Last edited by **Lukas Yu** 1 year ago

Scraper Pipeline

Article collection

There are set default values for each field. By doing this, indexing can be used, improving the performance of queries.

If we used non-existing fields to signify the absence of an attribute, we would have to use the "$exists" keyword to distinguish between articles that have a certain attribute. However, the *$exists* operator cannot use any index.

Architecture

File structure

The file structure of the program is organized like this:

- There is one *main.py* file to start from, which imports all the relevant modules from the *scraperpackage* and executes all the processes while also managing the logging of events and the insertion into the database.
- main.py
- .env
- scraperpackage
  - mongomanager.py
  - logger.py
  - scrapers
    - blickscraper.py
    - nzzscraper.py
    - srfscraper.py
    - tascraper.py
    - wozscraper.py
    - wwscraper.py
  - pipeline
    - duplicatedetector.py
    - normalizer.py

MongoManager

This module simplifies to establish a connection the the MongoDB database. It retrieves the credentials as well as the SSH connection details from the .env file.

To use the MongoManager, first import it with

```
From mongomanager import MongoManager
```

Then use the *with* keyword to open the connection. MongoManager will return a *MongoClient* object. If the configuration uses SSH, it will automatically close the connection once outside the *with* block.

If it configured without SSH, the manager will simply create a MongoClient object with the given address.

```
with MongoManager() as db:
    articles = db.articles.find({}).fetch()
```

Logger

Logging module for all loggin related stuff The logs are not stored into the database directly, but are put into a list first. To store it into the database and send an email if the list contains an error, call the function save_logs().

You can also use the class using the *with* directive. Once it reaches outside the *with* block, it will call the save_logs() function automatically. If an error occurs within the block, an additional error log will be added before the function is called.

```
with Logger() as logger:
    logger.log('Message text', 'Component', is_error=True)
```

Scrapers

The scrapers are imported by the main.py program and are executed by calling the scrape() function, which all scrapermodules have to implement to work.

The majority of the scrapers will follow a similar pattern:

1. Open the RSS feeds use feedparser to retrieve the url of articles and possibly other metadata.
2. Open each url, parse the HTML response with BeautifulSoup4 and scrape the remaining metadata
3. Collect all data and create a uniform article object (see article collection section)

The notable exception that does not use this pattern is the Weltwoche scraper.

Weltwoche scraper

As Weltwoche does not offer any RSS feeds, we cannot use feedparser to retrieve a list of the latest articles. Instead, the list has to be scraped from an edition overview HTML page, which displays a list of articles in the edition of the current week. The content of this list is loaded dynamically, so we use Selenium with Geckodriver to open the browser.

Alternatively, one could directly use the link which returns the relevant content in HTML to bypass having to use Selenium, but this method has not explored thouroughly yet because the current way still works flawlessly. The link would in questin is link

Open "https://www.weltwoche.ch/inhaltsverzeichnis.html" and store every article link in a list

Open the url of each article and scrape:

- title
- url
- author
- publishDate
- image
- lead
- body

SRF scraper

Using RSS feed for:

- url
- title
- lead
- image

GET request html with url and scrape:

- body
- publishDate

Blick scraper

Using RSS feed for:

- url
- title
- author

GET request html with url and scrape:

- primary and subcategories
- body
- image
- publishDate
- modifyDate

Pipeline

Duplication detection

Sometimes it can happen that news outlets change a small portion of the article text and publish it again. Those duplicates are not desirable in our application, so we aim to detect and purge those duplicates.

This is done by splitting the text of an article into single sentences, and then compare those sentences to every article that has been published in the last few days. If a large part of two articles is identical, the new article won't be added into the database as a new article.