Midterm Project Report

Advanced Computer Programming

Student Name : M Naufal Ihza Syahzada
Student ID : 112021223
Teacher : DINH-TRUNG VU

2024-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/NaufalIhza17
2) **Group Project Repository**: https://github.com/fadhlanharashta/ACP---Group-3

1.2 Overview

For this midterm project, in order to crawl a github repository page account I choose to use only Scrapy and of course with python for the programming language. Without any other advanced language feature and libraries I'm able to get almost all of the information I need inside this page https://github.com/NaufalIhza17. In this report you will see how I approach this problem using only Scrapy and what kind of problems I faced during the implementation.

Chapter 2 Implementation

2.1 Class

In this project I only made and use one class which is GithubSpider. Before we dive into the class, I import three things and those are scrapy itself, items that I define in the crawler project, and lastly urllib.parse from library urljoin. The crawler is built using several key classes, each serving a specific purpose in the scraping process. Below is a breakdown of these classes:

- GithubSpider (githubcrawler.py)
  - A Scrapy spider class responsible for crawling GitHub repositories for a given username.
  - Inherits from scrapy.Spider.

```python
import scrapy
from githubcrawler.items import GithubcrawlerItem
from urllib.parse import urljoin

class GithubSpider(scrapy.Spider):
    name = 'github'

    def __init__(self, username=None, *args, **kwargs):
        super(GithubSpider, self).__init__(*args, **kwargs)
        if username:
            self.start_urls = [f'https://github.com/{username}?tab=repositories']
        else:
            self.start_urls = ['https://github.com/yourusername?tab=repositories']

    def parse(self, response):
        # Extract all repository links
        repos = response.css('li.source div.d-inline-block div.mb-1 h3.wb-break-all a::attr(href)').getall()

        for repo in repos:
            repo_url = urljoin('https://github.com', repo)
            yield scrapy.Request(repo_url, callback=self.parse_repo)

        # Handle pagination
        next_page = response.css('div.paginate-container div.pagination a.next_page::attr(href)').get()
        if next_page:
            yield response.follow(next_page, callback=self.parse)

    def parse_repo(self, response):
        item = GithubcrawlerItem()

        # URL
        item['url'] = response.url

        # About - check description first, then repo name if empty
        about = response.css('div.Layout-sidebar div.BorderGrid-cell p.f4.my-3::text').get()
        if not about:
            about = response.css('strong.mr-2.flex-self-stretch a::text').get()
        item['about'] = about.strip() if about else None

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

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

        if not is_empty:
            languages = response.css('div.BorderGrid-cell ul.list-style-none li.d-inline a[href*="search"] span.text-bold::text').getall()
            item['languages'] = [lang.strip() for lang in languages] if languages else None
        else:
            item['languages'] = None
```

- GithubcrawlerItem (items.py)

- Defines the structure of the scraped data (items).
- Inherits from scrapy.Item.

```python
import scrapy


class GithubcrawlerItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()

    url = scrapy.Field()
    about = scrapy.Field()
    last_updated = scrapy.Field()
    languages = scrapy.Field()
    commits = scrapy.Field()

    pass

```

2.1.1 Fields

- GithubcrawlerItem (items.py)
  - This class defines the structure of the scraped data. It includes the following fields:
    - url (String): The URL of the GitHub repository.
    - about (String): A short description or the name of the repository.
    - last_updated (List): Timestamps indicating when the repository was last updated.
    - languages (List): Programming languages used in the repository.
    - commits (String): The number of commits made to the repository.

```python
url = scrapy.Field()
about = scrapy.Field()
last_updated = scrapy.Field()
languages = scrapy.Field()
commits = scrapy.Field()
```

- GithubSpider (githubcrawler.py)
  - This class contains the main logic for crawling GitHub. Its key fields include:
    - name = 'github': The name of the spider, used to run it via Scrapy.
    - start_urls: A dynamically generated list of URLs to start crawling (based on the provided GitHub username).

```
name = 'github'

def __init__(self, username=None, *args, **kwargs):
    super(GithubSpider, self).__init__(*args, **kwargs)
    if username:
        self.start_urls = [f'https://github.com/{username}?tab=repositories']
    else:
        self.start_urls = ['https://github.com/yourusername?tab=repositories']
```

2.1.2 Methods

- GithubSpider Methods (githubcrawler.py)
  - __init__(self, username=None, *args, **kwargs)
    - Initializes the spider with an optional GitHub username. If no username is provided, it defaults to a placeholder.
  - parse(self, response)
    - Processes the main repositories page, extracts repository links, and follows pagination if available.
  - parse_repo(self, response)
    - Extracts detailed repository information (description, languages, commits, etc.) and yields a structured GithubcrawlerItem.

2.1.3 Functions

The code doesn't define standalone functions. All logic is encapsulated within class methods.

2.2 Method/Function

2.2.1 parse(self, response)

- Input: An HTTP response from the initial GitHub repositories page.
- Process:
  - Uses CSS selectors (li.source div.d-inline-block ...) to extract repository links.
  - Constructs full URLs using urljoin and yields new scrapy.Request objects for each repository.
  - Checks for a "next page" link and follows it if available.
- Output: Generates requests for each repository page.

```
def parse(self, response):
    # Extract all repository links
    repos = response.css('li.source div.d-inline-block div.mb-1 h3.wb-break-all a::attr(href)').getall()

    for repo in repos:
        repo_url = urljoin('https://github.com', repo)
        yield scrapy.Request(repo_url, callback=self.parse_repo)

    # Handle pagination
    next_page = response.css('div.paginate-container div.pagination a.next_page::attr(href)').get()
    if next_page:
        yield response.follow(next_page, callback=self.parse)
```

2.2.2 parse_repo(self, response)

- Input: An HTTP response from a single repository page.
- Process:
  - Checks if the repository is empty (div.Blankslate).
  - Extracts metadata:
    - Description: Falls back to the repository name if no description exists.
    - Languages: Fetches from a list under the "Languages" section.
    - Commits: Extracts from the commits button (if available).
  - Populates a GithubcrawlerItem with the scraped data.
- Output: Yields a structured item containing repository details.

```python
def parse_repo(self, response):
    item = GithubcrawlerItem()

    # URL
    item['url'] = response.url

    # About - check description first, then repo name if empty
    about = response.css('div.Layout-sidebar div.BorderGrid-cell p.f4.my-3::text').get()
    if not about:
        about = response.css('strong.mr-2.flex-self-stretch a::text').get()
    item['about'] = about.strip() if about else None

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

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

    if not is_empty:
        languages = response.css('div.BorderGrid-cell ul.list-style-none li.d-inline a[href*="search"] span.text-bold::text').getall()
        item['languages'] = [lang.strip() for lang in languages] if languages else None
    else:
        item['languages'] = None

    if not is_empty:
        commits_link = response.css('a[href*="commits"] span[data-component="buttonContent"] span[data-component="text"] span.fgColor-default::text').get()
        item['commits'] = commits_link.strip() if commits_link else None
    else:
        item['commits'] = None

    yield item
```

2.2.3 process_item(self, item, spider)

- Input: A scraped GithubcrawlerItem.
- Process: Currently, it simply returns the item without modifications.
- Future Use: Can be extended to clean data, store in a database, or filter unwanted entries.

```python
class GithubcrawlerItem(scrapy.Item):
    # define the fields for your item here like:
    # name = scrapy.Field()

    url = scrapy.Field()
    about = scrapy.Field()
    last_updated = scrapy.Field()
    languages = scrapy.Field()
    commits = scrapy.Field()

    pass
```

Chapter 3 Results

3.1 Result

3.1.1 Execution Command

The crawler was executed using the following command to scrape repositories from the GitHub user Naufalihza17 and export the results to an XML file:

```
scrapy crawl github -a username='Naufalihza17' -o result.xml
```

3.1.2 Output Format

The crawler generates structured data in XML format, with each repository's metadata stored as an item with the following fields:

- <url>: Full URL of the repository (e.g., https://github.com/Naufalihza17/repo-name).
- <about>: Repository description or name (e.g., "A project for testing Scrapy").
- <last_updated>: List of timestamps in ISO format (e.g., <value>2023-10-15T12:34:56Z</value>).
- <languages>: List of programming languages (e.g., <value>Python</value><value>JavaScript</value>).
- <commits>: Number of commits (e.g., "42 commits").

3.1.3 Output (result.xml)

3.1.4 Behavior

- The spider starts from https://github.com/NaufalIhza17?tab=repositories.
- Iterates through all repositories, including paginated results.
- Skips empty repositories (no languages/commits detected).
- Exports all valid data to result.xml in the project directory.

3.1.5 Validation

- The XML output was verified for:
    - Correctness: All fields match the scraped data.
    - Structure: Hierarchical nesting of <item>, <value>, etc.
    - Completeness: No missing fields for non-empty repositories.

.

Chapter 4 Conclusions

4.1 Summary

- The crawler successfully extracts repository metadata from GitHub profiles.
- It handles pagination, empty repositories, and multiple data points (languages, commits, etc.).

4.2 Limitations

- No Rate-Limit Handling: May get blocked if too many requests are sent.
- No Authentication: Cannot access private repositories.
- Basic Pipeline: Currently, scraped data is not stored or processed further.

4.3 Future Improvements

- Rate Limiting: Add delays between requests or use proxies.
- Data Storage: Extend the pipeline to save data in a database (e.g., PostgreSQL, MongoDB).
- Error Handling: Improve robustness for failed requests or CAPTCHAs.
- More Metadata: Extract stars, forks, contributors, etc.