

****

**Midterm Project Report**

**Advanced Computer Programming**

**Student Name : Bhoomika S Horapeti**

**Student ID : 113710077**

**Teacher : DINH-TRUNG VU**

**2024-04**

# Introduction

## GitHub

1. **Personal GitHub Account**: https://github.com/113710077
2. **Group Project Repository**: https://github.com/113710077/Group-1

## Overview

This project is centered around utilizing Scrapy, a powerful Python-based web crawling and scraping framework, to collect structured data from a GitHub user profile. The specific case studied involves the public GitHub profile of user 113710077, which hosts two public repositories. The overarching objective is to programmatically gather metadata from each repository, including repository URL, description ('About'), last updated date, programming languages used, and the number of commits. This data is then exported to an XML format for further processing, visualization, or analysis.

Web scraping using frameworks like Scrapy is an advanced programming technique used extensively in data-driven applications. Scrapy enables asynchronous crawling, automatic request handling, and clean data extraction through a powerful selector syntax. Throughout this project, the technical focus remained on not only implementing a working scraper but also refining logic to adapt to different data conditions, managing missing values, and structuring outputs for XML serialization.

**1.3 Objective**

The primary objectives of this project were:

* To gain practical experience in using Scrapy for web scraping.
* To extract specific information from GitHub repositories such as:
  + Repository URL
  + Repository description ('About')
  + Last updated timestamp
  + Languages used in the repository
  + Number of commits in the repository
* To export the scraped data to an XML format.
* To manage real-world issues such as incomplete or missing fields in web data.
* To apply Pythonic techniques and best practices for scalable and readable code.

## 2. Implementation:

1. **Environment and Project Setup:**
   1. Scrapy was installed using pip

python3 -m pip install scrapy

* 1. A new project named midterm was initiated

scrapy startproject midterm

* 1. The default directory structure generated by Scrapy included folders for spiders, items.py, pipelines.py, and settings.py.
  2. A spider named repospider.py was created inside the midterm/spiders/ directory.

1. **Item Definition (midterm/items.py):**

This file contains the data structure that defines what we want to extract from GitHub.

import scrapy

class GithubScraperItem(scrapy.Item):

repo\_name = scrapy.Field()

url = scrapy.Field()

about = scrapy.Field()

last\_updated = scrapy.Field()

languages = scrapy.Field()

number\_of\_commits = scrapy.Field()

1. **Spider Implementation (midterm/spiders/repospider.py):**

import scrapy

from midterm.items import GithubScraperItem

class RepospiderSpider(scrapy.Spider):

name = "repospider"

allowed\_domains = ["api.github.com"]

start\_urls = ["<https://api.github.com/users/113710077/repos>"]

def parse(self, response):

repos = response.json()

for repo in repos:

item = GithubScraperItem()

item['repo\_name'] = repo.get('name')

item['url'] = repo.get('html\_url')

item['about'] = repo.get('description') or repo.get('name')

item['last\_updated'] = repo.get('updated\_at')

item['languages'] = None # We'll get this separately

item['number\_of\_commits'] = None # We'll also get this later

# Fetch languages

languages\_url = repo.get('languages\_url')

commits\_url = repo.get('commits\_url').replace('{/sha}', '')

yield scrapy.Request(

url=languages\_url,

callback=self.parse\_languages,

meta={'item': item, 'commits\_url': commits\_url}

)

def parse\_languages(self, response):

item = response.meta['item']

commits\_url = response.meta['commits\_url']

langs = response.json()

item['languages'] = list(langs.keys()) if langs else None

# Now get commit count

yield scrapy.Request(

url=commits\_url,

callback=self.parse\_commits,

meta={'item': item}

)

def parse\_commits(self, response):

item = response.meta['item']

commits = response.json()

item['number\_of\_commits'] = len(commits) if isinstance(commits, list) else None

yield item

1. **Running the Spider:**

After writing the spider, the scraper is executed with:

scrapy crawl repospider -o output.xml

This command runs the spider and exports the scraped data into output.xml in a structured format.

 





## 3. Results

* The spider successfully scraped metadata from both public repositories of user 113710077.
* Data fields such as description, languages, and number of commits were accurately captured or handled gracefully when missing.
* The exported XML file had clean and structured records for easy readability and future use.
* Below is an snippet from the output.xml file:

<?xml version="1.0" encoding="utf-8"?>

<items>

</items><?xml version="1.0" encoding="utf-8"?>

<items>

<item><repo\_name>Group-1</repo\_name>

<url>https://github.com/113710077/Group-1</url>

<about>Group repository for Advance Computer Programming course project reports – midterm and final documentation.</about>

<last\_updated>2025-04-16T06:51:01Z</last\_updated>

<languages>None</languages>

<number\_of\_commits>2</number\_of\_commits></item>

<item><repo\_name>advancedpython3\_2nd</repo\_name><

url>https://github.com/113710077/advancedpython3\_2nd</url>

<about>Materials supporting the Advanced Guide to Python 3 Programming 2nd Edition</about>

<last\_updated>2025-04-16T05:59:00Z</last\_updated>

<languages><value>Python</value></languages>

<number\_of\_commits>30</number\_of\_commits></item>

</items>

**4. Challenges and Solutions**

* Scrapy Installation Issues: Initially, there were issues with command recognition (scrapy: command not found). This was resolved by updating PATH variables and ensuring Scrapy was installed for the correct Python version.
* Import Errors: Import paths like from ..items caused confusion, later corrected to from midterm.items based on Scrapy project structure.
* CSS Selector Accuracy: Some fields such as description or commit count were not always available. By adding conditionals, the spider could bypass these gracefully without crashing.
* XML Export Format Issues: Improper spider or item configuration led to empty XML files initially. Debugging helped correct logic and selectors.

1. **Advanced Python Techniques Used**

* Scrapy Framework Architecture: Modular architecture using items, spiders, pipelines, and settings.
* Selectors: Advanced CSS selectors and chaining to accurately locate HTML elements.
* Conditional Parsing Logic: Used if, elif, and fallback conditions to ensure robustness.
* Data Serialization: Automatically converted complex Python structures to XML.
* Directory Navigation and Command Line Interface (CLI): Leveraged terminal commands for project setup and spider execution.
* Error Handling: Managed missing data and empty states using conditionals and None values.
* Project Scalability: Project is scalable to scrape more users or repositories by modifying the start\_urls.

**6. Conclusion**

This GitHub scraper project served as a valuable introduction to real-world web scraping using Scrapy. It offered deep insights into how websites structure their content and how automation tools can extract meaningful data at scale. From setting up Scrapy to dealing with incomplete HTML structures, this exercise demonstrated the relevance of Python in data acquisition, cybersecurity research, and AI-IoT applications.

Furthermore, the experience highlighted not only programming skills but also logical reasoning, problem-solving, and patience—essential qualities in any data science or software engineering role. This project has laid the foundation for more advanced scraping tasks and API-based data interactions in future endeavors.

.