>
  <commits>None</commits>
</item>
```

Result 2: The output XML file (repos.xml) was created using the -O flag in the Scrapy command. This ensured that the file was overwritten cleanly, avoiding multiple XML headers or invalid structures that occur with appending (-o).

The output was verified using both text editors and browsers. A malformed version caused XML rendering issues due to repeated declarations. This was fixed by cleaning up the file and always overwriting it during re-runs.

```
> python -m scrapy crawl github -O repos.xml
```

Chapter 2 Conclusions

This project successfully demonstrated how to use Scrapy for structured web scraping. It revealed both the strengths of Scrapy (clean scraping pipeline, robust data structuring) and limitations (handling dynamic JavaScript content like GitHub's commit counters and languages). For future improvements:

- Using the GitHub API would allow access to complete repository metadata including commits, contributors, and languages.
- Integrating Selenium could allow scraping of JavaScript-rendered pages.
- Output can be extended to other formats like CSV or JSON for easier analysis.

Overall, the project built a solid foundation in scraping and automation with Python.