

****

**Midterm Project Report**

**Advanced Computer Programming**

**Student Name : Edgar Joshua Effendi**

**Student ID : 113021147**

**Teacher : DINH-TRUNG VU**

**2024-04**

# Introduction

GitHub

Personal Github Account: https://github.com/113021147-EdgarJoshua

Group Project Repository: https://github.com/GH-Team-Duck/MidTerm\_Project/tree/main/midterm

# Overview

This project involved creating a web scraper using Python and the Scrapy framework to extract information about a specified user's public repositories from GitHub. Advanced language features and libraries utilized include Scrapy for asynchronous web crawling and scraping, CSS and XPath selectors for HTML parsing, regular expressions (re) for data extraction (like commit counts), Python's datetime module for handling timestamps, and Scrapy Items for structured data output. The program successfully extracts repository details such as URL, About description, primary languages used (with percentages), the total number of commits, and the last updated time. The collected data is then organized and exported into an XML file for further use. Logic was implemented to handle cases where repository descriptions might be missing and to differentiate between empty repositories and those with content but no description. The spider also includes basic pagination support and respects robots.txt rules (though this needed adjustment during development).

*(Self-Correction Example based on our discussion)*: Initially, the spider adhered strictly to robots.txt (ROBOTSTXT\_OBEY = True), which prevented scraping the target repository listing pages disallowed by GitHub's rules. The setting was changed to ROBOTSTXT\_OBEY = False to allow the scraping necessary for the project's goal, while maintaining politeness through download delays and auto-throttling.

# Implementation

Class 1: GithubRepoItem (Scrapy Item)

Description: A Scrapy Item class used to define the structured data fields for each repository scraped. This ensures consistent data handling and export.

Fields:

url (scrapy.Field): Stores the full URL of the repository.

about (scrapy.Field): Stores the repository's description.

last\_updated (scrapy.Field): Stores the ISO format timestamp of the last update.

languages (scrapy.Field): Stores a dictionary of languages and their percentages or None.

num\_commits (scrapy.Field): Stores the integer count of commits or None.

Methods: N/A

Functions: N/A

Class 2: GithubReposSpider (Scrapy Spider)

Description: The main spider class inheriting from scrapy.Spider. It handles the crawling logic, starting from the user's repository list page, following links to individual repositories, and extracting the required data.

Fields:

name: "github\_repos"

allowed\_domains: ['github.com']

base\_url: Template URL string.

custom\_settings: Dictionary containing settings for XML feed export, logging level, download delays, throttling, and ROBOTSTXT\_OBEY status.

start\_urls: List populated in \_\_init\_\_ based on the provided username.

username: Stores the target GitHub username.

Methods:

\_\_init\_\_(self, username=None, \*args, \*\*kwargs): Constructor that accepts the target username via command-line arguments (-a) and initializes start\_urls.

parse(self, response): Callback method to process the main repository list page. It identifies repository entries using CSS selectors, extracts initial data (URL, name, initial about, last updated), handles pagination by finding 'Next' links, and yields scrapy.Request objects to scrape individual repository pages, passing initial data via cb\_kwargs.

parse\_repo\_page(self, response, initial\_data): Callback method to process individual repository pages. It receives initial data, checks if the repository is empty, extracts languages and commit counts (if not empty) using CSS selectors and regex, applies the logic for the 'About' field (using repo name if the description was initially empty on a non-empty repo), populates the GithubRepoItem, and yields the final item.

Functions: No standalone functions defined within the class in the example, logic is contained within methods.

Results

Result : The spider successfully generated an XML file (repositories.xml) containing structured data for the public repositories of the target GitHub user (113021147-EdgarJoshua in our test case).

Conclusions

This project successfully demonstrated the use of the Scrapy framework and related Python libraries to build a functional web scraper capable of extracting specific data points from GitHub repository pages. The scraper handles data extraction, cleaning (like parsing dates and commit numbers), conditional logic (for empty repos/descriptions), and outputs structured data to an XML file. Key challenges included identifying robust selectors for GitHub's dynamic HTML structure and understanding the implications of robots.txt rules, which required adjusting Scrapy settings. The project provides a practical application of advanced computer programming techniques for web data acquisition. Future improvements could include adding error handling for network issues, supporting authentication for private repositories (via GitHub API integration, which is preferred), or expanding the data points collected.