Midterm Project Report

Advanced Computer Programming

Student Name : RADIAN TRY DARMAWAN
Student ID : 113021220
Teacher : DINH-TRUNG VU

2025-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/radiandrmwn
2) **Group Project Repository**: https://github.com/113021198/ACP/tree/main/midterm/radian-try

1.2 Overview

This Midterm project implements a GitHub repository scraper using the Scrapy framework in Python. The scraper extracts detailed information about all repositories belonging to a specified GitHub user. The implementation leverages several advanced Python language features and libraries:

- **Scrapy Framework**: Used in the process of creating the web crawler that navigates through GitHub pages
- **XML Processing**: Utilized to format and output the scraped data in a structured XML format
- **CSS Selectors**: Used for precise targeting of HTML elements containing repository data

The program successfully extracts repository information including URL, description, and last updated date. Data is exported as a well-structured XML file that can be easily parsed and analyzed further.

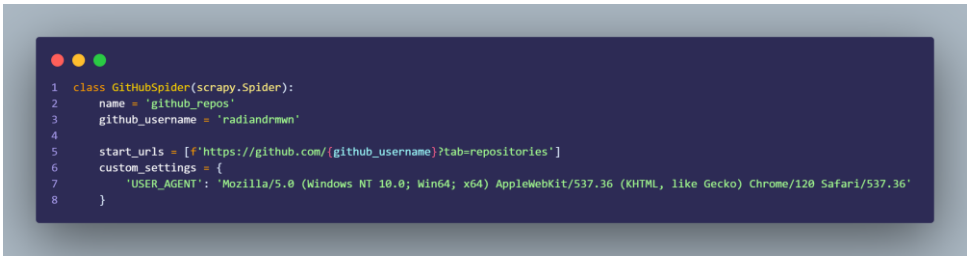
Chapter 2 Implementation

2.1 Class 1

2.1.1 The main spider class that inherits from scrapy(github_repos,py) and handles the crawling and data extraction process.

2.1.2 Fields

This snippet of the code is where we put our GitHub repositories adress



- name: Identifier for the spider, set to 'github_repos'
- github_username: GitHub username to scrape repositories from, set to 'radiandrmwn'
- start_urls: Initial URL to begin scraping from, constructed using the username

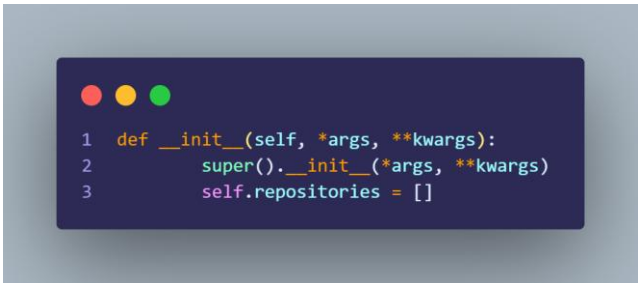
This is important as it is pointing the GitHub address to start the scraping process

2.1.3 Methods

init(self, *args, **kwargs)

Initializes the spider by:

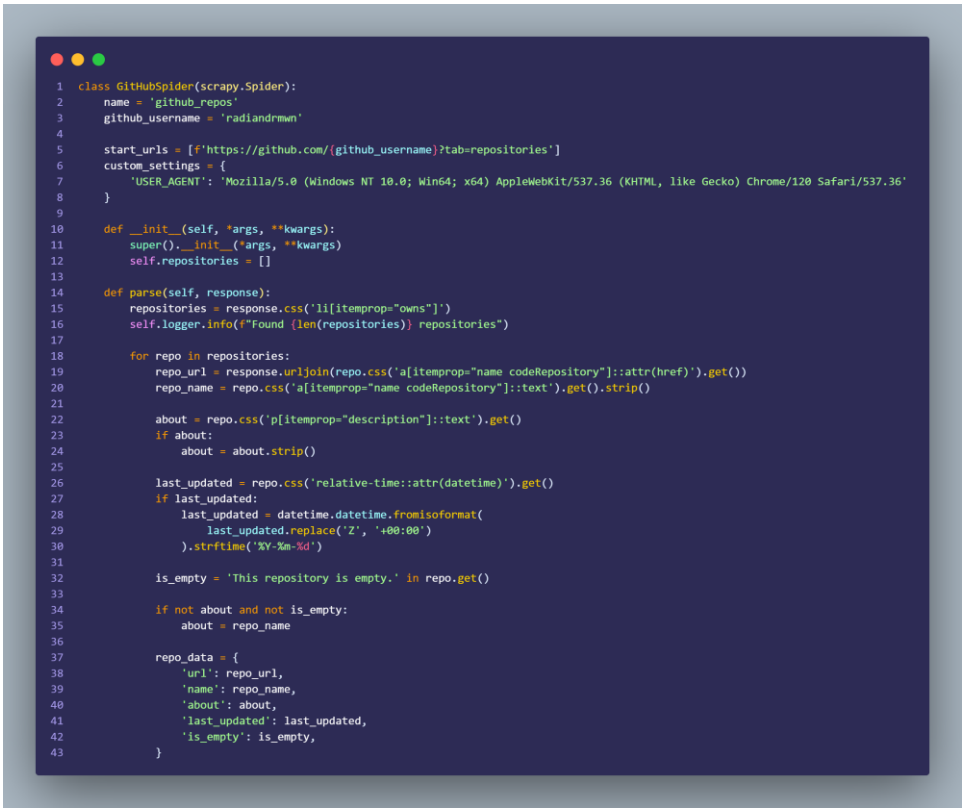
- Calling the parent class constructor
- Setting up an empty list to store repository data



parse(self, response)

The entry point method that processes the repository list page:

- Extracts basic information from each repository listing
- Determines if repositories are empty
- Creates follow-up requests for each repository to gather detailed information
- Handles pagination to crawl through all repository pages



```python
class GitHubSpider(scrapy.Spider):
    name = 'github_repos'
    github_username = 'radiandrmwn'

    start_urls = [f'https://github.com/{github_username}?tab=repositories']
    custom_settings = {
        'USER_AGENT': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120 Safari/537.36'
    }

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.repositories = []

    def parse(self, response):
        repositories = response.css('li[itemprop="owns"]')
        self.logger.info(f"Found {len(repositories)} repositories")

        for repo in repositories:
            repo_url = response.urljoin(repo.css('a[itemprop="name codeRepository"]::attr(href)').get())
            repo_name = repo.css('a[itemprop="name codeRepository"]::text').get().strip()

            about = repo.css('p[itemprop="description"]::text').get()
            if about:
                about = about.strip()

            last_updated = repo.css('relative-time::attr(datetime)').get()
            if last_updated:
                last_updated = datetime.datetime.fromisoformat(
                    last_updated.replace('Z', '+00:00')
                ).strftime('%Y-%m-%d')

            is_empty = 'This repository is empty.' in repo.get()

            if not about and not is_empty:
                about = repo_name

            repo_data = {
                'url': repo_url,
                'name': repo_name,
                'about': about,
                'last_updated': last_updated,
                'is_empty': is_empty,
            }
```

```python
    if not is_empty:
        yield scrapy.Request(
            url=repo_url,
            callback=self.parse_repository_details,
            meta={'repo_data': repo_data}
        )
    else:
        repo_data['languages'] = None
        repo_data['commits'] = None
        self.repositories.append(repo_data)

next_page = response.css('a.next_page::attr(href)').get()
if next_page:
    yield response.follow(next_page, callback=self.parse)
```

2.1.4 Functions

parse_repository_details(self, response)

This method handles the processing of individual repository pages:

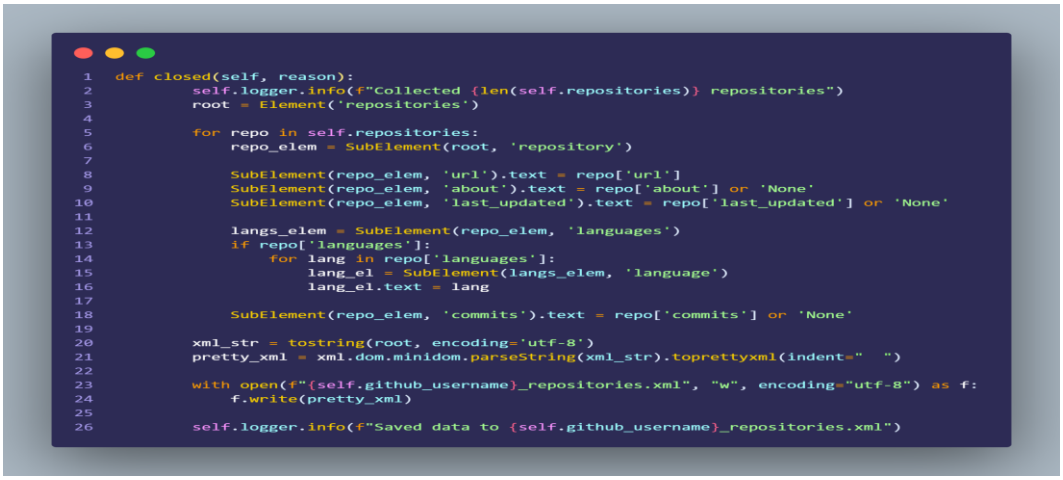
- Extracts programming languages used in the repository
- Extracts the number of commits using regular expressions
- Adds the detailed information to the repository data dictionary
- Stores the complete repository data for final output

```python
def parse_repository_details(self, response):
        repo_data = response.meta['repo_data']

        # Languages
        lang_data = response.css('.repository-content .Layout-sidebar span[itemprop="programmingLanguage"]::text').getall()
        if not lang_data:
            lang_data = ['None']
        repo_data['languages'] = lang_data

        # Commits
        commits_text = response.css('li.Commits .d-none.d-sm-inline::text').get()
        if commits_text:
            commits = re.search(r'(\d+)', commits_text)
            repo_data['commits'] = commits.group(1) if commits else '0'
        else:
            repo_data['commits'] = '0'

        self.repositories.append(repo_data)
```

closed(self, reason)

A special method that is called when the spider finishes crawling:

- Creates an XML structure to store all repository data
- Formats the data with proper indentation for readability
- Writes the formatted XML to a file named after the GitHub username
- Logs the completion of the data saving process



```python
def closed(self, reason):
        self.logger.info(f"Collected {len(self.repositories)} repositories")
        root = Element('repositories')

        for repo in self.repositories:
            repo_elem = SubElement(root, 'repository')

            SubElement(repo_elem, 'url').text = repo['url']
            SubElement(repo_elem, 'about').text = repo['about'] or 'None'
            SubElement(repo_elem, 'last_updated').text = repo['last_updated'] or 'None'

            langs_elem = SubElement(repo_elem, 'languages')
            if repo['languages']:
                for lang in repo['languages']:
                    lang_el = SubElement(langs_elem, 'language')
                    lang_el.text = lang

            SubElement(repo_elem, 'commits').text = repo['commits'] or 'None'

        xml_str = tostring(root, encoding='utf-8')
        pretty_xml = xml.dom.minidom.parseString(xml_str).toprettyxml(indent="  ")

        with open(f"{self.github_username}_repositories.xml", "w", encoding="utf-8") as f:
            f.write(pretty_xml)

        self.logger.info(f"Saved data to {self.github_username}_repositories.xml")
```

Chapter 3 Results

The spider generates an XML file named after the GitHub username (e.g., radiandrmwn_repositories.xml). The XML file contains a structured representation of all repositories and their details

```xml
<?xml version="1.0" ?>
<repositories>
  <repository>
    <url>https://github.com/radiandrmwn/radiandrmwn</url>
    <about>Config files for my GitHub profile.</about>
    <last_updated>2022-08-28</last_updated>
    <languages>
      <language>None</language>
    </languages>
    <commits>0</commits>
  </repository>
  <repository>
    <url>https://github.com/radiandrmwn/QUIZ1_Webpro_Radian-Try-Darmawan</url>
    <about>QUIZ1_Webpro_Radian-Try-Darmawan</about>
    <last_updated>2023-09-26</last_updated>
    <languages>
      <language>None</language>
    </languages>
    <commits>0</commits>
  </repository>
```

For each repository, the following information is successfully extracted:

- **URL**: Direct link to the repository
- **About**: The description of the repository or its name if no description exists
- **Last Updated**: The date when the repository was last modified

This data provides a comprehensive overview of a GitHub user's repositories, allowing for analysis of their programming preferences, activity level, and project topics.

Chapter 4 Conclusions

This project successfully implements a web scraper for GitHub repositories using the Scrapy framework. The scraper is designed to be robust, handling edge cases such as empty repositories and missing information. It also implements polite scraping practices by using a realistic user agent string to avoid being blocked by GitHub's anti-bot measures.

The project demonstrates the power of Python's ecosystem for web scraping tasks, showcasing how relatively few lines of code can extract significant amounts of structured data from complex websites. The resulting data can be used for various purposes, such as portfolio analysis, tracking project progress, or studying programming language trends among GitHub users.