亞洲大學
ASIA UNIVERSITY

Midterm Project Report

Advanced Computer Programming

Student Name : Julian Handoyo

Student ID : 113021134

Teacher : DINH-TRUNG VU

2024-04

Chapter 1 Introduction

1.1 Github

1) **Personal Github Account**: https://github.com/113021134
2) **Group Project Repository**: https://github.com/113021134/acp-1132

1.2 Overview

Brief description of advanced language features and libraries used in your work as well as results you have archieved. For example:

The data class, pprint, pattern matching, regulation expresion, and Beautiful Soup have used in my program. My program have extracted information about URL, About, Language, Number of Commits of repositories on the page https://github.com/trungvdhp.

Chapter 2 Implementation

2.1 Class 1: GithubSpider

This class is designed to scrape information from a GitHub user's public repositories. It extracts data such as repository URL, name, last updated time, README content, languages used, number of commits. The extracted data is saved to an XML file upon completion

2.1 Fields

name: spider name used to run it

allowed_domains: specifies the domain to restrict scraping

start_urls: initial URL to start crawling

root: XML root element where all repository entries are stored

idCount: counter to assign a unique ID to each repository in XML

currentDesc, currentLastUpdated: Temporary variables to store data between requests

2.2 Methods/Function

```
import scrapy
import xml.etree.ElementTree as ET

root = ET.Element("Repositories")
idCount = 0
currentLastUpdated = None
```

Import scrapy, and xml library.

initialize temporary variables to store data between each request.

```
class GithubSpider(scrapy.Spider):
    name = "github"
    allowed_domains = ['github.com']
    start_urls = ['https://github.com/113021134?tab=repositories']
```

This defines a new Scrapy spider class called GithubSpider.

name = "github" sets the name of the spider. You use this name to run the spider from the command line.

allowed_domains restricts the spider to only crawl pages from the specified domain.

start_urls is a list of URLs where the spider will begin crawling.

```
def parse(self, response):
        global currentLastUpdated
        repos = response.css('li[itemprop="owns"]')

        for repo in repos:
            # currentDesc = None if
len(repo.css('p[itemprop="description"]::text').get(default='').strip())
== 0 else
repo.css('p[itemprop="description"]::text').get(default='').strip()
            # currentDesc =
repo.css('p[itemprop="description"]::text').get(default='No
description').strip()
            currentLastUpdated =
repo.css('relative-time::attr(datetime)').get()

            relative_url = repo.css('a[itemprop="name
codeRepository"]::attr(href)').get()

            if relative_url:
                full_url = response.urljoin(relative_url)
                yield scrapy.Request(full_url, callback=self.parse_repo)
```

The repos variable is used to store the selected part of the HTML that contains <li> tags with the itemprop="owns" attribute. Then, the spider loops through each repository found. For each repository:

- It also checks for the "last updated" information. If it's missing, last_updated is set to None.

Finally, the spider follows the repository link and calls a function named parse_repo to continue parsing details from the repository page.

```
def parse_repo(self, response):
        global idCount, root, currentLastUpdated

        repoName = response.css('strong.mr-2.flex-self-stretch
a::text').get(default='').strip()
        readmeDiv = response.css('article[itemprop="text"]')

        file_rows = response.css('thead.Box-sc-g0xbh4-0.jGKpsv')  # these
are the file entries
        isEmpty = len(file_rows) == 0

        idCount += 1
        rep = ET.SubElement(root, "repository")
        rep.set("id", str(idCount))

        ET.SubElement(rep, "URL").text = response.url
        ET.SubElement(rep, "last_updated").text = currentLastUpdated

        ET.SubElement(rep, "name").text = repoName

        currentDesc = response.css('p.f4.my-3::text').get(default='No
Description').strip()

        if len(currentDesc) != 0:
            ET.SubElement(rep, "description").text = currentDesc


        if isEmpty:
            ET.SubElement(rep, "languages").text = "Empty"
            ET.SubElement(rep, "commits").text = "Empty"
            return
        else:
            readme = ET.SubElement(rep, "README")

            if readmeDiv:
                readmeText = '
'.join(readmeDiv.css('::text').getall()).replace("\n", "").strip()
                readme.text = readmeText
            else:
                readme.text = 'no README'

            languageStats =
response.css('div.BorderGrid-row:contains("Languages")')
            commits_text =
response.css('span.fgColor-default::text').get(default='').strip()

            langEl = ET.SubElement(rep, "languages")
            commitsEl = ET.SubElement(rep, "commits")
            if languageStats:
                lang_list =
languageStats.css('span.color-fg-default::text').getall()
                for lang in lang_list:
                    lang = lang.strip()
                    if lang:
                        ET.SubElement(langEl, "language").text = lang
            else:
                langEl.text = "None"

            commitsEl.text = commits_text if commits_text else "None"

            yield {
                'repositoryUrl': response.url,
                'name': repoName,
            }
```

The parse_repo function is used to handle each individual repository page. First, it gets the repository name using a CSS selector and stores the README section if it exists. Then it checks if the repo is empty by checking the file rows.

If the repo is empty, it sets the name to the description if available, otherwise it uses the repo name. It also sets languages and commits to "Empty" and stops there.

If the repo is not empty, it continues by adding the repo name, README text (or "no README" if missing), and collects the programming languages used. It checks the number of commits and adds that too.

Lastly, it yields the repository URL and name to Scrapy's output and adds everything to the XML structure using global variables.

```
def closed(self, reason):
        tree = ET.ElementTree(root)
        tree.write("repositories.xml", encoding="utf-8",
xml_declaration=True)
```

The closed function runs when the spider finishes crawling. It takes a reason parameter that tells why the spider stopped. Inside the function, it creates an XML tree using the root element, then saves it as a file called repositories.xml with UTF-8 encoding and an XML declaration at the top.

2.3 Code Structure

The scraper begins by storing metadata for each repository. Specifically, it extracts the last updated time, which is saved in the currentLastUpdated variable, and the description, stored in currentDesc.

Next, it fetches the repository’s URL and sends a request to visit each repository's individual page.

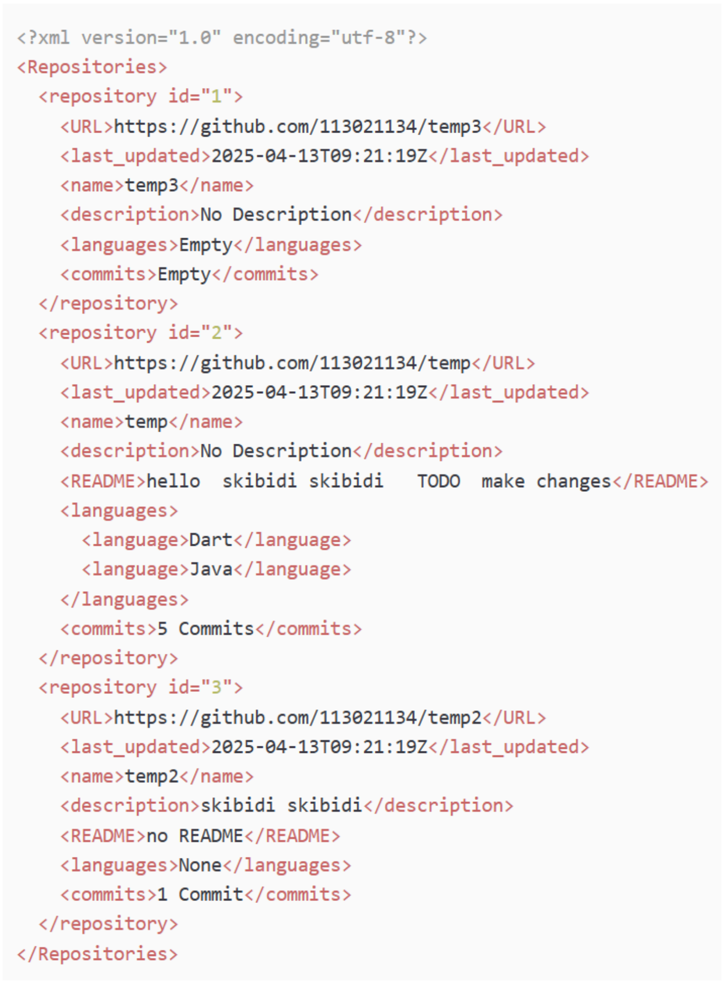
Once on the repository page, the scraper checks whether the repository is empty or not:

- If the repository is empty:
    - Set the languages and commits to Empty
- If the repository is not empty:
    - It checks for the presence of a README file and stores its contents.
    - Then it searches for programming languages used in the repository.
    - Finally, it retrieves the number of commits.

All this information is then structured and saved into an XML file.

Chapter 3 Results

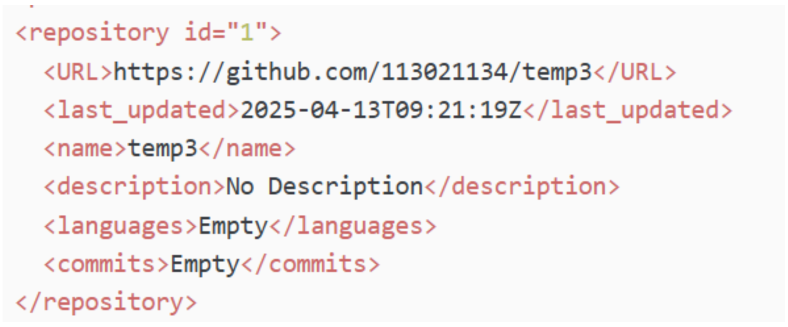
3.1 Results



```xml
<?xml version="1.0" encoding="utf-8"?>
<Repositories>
  <repository id="1">
    <URL>https://github.com/113021134/temp3</URL>
    <last_updated>2025-04-13T09:21:19Z</last_updated>
    <name>temp3</name>
    <description>No Description</description>
    <languages>Empty</languages>
    <commits>Empty</commits>
  </repository>
  <repository id="2">
    <URL>https://github.com/113021134/temp</URL>
    <last_updated>2025-04-13T09:21:19Z</last_updated>
    <name>temp</name>
    <description>No Description</description>
    <README>hello  skibidi skibidi   TODO  make changes</README>
    <languages>
      <language>Dart</language>
      <language>Java</language>
    </languages>
    <commits>5 Commits</commits>
  </repository>
  <repository id="3">
    <URL>https://github.com/113021134/temp2</URL>
    <last_updated>2025-04-13T09:21:19Z</last_updated>
    <name>temp2</name>
    <description>skibidi skibidi</description>
    <README>no README</README>
    <languages>None</languages>
    <commits>1 Commit</commits>
  </repository>
</Repositories>
```

3.2 Result 1

Empty repositories

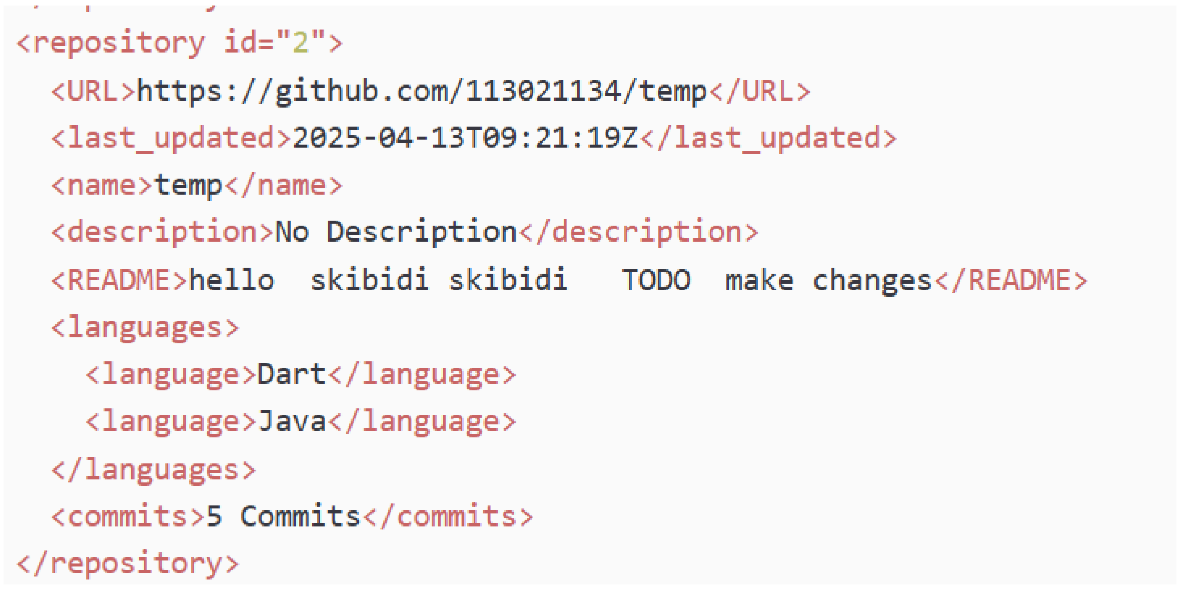


```xml
<repository id="1">
  <URL>https://github.com/113021134/temp3</URL>
  <last_updated>2025-04-13T09:21:19Z</last_updated>
  <name>temp3</name>
  <description>No Description</description>
  <languages>Empty</languages>
  <commits>Empty</commits>
</repository>
```

.

3.3 Result 2

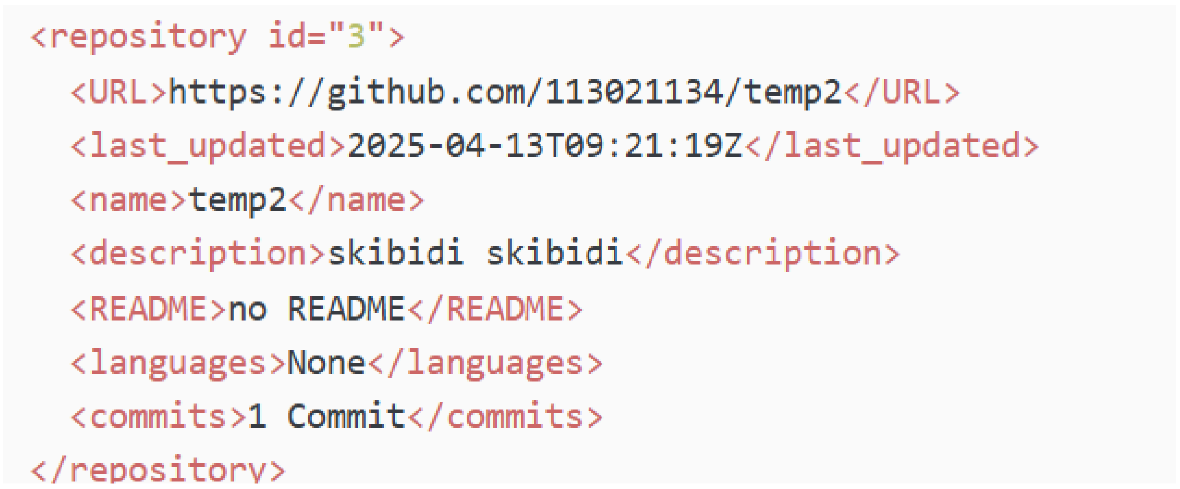
Full repositories named “temp” with Java and Dart files, also a README file, and 5 commits



```xml
<repository id="2">
  <URL>https://github.com/113021134/temp</URL>
  <last_updated>2025-04-13T09:21:19Z</last_updated>
  <name>temp</name>
  <description>No Description</description>
  <README>hello  skibidi skibidi   TODO  make changes</README>
  <languages>
    <language>Dart</language>
    <language>Java</language>
  </languages>
  <commits>5 Commits</commits>
</repository>
```

3.4 Result 3

Empty repositories where all the files were deleted and no README



```xml
<repository id="3">
  <URL>https://github.com/113021134/temp2</URL>
  <last_updated>2025-04-13T09:21:19Z</last_updated>
  <name>temp2</name>
  <description>skibidi skibidi</description>
  <README>no README</README>
  <languages>None</languages>
  <commits>1 Commit</commits>
</repository>
```

.

Chapter 4 Conclusions

This project successfully demonstrates the use of the Scrapy framework to extract structured data from GitHub user repositories and export it into an XML format. The scraper efficiently navigates the repository listing, gathers detailed information about each repository such as the name, description, last updated time, README content, programming languages, and commit counts.

The program handles both regular and empty repositories by checking for file listings and adjusting the output accordingly. This ensures that the data collected is as complete and accurate as possible, even in edge cases.

By automating this data collection process, the implementation showcases how web scraping can be used to gather and analyze information from developer platforms like GitHub. The use of structured XML output also makes the data easy to store, share, or use in other applications.