# Web Scraping Application Documentation

## Overview

This web scraping application extracts data from three specific web pages:

1. AJAX Data: `https://www.scrapethissite.com/pages/ajax javascript/#2015`
2. Forms Data: `https://www.scrapethissite.com/pages/forms/`
3. Advanced Data: `https://www.scrapethissite.com/pages/advanced/`

The extracted data is then stored in a MongoDB database.

## Prerequisites

Ensure you have the following installed:

1. Python 3.7 or later
2. MongoDB

## Setup Instructions

Step 1: Clone the Repository

Clone the repository containing the scraping application code to your local machine:

Step 2: Create a Virtual Environment

Create and activate a Python virtual environment to manage dependencies.

Step 3: Install Dependencies

Install the required Python packages using `pip`: pip

install  r requirements.txt

The `requirements.txt` file should include:

beautifulsoup4 pymongo

requests

If you don't have a `requirements.txt` file, create one with the following content:

plaintext

beautifulsoup4==4.9.3

pymongo==3.11.4 requests==2.25.1

Step 4: Configure MongoDB

Ensure MongoDB is running on your local machine. You can start MongoDB using the following command: mongod  dbpath /path/to/your/mongodb/data

Replace `/path/to/your/mongodb/data` with the actual path to your MongoDB data directory.

Step 5: Run the Scraper

Execute the scraping script

**How the Script Works**

1. Fetching Data :
    - The script sends HTTP GET requests to the specified URLs.
    - For AJAX data, the script handles JSON responses.
    - For Forms and Advanced data, the script uses BeautifulSoup to parse the HTML and extract the required information.

2. Storing Data :
    - The extracted data is stored in the MongoDB database named `scraped_data`.
    - Collections in the database:

    `ajax_data`

    `forms_data`

    `advanced_data`

3. Logging :

- Logs are saved in the `scraping_errors.log` file, detailing the fetching, parsing, and storing process, as well as any errors encountered.

**Verifying Data Storage**

Verify that the data has been stored correctly in MongoDB:

1. Start the MongoDB shell :

   Mongo

2. Switch to the `scraped_data` database :

   use scraped_data

3. Check the collections :

   show collections

4. View the data in each collection :

   db.ajax_data.find

**Troubleshooting**

**No Data in MongoDB :** • Ensure

MongoDB is running.

- Check the `scraping_errors.log` file for any errors during the data fetching or insertion process.
- Verify the correct structure of the HTML on the target pages and ensure the BeautifulSoup selectors match this structure.

**Connection Issues :**

- Ensure the MongoDB connection string in the script matches your local setup. The default is `"mongodb://localhost:27017/"`.

**Dependencies Issues :**

- Ensure all dependencies are installed in your virtual environment. Use `pip list` to check the installed packages.

**Logging :**

- The `scraping_errors.log` file contains detailed logs for debugging purposes. Ensure logging levels are set appropriately for your needs.

**Optional performance enhancement**

To optimize the web scraping process and minimize the load on the target website's servers, follow these best practices:

1. Respect the Website’s robots.txt

Always check the website’s robots.txt file to understand which pages are allowed or disallowed for scraping. Although robots.txt is not legally binding, respecting it is considered good practice.

2. Implement Rate Limiting

To avoid overwhelming the server with requests, implement rate limiting by introducing delays between requests.

3. Use Efficient Parsers

Use efficient HTML parsers like lxml instead of the default HTML parser in BeautifulSoup, which is faster and more efficient.

4. Use Caching

Cache the responses of pages that are unlikely to change frequently to avoid repeated requests to the same pages.

5. Parallelize Requests (with caution)

Make parallel requests using libraries like concurrent.futures, but be mindful of the server load and use rate limiting in combination.

**Parallel scraping-**

Parallel Scraping with Celery- Celery is an asynchronous task queue/job queue that can be used to execute tasks concurrently, improving the performance of our web scraping application.

By implementing parallel scraping with Celery and caching mechanisms with Requests-Cache, the performance of the web scraping application can be significantly improved, reducing the load on the target website's servers and speeding up the scraping process.

**Contact**

For any further questions or issues, please contact:

Name : Priyanshi Jain

Email : priyanshijainindore@gmail.com

GitHub : https://github.com/priyanshij20

***This documentation provides all necessary steps to set up, run, and verify the scraping application, ensuring smooth installation and operation.***