Midterm Project Report

Advanced Computer Programming

Student Name : Tselmeg Gantulga
Student ID : 113021201
Teacher : DINH-TRUNG VU

2025-04-14

Chapter 1: Introduction

1.1 GitHub

1. Personal GitHub Account: https://github.com/113021201
2. Group Project Repository: https://github.com/Jantsagdorj/ACP-AU-1132

1.2 Overview

This project uses **Scrapy**, a powerful Python web scraping framework, to extract repository data from a GitHub user profile. The goal was to collect the following details for all public repositories:

- Repository URL
- Description (About)
- Last Updated date
- Programming Languages used
- Number of Commits

To implement the scraper, I used the following advanced libraries and techniques:

1. **Scrapy**: A powerful web scraping library in Python. It provides efficient tools to interact with APIs, scrape data, and manage requests and responses.
2. **JSON**: To process the responses from GitHub's API, which returns data in JSON format.
3. **Callbacks**: The scraper was structured using multiple callback methods (parse(), parse_languages(), parse_commits()), which allowed me to handle different stages of data extraction for each repository.
4. **Meta Data**: I used Scrapy's meta parameter to pass data between requests, ensuring that repository details were retained throughout the scraping process.

Chapter 2: Implementation

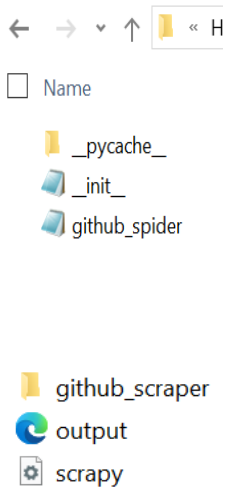
Setup / Environment Preparation

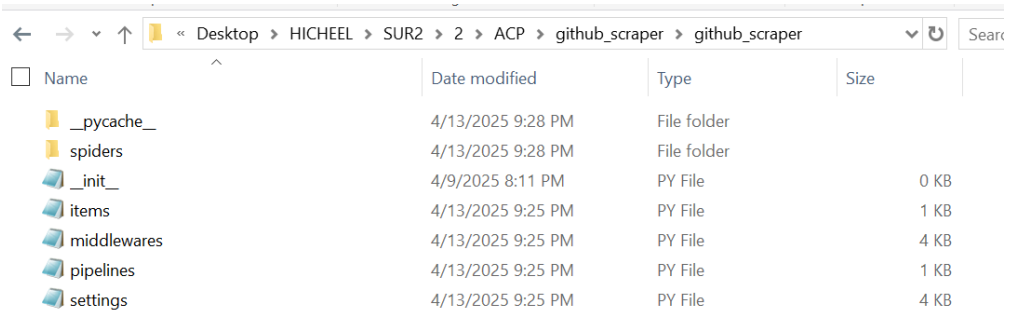
I initially began working on this project locally on my notebook, using the Windows Command Prompt. I created a Scrapy project folder and wrote all necessary Python files, including github_scraper.py.

Step 1: Install Required Library in the Command Prompt: **pip install scrapy**

Step 2: Create a Scrapy Project: **scrapy startproject github_scraper**
This created the following project structure:

```
github_scraper/
│
├── scrapy.cfg
└── github_scraper/
    ├── __init__.py
    ├── items.py
    ├── middlewares.py
    ├── pipelines.py
    ├── settings.py
    └── spiders/
        ├── __init__.py
        └── github_spider.py
```





Step3: Write the project main code in **github_spider.py.**
Step4: **scrapy crawl github_spider -o output.xml** — This executed my spider and saved the extracted GitHub repository data into an output.xml file, which included details such as the repository URL, about/description, last updated date, programming languages, and commit counts.

2.1 Class 1: GitHubAPI This class is the core of the scraping project, which connects to GitHub's API and retrieves repository data.

```
class GithubSpider(scrapy.Spider):
```

2.1.1 Fields

- name: The name of the spider (in this case, "github_spider").
- start_urls: The starting URL where the scraping begins, which is the GitHub API endpoint to retrieve repositories.

2.1.2 Methods

- parse(): This method is the main parsing function that processes the response from the start URL. It extracts basic information about each repository, such as the name, URL, description, and last updated time. It checks if the repository is empty and initiates requests to fetch languages and commit data if the repository is not empty.
- parse_languages(): This method is responsible for extracting the programming languages used in each repository by querying the GitHub API for each repository's language data.
- parse_commits(): This method handles the commit data for each repository. It fetches the number of commits associated with each repository by sending a request to the appropriate GitHub API endpoint.

2.2 Method Breakdown

2.2.1 Method 1: *parse()*

parse():. It reads the JSON response from the GitHub API (which contains a list of repositories) and prepares to gather more detailed data.

```
def parse(self, response):
    repos = json.loads(response.text)

    for repo in repos:
        repo_name = repo.get("name")
        html_url = repo.get("html_url")
        about = repo.get("description")
        updated_at = repo.get("updated_at")
        is_empty = repo.get("size", 0) == 0

        # If about is empty but repo is not, use repo name as about
        if not about and not is_empty:
            about = repo_name

        item = {
            "name": repo_name,
            "url": html_url,
            "about": about if about else "None",
            "last_updated": updated_at,
            "languages": None,
            "commits": None,
        }

        if not is_empty:
            languages_url = f"https://api.github.com/repos/113021201/{repo_name}/languages"
            commits_url = f"https://api.github.com/repos/113021201/{repo_name}/commits"

            request = scrapy.Request(
                url=languages_url,
                callback=self.parse_languages,
                meta={"item": item, "commits_url": commits_url}
            )
            yield request
        else:
            yield item
```

- **def parse(self, response):** This is the main parsing method automatically triggered by Scrapy when the spider hits a URL from start_urls.
- **repos = json.loads(response.text):** The response is in JSON format (a string), so this converts it into a Python list of dictionaries using json.loads().

- **for repo in repos:** Loops through each repository dictionary in the list.
  - Extracts specific values from the JSON: the name, URL, description, and last updated time of each repository.

```
repo_name = repo.get("name")
html_url = repo.get("html_url")
about = repo.get("description")
updated_at = repo.get("updated_at")
```

  - **is_empty = repo.get("size", 0) == 0** : Checks if the repository is empty (size = 0). If size is missing, it defaults to 0.
  - **if not about and not is_empty:** If there's no description but the repo isn't empty, **about = repo_name** it uses the repo name as the "about" value to avoid having a blank description.
  - Creates a dictionary (item) to store all the collected data. Some values are still None for now, because more requests are needed to get them.

```
item = {
    "name": repo_name,
    "url": html_url,
    "about": about if about else "None",
    "last_updated": updated_at,
    "languages": None,
    "commits": None,
}
```

  - **if not is_empty:** If the repo has content (not empty), then we continue to fetch more data.
  - **languages_url, commits_url:** Prepares two URLs to fetch programming languages and commits from the GitHub API.

```
languages_url = f"https://api.github.com/repos/113021201/{repo_name}/languages"
commits_url = f"https://api.github.com/repos/113021201/{repo_name}/commits"
```

    - **request = scrapy.Request():** Creates a new Scrapy request to get languages. The meta dictionary is used to pass data between methods.

```
request = scrapy.Request(
    url=languages_url,
    callback=self.parse_languages,
    meta={"item": item, "commits_url": commits_url}
)
```

  - **yield request:** Sends out the request for languages. This also suspends the current method until parse_languages finishes.
  - **else: yield item:** If the repo is empty, there's no need to fetch more data. So it directly yields the item.

2.2.2 Method: *parse_languages(self, response)*

This method gets the programming languages used in a specific (languages)repo.

```
def parse_languages(self, response):
    item = response.meta["item"]
    commits_url = response.meta["commits_url"]

    languages = list(json.loads(response.text).keys())
    item["languages"] = ", ".join(languages) if languages else "None"

    yield scrapy.Request(
        url=commits_url,
        callback=self.parse_commits,
        meta={"item": item}
    )
```

- **def parse_languages(self, response):** Callback function triggered when the languages URL is fetched.
    - This 2 line retrieves the partially completed item and commits the URL passed from the previous method.

      ```
      item = response.meta["item"]
      commits_url = response.meta["commits_url"]
      ```

    - **languages = list(json.loads(response.text).keys()):** Loads the response as a dictionary and extracts the keys (language names), converting them into a list.
    - **item["languages"] = ", ".join(languages) if languages else "None"** :Joins the language names into a string (e.g., "Python, HTML"). If the list is empty, sets it to "None".
    - **yield scrapy.Request():** Sends another request to get the commit data, passing the updated item along using meta.

📜 2.2.3 Method: parse_commits(self, response)

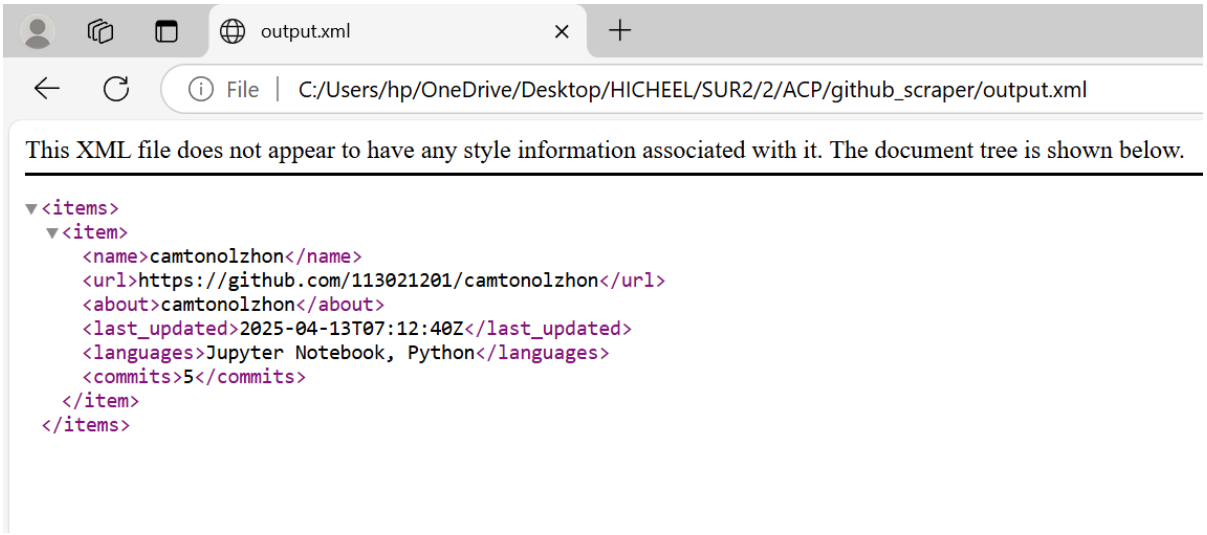
This is the final method that gets commit history and completes the data.

```
def parse_commits(self, response):
    item = response.meta["item"]
    commits = json.loads(response.text)
    item["commits"] = len(commits) if isinstance(commits, list) else "None"
    yield item
```

- **def parse_commits(self, response):** Callback function for handling the response from the commits API.
    - **item = response.meta["item"] :** Gets the current item passed from parse_languages.
    - **commits = json.loads(response.text):** Parses the JSON response, which is usually a list of commit objects.
    - **item["commits"] = len(commits) if isinstance(commits, list) else "None":** If the response is a valid list, it counts the number of commits. Otherwise, it sets it to "None".
    - **yield item:** It yields the completed item with all details filled in.

Chapter 3: Results

After running the Scrapy spider on my personal GitHub account, the scraper successfully retrieved detailed information from each of my public repositories. The spider followed a multi-step process: it first collected general repository metadata, then made additional requests to fetch the programming languages and commit histories for non-empty repositories.



Summary of Results

- **Total repositories scraped**: All public repositories linked to my username /113021201/.
- **Non-empty repositories**: Languages and commits were retrieved and included.
- **Empty repositories**: Only basic information (name, URL, last updated, etc.) was collected. No additional API requests were made for these to save time and resources.
- **Missing descriptions**: If a repository had no description, the scraper automatically used the repo name as a fallback for the "About" section.

Chapter 4: Conclusion

For this project, I built a GitHub scraper using Scrapy and the GitHub API to collect information about my repositories, like the URL, description, last updated date, languages, and number of commits. At first, I tried doing everything using only Scrapy to scrape GitHub's website directly. I even tried using just the requests library without Scrapy at all. But after spending a few days stuck, I realized GitHub hides some details like languages and commits history on the normal webpage. That's why those fields kept showing up as None even when the repo wasn't empty. Later, I found out that using the GitHub API was the best way to get all the information I needed. Once I combined Scrapy with the API, everything worked smoothly and I was able to finish the project and meet all the requirements.