亞洲大學
ASIA UNIVERSITY

Midterm Project Report

Advanced Computer Programming

Student Name : Dimas Ahmad Fahreza

Student ID : 113021199

Teacher : DINH-TRUNG VU

2025-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/dimasfahrza
2) **Group Project Repository**: https://github.com/113021198/ACP/tree/main

1.2 Overview

In this project, Scrapy was used as the main framework for web scraping tasks. Scrapy provides a structured and efficient way to define crawlers and manage requests. CSS selectors were employed to navigate and extract HTML elements representing the desired repository data. The crawler was customized to simulate human behavior using a specific user-agent string and a slight delay between requests to avoid triggering GitHub's anti-bot mechanisms.

The extracted data includes the repository URL, a short description (or the repository name if no description exists), the last updated timestamp, a list of languages with usage percentages, and the total number of commits. These elements provide a comprehensive overview of each repository's metadata. The results are stored in my_github_output.xml, an XML file generated by Scrapy's built-in export system.

Implementation

1.1 Class 1

MyGithubRepoSpider is the main class that defines the spider behavior. It inherits from scrapy.Spider and manages the crawling and data extraction process

1.1.1 Fields

The class defines several important fields:

```
name = "github_spider"
    allowed_domains = ["github.com"]
    start_urls = ["https://github.com/dimasfahrza?tab=repositories"]
```

These fields specify the spider's identifier, the domain to stay within, and the initial URL to begin scraping from.

1.1.2 Methods

The custom_settings dictionary is defined within the class to customize behavior such as output format, user agent string, and download delay:

```
custom_settings = {
        'FEED_FORMAT': 'xml',
        'FEED_URI': 'my_github_output.xml',
        'FEED_EXPORT_ENCODING': 'utf-8',
        'USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)',
        'ROBOTSTXT_OBEY': False,
        'DOWNLOAD_DELAY': 1.5,
        'CONCURRENT_REQUESTS': 1,
    }
```

1.1.3 Functions 1

The function parse(self, response) handles the first level or parsing. It selects each repository listed on the user’s GitHub profile, collects basic information such as URL, repository name, description, and last updated timestamp, then sends a new request to the repository page for more detailed data.

```
    def parse(self, response):
        self.logger.info(f"🔍 Looking through: {response.url}")

        repo_blocks = response.css('li[itemprop="owns"], div.Box-row, 
div[data-test-id="repository-list-item"]')

        for block in repo_blocks:
            repo_href = block.css('a[itemprop="name 
codeRepository"]::attr(href)').get() \
                        or 
block.css('a[data-test-id="repository-link"]::attr(href)').get()
            repo_url = response.urljoin(repo_href) if repo_href else 
None

            repo_name = (block.css('a[itemprop="name 
codeRepository"]::text').get() or 

block.css('a[data-test-id="repository-link"]::text').get() or 
"").strip()

            repo_about = 
block.css('p[itemprop="description"]::text').get() \
                        or 
block.css('p[data-test-id="repository-description"]::text').get()
            repo_about = repo_about.strip() if repo_about else 
repo_name

            last_edit_time = 
block.css('relative-time::attr(datetime)').get()
```

```
            scraped_data = {
                'url': repo_url,
                'about': repo_about,
                'last_updated': last_edit_time
            }

            if repo_url:
                yield response.follow(
                    repo_url,
                    callback=self.parse_repo_details,
                    meta={'repo_data': scraped_data}
                )
            else:
                scraped_data.update({'languages': None, 'commits':
None})
                yield scraped_data
```

1.1.4 Function 2

This function is responsible for visiting each individual repository and extracting more specific information, such as programming languages and commit count. It also handles the case where the repository may be empty.

```
def parse_repo_details(self, response):
        repo_data = response.meta['repo_data']
        self.logger.info(f"➡️  Entering repo: {repo_data['url']}")

        is_repo_empty = bool(response.css('.blankslate,
.BlankState'))

        if is_repo_empty:
            repo_data.update({'languages': None, 'commits': None})
        else:
            # Get all programming languages used
            lang_items = response.css('li.d-inline
span::text').getall()
            langs = [lang.strip() for lang in lang_items if
lang.strip()]
            repo_data['languages'] = langs if langs else None

            # Try to get commit count
            commit_text = response.css('a[href*="commits"]
span::text').get() \
                        or
response.css('strong[data-test-id="commits"]::text').get()

            if commit_text:
                commit_count = commit_text.strip().replace(",", "")
                if "Commit" not in commit_count:
                    commit_count += " Commits"
                repo_data['commits'] = commit_count
            else:
                repo_data['commits'] = "None"

        yield repo_data
```

This structure separates the scraping into two main responsibilities: parsing the list of repositories and parsing the details of each repository. It is modular and clean, allowing for easy debugging and scalability.

Chapter 2 Results

The spider successfully scraped multiple repositories from my GitHub account https://github.com/dimasfahrza. For example, the repository quiz1 was found to use HTML, CSS, and JavaScript, while the repository midterm was identified to use PHP and Blade. These repositories were not empty and included the appropriate number of commits and languages used.

The resulting data was stored in a structured XML file named my_github_output.xml, which contains entries like the following:

```
<items>
  <item>
    <url>https://github.com/dimasfahrza/midterm</url>
    <about>midterm</about>
    <last_updated>2023-10-25T16:36:34Z</last_updated>
    <languages>
      <value>PHP</value>
      <value>62.0%</value>
      <value>Blade</value>
      <value>37.6%</value>
      <value>Other</value>
      <value>0.4%</value>
    </languages>
    <commits>1 Commit</commits>
  </item>
  <item>
    <url>https://github.com/dimasfahrza/quiz1</url>
    <about>quiz1</about>
    <last_updated>2023-09-28T09:46:13Z</last_updated>
    <languages>
      <value>CSS</value>
      <value>58.2%</value>
      <value>HTML</value>
      <value>41.1%</value>
      <value>JavaScript</value>
      <value>0.7%</value>
    </languages>
    <commits>1 Commit</commits>
  </item>
</items>
```

.

Chapter 3 Conclusions

This midterm project demonstrates the ability to design and implement a web scraper using the Scrapy framework. It covers key concepts such as request handling, HTML parsing with CSS selectors, and structured data export. The scraper was made robust through fallback mechanisms for missing data and polite scraping behavior using headers and delays. From this project, we can collect our GitHub basic information such as URL, repository name, description, and last updated timestamps. As a result, the program was able to collect comprehensive data on GitHub repositories for further study or analysis