5.1 Open-source availability and infrastructure

The corgibrowser framework is an open-source project, and it's expected to be used by the Python community and start getting new features from users integrated into the modules of Web Crawling, Web Scraping, Data Management, and Data Analysis.

Currently on the Data Management module it's only supported one type of storage for data, but with more users on boarding on the project, it's expected to start getting more possibilities to integrate with cloud providers, or use local storage services to manage the tables, queues, or objects.

For new users to add features to the framework they will need to start a new Pull Request, pass all tests from the Pipeline, and get approval of one of the repository owners.

- Framework Tests: https://github.com/j-enriquez/corgibrowser/tree/main/tests

Users can use the framework directly installing it from the Python Package Index:

pip install corgibrowser

https://pypi.org/project/corgibrowser/

Or accessing the code from the GitHub repository:

gh repo clone j-enriquez/corgibrowser

https://github.com/j-enriquez/corgibrowser

When a Pull request is created, it will trigger a Pipeline to run validations for each change made, and at least 1 reviewer will be required to allow pull request to be merged:

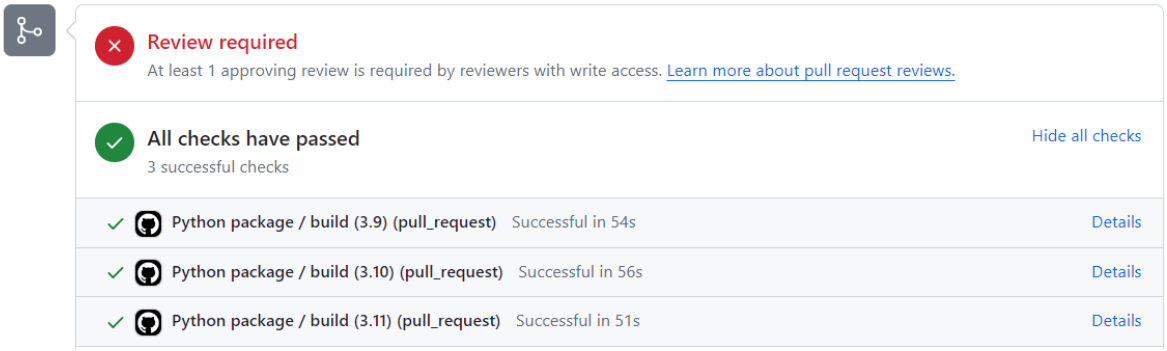


Figure 1: Example of pipeline requirements to merge pull request

And all tests for this specific pull request need to pass for build to pass.

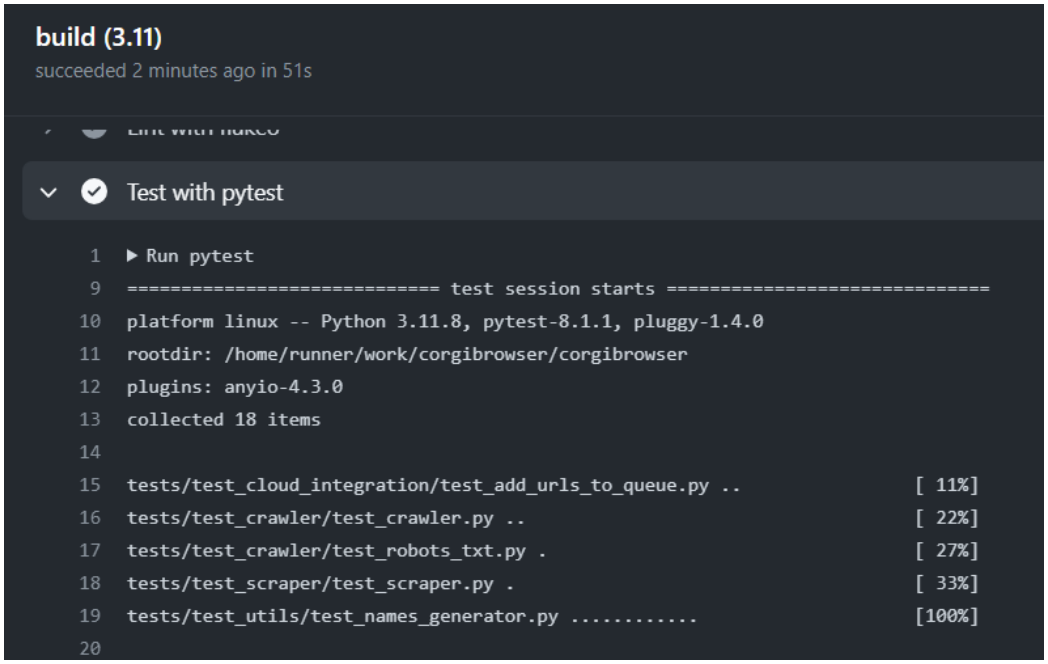


Figure 2: Example of Tests needed by the pipeline

5.2 Build in tools available to end-users for framework to be considered easy to use.

To start using the framework with default configurations, the users need to do the following steps:

- A) Set up Azure Storage Account
- B) Create a Crawler
- C) Create a Scraper

A) Instructions to set up an Azure Storage Account can be found in the Following url: https://learn.microsoft.com/en-us/azure/storage/common/storage-account-create?tabs=azure-portal

A.1 Create storage account

https://portal.azure.com/#create/Microsoft.StorageAccount-ARM

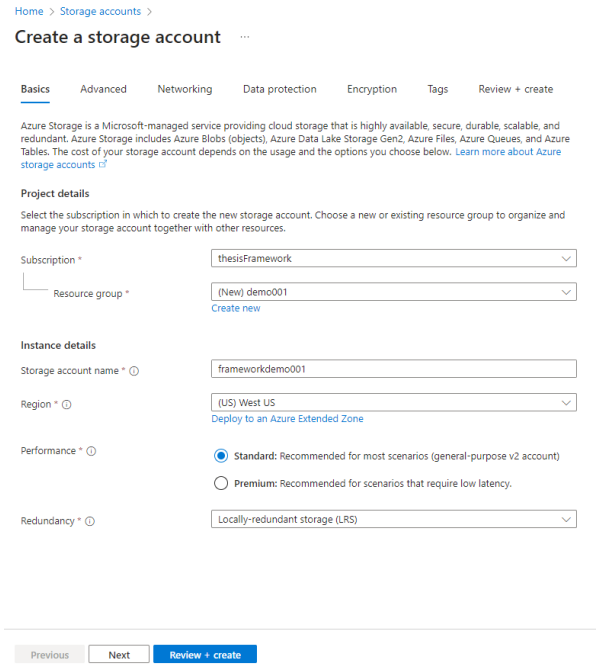


Figure 3: How to create a Storage Account

A.2 Review and create Storage Account

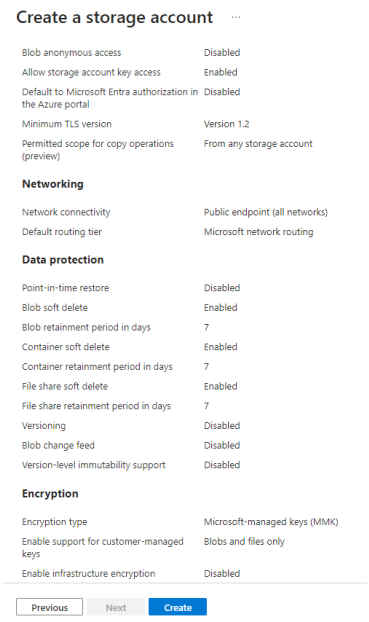


Figure 4: Review and create Storage Account

A.3) Retrieve access keys

https://learn.microsoft.com/en-us/azure/storage/common/storage-account-keys-manage?tabs=azure-portal

The account name and key1 will be used to access the Storage Account, this keys will allow the framework to communicate with the tables, queues and object storage.

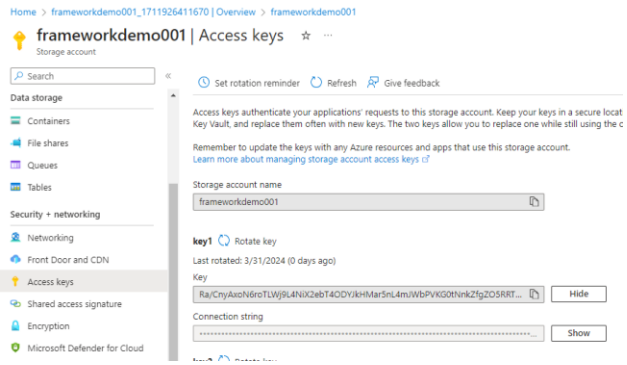


Figure 5: Where to find Access Keys of Storage Account

B) Create a Crawler

https://github.com/j-enriquez/corgibrowser/blob/main/user/default/demo_crawler.py

In the following example, from lines 1-17 the user is importing the framework and using the SettingsManager to update the default values to connect with the Storage Account. With the Cloud integration is creating an instance where it will be connecting the Storage Account.

On lines 20-21, is using the DataSetsManager that contains default datasets that are useful to initialize the crawler with certain domain urls.

On the lines 24-26 user is initializing the crawler and this is all the work needed from users to visit websites.

Code Blame 26 lines (22 loc) · 1019 Bytes Raw

```
1  import os
2  from dotenv import load_dotenv
3  from corgibrowser.corgi_cloud_integration.cloud_integration import CloudIntegration
4  from corgibrowser.corgi_datasets.DataSetsManager import DataSetsManager
5  from corgibrowser.corgi_settings.SettingsManager import SettingsManager
6  from corgibrowser.corgi_crawler.crawler import *
7
8  # Load Settings Manager
9  settings_manager = SettingsManager()
10 load_dotenv()
11 settings_manager.CLOUD["AZURE_STORAGE_ACCOUNT_NAME"] = os.getenv("AZURE_STORAGE_ACCOUNT_NAME")
12 settings_manager.CLOUD["AZURE_STORAGE_ACCOUNT_KEY"] = os.getenv("AZURE_STORAGE_ACCOUNT_KEY")
13
14 # Set Up cloud
15 CloudIntegration(settings_manager = settings_manager)
16 cloud_integration = CloudIntegration( settings_manager = settings_manager )
17 cloud_integration.initialize()
18
19 # Add Initial URLs
20 for url in DataSetsManager.load_usa_newspaper_urls():
21     cloud_integration.add_url_to_queue(url)
22
23 # Crawl
24 crawler = WebCrawler(cloud_integration = cloud_integration, settings_manager=settings_manager )
25 crawler.initialize()
26 crawler.start()
```

Figure 6: Code of example default crawler

C) Create a Web Scraper:

https://github.com/j-enriquez/corgibrowser/blob/main/user/default/demo_scraper.py

Lines 1-16 follow the similar steps from the crawler to import the packages, configure the settings manager, and connections with cloud provider.

From line 19-21 it's what user will run to start working on web scraping.

Code Blame 21 lines (18 loc) · 837 Bytes Raw

```
1   import os
2   from dotenv import load_dotenv
3   from corgibrowser.corgi_cloud_integration.cloud_integration import CloudIntegration
4   from corgibrowser.corgi_settings.SettingsManager import SettingsManager
5   from corgibrowser.corgi_webscraping.scraper import Scraper
6
7   # Load Settings Manager
8   settings_manager = SettingsManager()
9   load_dotenv()
10  settings_manager.CLOUD["AZURE_STORAGE_ACCOUNT_NAME"] = os.getenv("AZURE_STORAGE_ACCOUNT_NAME")
11  settings_manager.CLOUD["AZURE_STORAGE_ACCOUNT_KEY"] = os.getenv("AZURE_STORAGE_ACCOUNT_KEY")
12
13  # Set Up cloud
14  CloudIntegration(settings_manager = settings_manager)
15  cloud_integration = CloudIntegration( settings_manager = settings_manager )
16  cloud_integration.initialize()
17
18  # Scrape
19  scraper = Scraper(cloud_integration = cloud_integration, settings_manager=settings_manager )
20  scraper.initialize()
21  scraper.start()
```

Figure 7: Code of example default scraper

5.3 Compliance modules to assure Robots.txt file is respected.

Robots.txt is a file provided by owners of websites to allow or disallow a particular use of a user-agent.

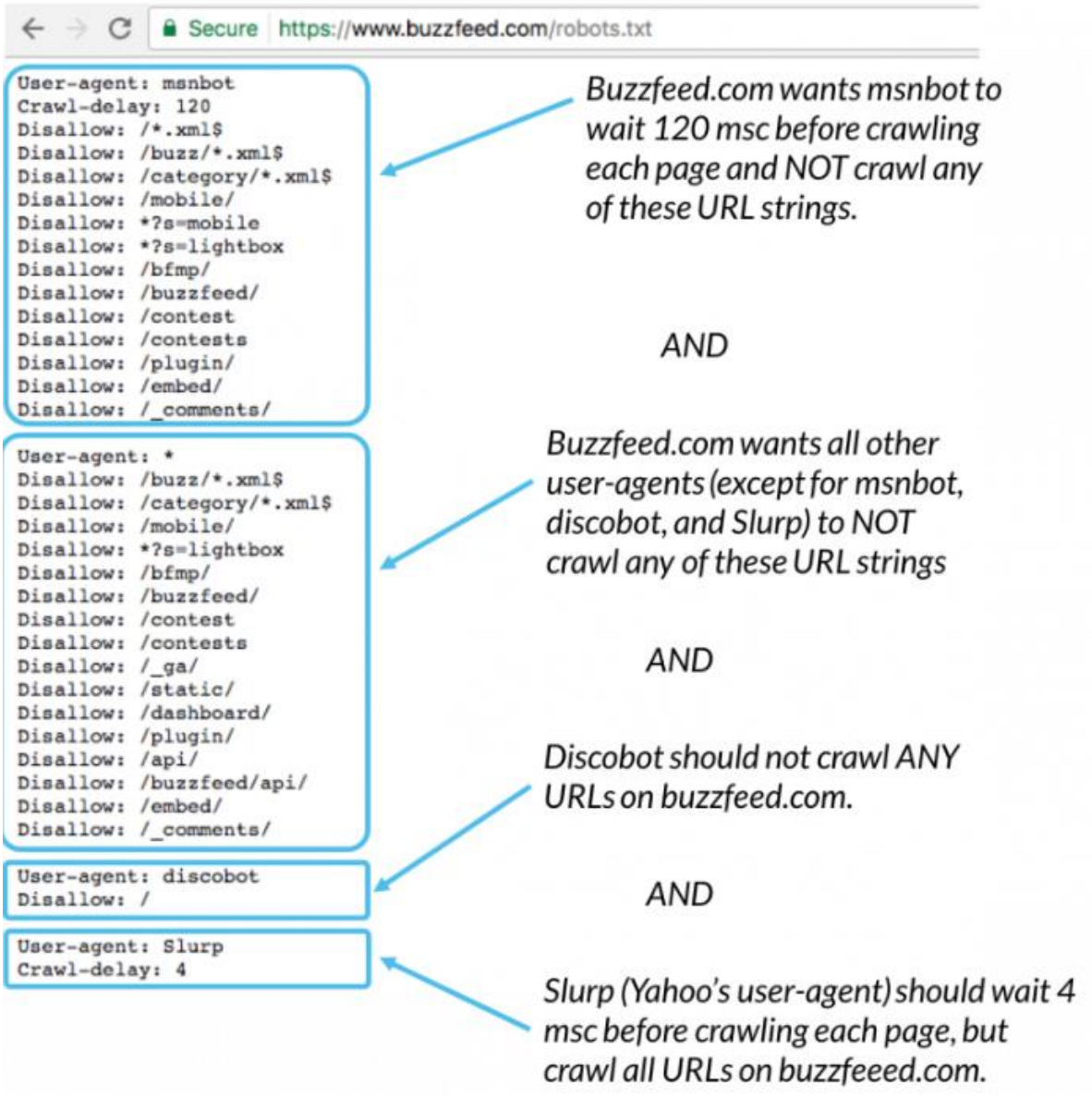


Figure 8: Example of robots.txt file by moz.com

On the previous example from moz.com, we can see how buzzfeed wants each of the user agents to interact with the website, allowing certain paths to be visited, and including the crawl-delay.

On Corgibrowser this is managed by the "RobotsCache", this class uses the library urllib.robotparser to determine if crawler can fetch.

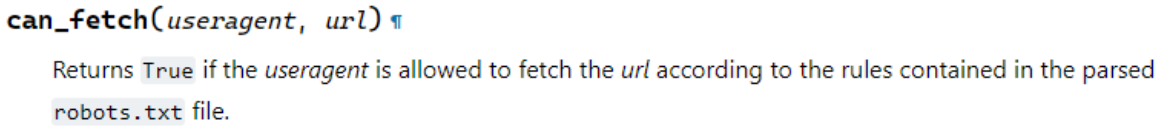


Figure 9: Python urllib.robotparser method

At the initialization of each crawler before making a visit to a website, the framework reviews the /robots.txt file and determines if the call can be made based on the can_fetch parameters, and if we are compliant with the website crawl-delay (if website does not contain a crawl-delay, default by the framework is 1).

To be compliant with the crawl-delay rules from the website, corgibrowser implemented a table called corgiwebtrottling, where based on the last timestamp we can determine the last time the website was visited. Access to this table is by PartitionKey and RowKey, which makes the access of the information faster by calling directly to the index we want to retrieve. Logic of can_process_domain is on the Cloud_Integration module. There are future improvements in this area to allow the use of an in-memory cache like Redis instead of a table and thanks to the modular and domain area approach of the framework, the logic will stay the same for crawler, and only adaptions on the DataManagement domain will be needed.

Tables > corgiwebthrottling

Authentication method: Access key (Switch to Microsoft Entra user account)

Add filter

Sorting all 1228 items

PartitionKey	RowKey	Timestamp	ThrottlingLimitSeconds ↓
RateLimit	www.wcgazette.com	2024-03-25T19:23:45.09...	10
RateLimit	www.westseattleherald.com	2024-03-25T09:26:31.25...	10
RateLimit	www.whyy.org	2024-03-25T19:22:19.24...	10
RateLimit	thechampionnewspaper.com	2024-03-25T19:22:00.68...	5
RateLimit	longislandweekly.com	2024-03-25T19:21:50.42...	3
RateLimit	www.dallasvoice.com	2024-03-25T19:23:26.28...	3
RateLimit	www.decorahnewspapers.com	2024-03-25T19:22:22.35...	3
RateLimit	www.georgiabulletin.org	2024-03-25T19:22:38.81...	3
RateLimit	www.kten.com	2024-03-25T19:23:32.98...	3
RateLimit	www.wgmd.com	2024-03-25T19:22:18.80...	3
RateLimit	www.whitewaterbanner.com	2024-03-25T19:23:45.89...	3

Figure 10: corgiwebthrottling table

For compliance and to be able to have proof of all requests made, on the table corgiwebrequestslog will be stored each http call made to an external service, if website specific url was not visited due to website being throttled or robots.txt not allowing a specific path, this data will also be stored on the logs table. Each log will consist of domain partition key for the specific website, the timestamp when this call was made as the row key to allow queries by time to be made easier, instance id of the unique process, original url from the website to visit, and the status code responded from website or the reason why website was not visited.

Tables > corgiwebrequestslog

Authentication method: Access key (Switch to Microsoft Entra user account)

Add filter

Showing the first 100 items

PartitionKey	RowKey	Timestamp	InstanceId	StatusCode	Url
abc30.com	20240325T084039Z	2024-03-25T08:40:39.98...	demo001-20240325084...	200	http://abc30.com/
abc30.com	20240325T085421Z	2024-03-25T08:54:21.23...	demo001-20240325085...	200	http://abc30.com/
abc30.com	20240325T090646Z	2024-03-25T09:06:46.70...	demo001-20240325090...	200	http://abc30.com/
abc7chicago.com	20240325T084040Z	2024-03-25T08:40:40.30...	demo001-20240325084...	200	http://abc7chicago.com/
abc7chicago.com	20240325T085421Z	2024-03-25T08:54:21.88...	demo001-20240325085...	200	http://abc7chicago.com/
abc7chicago.com	20240325T090647Z	2024-03-25T09:06:47.70...	demo001-20240325090...	200	http://abc7chicago.com/
abcnews.go.com	20240325T084039Z	2024-03-25T08:40:39.43...	demo001-20240325084...	200	https://abcnews.go.com/
abcnews.go.com	20240325T084040Z	2024-03-25T08:40:40.52...	demo001-20240325084...	Throttled	https://abcnews.go.com/
abcnews.go.com	20240325T085422Z	2024-03-25T08:54:22.27...	demo001-20240325085...	200	https://abcnews.go.com/
abcnews.go.com	20240325T090648Z	2024-03-25T09:06:48.14...	demo001-20240325090...	200	https://abcnews.go.com/
acuoptimist.com	20240325T084039Z	2024-03-25T08:40:39.83...	demo001-20240325084...	NotAllowed	http://acuoptimist.com/
acuoptimist.com	20240325T084040Z	2024-03-25T08:40:40.81...	demo001-20240325084...	NotAllowed	http://acuoptimist.com/
acuoptimist.com	20240325T085422Z	2024-03-25T08:54:22.64...	demo001-20240325085...	NotAllowed	http://acuoptimist.com/
advancetitan.com	20240325T084039Z	2024-03-25T08:40:39.93...	demo001-20240325084...	NotAllowed	https://advancetitan.com/

Figure 11: corgiwebrequestslog table

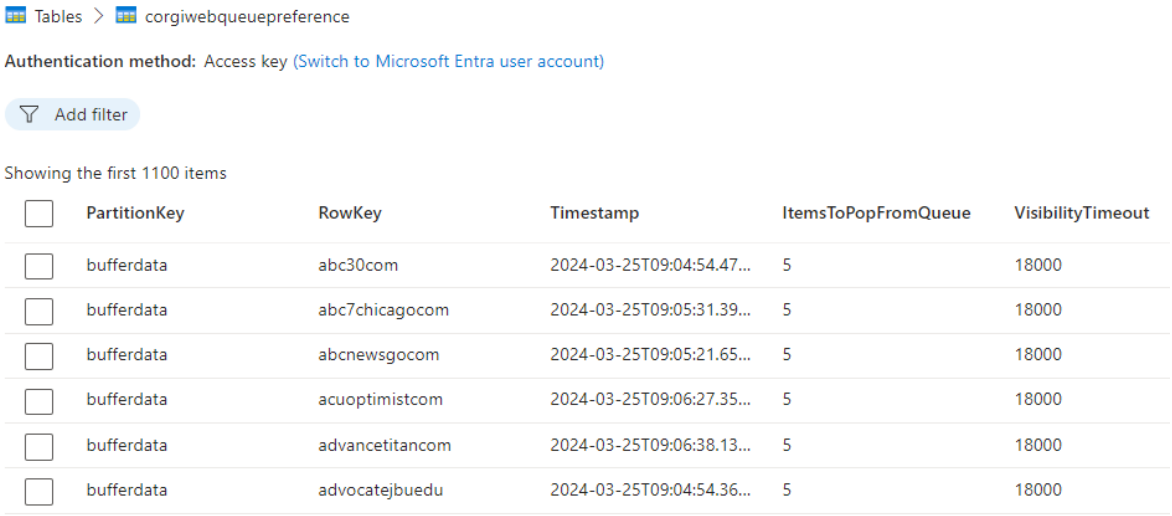
On the web scraping domain, when we are reviewing all the new urls to see if this can be added to the queue, the framework also validates if this URL's can be visited by Robots.txt by making a call to the **can_fetch** method before processing a new url found.

5.4 Partitioned Tables/Queuing system to allow customized search patterns to target specific websites

Users will be able to decide which website domains the crawler/scraper instances will be running and getting values from its queues. For each website domain, the DataManagement area will create a Table, a Queue, and a Container.

Because each domain is partitioned the user can decide how to split the work between instances and prioritize the search of certain website domains based on the business needs.

If the setting “QUEUE_ONLY_DOMAINS” variable is empty, the default value will be to retrieve all domains available to visit from the table “corgiwebqueuepreference”.


Tables > corgiwebqueuepreference

Authentication method: Access key (Switch to Microsoft Entra user account)

Add filter

Showing the first 1100 items

PartitionKey	RowKey	Timestamp	ItemsToPopFromQueue	VisibilityTimeout
bufferdata	abc30com	2024-03-25T09:04:54.47...	5	18000
bufferdata	abc7chicagocom	2024-03-25T09:05:31.39...	5	18000
bufferdata	abcnewsgocom	2024-03-25T09:05:21.65...	5	18000
bufferdata	acuoptimistcom	2024-03-25T09:06:27.35...	5	18000
bufferdata	advancetitancom	2024-03-25T09:06:38.13...	5	18000
bufferdata	advocatejbuedu	2024-03-25T09:04:54.36...	5	18000

Figure 12: Example of corgiwebqueuepreference table

On the following example the user is creating a crawler that will only be visiting the websites from the specified domains and using the values on the table "corgiwebqueuepreference" for ItemsToPopFromQueue and VisibilityTimeout:

```
# Crawl
settings_manager.CRAWLER["QUEUE_ONLY_DOMAINS"] = [
    "fbrefcom",
    "wwweluniversalcommx",
    "abcnewsgocom",
    "wwwcnncom"]
crawler = WebCrawler(cloud_integration = cloud_integration, settings_manager=settings_manager )
crawler.initialize()
crawler.start()
```

Figure 13: Code example of Crawler with specific domains

The following image shows an example of the queues by domain:

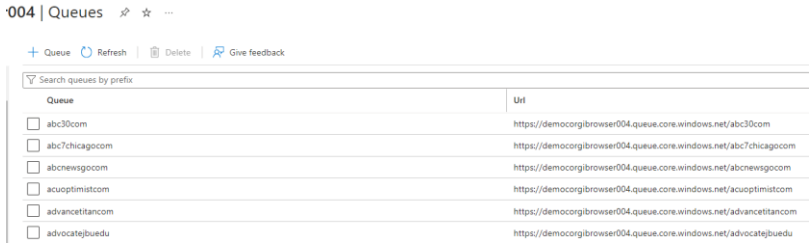


Figure 14: Domain Specific Queues

And each of the queues will store the messages of the new URL's to visit following the schema of corgi_web_queue_version_1

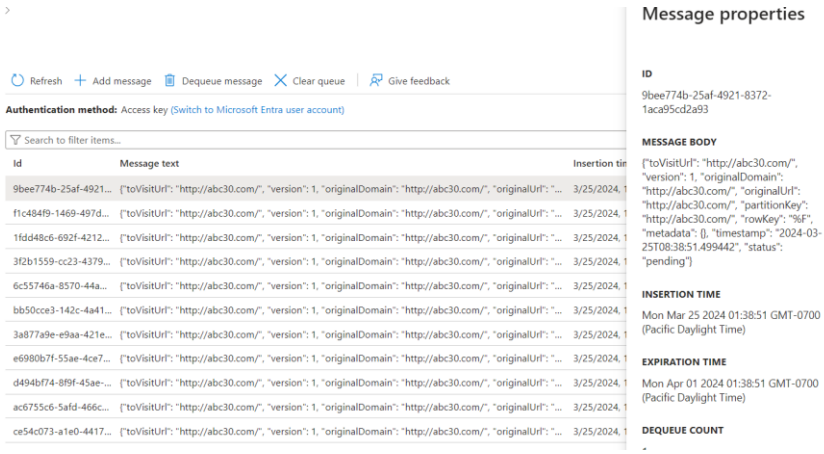


Figure 15: Message properties of queues

The tables are also partitioned by domain

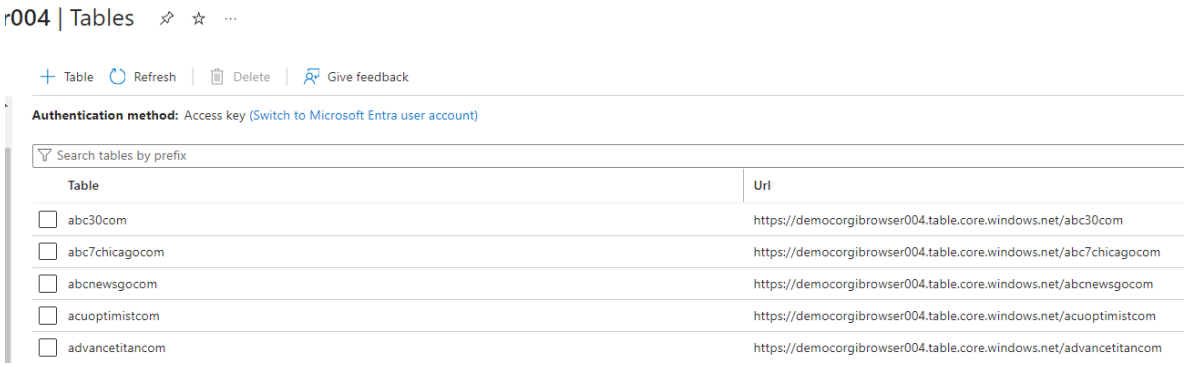


Figure 16: Domain Specific Tables

And each of the tables will follow the shema of [corgiweb_queuepreference], containing relevant information about the visited urls and the urls to visit. This table can help in case re-visit rules are needed to add visited URLs to the queue.

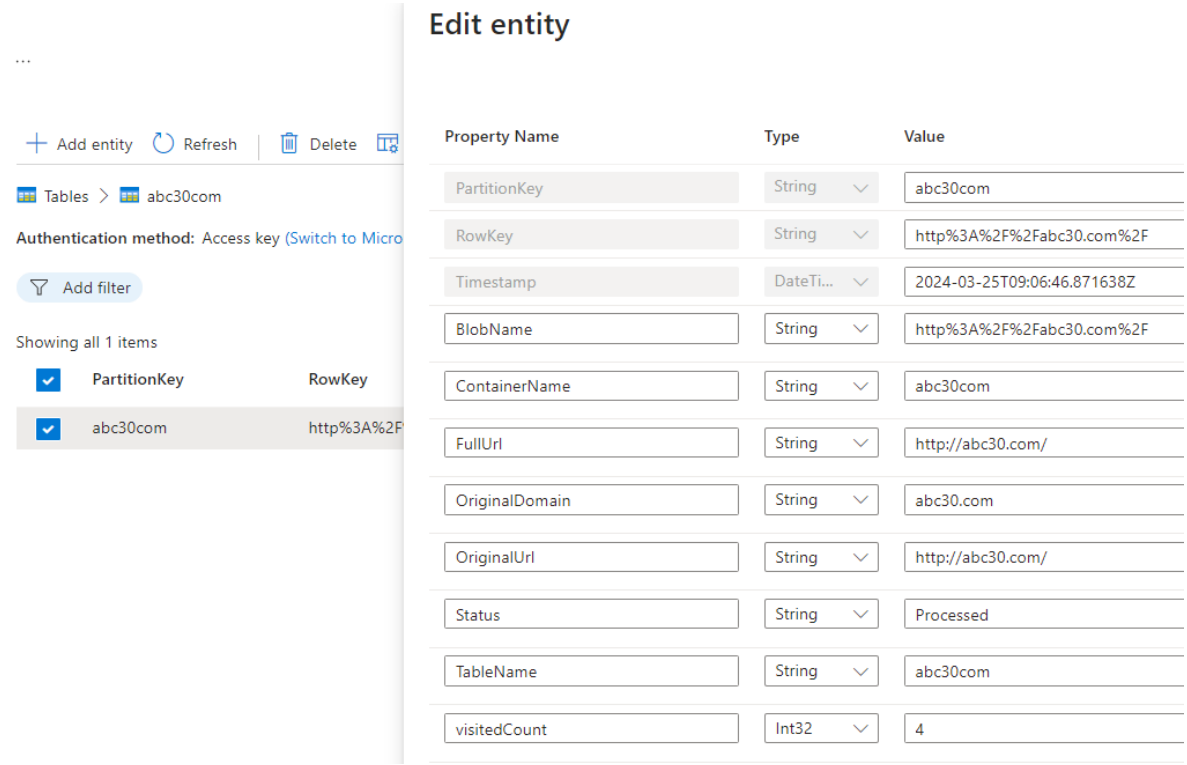


Figure 17: corgiweb_queuepreference entity values

Containers are also partitioned by domain.

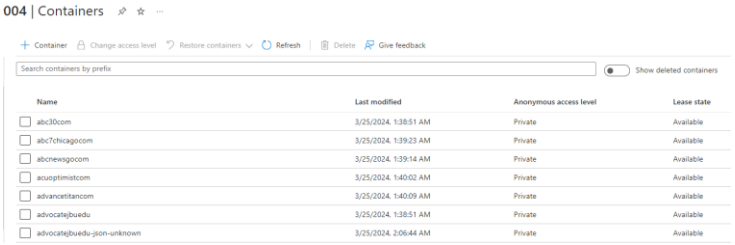


Figure 18: Domain Specific Containers

And initially from the crawler the data is stored in the form of the original HTML

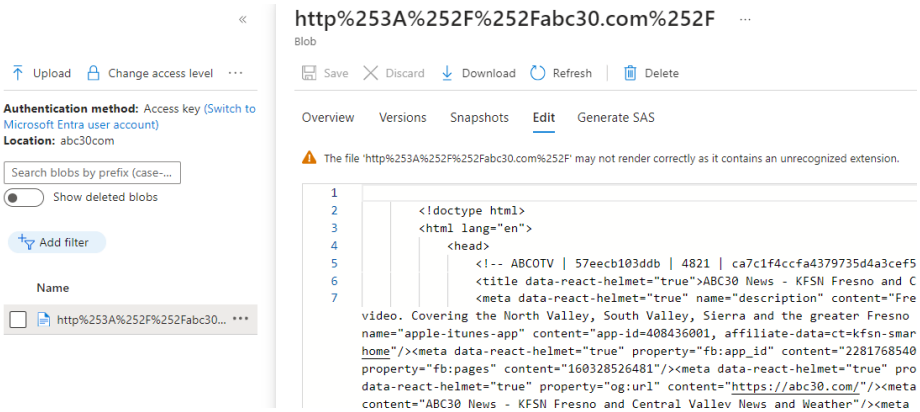


Figure 19: Example of downloaded url as HTML

After if the url is already processed by the Web scraping module, the data will be stored in form of a JSON.

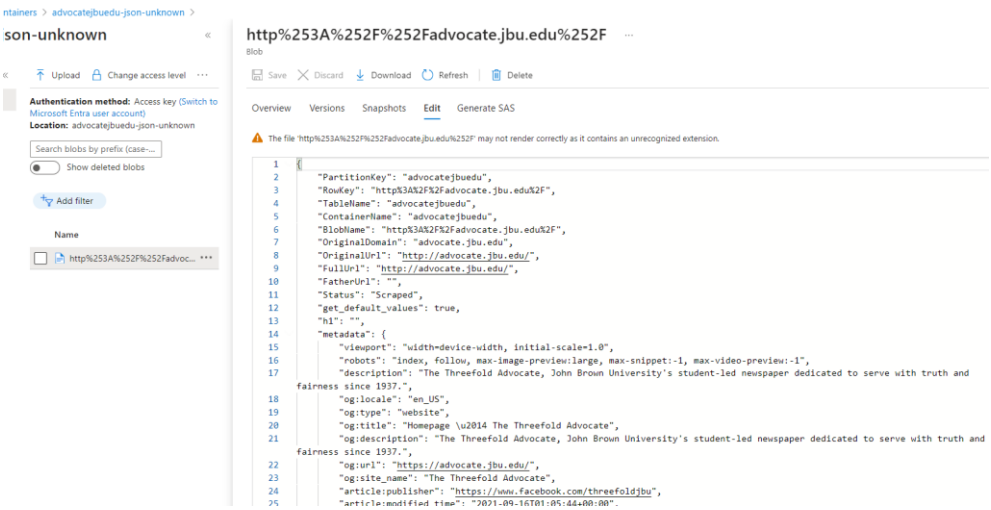


Figure 20: Example of Downloaded and processed URL as a JSON

5.5 Visited websites hash table

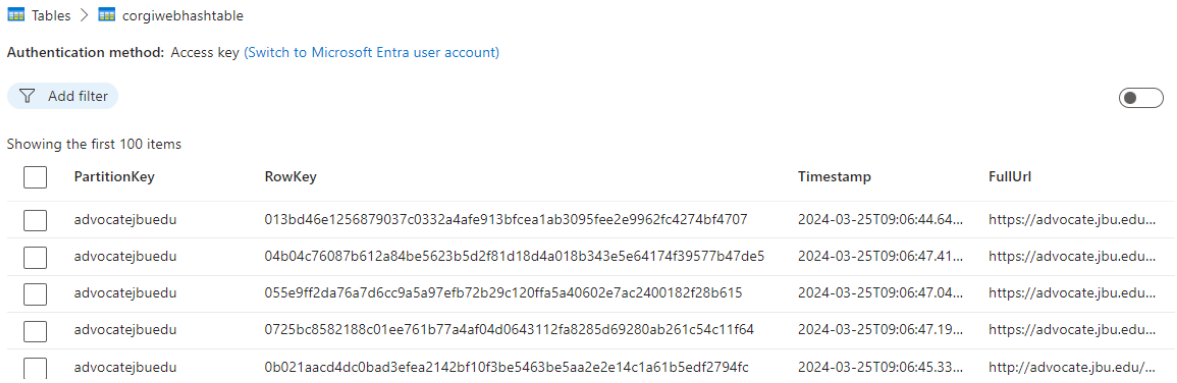
For each url that is added to the webcrawling tables or queues, the framework encodes the url with SHA-256 using UrlHash class.

Code Blame 26 lines (19 loc) · 645 Bytes

```
1
2     import hashlib
3
4     class UrlHash:
5         @staticmethod
6         def encode_url(url):
7             """
8             Encodes a URL into a unique hash string using SHA-256.
9
10            Parameters:
11            - url (str): The URL to be encoded.
12
13            Returns:
14            - str: A SHA-256 hash of the URL.
15            """
16            # Encode the URL to a bytes object required by hashlib
17            url_bytes = url.encode( 'utf-8' )
18
19            # Create a sha256 hash object
20            hasher = hashlib.sha256()
21
22            # Update the hash object with the bytes of the URL
23            hasher.update( url_bytes )
24
25            # Return the hexadecimal digest of the URL
26            return hasher.hexdigest()
```

Figure 21: Code of UrlHash

Each Hash is added to the corgiwebhashtable, stored by partition and hash as the rowkey.



PartitionKey	RowKey	Timestamp	FullUrl
advocatejbuedu	013bd46e1256879037c0332a4afe913bfcea1ab3095fee2e9962fc4274bf4707	2024-03-25T09:06:44.64...	https://advocate.jbu.edu...
advocatejbuedu	04b04c76087b612a84be5623b5d2f81d18d4a018b343e5e64174f39577b47de5	2024-03-25T09:06:47.41...	https://advocate.jbu.edu...
advocatejbuedu	055e9ff2da76a7d6cc9a5a97efb72b29c120ffa5a40602e7ac2400182f28b615	2024-03-25T09:06:47.04...	https://advocate.jbu.edu...
advocatejbuedu	0725bc8582188c01ee761b77a4af04d0643112fa8285d69280ab261c54c11f64	2024-03-25T09:06:47.19...	https://advocate.jbu.edu...
advocatejbuedu	0b021aacd4dc0bad3efea2142bf10f3be5463be5aa2e2e14c1a61b5edf2794fc	2024-03-25T09:06:45.33...	http://advocate.jbu.edu/...

Figure 22: corgiwebhashtable example

The user can decide on the initialization of each of the scraping instances which HASH_PARTITIONS wants to retrieve and manage in-memory to reduce the number of calls to the tables to know if an url was already visited. On the following example the user is specifying 4 partitions to initialize the Scraper with in memory storage of the urls hash list.

```
28    settings_manager.SCRAPER["HASH_PARTITIONS"] = [
29        "fbrefcom",
30        "wwweluniversalcommx",
31        "abcnewsgocom",
32        "wwwcnncom"]
33  > scraper_dict = scraper_dict = {  ···
38    }
39    scraper = Scraper(cloud_integration = cloud_integration, settings_manager=settings_manager, scraper_dict=scraper_dict )
40    scraper.initialize()
41    scraper.start()
```

Figure 23: Code of simple Scraper using HASH_PARTITIONS

5.6 Availability to integrate customized scraping templates.

For Web crawling and Web scraping, users need the possibility to use their own customized templated for scraping, and in some cases use their preferred library to manipulate the html retrieved files.

On the following example user/customTemplates/demo_scraper_templates.py the custom user is making a scraper targeting 4 domains. User is providing values in ONLY_DOMAINS to determine from which containers wants to read html files. HASH_PARTITIONS to use an in-memory cache of visited urls and identify visited urls in memory, and scraper_dict which maps each domain with a customized class that will determine the different rules for scraping each of the websites.

```
# Scrape
settings_manager.SCRAPER["ONLY_DOMAINS"] = [
    "fbrefcom",
    "wwweluniversalcommx",
    "abcnewsgocom",
    "wwwcnncom"]
settings_manager.SCRAPER["HASH_PARTITIONS"] = [
    "fbrefcom",
    "wwweluniversalcommx",
    "abcnewsgocom",
    "wwwcnncom"]
scraper_dict = scraper_dict = {
    "fbrefcom": fbrefcom,
    "wwweluniversalcommx": wwweluniversalcommx,
    "abcnewsgocom": abcnewsgocom,
    "wwwcnncom": wwwcnncom
}
scraper = Scraper(cloud_integration = cloud_integration, settings_manager=settings_manager, scraper_dict=scraper_dict )
scraper.initialize()
scraper.start()
```

Figure 24: Sharding of database example

On the following file there is a customized example of one of the customized scrapings class:

[user/customTemplates/scraping_templates/abcnewsgocom.py]

Where the user uses the ScrapingTemplate as base, and then provides customized rules on how this website data is retrieved, all the data saved on .extra_keys will automatically be stored on the result container in the JSON file created from the scraper.

Code Blame 43 lines (30 loc) · 1.96 KB

```python
from urllib.parse import unquote

from bs4 import BeautifulSoup
from parsel import Selector

from corgibrowser.corgi_webscraping.default_scrape_template import ScrapingTemplate


class abcnewsgocom(ScrapingTemplate):
    def initialize(self, ):
        self.soup = BeautifulSoup( self.html_text, "lxml" )

    def extra_data(self, ):
        self.sel = Selector( text = self.html_text )

        if self.sel.xpath("//meta[@property='og:type' and @content='article']" ):
            self.handle_article()
        else:
            self.extra_keys[ "ContainerSuffix" ] = "unknown2"
        self.extra_keys[ "html_text" ] = self.html_text


    def handle_homepage(self, ):
        self.extra_keys["ContainerSuffix"] = "homepage"
        self.extra_keys[ "images" ] = ""
        self.extra_keys[ "paragraphs" ] = ""
        self.extra_keys[ "html_text" ] = ""

    def handle_article(self, ):
        self.extra_keys[ "ContainerSuffix" ] = "article"

        content_sel = self.sel.xpath("//div[contains(concat(' ', normalize-space(@class), ' '), ' FITT_Article_main__body')]")
        self.extra_keys["h1"] = self.get_image_urls_by_xpath(self.sel,"//meta[@property='og:title']","/@content")
        self.extra_keys[ "images" ] = self.get_image_urls_by_xpath(content_sel,"//img","/@src")
        self.extra_keys[ "category" ] = self.extract_segment_in_path( unquote(unquote(self.row_key)),0)

        self.extra_keys[ "author" ] = self.get_image_urls_by_xpath( self.sel, "//meta[@name='author']","/@content" )
        self.extra_keys[ "author_date" ] = self.get_image_urls_by_xpath( self.sel,"//meta[@property='lastPublishedDate']","/@content" )

        self.extra_keys[ "paragraphs" ] = self.extract_all_text(content_sel,"//div[contains(concat(' ', normalize-space(@data-testid), ' '), ' prism-article-body')]")

        self.extra_keys["SourceDataField"] = self.extra_keys[ "h1" ] + " " + self.extra_keys["paragraphs"]
        self.extra_keys[ "SourceDataField" ] = self.extra_keys[ "SourceDataField" ][ : 2000 ]
```

Figure 25: Code of abcnewsgocom scraping template

If the key ["ContainerSuffix"] is updated, this will create a new container with the specified name and store the JSON in this direction following these rules:

container_suffix = "-" + "json" + "-" + blob_row["ContainerSuffix"])

On the following example we can see how some of the customized values were retrieved and stored in the JSON for the url.

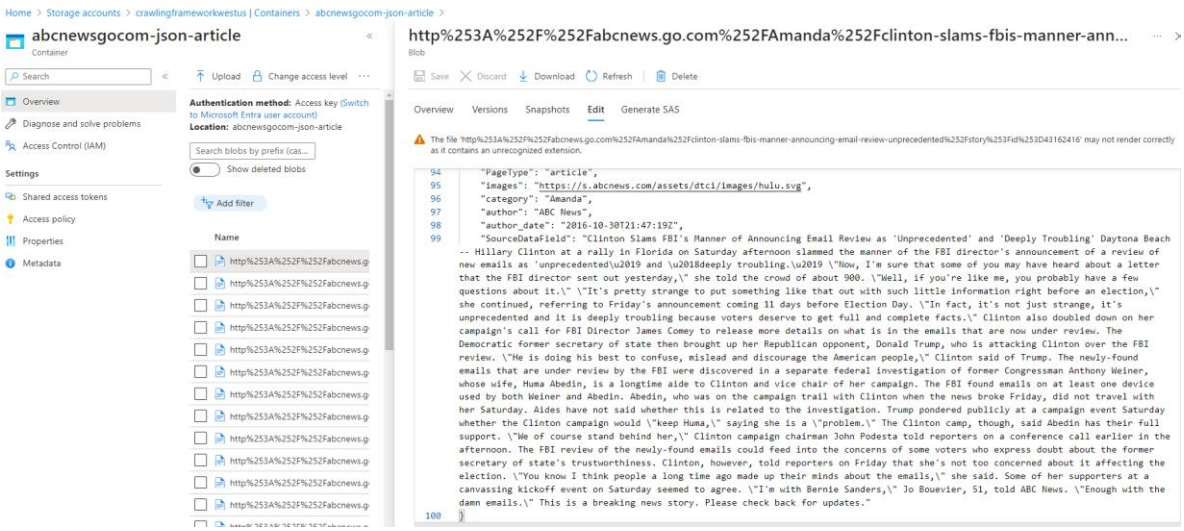


Figure 26: Processed url data from abcnewsgocom

Example of [user/customTemplates/scraping_templates/wwwcnncom.py]

Code Blame 43 lines (31 loc) · 1.94 KB

```python
from bs4 import BeautifulSoup
from parsel import Selector

from corgibrowser.corgi_webscraping.default_scrape_template import ScrapingTemplate


class wwwcnncom(ScrapingTemplate):
    def initialize(self, ):
        self.soup = BeautifulSoup( self.html_text, "lxml" )

    def extra_data(self, ):
        self.sel = Selector( text = self.html_text )

        # if self.sel.xpath( "//meta[@property='og:type' and @content='article']" ):
        #     self.handle_homepage()
        if self.sel.xpath("//main[@class='article__main']" ):
            self.handle_article()
        else:
            self.extra_keys[ "ContainerSuffix" ] = "unknown2"
        self.extra_keys[ "html_text" ] = self.html_text


    def handle_homepage(self, ):
        self.extra_keys["ContainerSuffix"] = "homepage"
        self.extra_keys[ "images" ] = ""
        self.extra_keys[ "paragraphs" ] = ""
        self.extra_keys[ "html_text" ] = ""

    def handle_article(self, ):
        self.extra_keys[ "ContainerSuffix" ] = "article"

        self.extra_keys["h1"] = self.get_image_urls_by_xpath(self.sel,"//meta[@property='og:title']","/@content")
        self.extra_keys[ "images" ] = self.get_image_urls_by_xpath(self.sel,"//meta[@name='twitter:image']","/@content")
        self.extra_keys[ "category" ] = self.get_image_urls_by_xpath(self.sel,"//meta[@name='twitter:image']","/@content")

        self.extra_keys[ "author" ] = self.get_image_urls_by_xpath(self.sel,"//meta[@name='author']","/@content")
        self.extra_keys[ "author_date" ] = self.get_image_urls_by_xpath(self.sel,"//meta[@property='article:published_time']","/@content")

        article_sel = self.sel.xpath( "//div[@class='article__content']" )
        self.extra_keys[ "paragraphs" ] = self.extract_all_text(article_sel,"//p")[: 2000 ]

        self.extra_keys["SourceDataField"] = self.extra_keys[ "h1" ] + " " + self.extra_keys["paragraphs"]
        self.extra_keys["SourceDataField"] = self.extra_keys["SourceDataField"][: 2000 ]
```

Figure 27: Code of wwwcnncom scraping template

Example of [user/customTemplates/scraping_templates/wwweluniversalcommx.py]

Code Blame 44 lines (33 loc) · 2.76 KB Raw

```python
from bs4 import BeautifulSoup
from parsel import Selector

from corgibrowser.corgi_webscraping.default_scrape_template import ScrapingTemplate


class wwweluniversalcommx(ScrapingTemplate):
    def initialize(self, ):
        self.soup = BeautifulSoup( self.html_text, "lxml" )

    def extra_data(self, ):
        self.sel = Selector( text = self.html_text )

        if self.sel.xpath( "//meta[@name='mrf:sections' and @content='homepage']" ):
            self.handle_homepage()
        if self.sel.xpath( "//body[contains(concat(' ', normalize-space(@class), ' '), ' homepage')]" ):
            self.handle_homepage()
        if self.sel.xpath( "//h1[contains(concat(' ', normalize-space(@class), ' '), 'home-custom-title')]" ):
            self.handle_homepage()
        if self.sel.xpath( "//meta[@property='og:type']/@content" ) and self.sel.xpath( "//div[contains(concat(' ', normalize-space(@class), ' '), ' encabezado ')]" ) and self.sel.xpat
            self.handle_article()

    def handle_homepage(self, ):
        self.extra_keys["ContainerSuffix"] = "homepage"
        self.extra_keys[ "images" ] = ""
        self.extra_keys[ "paragraphs" ] = ""
        self.extra_keys[ "html_text" ] = ""

    def handle_article(self, ):
        self.extra_keys[ "ContainerSuffix" ] = "article"

        headers_sel = self.sel.xpath("//div[contains(concat(' ', normalize-space(@class), ' '), ' encabezado ')]")
        self.extra_keys["h1"] = self.get_text_by_xpath(headers_sel,"//h1[contains(@class, 'title') and contains(@class, 'font-bold')]")
        self.extra_keys[ "images" ] = self.get_image_urls_by_xpath(headers_sel,"//picture[contains(@class, 'story__pic') and contains(@class, 'block') and contains(@class, 'w-full') ar
        self.extra_keys[ "category" ] = self.get_text_by_xpath( headers_sel,"//a[contains(concat(' ', normalize-space(@class), ' '), 'sc__author--category')]")

        headers_sel = self.sel.xpath( "//div[contains(concat(' ', normalize-space(@class), ' '), ' sc__author ')]" )
        self.extra_keys[ "author" ] = self.extract_all_text( headers_sel,"//div[contains(concat(' ', normalize-space(@class), ' '), 'sc__author-nota')]" )
        self.extra_keys[ "author_date" ] = self.extract_all_text( headers_sel,"//span[contains(concat(' ', normalize-space(@class), ' '), 'sc__author--date')]" )

        self.extra_keys[ "paragraphs" ] = self.get_text_by_xpath(self.sel,"//div[contains(@class, 'sc') and contains(@class, 'pl-3')]//p[@itemprop='description']")
        self.extra_keys[ "html_text" ] = ""

        self.extra_keys["SourceDataField"] = self.extra_keys[ "h1" ] + " " + self.extra_keys["paragraphs"]
```

Figure 28: Code of wwweluniversalcommx scraping template