



**Midterm Project Report**

**Advanced Computer Programming**

**Student Name : Munkhtselmeg**

**Student ID : 113021193**

**Teacher : DINH-TRUNG VU**

**2024-04-14**

# Introduction

## Github

1. **Personal Github Account**: your personal Github Account.
2. **Group Project Repository**: your group’s Project Repository

## Overview

In this project, I built a web scraping tool to collect detailed information about repositories from a GitHub user's profile. The main goal was to efficiently gather key data like repository URLs, descriptions, languages used, and the number of commits and organize it in a clean, structured way.

To make this happen, I used Python and the Scrapy framework, which made handling requests, navigating through GitHub pages, and extracting data much easier. I combined both XPath and CSS selectors to target the specific parts of each page using XPath especially for dynamic data like commit counts, and CSS for things like URLs, descriptions, and timestamps.

I also took advantage of Scrapy’s meta feature to pass information between different steps of the scraping process, which kept things well-organized. On top of that, I used regular expressions to handle inconsistencies in how the data is formatted for example, pulling out commit numbers and filtering out the languages used in each repo.

Additionally, the project was a great exercise in using advanced Python tools to automate data collection and make sense of structured data on the web.

<https://github.com/trungvdhp>.

# Implementation

## Class 1

In this project, I used a repository scraper class to handle the heavy lifting of crawling through a GitHub user's public repositories. This class was responsible for collecting all the important details — things like the repository URL, a brief description, the programming languages used, the last time each repo was updated, and the number of commits. After gathering all that data, it automatically organized everything into a clean and well-structured XML file, making it easy to store, read, or use for any further analysis.

### Fields

In this project, I used a custom class called GithubScraperItem to neatly store the data I pulled from each GitHub repository. It worked like a container that kept all the important bits of information organized and easy to manage. Each instance of this class held the following details:

* **url** – The link to the repository on GitHub.
* **about** – A short description of what the repository is about.
* **last\_updated** – A timestamp showing when the repository was last updated.
* **languages** – A list of programming languages used in the repo.
* **commits** – The total number of commits made to the repository.

This structure made it much easier to handle the scraped data and prepare it for exporting into a clean XML format.

### Methods

The GithubScraperItem class plays a simple but essential role in the scraping workflow. It doesn’t include any complex logic or behavior—instead, it acts as a structured data model designed specifically to hold the information gathered from each repository. By defining clear fields for each piece of data, this class helps keep everything organized as the scraper collects information across multiple GitHub pages. It serves as a reliable blueprint for storing repository details like URLs, descriptions, update timestamps, programming languages, and commit counts, making the data easy to process, export, or analyze later on.

### Functions

The GithubScraperItem class does not include any special functions on its own. Instead, it works together with Scrapy’s framework to manage and organize the data. This class is mainly used to hold the information that is collected during the scraping process. It stores the data temporarily while it is being passed through different parts of the Scrapy spider before it is saved or exported.

## Class 2 GithubReposodorySpider

I used **GithubRepositorySpider** class is built with Scrapy and is responsible for collecting repository data from a GitHub user's profile. It manages how web requests are sent, how responses are processed, and how key information is extracted during the scraping process.

## Method/Function parse(self, response)

The parse method acts as the starting point for the spider when it first lands on the GitHub profile page. It looks through the list of repositories using CSS selectors to grab basic details like URLs and descriptions. For each repository found, it sends out a new request to visit that specific repo’s page. While doing so, it also passes along some info using Scrapy’s meta feature so the next method has context to work with and can collect even more detailed data.

## Method/Function 2 scrapy.Request()

The scrapy.Request() function is used to tell the spider, “Hey, go visit this URL and do something with the response.” In this project, it's used to jump from the main GitHub profile page to each individual repository page. Along with the request, you can attach a callback function that handles the response (like parse\_repo), and you can pass extra data using the meta dictionary so it doesn’t get lost between requests. It’s basically how Scrapy moves around and carries information between pages.\

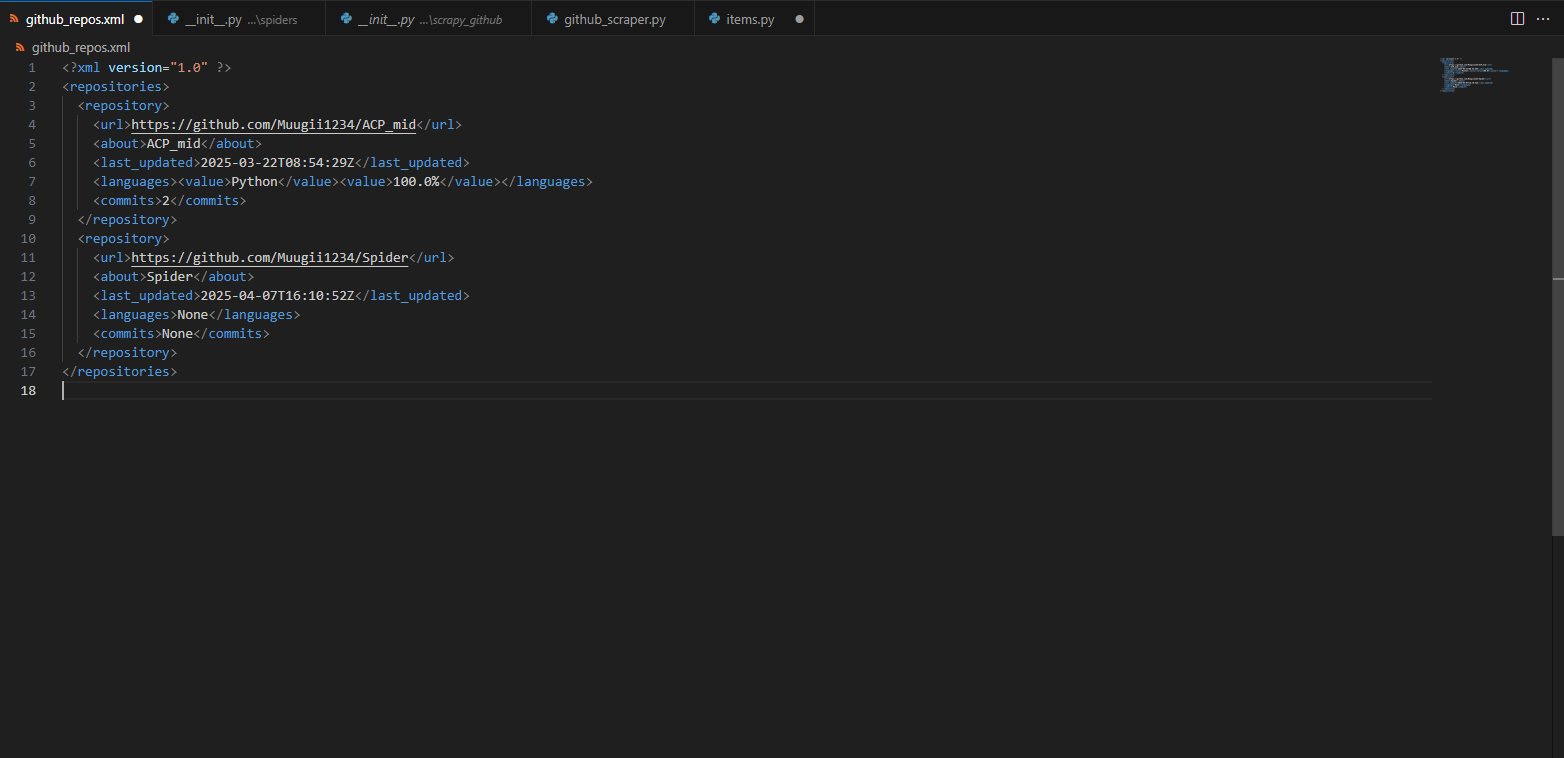
## Method/Function 3

In my project, I used the ET.SubElement() function from Python’s xml.etree.ElementTree module to help build the final XML file. I used it inside the loop where each repository’s data is being processed. Basically, this method let me create XML tags like <url>, <about>, <last\_updated>, and so on, under a main <repository> tag. It was a simple and clean way to structure all the scraped data, so the final output file is organized and easy to read or work with later.

.

# Results

The output of my web scraping program is structured in XML format, with each repository tag representing a repository scraped from the specified GitHub user profile. The program successfully extracted the relevant information from each repository, demonstrating the accuracy and functionality of the implemented spider.  
 In the current result, the scraper collected data from two repositories:



The result confirms that the scraper functions as expected, accurately navigating to the user's GitHub profile, parsing the necessary data from both the main repositories page and the individual repository pages and outputting it in a clean, structured XML format. This format is suitable for further processing, analysis, or integration with other systems and final results shown like this.

# Conclusions

In this project, I developed a GitHub repository scraper using the Scrapy framework in Python. The goal was to automate the extraction of key repository information such as URLs, descriptions, last updated dates, programming languages, and the number of commits from a GitHub user's profile.

The scraper navigates through the GitHub profile, gathers repository data, and processes it by sending requests to the appropriate GitHub API endpoints. It efficiently handles different scenarios, including empty repositories, and formats the extracted information into an organized XML structure for easy reuse and further analysis.

The project showcases the power of the Scrapy framework for automating data extraction tasks. By applying Python programming techniques, the scraper can be extended to scrape additional fields or customized for different GitHub profiles. Overall, this project demonstrates how web scraping can be effectively utilized to collect and manage repository data for practical purposes like data analysis and integration into larger systems.