

FastAPI Web Scraping

DEMO:

 1728868198100764.MP4

Setting up the project:

Clone the github repository into your local PC and run the following commands:

Installation

Create a virtual environment:

```
python3 -m venv env  
source env/bin/activate
```

All the required modules have been included in the requirement.txt, use the following command to install.

```
pip install -r requirements.txt
```

Configuration

Create a .env file in project's root directory, using below command in terminal
touch .env

Enter the values of required fields to .env

```
STATIC_TOKEN=itssafehere
```

```
# Email Configuration
```

```
SMTP_SERVER=smtp.gmail.com
```

```
SMTP_PORT=587
```

```
SMTP_USERNAME=your_email@gmail.com
```

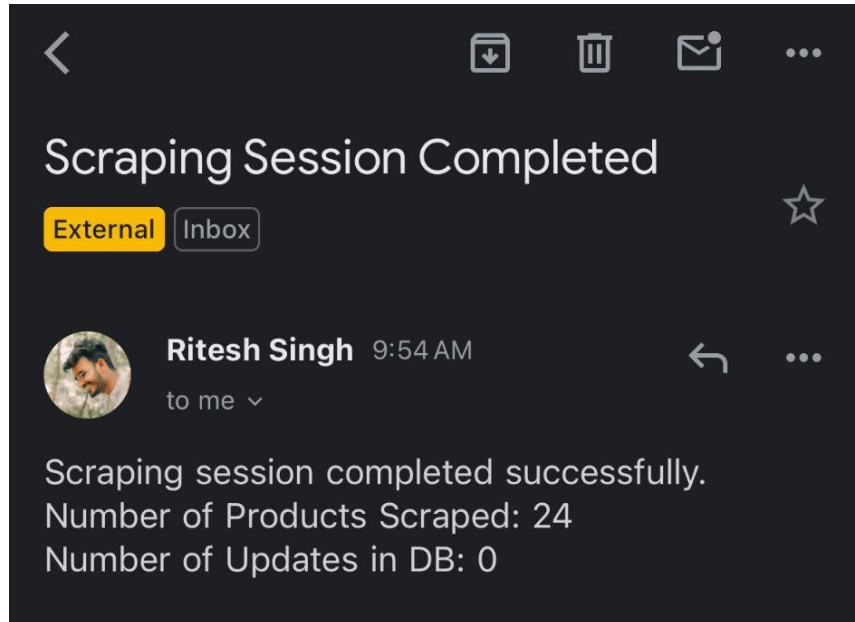
```
SMTP_PASSWORD=your_google_password
```

```
FROM_EMAIL=your_email@gmail.com
```

```
SMTP_USE_TLS=True
EMAIL_RECIPIENTS= rsing116@ucr.edu

# AWS Configuration
AWS_ACCESS_KEY_ID=your_new_access_key_id
AWS_SECRET_ACCESS_KEY=your_new_secret_access_key
REGION_NAME=eu-north-1
BUCKET_NAME=my-product-images-atlys
```

Static token is being used for simple authentication at the endpoint here.
Email configuration is used for notification. Configure the SMTP server with credentials.
We can also use slack, discord or teams using their APIs for extending the application.
The sample email while testing the application is below:



AWS access key id and AWS secret access key is used to authenticate AWS S3, where I have downloaded and stored the images of each product.

Running the application

Start the FastAPI server using **Uvicorn**:

```
uvicorn main:app --reload
```

Scraping:

To start the scraping process access the **/scrape/** endpoint with the required query parameters.

You can use your web browser or Postman to check this.

Using a web browser:

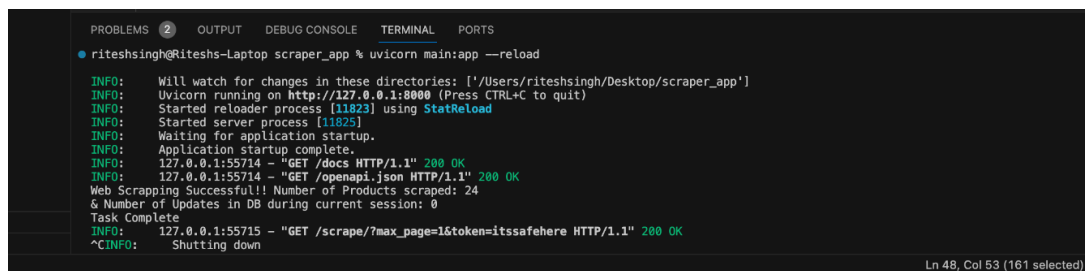
Use the following url in your browser and the value for desired query.

http://127.0.0.1:8000/scrape/?max_page=1&token=itssafehere

We can also add another query here, `proxy_string`, we can either use a proxy service provider or use some [free proxies](#) to test our project.

Notification:

Upon successful scraping, the console will display logs indicating the number of products scraped and the number of database updates. We will be notified via console and e-mail.

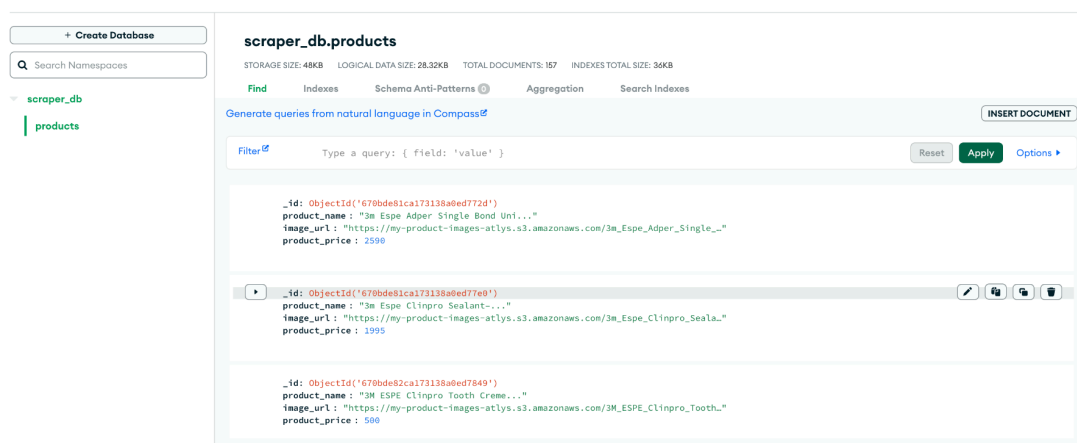


```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
riteshsingh@Riteshs-Laptop scraper_app % uvicorn main:app --reload

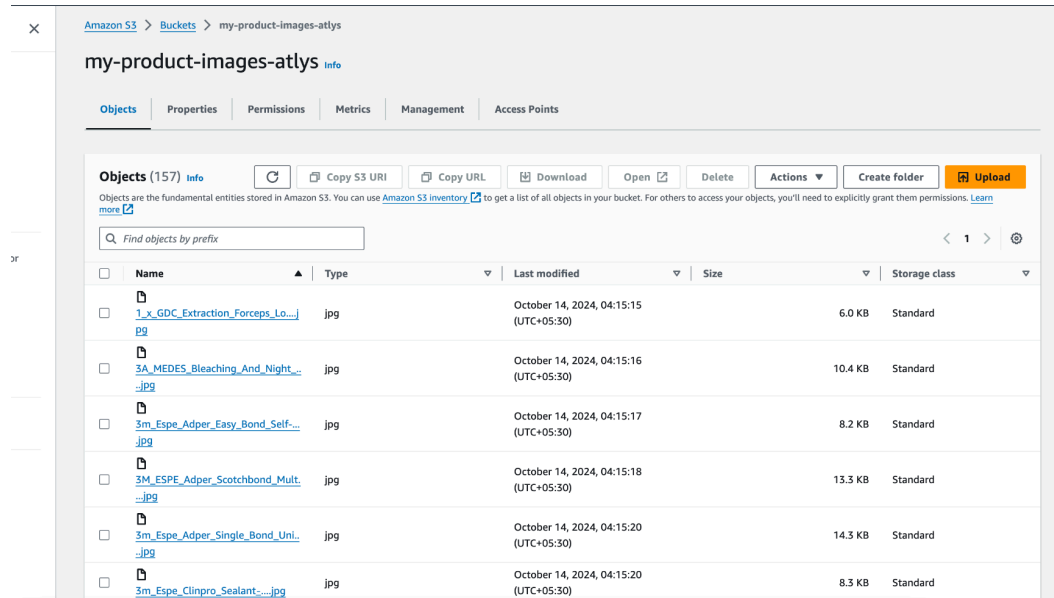
INFO: Will watch for changes in these directories: ['/Users/riteshsingh/Desktop/scraper_app']
INFO: Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO: Started reloader process [11823] using StatReload
INFO: Started server process [11825]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 127.0.0.1:55714 - "GET /docs HTTP/1.1" 200 OK
INFO: 127.0.0.1:55714 - "GET /openapi-json HTTP/1.1" 200 OK
Web Scrapping Successful!! Number of Products scraped: 24
& Number of Updates in DB during current session: 0
Task Complete
INFO: 127.0.0.1:55715 - "GET /scrape/?max_page=1&token=itssafehere HTTP/1.1" 200 OK
^CINFO: Shutting down
```

Storage:

The scraped information will be stored in the MongoDB Atlas cloud inside the products database.



For each product in the website, we are storing the Title for the product, image link(downloaded and stored in AWS S3), and price. Type validation is kept in mind while scraping and handling the product information.



In-Memory Cache:

In this project, I have implemented in-memory cache using redis. It ensures that if a product's information is already stored in the cache or database memory then it doesn't update the product and skips it. This is the reason why the notification shows the number of updates as Zero, because the product information was already stored in the cache. Whereas products checked is 24(number of products in one page.)

Future Goals:

There are other approaches that can be used to make this application more efficient and accessible.

Currently I am using free service or proxies, thus providing a bit slower connection. Upgrading to a better service provider will be more reliable and faster.

We can also use docker containerization for environment consistency, scalability, and ease of deployment.