Documentation - WebScrapper

Application Overview

This web application provides an interface for users to scrape articles from a given website. Post scraping, users can filter, search, and download the articles.

API Endpoints

1. POST /scrape

Description: Scrapes articles from a provided URL.

Request Body:

```
const url = formData.get('url');
```

Response:

Returns a list of articles with the following structure:

```
"title": "The Joys of Gardening",
"short_description": "Explore the enriching world of gardening and discover its positive impact on mood and well-being.",
"author": "Alex Green",
"author_proffesion": "Gardening Enthusiast",
"linkkRL": "https://wsa-test.vercel.app/blog/the-joys-of-gardening",
"imgURL": "https://wsa-test.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Fgarden.8d6b6c5f.webp&w=3840&q=75",
"sentiment": "Positive"
 "wordCount": 381,
"textFromSecondPage": "The Joys of Gardening Discover the Blissful Moments in Gardening Gardening is indeed a joyful and rewarding hobby. It is not just an activity bu
"date": "September 15, 2023",
"title": The Challenges of Urban Living",
"short_description": "A candid look at the challenges of urban living, with insights into coping strategies.",
 'author_proffesion": "City Life Blogger",
aucho_broffesion . City Life Biogger ,
"linkURL": "https://wsa-test.vercel.app/log/the-challenges-of-urban-living",
"imgURL": "https://wsa-test.vercel.app/_next/image?url=%2F_next%2Fstatic%2Fmedia%2Furban.b1d13747.webp&w=3840&q=75",
"sentiment": "Negative",
"type_of_article": "Urban Life",
 "wordCount": 247,
"textFromSecondPage": "The Challenges of Urban Living Navigating the Urban Jungle Living in a bustling city comes with its own set of challenges. The continuous hustle
"date": "September 14, 2023",
"title": "The Radiant Days of Summer"
title: Ine Kasalant Days or Summer, "short_description": "Dive into the vibrant and joyful moments that the summer season brings to our lives.", "author": "Sunny Ray", |
"author proffesion": "Seasonal Blogger",
"linkURL": "https://wsa-test.vercel.app/blog/the-radiant-days-of-summer",
"imgURL": "https://wsa-test.vercel.app/_next/image?url-%2F_next%2Fstatic%2Fmedia%2Fsummer.8f9285fa.webp&w=3840&q=75",
"sentiment": "Positive",
"sentiment": "Positive",
"sentiment": "Positive",
"type_of_article": "Seasons",
 'textFromSecondPage": "The Radiant Days of Summer Embracing the Golden Rays Summer is a season full of vibrant and joyful moments. It is the time of year where everyth
```

2. GET /download

Description: Allows users to download the scraped articles in JSON format.

Response: A .txt file containing the articles in JSON format.

Frontend Interface

The main UI consists of:

- An input field for the URL to scrape.
- A button to start scraping.
- Information about the articles in the blog
- A search field to filter articles based on keywords.
- A dropdown filter to select articles based on sentiment.
- A download button to get articles in JSON format.

Explanation

Choices:

Puppeteer & Cheerio: I chose these libraries because they provide a combination of real browser navigation (via Puppeteer) and DOM working capabilities (via Cheerio). This allows to effectively scrape and parse web pages.

Tailwind CSS: I chose this utility-first CSS framework for rapid interface design and also to work with something that I haven't before in order to step out of my comfort zone. It's flexible and allows for easy customization without leaving the HTML.

Standout Features:

Sentiment Analysis: The main text of each article is scanned with my sentiment analysis algorithm, categorizing them into Positive, Neutral, or Negative sentiments. It's a simple algorithm in order to follow the projects requirements however with some touchups like a vaster word base it could be a fast way to get the general sentiment of an article without analyzing the context.

Article Word Count: Alongside the basic details, the application also provides word count statistics for each article giving us an idea of the length of the article.

Search bar: Implemented a search bar in order for a user-friendly and quick way for users to find desired information from an article faster.

Sentiment filter: Implemented a filter for the user to filter out the sentiments and search for specific sentiment in articles.

Download JSON: A button to quickly get all the scrapped data in a JSON format directly in a file without issues.

Learning Experience:

Complexity of Web Scraping: The scrapping part was a difficult process at the beginning. I had to think of a very general way of getting all the data. Firstly I tough of getting the articles by their position on the website but then I checked and saw that the website is responsive so that wouldn't work. Then I decided to get all the data needed from the <time> selector. Since on the website the time selector was used only once each time for each article I searched for that and then I got the other data that I needed in correlation with the <time> selector. For a bigger project I would think that first we could scan the page for its structure and then with AI(or a more general approach) get the data from each article we need. We could set some models for our data (for example we could now that the time of something is usually in a specific format or that the Title usually is bigger than the rest of the data on the page)

Performance Concerns: While Puppeteer gives me a real browser environment, it can be resource-intensive. However for this project was the right choice (I don't think I would use this in a bigger project like yours). Another thing was that sometimes puppeteer didn't get the page ok, and I found out that it didn't have enough time to get the page so I just added a timeout of half a second to make sure it has enough time. The same applies to the second time I had to access the link of each article to get the individual page of each article to get the entire content (added a timeout of 2 second there)

Sentiment analysis: The main issue was how would I design this algorithm. I didn't really find a better way than just comparing each word of the content of the article with some positive and negative words provided by myself and then make a score for this. Another issue that I had to fix later was that in the neutral observation on modern art Article, there are the same number of positive words as negative but the word: bad has a ";" right after it, and when my first version of my algorithm checked on this didn't find it well and returned that the article was positive. After this I modified the preprocess so that I get only letters in words and nothing else. And now it works fine!

DOM: Working with DOM and finding the structure of each article was an interesting process ©. Also another issue was that I wanted to get only the content of the individual page and when I accessed that

firstly I got as words the type of article and the 'Back to Articles' so I had to modify that so that it didn't count as words or in the sentiment analysis.

Tailwind CSS: Finding the right class for each div, selector, input, span and, p was a real challenge however there were some useful videos on Youtube for that. (Getting everything to be well placed in the page and the page to be responsive was also a big deal)

The other implementations weren't something that I had difficulties in.

Improvements

What I could see improved or added:

Additional Filters:

- Date Range: Allow users to filter articles based on a specific date range.
- Author: Allow filtering by the author's name.

Database Integration:

- Save Articles: Instead of saving articles to a file, I would consider using a database to store them, allowing more complex queries and persistent storage.
- User Accounts: With a database, I could also introduce user accounts where users can save favorite articles, set preferences, etc...

Performance and Scalability:

• Caching: I would implement caching to avoid scraping the same site repeatedly within a short period, reducing the load on both your server and the target website.