Rajiv Gandhi University of Knowledge Technologies

R.K Valley, Y.S.R Kadapa (Dist) - 516330

A
Project Report
on

Web Scrapping

Submitted by

Mutyala Balaji R170052

Under the guidance of

MISS.M HIMABINDU
(Assistant Professor, CSE)

Department of Computer Science Engineering

This project report has been submitted in fulfillment of the requirements for the Degree of Bachelor of Technology in Software Engineering.

Rajiv Gandhi University of Knowledge Technologies
IIIT, R.K. Valley, YSR Kadapa (Dist) - 516330

<u>CERTIFICATE</u>

This is to certify that report entitled "**Web Scrapping**" Submitted by Mutyala Balaji (R170052) in partial fulfillment of the requirements of the award of bachelor of technology in Computer Science Engineering is a bonafide work carried by the them under the supervision and guidance. The report has been not submitted previously in part or full to this or any other university or institue for the award of any degree or diploma.

GUIDE	**HEAD OF THE DEPARTMENT**
MISS.M HIMABINDU	MR.N SATYANANDARAM
Assistant Professor,	Assistant Professor,
Department of CSE,	Department of CSE,
RGUKT RK Valley.	RGUKT RK Valley.

ACKNOWLEDGEMENT

The satisfaction that accompanies the succeful completion of any task would be incomplete without the mention of the people who made it possible and who’s constant guidance and encouragement crown all the efforts success. I would like to express my sincere gratitude to **Miss.M Himabindu**, my project guide for valuable suggestions and keen interest throughout the progress of my project. I am grateful to **Mr.Satyanandaram HOD CSE**, for providing excellent computing facilities and congenial atmosphere for progressing my project. At the outset, I would like to thank **Rajiv Gandhi Of University of Knowledge Technologies (RGUKT),** for providing all the necessary resources and support for the successful completion of my course work.

DECLARATION

We hereby declare that this report entitled **“Web Scrapping”** Submitted by me under the guidance and Supervision of **Miss.M Himabindu**, is a bonafide work. We also declare that it has not been of Submitted previously in part or in full to this University or other institution for the award of any degree or diploma.

Date:- 01-05-2022

Place:-RK VALLEY Mutyala Balaji(R170052)

ABSTRACT

Web scrapping is the process of extracting data and content from web pages. When we use traditional aproach of web scrapping for data delivery it takes more time and some data or details may be excluded. So inorder to extract each and every minute details or data from the web page, we implemented the same web scrapping process in 6 phases in our project, namely **recrawl, feedcrawl**, **extraction**, **dedup**, **normalisation**, **upload**. By following this approach we can extract each and every minute details or data from the web page in a short time.

The primary objective of this project is to provide a complete, accurate, reliable data to the customer in a short time and in an efficient manner.

INDEX

INTRODUCTION

As we can see in our daily lives, we came across huge volumes data from web pages through the internet. Most of the websites do not provide a mechanism such as web services, APIs to collect their data, or the provided APIs are either poorly documented or difficult to use. So for extracting these huge volumes of data we are using the process of web scrapping. Web scrapping is a process of extracting the content and data in large quantities from the web pages. Most of this data is in unstructured format i,e in HTML format which is needed to be converted into structured data so that it can be used in various applications.

By using this process of web scrapping we can extract huge volumes of data from the web page and provide that data in a structured format like xml, json, csv to the customer. Another conern while extracting data is some data may not be included in the process of extraction. So we implemented this web scrapping process in a 6 pahse approach namely **recrawl**, **feedcrawl**, **extraction**, **dedup**, **normalisation**, **upload**. When we scrape huge volumes of data through this approach, we can extract each and every minute details and data from the web page that too in a short time and in an efficient manner.

PURPOSE

Now a days most of the data we are seeing is unstructured and to process that data we need to convert that data into structured format. This is the main purpose of using this web scrapping. Using this web scrapping methods we convert the unstructured data into structured format and will deliver those structured data to clients location.

PRELIMINARIES

RUBY

Ruby is a dynamic, reflective, object-oriented, general-purpose programming language. Ruby is a pure Object-Oriented language developed by Yukihiro Matsumoto. Everything in Ruby is an object except the blocks but there are replacements too for it i.e procs and lambda. The objective of Ruby's development was to make it act as a sensible buffer between human programmers and the underlying computing machinery.

XPATH

XPATH stands for Xml Path Language which uses path like syntax to identify and and navigate nodes in an Xml or HTML document. It contains over 200 built-in functions. Xpath is generally used to uniquely identify the elements or nodes of a HTML documents.

NOKOGIRI

Nokogiri is a ruby tool for parsing the HTML or Xml documents. It is not possible to directly extract the data froma HTML page, first it need to be converted into Dom. To do so we are using this Nokogiri.

REDMINE

Redmine is a free and open source, web-based project management and issue tracking tool. It allows users to manage multiple projects and associated subprojects. It features per project wikis and forums, time tracking, and flexible, role-based access control.

DOM

DOM stands for Document Object Model. It is a tree like structure of a HTML or XML Document. It represents all the nodes of HTML or XML Document in a heirarchy such that it look like a tree structure. Which make easier to navigate through the nodes of HTML page.

ELASTIC SEARCH

Elastic Search is a distributed, free and open search and analytics engine for all types of data, including textual, numerical, geospatial, structured and unstructured. ES allows us to store and search and analyze huge volumes of data quickly.

JSON

JSON stands for JavaScript Object Notation. JSON is a lightweight format for storing and transporting data. JSON is often used when data is sent from a server to a web page. JSON is "self-describing" and easy to understand.

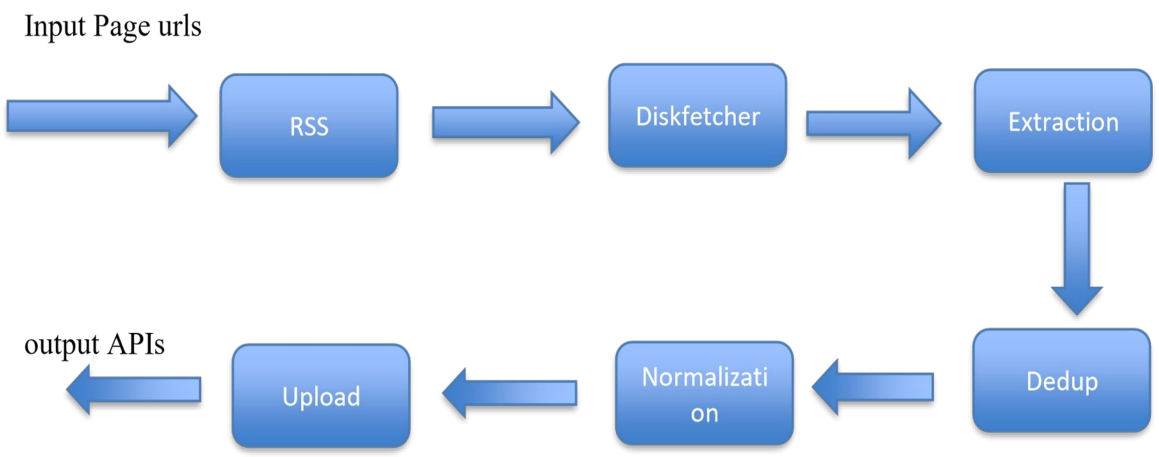
CRAWLBOARD

Crawlboard is a dedicated management system for managing all the crawls that have been pushed. We are calling this as app.promptcloud and it contains major feature called as internal dashboard which is used to get the crawl data of any sites that we are crawling. And we can collect failed urls, respush failed urls and also lot of functionalities with this crawlboard.

KIBANA

Kibana is a free and open frontend application that sits on top of the Elastic Stack, providing search and data visualization capabilities for the data indexed in Elastic Search. Commonly knows as the charting tool for the Elastic Stack. Kibana also acts as the user interface for monitoring, managing and securing an Elastic Stack cluster.

PROPOSED MODEL



Here we proposed a model consisting of 6 stages or phases. Where each step is defined with some purpose and some task. In our pipeline we follow above mentioned six stages inorder to extract data from a page which is required by the client. Each phase is interdepend on next phases, where output of one phase iss given as input to next phase.

It starts with rss which takes site url as input and ends with last stage called as upload where data is uploaded to client's location.

IMPLEMENTATION

We implement this web scrapping in our pipeline in 6 stages, they are:

RECRAWL(RSS):

Here we extract the actual page URL's using a plugin called rss plugin. This plugin creates a dom out of a file and extracts page urls using Xpath and then pushes it to a queue from where feed crawl happens.

In this initial stage we follow depth rules. In the depth0 we take the home page url (seed url) of a website and extract all the category urls from the website. In the depth1 we take the total page count of a category. In this depth2 we extract job urls from each page.

Based on the configuration given in yml file of rss, product urls will be pushed to rss_queue. The RSS plugin will creates a rss_queue having final urls. The rss_queue is located in the following path:

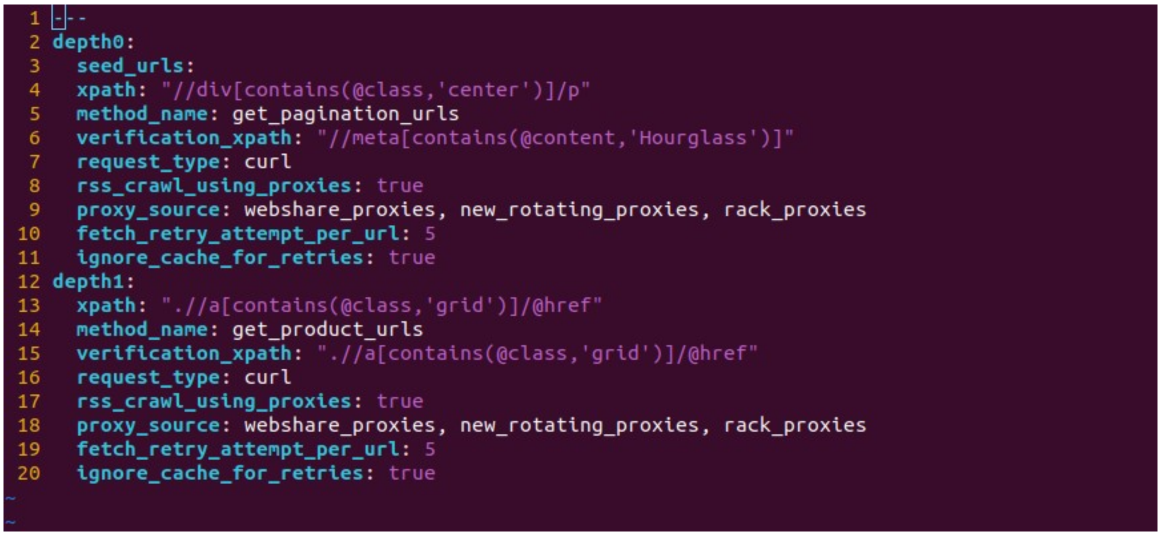
<crawl_home><pipeline_v2><rss_crawl><site_name>_<yyyy_mm_dd>_<crawl_time stamp>

The linux command to test this rss code is :

```
rtest -t r -s <site_name>
```

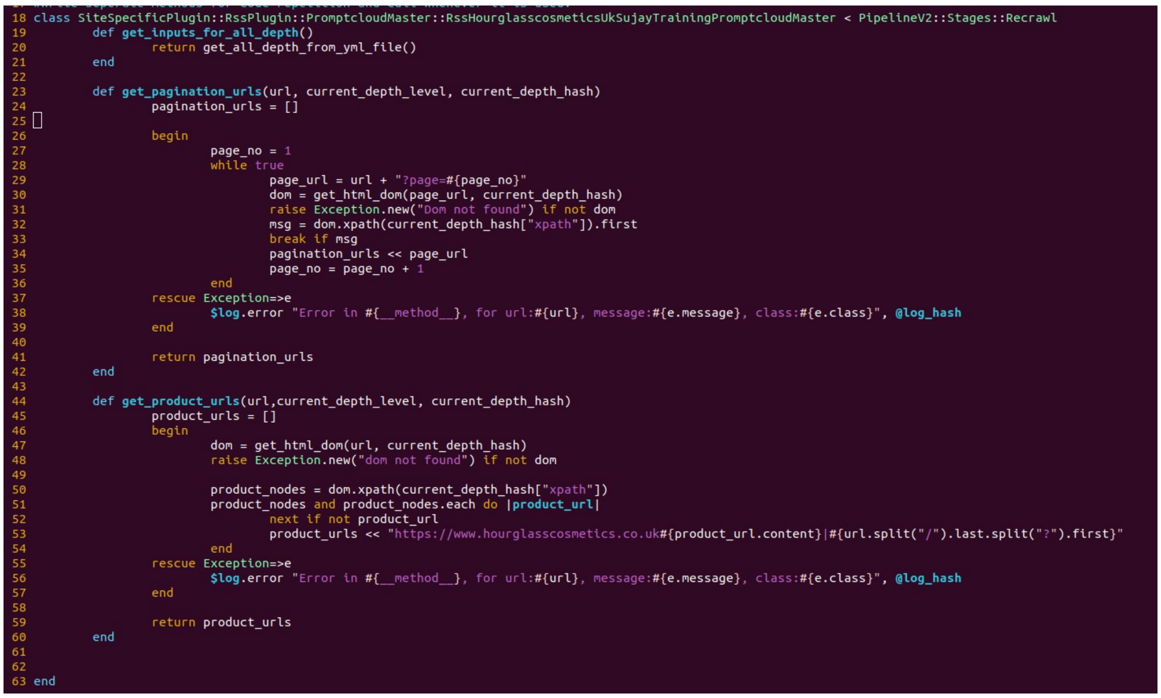
Our pipeline first begins with this stage. In this stage we will collect all the product urls from the given seed_url by the client. And then we will store these urls in a temporary storage called as rss_queue.

Example:



```yaml
---
depth0:
  seed_urls:
  xpath: "//div[contains(@class,'center')]/p"
  method_name: get_pagination_urls
  verification_xpath: "//meta[contains(@content,'Hourglass')]"
  request_type: curl
  rss_crawl_using_proxies: true
  proxy_source: webshare_proxies, new_rotating_proxies, rack_proxies
  fetch_retry_attempt_per_url: 5
  ignore_cache_for_retries: true
depth1:
  xpath: ".//a[contains(@class,'grid')]/@href"
  method_name: get_product_urls
  verification_xpath: ".//a[contains(@class,'grid')]/@href"
  request_type: curl
  rss_crawl_using_proxies: true
  proxy_source: webshare_proxies, new_rotating_proxies, rack_proxies
  fetch_retry_attempt_per_url: 5
  ignore_cache_for_retries: true
```

rss yml file



```ruby
class SiteSpecificPlugin::RssPlugin::PromptcloudMaster::RssHourglasscosmeticsUkSujayTrainingPromptcloudMaster < PipelineV2::Stages::Recrawl
        def get_inputs_for_all_depth()
                return get_all_depth_from_yml_file()
        end

        def get_pagination_urls(url, current_depth_level, current_depth_hash)
                pagination_urls = []

                begin
                        page_no = 1
                        while true
                                page_url = url + "?page=#{page_no}"
                                dom = get_html_dom(page_url, current_depth_hash)
                                raise Exception.new("Dom not found") if not dom
                                msg = dom.xpath(current_depth_hash["xpath"]).first
                                break if msg
                                pagination_urls << page_url
                                page_no = page_no + 1
                        end
                rescue Exception=>e
                        $log.error "Error in #{__method__}, for url:#{url}, message:#{e.message}, class:#{e.class}", @log_hash
                end

                return pagination_urls
        end

        def get_product_urls(url,current_depth_level, current_depth_hash)
                product_urls = []
                begin
                        dom = get_html_dom(url, current_depth_hash)
                        raise Exception.new("dom not found") if not dom

                        product_nodes = dom.xpath(current_depth_hash["xpath"])
                        product_nodes and product_nodes.each do |product_url|
                                next if not product_url
                                product_urls << "https://www.hourglasscosmetics.co.uk#{product_url.content}|#{url.split("/").last.split("?").first}"
                        end
                rescue Exception=>e
                        $log.error "Error in #{__method__}, for url:#{url}, message:#{e.message}, class:#{e.class}", @log_hash
                end

                return product_urls
        end


end
```

rss rb file

DISKFETCHER(FEEDCRAWL):

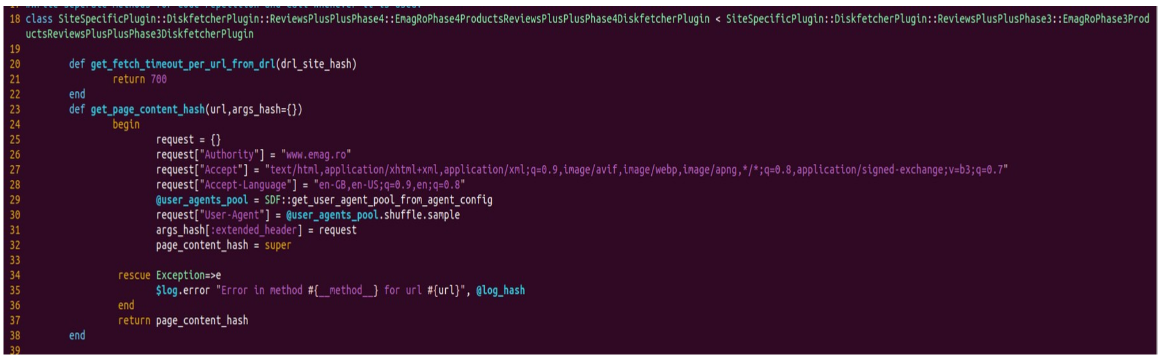
It is the second phase of our pipeline. Where we fetch the pages of urls which are stored in rss_queue. It takes rss_queue as an input and fetches all the pages in that queue and then it stores that pages.

Diskfetcher takes popped urls from rss queues as Input. All the URLs from rss queue pushed as input and we are hitting that particular Product/ review URLs with different types of request types those are **get, post, curl, open_uri, scraper_api**. In this stage it checks thedaily_recrawl_lists(drl) for reading the inputs of frequency time, verification xpaths, proxy country and other required fields for a project nothing but all the configuration. By taking all these fields we can fetch the page source of the url and send the page source file to next pipeline stage. In this output will be stored in our local. This page source is send to next pipeline stage i.e; Extraction.

Command:

rtest -t f -s <site_name>

Example:



```
class SiteSpecificPlugin::DiskfetcherPlugin::ReviewsPlusPlusPhase4::EmagRoPhase4ProductsReviewsPlusPlusPhase4DiskfetcherPlugin < SiteSpecificPlugin::DiskfetcherPlugin::ReviewsPlusPlusPhase3::EmagRoPhase3Prod
uctsReviewsPlusPlusPhase3DiskfetcherPlugin

        def get_fetch_timeout_per_url_from_drl(drl_site_hash)
                return 700
        end
        def get_page_content_hash(url,args_hash={})
                begin
                        request = {}
                        request["Authority"] = "www.emag.ro"
                        request["Accept"] = "text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7"
                        request["Accept-Language"] = "en-GB,en-US;q=0.9,en;q=0.8"
                        @user_agents_pool = SDF::get_user_agent_pool_from_agent_config
                        request["User-Agent"] = @user_agents_pool.shuffle.sample
                        args_hash[:extended_header] = request
                        page_content_hash = super

                 rescue Exception=>e
                        $log.error "Error in method #{__method__} for url #{url}", @log_hash
                 end
                 return page_content_hash
        end
```

Diskfetcher file

EXTRACTION:

It is the main phase where the actual data is being collected. We apply the xpaths on the DOM of the pages which are fetched in feedcrawl stage. We extract the fields which are required by the client and then store them in xml format.

We will extract the required fields from the source code and parse the text to required format. It consider the output of Diskfetcher as input which is stored in local. Here we write all the methods for each and every field which client required separately. We use Xpaths to extract the field from the source page. We will extract the required fields from the source code and parse the text to required format. For any website uniq_id and crawl_timestamp are the required fields these two fields are generated by md5 hash. If any required fields have not generated for the url then it will drop in extraction stage itself. Those urls data neet not send to next pipeline stage.

Extraction plugin will create an ext segment of having the details of a web page url. Along with that it also contains the Quality Analysis report, start time and end time required to crawl the data.

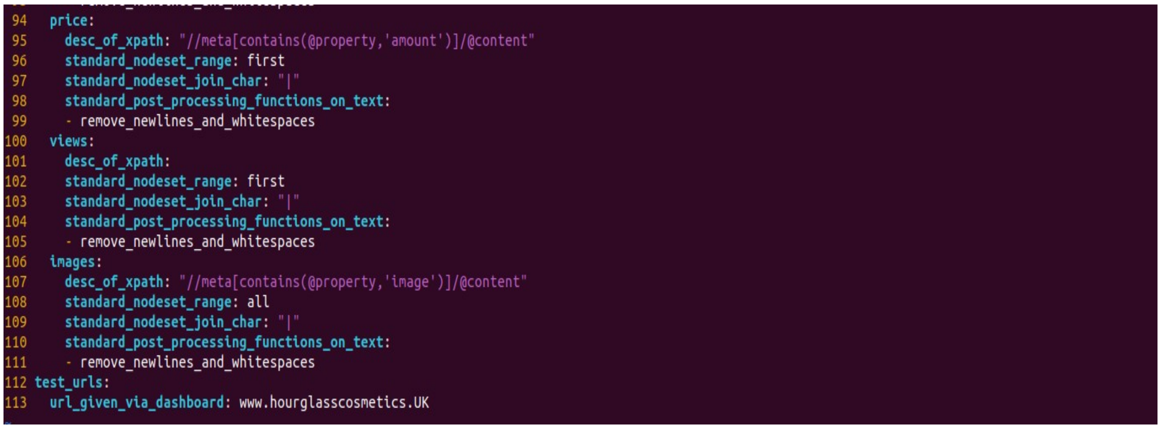
Command:

Testing a single webpage url : *rtest -t e -s <site_name> -u <url>*

Testing an entire segment : *rtest -t e -s <site_name> -q*

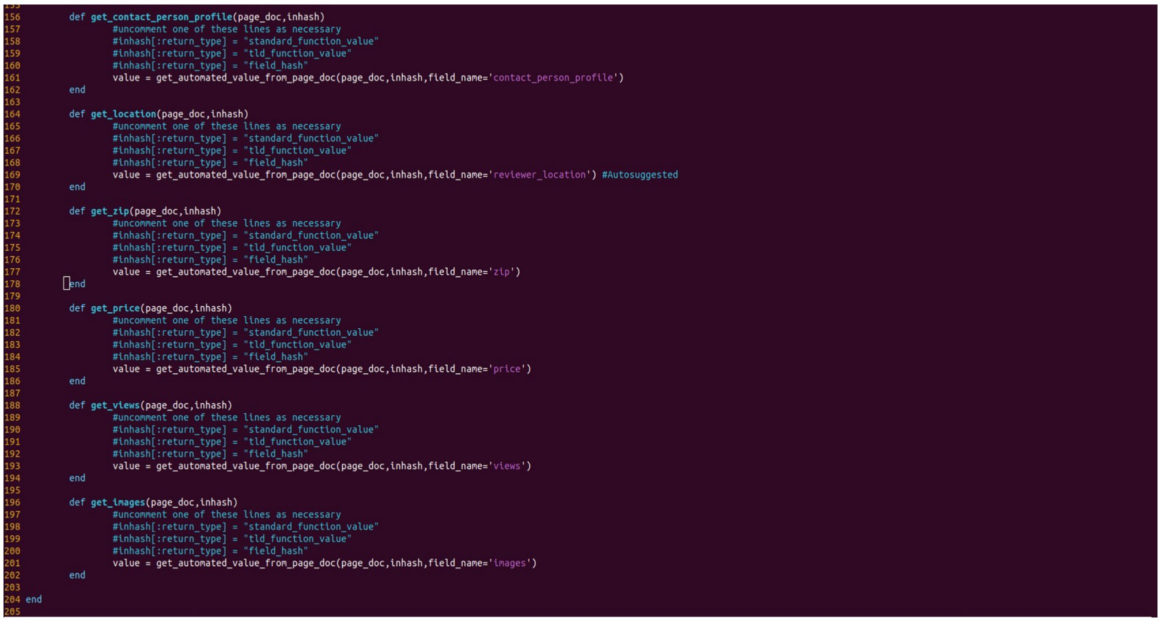
After ruuning above commands, if all the code is proper and error free then it will generate xml file consisting of required fields. Then this xml file will be passed to next phase called as dedup.

Example:



```yaml
  price:
    desc_of_xpath: "//meta[contains(@property,'amount')]/@content"
    standard_nodeset_range: first
    standard_nodeset_join_char: "|"
    standard_post_processing_functions_on_text:
    - remove_newlines_and_whitespaces
  views:
    desc_of_xpath:
    standard_nodeset_range: first
    standard_nodeset_join_char: "|"
    standard_post_processing_functions_on_text:
    - remove_newlines_and_whitespaces
  images:
    desc_of_xpath: "//meta[contains(@property,'image')]/@content"
    standard_nodeset_range: all
    standard_nodeset_join_char: "|"
    standard_post_processing_functions_on_text:
    - remove_newlines_and_whitespaces
test_urls:
  url_given_via_dashboard: www.hourglasscosmetics.UK
```

ext yml file



```ruby
        def get_contact_person_profile(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='contact_person_profile')
        end

        def get_location(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='reviewer_location') #Autosuggested
        end

        def get_zip(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='zip')
        end

        def get_price(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='price')
        end

        def get_views(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='views')
        end

        def get_images(page_doc,inhash)
                #uncomment one of these lines as necessary
                #inhash[:return_type] = "standard_function_value"
                #inhash[:return_type] = "tld_function_value"
                #inhash[:return_type] = "field_hash"
                value = get_automated_value_from_page_doc(page_doc,inhash,field_name='images')
        end

end
```

ext rb file

DEDUP

DEDUP is the stage where all the duplicates are removed from the data. If there are any duplicates present in the data then we eliminate this with the help of elastric search.

In this stage we remove the redundant or duplicate records from the extracted records. The duplication process can be identified by using the uniq_id which has been generated from the extraction. Dedup plugin will create a dedup segment of having the details of a web page url. Along with that it also contains the start time and end time required to crawl the data.

Command:

```
rtest -t d -s <site_name> -q
```

NORMALIZATION

Normalization is the process where the format of data is being converted. By default the data will be in xml format, if client want data to be delivered in other formats like json, csv then we will convert that xml into required format.

Command:

```
rtest -t n -s <site_name> -q
```

UPLOAD

It is the final stage where the data is being delivered to the clients location. In this stage we upload the extracted data to our API server. If client wanted to deliver to their location the we will upload the data to clients location.

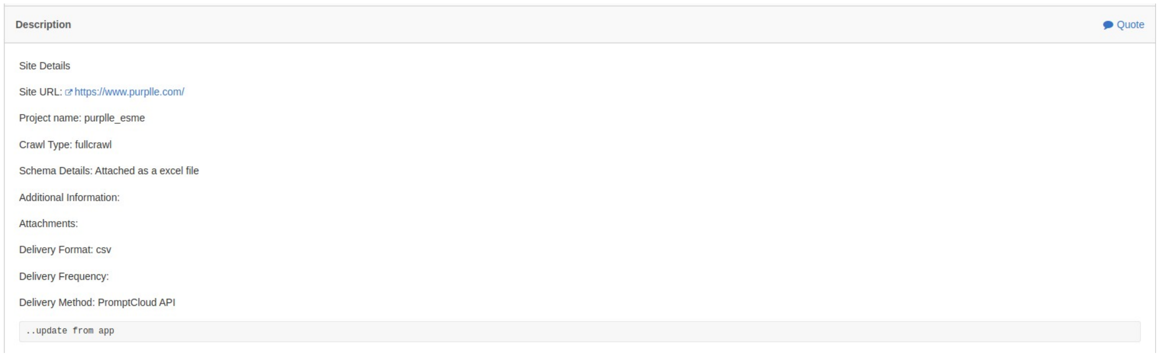
Command:

```
rtest -t u -s <site_name> -q
```

INPUT AND OUTPUT

INPUT

We take seed_url(site url) and the fields to be extracted as an input from the client.



Description Quote

Site Details

Site URL: https://www.purplle.com/

Project name: purplle_esme

Crawl Type: fullcrawl

Schema Details: Attached as a excel file

Additional Information:

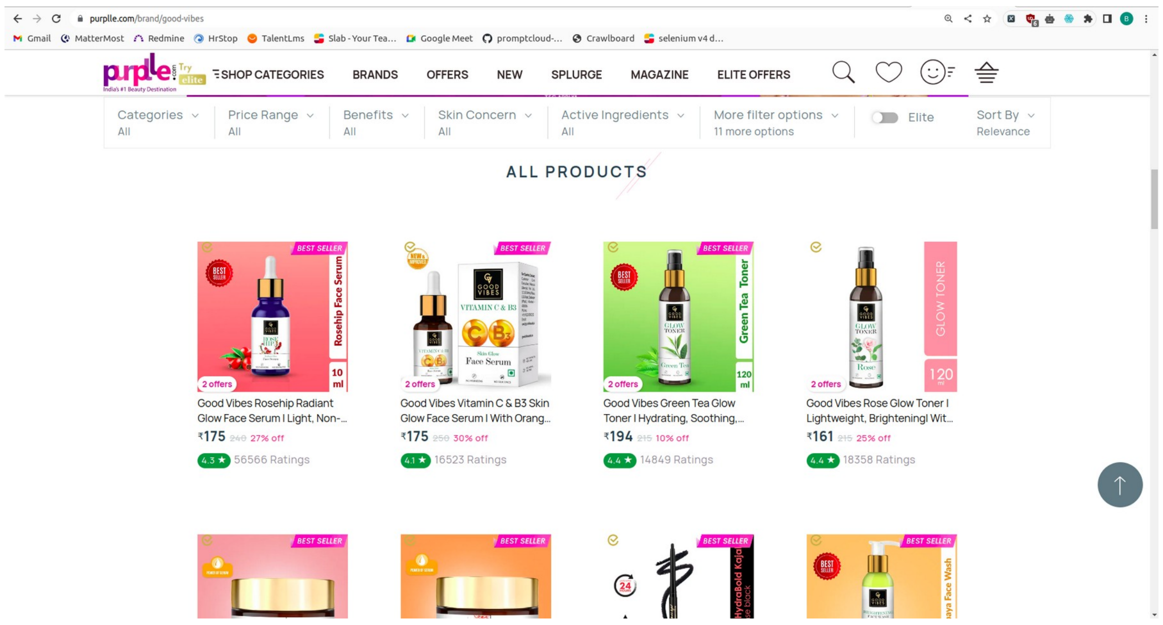
Attachments:

Delivery Format: csv

Delivery Frequency:

Delivery Method: PromptCloud API

```
..update from app
```



Good Vibes Rosehip Radiant Glow Face Serum I Light, Non-...
₹175 ~~240~~ 27% off
4.3 ★ 56566 Ratings

Good Vibes Vitamin C & B3 Skin Glow Face Serum I With Orang...
₹175 ~~250~~ 30% off
4.1 ★ 16523 Ratings

Good Vibes Green Tea Glow Toner I Hydrating, Soothing,...
₹194 ~~215~~ 10% off
4.4 ★ 14849 Ratings

Good Vibes Rose Glow Toner I Lightweight, BrighteningI Wit...
₹161 ~~215~~ 25% off
4.4 ★ 18358 Ratings

OUTPUT

We take site urls as an input and produce an xml document containing the required fields of data as an output.

```
<root>
<page>
  <pageurl>https://www.purplle.com/product/good-vibes-papaya-brightening-face-wash-
deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-
testing-120-ml</pageurl>
  <entity>
    <record>
      <uniq_id>219fde1483c1bb8a8510fb6778530aac</uniq_id>
      <crawl_timestamp>2023-05-04 05:12:22 UTC</crawl_timestamp>
      <pageurl>https://www.purplle.com/product/good-vibes-papaya-brightening-face-
wash-deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-
animal-testing-120-ml</pageurl>
      <product_name>Good Vibes</product_name>
      <list_price type="float">215.0</list_price>
      <sale_price type="float">178.0</sale_price>
      <discount type="float">37.0</discount>
      <brand>Good Vibes</brand>
      <avg_rating type="float">4.3</7avg_rating>
      <number_of_reviews type="float">11223.0</number_of_reviews>
      <seller_name>Avni Beauty Distributors LLP (MP) - PB MUM</seller_name>
      <category>Skincare</category>
      <sub_category_1>Cleansers</sub_category_1>
      <sub_category_2>Face Washes</sub_category_2>
      <sub_category_3>Good Vibes Face Washes</sub_category_3>
      <best_seller_ranking>#4 Best Seller</best_seller_ranking>
      <available_offers>Additional Free Gift of your choice on Rs.799 across Good Vibes.
Free Gift will be auto-added to your cart. This Offer is not valid on Sheet Masks|Choose
your free gift on cart orders above Rs.399 across Good Vibes. FREE Product will be
auto-added to your cart. Offer is not valid on Sheet Masks</available_offers>
```

```
<image_urls>https://media6.ppl-media.com/tr:h-75,w-75,c-at_max,dpr-2/static/img/
product/339004/good-vibes-papaya-brightening-face-wash-deep-pore-cleansing-non-
drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-testing-120-
ml_1_display_1681136508_b256c0ff.jpg|https://media6.ppl-media.com/tr:h-75,w-75,c-
at_max,dpr-2/static/img/product/339004/good-vibes-papaya-brightening-face-wash-deep-
pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-testing-
120-ml_2_display_1681136509_c5146f19.jpg|https://media6.ppl-media.com/tr:h-75,w-
75,c-at_max,dpr-2/static/img/product/339004/good-vibes-papaya-brightening-face-wash-
deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-
testing-120-ml_3_display_1681136510_53bb3ffd.jpg|https://media6.ppl-media.com/tr:h-
75,w-75,c-at_max,dpr-2/static/img/product/339004/good-vibes-papaya-brightening-face-
wash-deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-
animal-testing-120-ml_4_display_1681136511_b170f77c.jpg|https://media6.ppl-
media.com/tr:h-75,w-75,c-at_max,dpr-2/static/img/product/339004/good-vibes-papaya-
brightening-face-wash-deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-
mineral-oil-no-animal-testing-120-ml_5_display_1681136512_6763bd76.jpg|https://
media6.ppl-media.com/tr:h-75,w-75,c-at_max,dpr-2/static/img/product/339004/good-
vibes-papaya-brightening-face-wash-deep-pore-cleansing-non-drying-with-mulberry-no-
parabens-no-mineral-oil-no-animal-testing-120-
ml_6_display_1681136513_076e4525.jpg|https://media6.ppl-media.com/tr:h-75,w-75,c-
at_max,dpr-2/static/img/product/339004/good-vibes-papaya-brightening-face-wash-deep-
pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-testing-
120-ml_7_display_1681136514_7f1fd247.jpg|https://media6.ppl-media.com/tr:h-75,w-
75,c-at_max,dpr-2/static/img/product/339004/good-vibes-papaya-brightening-face-wash-
deep-pore-cleansing-non-drying-with-mulberry-no-parabens-no-mineral-oil-no-animal-
testing-120-ml_8_display_1681136515_92e7f4de.jpg|https://media4.purplle.com/static/
img/brand//__1620634041_e49bdaf1.jpg</image_urls>
    </record>
  </entity>
</page>
</root>
```

CONCLUSION

We demonstrated the process of web scrapping in our pipeline which takes site_url and fields to be extracted as an input and produces an xml document containing all the required fields of data. In this we taken a site_url from client and then fetched that page. And extracted the data fields which are required by the client. Then normalized the data into required format. And then uploades the data to the client location.

REFERENCES

1. https://redmine.promptcloud.com/issues/328349
2. https://www.geeksforgeeks.org/what-is-web-scraping-and-how-to-use-it/
3. https://promptcloud.slab.com/posts/complete-project-setup-in-the-pipeline-step-by-step-d-scraper-api-0464lbrv
4. https://redmine.promptcloud.com/projects/sdftech/wiki/SE_SSE_Training_Module_2
5. https://www.tutorialspoint.com/ruby/index.htm