## Project Overview

This Django project is web scraping, data processing, data visualization and a MySQL-backed Django application.

Project link: https://github.com/shown440/scraping\_project

## Prerequisites

Before you begin, ensure you have the following installed:

* Python 3.12+ (recommended)
* Git
* MySQL server (or another supported database)
* Virtual environment tool (venv, virtualenv)
* System dependencies for mysqlclient (on Ubuntu/Debian):
* sudo apt-get update  
  sudo apt-get install default-libmysqlclient-dev build-essential
* sudo apt install python3.12 python3.12-venv python3.12-dev python3-pip -y

## Installation Steps

1. **Clone the repository**

* git clone git@github.com:shown440/scraping\_project.git  
  cd scraping\_project

1. **Create a virtual environment and activate it**

* python3.12 -m venv sebpo\_scrap\_env\_312  
  source sebpo\_scrap\_env\_312/bin/activate

1. **Install Python dependencies**

* pip install --upgrade pip  
  pip install -r requirements.txt

## Configuration

1. **Environment variables**

Already inserted inside the code, because It’s anon production project.

## Database Setup and Migrations

1. **Create the database** in MySQL:

* 1. Login inside mysql database
* 2. CREATE DATABASE sebpo\_scrap\_db;
* 3. GRANT ALL PRIVILEGES ON sebpo\_scrap\_db.\* TO 'ffwc'@'localhost';
* 4. FLUSH PRIVILEGES;
* 5. Exit
* 6. Restore dumped mysql db: mysql -u db\_username -p sebpo\_scrap\_db < sebpo\_scrap\_db.sql

1. **Apply migrations**:

* 1. python3.12 manage.py makemigrations
* 1. python3.12 manage.py migrate

1. **Create a superuser** (for admin access):

* python3.12 manage.py createsuperuser

## Running the Development Server

Start Django’s built-in development server:

Python3.12 manage.py runserver then it will run is: 0.0.0.0:8006

Visit http://127.0.0.1:8006/ in your browser.

## Running Scheduled and Custom Commands

* **Daily scraping task**:
* python3.12 manage.py fetch\_cia\_data
* **Daily scraping task** via cron:
* 1. goto project folder
* 2. go to: /accessories/bash\_script\_and\_crontab/bash\_script/fetch\_cia\_data.sh
* 3. find: “/var/www/prod/scraping\_project” then replace with your project path
* 4. find: “/var/www/Project\_Environments/python\_environments/sebpo\_scrap\_env\_312
* ” then replace with your python environment path
* 5. then provide permission update cia\_data.sh: sudo chmod 777 /var/www/prod/scraping\_project/accessories/bash\_script/fetch\_cia\_data.sh
* 6. now write in terminal: sudo crontab -e
  + Then: 20 15 \* \* \* /var/www/prod/scraping\_project/accessories/bash\_script/fetch\_cia\_data.sh
  + Then: save and exit

## Static Files (Production)

1. Collect static assets:

* python3.12 manage.py collectstatic

1. Configure your web server (e.g., Nginx) to serve the static/ directory.

## Deployment

For deploying to production, consider using:

* **Gunicorn + Nginx**

**Gunicorn service:** Goto the path: ../scraping\_project/accountable\_services/ubuntu\_server/scrap\_webservice\_port.service

1. find: “/var/www/Project\_Environments/python\_environments/sebpo\_scrap\_env\_312” then replace with your python environment path
2. Create Command: sudo nano scrap\_webservice\_port.service

And paste updated command inside here from: ../scraping\_project/accountable\_services/ubuntu\_server/scrap\_webservice\_port.service

1. Enable Command: sudo systemctl enable scrap\_webservice\_port.service
2. Start Command: sudo systemctl start scrap\_webservice\_port.service
3. Restart Command: sudo systemctl scrap\_webservice\_port.service
4. Monitoring Command: sudo systemctl status scrap\_webservice\_port.service

**Nginx configuration process:** Goto the path: ../scraping\_project/accountable\_services/ubuntu\_server/scrap\_ng\_port

1. find: “/var/www/prod/scraping\_project” then replace with your python project path
2. find: “scrap.sebpo.com” then replace with your own domain
3. then copy the whole command and goto: sudo nano /etc/nginx/sites-available/scrap\_ng\_port and paste and save it
4. Link the Configuration File: sudo ln -s /etc/nginx/sites-available/ scrap\_ng\_port /etc/nginx/sites-enabled/
5. sudo nginx -t
6. sudo systemctl restart nginx
7. If need to https the we can use certbot to do that:
   1. Then install: pip install certbot certbot-nginx
   2. Renew https domain: sudo certbot –nginx
   3. sudo systemctl restart nginx

## Explanations of features:

1. **How the scraper works?:**

The scraper fetches CIA agreement data from the OIG website in three key steps:

1. **Data Extraction**
   * Iterates through A-Z pages to scrape provider records
   * Parses HTML tables using BeautifulSoup
   * Captures provider names, locations, effective dates, and document links
2. **Change Detection**
   * Generates SHA-256 hashes for each record
   * Compares hashes with previous database pull
   * Identifies added/removed records since last run
3. **Database Operations**
   * Creates new DataPull entry for each execution
   * Stores raw data in CIAData table
   * Logs changes in DataChange table (additions/removals)
   * Marks pull as processed after completion

The process tracks data evolution by comparing cryptographic hashes between runs, efficiently detecting changes without storing full historical datasets.

1. **How the refresh button works?:**
   1. User Click:

* Clicking the "Refresh Data" button triggers a JavaScript event.
  1. Background Process Start:
* The button hides, and a "Processing..." spinner appears.
* A request is sent to Django's start\_processing URL.
* Django launches a background thread to scrape/process CIA data without blocking the user.
  1. Data Processing:
     + Scrapes the latest CIA data from the OIG website.
     + Compares new data with the previous pull to detect additions/removals.
     + Logs changes in the database (DataChange model).
  2. Auto-Refresh on Completion:
     + JavaScript checks the dashboard every 3 seconds.
     + When the "Latest Pull" timestamp updates, the page reloads automatically.
     + The UI displays new changes (added/removed records).
  3. Key Flow:

Click → Hide Button → Start Thread → Scrape → Compare → Save Changes → Detect Update → Reload Page

1. **How differences are displayed?:**

The differences are displayed in a **categorized, side-by-side format** based on change type:

1. **Added Items**:  
   Added: [New Data]  
   *(Shows only the new state)*
2. **Removed Items**:  
   Removed: [Old Data]  
   *(Shows only the previous state)*
3. **Modified Items**:  
   Changed from: [Old Data] to [New Data]  
   *(Directly compares previous → current values)*

Key features:

* Color-coded change types (visual distinction)
* Raw HTML rendering (|safe filter preserves formatting)
* Tabular layout with provider context
* Summary statistics at top (added/removed/modified counts)

1. **How latest data are displayed?:**

Here's a concise explanation of how the latest data is displayed:

1. **Latest Data Pull**: The most recent DataPull entry is fetched from the database.
2. **Time Conversion**: The pull time is converted to GMT+6 and displayed in YYYY-MM-DD HH:MM format.
3. **Paginated Records**: Associated records from CIAData are displayed in a table with 25 entries per page.
4. **Table Structure**: Each row shows:
   * Provider
   * City
   * State
   * Effective Date
5. **Pagination Controls**: Users can navigate between pages using first/previous/next/last links.
6. **Fallback Handling**: Shows "No records found" if data is empty or "No data pulls available" if no pull exists.

Output is always timezone-adjusted and paginated for clarity.

1. **How authentication is enforced?:**

Authentication is enforced by applying the @login\_required**decorator** to every view function. This:

* 1. **Restricts access** to authenticated users only.
  2. **Redirects unauthenticated users** to Django's default login page (handled internally).
  3. **Uses Django's built-in session-based authentication** system (no custom logic needed).
  4. **Result:** All routes (e.g., /dashboard, /history) are automatically protected. Unauthenticated requests get redirected to login.

**(Implementation: Decorators wrap each view, checking request.user.is\_authenticated behind the scenes.)**

1. **How to set up scheduled scraping?:**
   1. Customize Paths

Edit fetch\_cia\_data.sh:

* + - Replace /var/www/prod/scraping\_project with your project path.
    - Replace /var/www/Project\_Environments/... with your Python environment path.
  1. Grant Script Permissions

Run:

* + - sudo chmod 777 /path/to/fetch\_cia\_data.sh
  1. Schedule in Cron

Execute:

* + - sudo crontab -e

Add this line (runs daily at 15:20 or 3:20 pm):

* + - 1. 20 15 \* \* \* /path/to/fetch\_cia\_data.sh

Save and exit.

* 1. Schedule Syntax: 20 15 \* \* \*
     + 20 → Minute 20 (of the hour)
     + 15 → Hour 15 (3 PM in 24-hour time)
     + \* → Every day of the month
     + \* → Every month
     + \* → Every day of the week (Monday-Sunday)

## Contributing

1. Fork the repository
2. Create a feature branch (git checkout -b feature/XYZ)
3. Commit your changes (git commit -m "Add XYZ feature")
4. Push to the branch (git push origin feature/XYZ)
5. Open a Pull Request