CRAWL SCRAP

TOOLS & TECHNOLOGY

Python
Flask

WHO

Dhruv Patel

INDEX · STEPS · FETCH · CRAWL · SCRAP · DATA · FRONT-END

RUNDHRUV.COM

adafruit

STEPS

FETCH AN URL

https://www.adafruit.com/categories

CRAWLING

CRAWL AND SCRAP 35 VARIOUS CATEGORIES

DATABASE SCRAPING

CREATE A TABLE SCHEMA AND STORE VALUES OF EACH URL TO IT

INDEX.HTML

MAKE ROUTES FROM FRONT END TO QUERY DATABASE

FETCH AN URL

We are given master url to fetch adafruit.com/categories which is attached to our download button on front end. I have chosen python's flask framework

I have iterated through various option to perform this task a like urllib, urllib2, urllib3. The urllib3 seems dominating among all and gives a much better performance compared to the old versions. The python has requests package which internally calls the urllib3 without worrying about HTTP methods and other overheads. With simple two liner solution to hit URL, fetch a response and dump into the beautifulsoup4 to parse DOM tree.

I have studied to understand the DOM of a website and tried to learn about various tags and other attributes arrangements to understand it better.

Initially, I was fetching only one category to see the code correctness and its performance towards fetching and processing the data.

CRAWLING

The in total categories are of 35 to fetch with the little twists and edge cases to look over. Some products are on sale and were under the specialized class of red-sale-price span. We have to override product price in such case to handle this one and put the exact price into the database.

There were some products which are without any description of stock, qty and price where treated as default values.

The third one was hidden edge case where there was no description of a product at all. However, I verified manually for each link to get the accurate ~ 4194 records.

The data structure in total for a site is in very good quality and able to understand it after a few trials and knowing it well.

The fetching process for all data and records is insane. After a while, I realized the power of search engine and get an idea of its massive scale. The simple critique for that data centers and infrastructures might be caching the results and serving them right away I guess. Also, there might be multithreading and multiprocessing environment which will enable this code at full pace.

DATABASE & SCRAPING

Before crawling an URL, we create table schema with a sqlite3 database. Which is again impressively work better for this smaller scale app.

There is also a ":memory" option if in a case to deal with extremely small data without worrying about dependencies and drivers. Using this utility, it makes python as a standout champ for the developer. I simply put a logic flow inside web_crawler.py which calculates the 6 data-values to put into the following schema of a database.

Category	ID	Name	Price	Quantity	Stock

I have used regular expression to make this execution faster throughout a code wherever I can.

There will be default values if a parameter is missing. The data would be permanent as executing .commit() command after the data is stored into the table.

As the database is permeant, we can access from various routes' decorative declarations. I was able to process the query to the each and every routes with almost a zero latency.

INDEX.HTML

The Front end looks simple and lightweight have a card with note to my website with a link

Underneath, I have put the download button to fetch a record and feed into the table which takes a huge time. Thus, while shipping this app I would feed table and you can verify it by hitting any command or shooting a query.

My download button is color blind. The color choices I made accordingly to the nature of a query. If an item is a top seller than choosing the green color and red for out of stock. Black for clearing an output and cyan for common items.

The table might grow large as having many records. However, I put a custom option to query database where you might play with the database.

Overall, I consider the front end is simple and can be added several features beyond the scope of the commercial platform and make it richer without a doubt! But, I have tried to demonstrate a common flow and a pattern behind small and large functions by writing this small yet wonderful app.