APKMirror Crawler

Report

Version 1.0
Course: Short Programming Project

rijksuniversiteit
groningen

Student:
Cristian Savin

Supervisor:
Mohsen, F.F.M. Dr. Fadi

Contents

1 Introduction

The APKMirror Crawler is a tool developed to crawl/scrape Android applications from `https://www.apkmirror.com`.
APKMirror is a website owned by AndroidPolice - one of, if not the most prominent Android news website. APKMirror contains free applications, which are thoroughly checked by the staff to be the original application and not some infected or cracked application by a malicious third party. New applications on APKMirror are checked as follows:

- New versions of an application are matched to previous versions (to ensure the true developers signed them).
- New apps are matched to other apps from the same developer (to ensure the true developer signed them).

The two big reasons why APKMirror was chosen are the extensive library of applications that contain old versions of the same application and the simple fact that APKMirror allows Web Crawlers/Scrapers, as long as the Scraper is not overloading the servers. This will be discussed later in the report in the Technologies and Challenges sections.
This document will go over the technologies used, the project's architecture, the tool's functionalities, applications on the crawler, challenges, and finally, possible improvements that can be made in the future.

Some general instructions might be discussed in the report, but please refer to README.md for instructions on installing and running the project. The README is considered to be part of the report.

2 Technologies

In this section, we will discuss the technologies used to develop this project and the reasoning for the chosen technologies.

2.1 Technology Stack

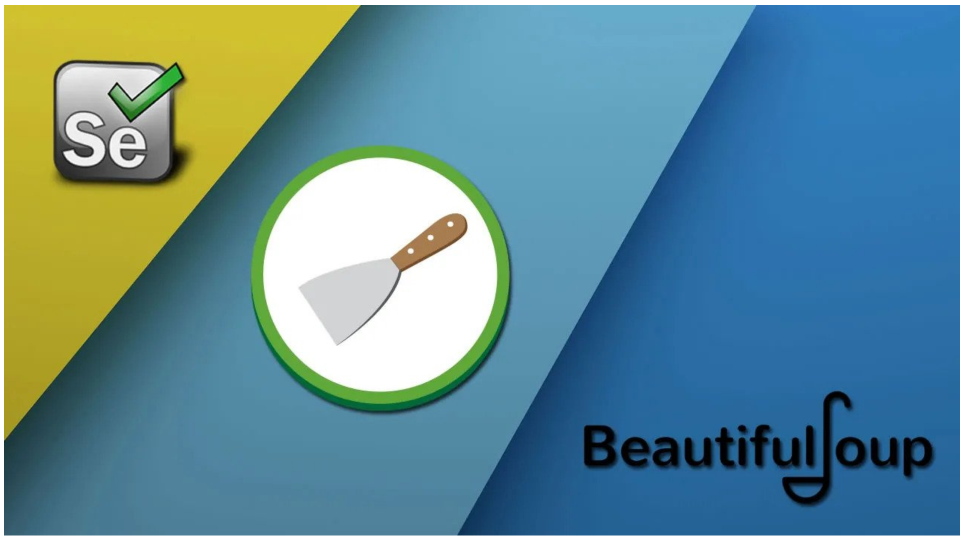
- Languages
  - Python
  - XPath
- Libraries

  - Scrapy Framework

- Environment
  - Python Virtual Environment

1. The programming language was the first easy choice for this project. The student and the supervisor agreed on the preferred language to be Python. One of the advantages of using Python is the numerous prebuilt libraries and available frameworks for any scenario. Secondly, Python is considered one of the more accessible languages to learn, read, and code.

2. The Web Scraping tool was the next choice that had to be made for this project. Python has a good number of frameworks, each with its advantages and disadvantages. We will go over the three options that were considered.



   - First, we will discuss the Beautiful Soup library. At first glance, this might seem like the perfect framework for this project, as the advantages of Soup are that it is easy to learn and has a ton of simple methods for pulling, navigating, searching, and modifying data out of HTML. It sounds like the perfect choice for this project. The disadvantage, however, is that Soup is recommended for small crawling projects, has some missing built-in functionalities, and is not easily extendable.

- Second, the Selenium framework was considered for its ability to be used with JavaScript concepts. Selenium's functionality is the ability to access the Selenium WebDriver - this framework allows the developer to execute cross-browser requests. The disadvantage of Selenium is that it was developed for Web Testing, and some of the required functionalities for Web Scraping are missing.
- Finally, the Scrapy framework is the one that was chosen for this project, as the tools provided seemed most fitting. Scrapy is a high-level, fast tool developed for Web Crawling and Web Scraping. The tool is considered multi-purpose as it has the necessary functions for projects from data mining, to web crawling, to web scraping, and even web testing. Advantages of Scrapy over the two options are that Scrapy projects are easily extendable and that it is possible to build a robust scraper that does not use a ton of memory and CPU storage.

3. As discussed in the previous section, Scrapy has a built-in mechanism for extracting data. XPath - an expression language designed to support the query or transformation of XML documents, is used to select parts of the HTML document. In short, when a user scrapes a website, he is prompted with a .html file of the page containing the HTML code for the whole page. XPath queries are used as a selector for specific parts of the website to extract only the desired information, i.e., application information. "XPath Helper" - a chrome extension was used to extract, edit, and evaluate XPath queries on APKMirror.

4. In the following section, we will discuss the possibility of speeding up the scraping speed. As mentioned earlier in the Introduction, APKMirror does allow Web Crawlers and Scrapers, as seen from **robots.txt** and as mentioned by the website owner in an email. The problem in allowing Crawlers free reign over a website without any protection is a disaster waiting to happen. As such, APKMirror uses Cloudflare Protection services. In the scenario where many requests are sent in minutes from a single IP address, the IP address will be rate-limited for an hour. In such a scenario, the scraper program cannot send any requests for that time. This is a scenario that any user wants to avoid, below we will discuss the solutions:

   - First, the straightforward way to solve this issue and avoid the rate limitation is to set request delays. The specific projects delays will be discussed later in the Settings sub-section from the Architectural Overview section. This solution involves settings limitations to the scraper, such as limiting the concurrent requests on the website, adding a download delay, and even a caching option, similar to what browsers use.

- The second solution might sound great in theory but is terrible in practice. The solution is to use rotating IP addresses, also known as rotating proxies. In this scenario, the user has a pool of proxies, and he sets the Scrapy project to send requests from the specific proxies. This will avoid the Cloudflare rate limitation as the Proxy will be rate-limited and not the user's IP address. If one Proxy is blocked, "rotate" to the next one. This option might overload a smaller website(with a cheaper host), and in that scenario, it can be considered a DDoS(Denial-of-service) attack. Another negative regarding this solution is that good proxies cost money, as they, for the most part, are private. The free ones on the Web have a weak connection and, in most cases, cannot even send the request. An advantage of using the rotating proxies method is the anonymity it provides to the user.
- Finally, the last solution that will be discussed in this report is the Zyte Smart Proxy Manager, a paid service that allows the user access to rotating proxies. The difference between this solution and the previous one is that services like the one provided by Zyte take care of making the requests, so the user can focus on parsing clean. The trial version provided by Zyte Smart Proxy Manager was used for a short period to develop the final touches of this project. Testing functionality in the latter parts of the project was complex with the first solution, as output times for some functionality can take hours with the first solution and minutes with the third. Problems encountered during the project development will be discussed in detail in the Challenges section.

3 Architectural Overview

In this section, we will discuss the scraper's main components. Below a directory tree of the scraper discussed components is provided. Clicking a component will forward the document to the specific sub-section.

```
apkmirror_crawler
 |__ apkmirror_crawler
 |    |__ item.py
 |    |__ pipeline.py
 |    |__ settings.py
 |    |__ spiders
 |         |__ apkdownloader_spider.py
 |         |__ apkinfo_spider.py
 |__ apkmirror_script.py
 |__ script_functions
      |__ read_input.py
```

Some of the components are part of the Scrapy framework. An explanation for them will be discussed beforehand. The report will not discuss specific codes, as the code is thoroughly documented.

3.1 apkmirror_script.py

This component is the main script the user will use to run the project. It combines the two developed spiders apkinfo_spider.py and apkdownloader_spider.py and with the help of the read_input.py and CrawlerRunner - a Scrapy class, the user has access to run the scraper in a personalised manner. Some of the more important options are:

- Run spider to crawl a single application, a single category or all categories.
- Choose a number of versions for every application crawled to parse
- Run the apkinfo_spider.py, apkdownloader_spider.py, or both in a sequential order.

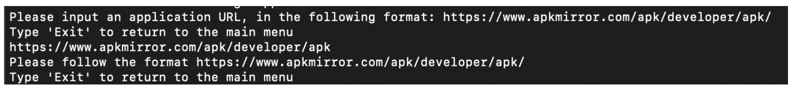
More in regards to the scraper's options is discussed in the Functionality section.
The CrawlerRunner class is a required class for scripts. This class allows the user to run a spider or multiple spiders from a script. The read_input.py file provides the necessary variables to be sent to the spiders based on the user's input.

3.2 read_input.py

This component has been developed to provide the full Command Line Interface to the user and subsequently parse the input. The parsed input is

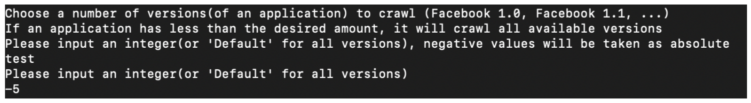
returned to the apkmirror_script.py as option variables used for the spiders. Numerous (smaller) components read_input_helper.py, menu_prints.py have been added during the refactoring phase to make the code readable and will only be discussed in this section.
The input parsing is developed to be robust, comprehensive, and exception-free. Below are some possible scenarios:

- In the first scenario, the user is asked to input a link for the desired application to scrape. The parser will require to follow the specific format for the URL. In this test scenario, the user input is missing a "/" at the end of the inputted URL, prompting a follow-up text to input following the URL format.



```
Please input an application URL, in the following format: https://www.apkmirror.com/apk/developer/apk/
Type 'Exit' to return to the main menu
https://www.apkmirror.com/apk/developer/apk
Please follow the format https://www.apkmirror.com/apk/developer/apk/
Type 'Exit' to return to the main menu
```

- In the following scenario, the user is asked to input the number of versions desired to be scraped for each application. The input parser will require an integer. In the scenario that the user inputs a string, it will prompt an error. If the user inputs a negative integer, the absolute value will be taken instead.



```
Choose a number of versions(of an application) to crawl (Facebook 1.0, Facebook 1.1, ...)
If an application has less than the desired amount, it will crawl all available versions
Please input an integer(or 'Default' for all versions), negative values will be taken as absolute
test
Please input an integer(or 'Default' for all versions)
-5
```

More details in regards to the scraper functionality as mentioned above, are discussed in the Functionality section.

3.3 item.py

This component aims to define the items extracted from APKMirror as structured data and later outputted in a structured CSV(comma-separated values) file by Scrapy. After an Item is scraped by the apkinfo_spider.py it is sent to the pipeline.py component.

3.4 pipeline.py

The Item Pipeline in this project was used to process the downloaded applications. By default, in the Scrapy framework, the downloaded files are stored with a SHA1 hash of their URLs as the file names. The downloaded Items are processed through the Item Pipeline and have their file name changed from `1b6c0a648f8eceb9b8e8d71e5ff185ace7c479fa` to something like `com.example.app123_1.0-1_minAPI10(nodpi).apk`

3.5 settings.py

This component allows the developer to customize the behavior of the Spiders. The critical behavior that was changed from the default Spiders is the following

- ROBOTSTXT_OBEY - This option is set to True, as we want to follow the rules indicated by the website in the robots.txt file
- RETRY_HTTP_CODES - This option is used to retry specific return codes in the scenario of a return code other than 200 - Success or 304 - Redirect.
- ITEM_PIPELINES - This option points to the pipeline.py class. For this project, that class is used to rename the downloaded files.
- CONCURRENT_REQUESTS, DOWNLOAD_DELAY, and CONCURRENT_REQUESTS_PER_DOMAIN are all used to set limits and time limits for the scraper. As a result, Cloudflare does not rate-limit the IP address based on the limitations set.

3.6 apkinfo_spider.py

This Scrapy component is the main spider used to crawl and scrape the APKMirror website. In combination with XPath queries, the spider crawls the website similar to a real user. The spider will receive a request URL depending on the user's input, and following user's options to the spider, it will follow the links and scrape the specified information by XPath(links, application information, metadata).
A quick run-down of the APKInfo Spider:
When Scrapy makes a `request`, it outputs a `response` variable, the `response` variable is the HTML of the website. Using XPath queries the spider extracts the links it has to follow, until it finds the application information and the application download link. The information is parsed in the `parse_info` method, the data is extracted to the `ApkMirrorItem` from item.py component, and finally outputted to the `output.csv` file.

3.7 apkdownloader_spider.py

The APKDownloader Spider requires an `output.csv` file with a download_link item in the file. The spider requests the download_link URL and parses the information to an ApkMirrorDownloader Item from item.py component, and downloads the APK file. Following is the Item Pipeline in pipeline.py here, the APK file is renamed as discussed in the pipeline section.

4 Functionality

In this section, we will discuss the functionality and possibilities of the scraper, this functionality is input-dependent (through the Command Line Interface), and it is checked in the apkmirror_script.py component before starting the crawler.

- First, we will discuss the project's main functionalities:
  1. Crawl a single application - For this option, the user will have to input a link following the format `https://www.apkmirror.com/apk/developer/application/`. This crawler option is precise and should be used for applications with many versions 1000-5000+, i.e., Facebook, with the intent to analyze a single application's development over the years.
  2. Crawl a single category - For this option, the user will have to input a link following a similar format `https://www.apkmirror.com/categories/category/`. This option will crawl applications of a category. It should be used when the user requires a significant number of applications(1000-15000+)
  3. Crawl all categories (DEFAULT) - This is the default option. If the User does not specify any link for the above two options, the script will default to this option and crawl all applications on the website. This is the option to scrape everything on APKMirror, and the User should expect to download 150000+ applications.
- Second, we will discuss the option to choose the number of versions for each app to scrape and how it affects the main functionality of the crawler. Most applications on APKMirror, as mentioned in the Introduction section, have multiple versions. The number of versions

depends on the application itself, the popularity of the application, and whether or not the application is still in development. An application like Facebook has over 3000 versions. Below we will discuss some scenarios on how to use this option in conjunction with the previously discussed functionalities.

1. Crawl a single application with a chosen number of versions - these two options can be used for the following scenario: scrape 1000 Facebook versions
2. Crawl a single category with a chosen number of versions. These two options can be used for the following scenario: scrape 100 versions(if available) from each application in the Finance category.
3. Crawl all categories with a chosen number of versions - these two options have a similar scenario to ”single category.” This option can also be used when the user wants unique applications. The user will input ”1” for this option in such a scenario.

If an application has less than the desired amount, it will crawl all available versions.

- Third, we will discuss the option to choose which spiders to run, by default both apkinfo_spider.py, and apkdownloader_spider.py will run a sequential order. The user can choose to run only apkinfo_spider.py to scrape the required applications, and later if the output file satisfies the user's requirements, apkdownloader_spider.py can be used to download the applications.
  The apkdownloader_spider.py will search for an `output.csv` file.

- Finally, we will discuss the option to choose to overwrite or append the results from the apkinfo_spider.py to the `output.csv` file. The default option is to overwrite the `output.csv` file. The append option can be used in the scenario where the user wants to crawl two applications and append the specific applications to a single `output.csv` file. A similar scenario can be used on two categories.

5 Applications of the Crawler

In the following section, we will discuss the possibilities of the crawler and how to combine all options to get the desired results. By default the apkmirror_script has the following settings:

1. Scrape all categories
2. No link provided
3. Scrape all versions
4. Run both spiders sequentially
5. Overwrite the output file

One scenario is to scrape all versions of a specific application and download them, the settings should be as follows:

1. Scrape a single application
2. Provide a link, i.e., "`https://www.apkmirror.com/apk/twitter-inc/twitter/`"
3. Scrape all versions
4. Run both spiders sequentially
5. Overwrite the output file

The APKMirror has a search bar, which can search for a specific application. In this scenario, only the first option discussed in Functionality section was changed. The example URL follows the format discussed,
"https://www.apkmirror.com/apk/developer/application/" .

A second scenario is to scrape at most 500 versions for each application of a specific category, with the intent to download them later. For this scenario the apkmirror_script.py has to run two times.
First:

1. Scrape a single category
2. Provide a link, i.e., "`https://www.apkmirror.com/categories/finance/`"
3. Scrape 500 versions for each application(if available)

4. Run only APKInfo spider
5. Overwrite the output file

In this scenario, where the user chooses the option to download 500 versions for each application, the crawler will scrape all versions available if the application has less than the desired amount. The example URL follows the format discussed in the Functionality section,
"https://www.apkmirror.com/categories/category/". The result of this script will be an `output.csv` file, but no APKs downloaded.
Next, the user may scrape a different category and append the results to the previous `output.csv` file or download the APKs for the scraped applications. We will follow the former scenario. Next,

1. Scrape a single category
2. Provide a link, i.e.,
   "`https://www.apkmirror.com/categories/lifestyle/`"
3. Scrape 300 versions for each application(if available)
4. Run only APKInfo spider
5. Append to the output file.

In this run, all the options are were changed from the default ones. The results from this script will be appended to the `output.csv` with the Finance applications. Finally, we may choose to download the `output.csv` file with the Finance and Lifestyle applications.

1. Scrape all categories - Not relevant
2. No link provided - Not relevant
3. Scrape all versions - Not relevant
4. Run only APKDownload spider
5. Overwrite the output file - Not relevant

In the scenario, where a user wants to download from an `output.csv` file, only the 4th option is relevant, and the rest of the options are ignored.

6 Future development

This section is provided to students who will want to improve this project. I will provide some suggestions for some improvements as well, as I had planned to implement some of them.

- List of applications desired to crawl - For this option, the user can provide either to the Command Line Interface or in a file, i.e., input_links.txt multiple applications to crawl. Currently, the only way to crawl specific applications in one `output.csv` file is to run the script for every application.
  This improvement can easily be implemented in a similar way to how apkdownloader_spider.py reads the `download_links` from a file.

- Crawl by Developers. The current crawler can crawl a single application, a single category, and all categories. A new Spider can be developed to crawl, a single developer and all developers.
  Parts of the APKInfo spider can be used to code the two new functionalities.

- The Application information page for most applications provides a Google Play link. For many applications, the link is unavailable, but if the link works, the crawler could follow the Google Play link and scrape any relevant information on the page.

- Test for better settings in the delay between requests to have a faster crawling speed. The settings present in the project can be improved, specifically, the DOWNLOAD_DELAY, AUTOTHROTTLE_START_DELAY, and AUTOTHROTTLE_MAX_DELAY.

- Order the `output.csv`, based on a specific item, so the APKDownloader spider crawls in sequential order of the released applications.

- Choose an output file name, a relatively simple improvement, but it is a great option when you want to run the script multiple times.

7 Challenges

During the development of this project I encountered two big challenges:

1. Familiarising myself with Scrapy, as it is a easy-to-use framework when you understand how everything works. StackOverflow had some an-

swers to my question, but for the most part I had to read the Scrapy Docs. Luckily, the Docs provided are thorough, and well explained.

2. The second challenge I encountered, was the CF(Cloudflare) rate limitation. While working on the project, if I had too many requests in a few minutes, CF would block my IP for an hour. This definitely was the biggest challenge, as in some days, while testing for the correct settings, I was blocked every hour.

8 Lessons Learned

- Do much research before starting on the project.
  I started to work with the Beautiful Soup framework as it was suggested the most on StackOverflow. After one week of development, I realized that it has many limitations, so I had to switch to Scrapy.

- Not all answers can be found on StackOverflow(SOf).
  I had a few struggles during the development phase, and I tried searching for the solution on SOf. In the later development phase, I realized that Scrapy Docs contained much more information. You just have to learn how to filter for the necessary information.