PyCK Project

News Aggregator App

-by Saksham Katiyar

Introduction

In today's world, where time is everything, reading news from different websites can be a tedious task. Also, all the news articles are not always interesting to you. This project is an attempt to simplify the task of getting all the latest news that you are interested in.

This web application collects news from multiple online sources(predefined), filters them according to a list of keywords provided by the user, and then displays the news on a web page(hosted on your server). Each headline/image shown on the web page is a link to the complete article of that headline.

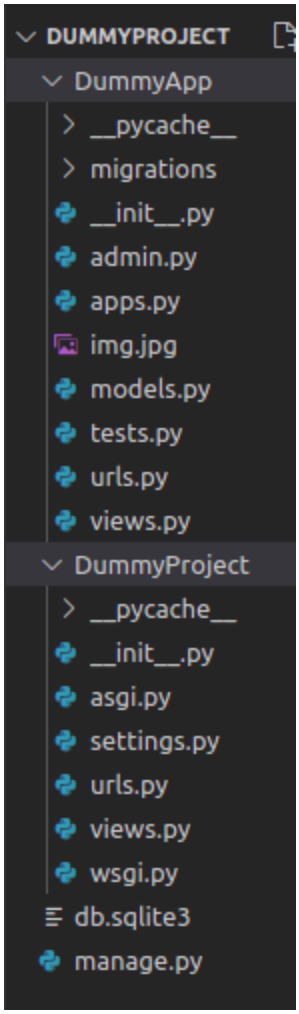
Implementation

The project is implemented using the Django framework. Web scraping is implemented using libraries like “requests” and “BeautifulSoup”. And then the collected data is displayed on a well-structured template made using HTML, CSS, JS, and Bootstrap.

The project can be divided into several phases as follows:

1. Learning Django basics and making a basic dummy application and model to learn about the general concepts
2. Making a project-oriented app i.e. implementing web scraping in the app.
3. Making a final web template using HTML/CSS to showcase the news scraped from different websites.

A virtual environment for python3 is used to keep our project dependencies organised. The environment folder is named “env” and is present inside the main project repository.



Django Basics

The Geeksforgeeks tutorial series for learning Django basics is really good if one is searching for a starting point.

Django simplifies the process of creating web applications. We can just use “django-admin” commands from the terminal to create a new project or an app inside an existing project. All the necessary and some extra files are automatically created inside the project/app folder. We just need to put our codes inside appropriate files and the app will be ready to run.

The main project contains:

- Project folder (“DummyProject”)
    - settings.py - contains all configuration data for our main project. This includes a list of all the installed apps and we need to add our app’s name in the list so that it works properly.

- urls.py - contains the list of url paths. It associates the URLs with their corresponding caller functions. In case we want to include the urls defined in another app (urls.py of the app) then we have to specify it using include as shown.

```
urlpatterns = [
    path('admin/', admin.site.urls),
    path('hello/', hello_world),
    path('', include("DummyApp.urls"))
]
```

- views.py - here we define all the caller functions associated with the URLs in urls.py. The functions usually take a “request” as parameter and return an HTTP response which is then rendered by the browser.

```
from django.http import HttpResponse

def something (request) :
    return HttpResponse("This is something")
```

- Other files that are either used in deployment of the web app or background low-level files. As far as this project is concerned, these files don’t matter as long as we don’t disturb them.

- App folder (“DummyApp”)
  - urls.py and views.py work exactly the same way as in project-folder.
  - models.py - contains all the models (data structures) that will be used to store the data. All the models inherit from django.db.models.Model

```
from django.db import models

class DummyModel(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    last_modified = models.DateTimeField(auto_now_add=True)
    img = models.ImageField(upload_to = "images/")

    def __str__(self):
        return self.title
```

  - admin.py - used to register the app on admin interface.

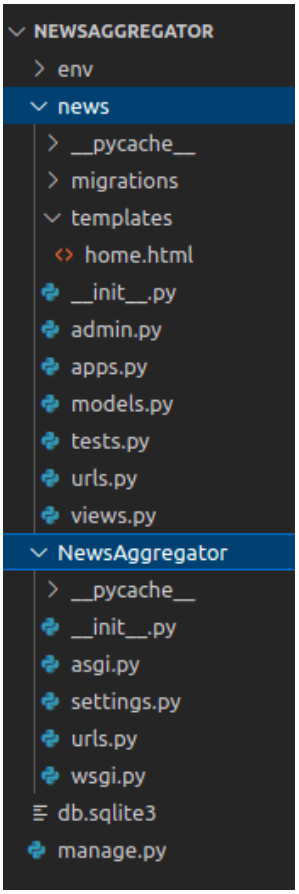
```
from django.contrib import admin
from .models import DummyModel

admin.site.register(DummyModel)
```

    - apps.py - contains configuration details of the app. However, we don't need to edit anything inside this file as per our purpose is concerned.
- manage.py
    - This python program lets us execute several commands from the terminal. The commands are sent as arguments to the program and they are executed as commands in the terminal. For example, "runserver", "shell", "startapp","createsuperuser", etc. The list of all commands can be found using "python3 manage.py help" command.

Whenever a create/update our models.py file in the app, then we need to execute 2 commands, `makemigrations` and `migrate`. These commands generate and execute the SQL commands in the database. We can notice the "migrations" folder being updated inside the app folder after these 2 commands. (This step is not mentioned in the data-flair blog which I was following to make the web scraper, and therefore caused a lot of errors)

Instead of sending direct HTTP response from functions in views.py, we can call a render function which will render the HTML file passed as argument. The path of file is relative to "templates" folder inside the app.



Web Scraping

To make the project oriented towards the goal, a "news" app is made inside the "NewsAggregator" project.

"models.py" inside the app contains a model named "Headline" which will store the details of each headline in our news aggregator, namely title, URL, and image source of the complete article. As mentioned before, we need to execute `makemigrations` and `migrate` command after adding the model.

"View.py" inside the app contains two functions, `scrape` and `news_list`

Scrape function is called whenever we press the "Get news" button on the web page. It uses `BeautifulSoup` and `requests` libraries for web scraping. `requests` module allows us to access websites by starting a

session. For this project, I am using a Google bot which mimics an actual user accessing the website.

The website used for this project is "www.theonion.com". The same website was being used in the data-flair blog so I didn't change it. However, the blog was written a long time ago, and the website has changed a lot since then. Because of this, the code given there doesn't work today. But using the inspect element, we can check the website HTML and find the correct details to extract the correct elements. HTML of such pages are quite large and I spent a lot of time figuring out the classes of the elements I wanted to extract. In future, I intend to add the code to scrape more news websites (this time some famous ones); the code will be very similar, only the classes' and elements' names will change.

`scrape` function adds Headline objects to the existing data. However, I have also added a command to delete all Headline objects before adding new ones, otherwise the number of news articles will just keep increasing as we run it again and again.

In the same function, we can use simple string functions to check whether the `title` contains any word from a given list of words and save the Headline object only if it does.

`news_list` function renders the HTML page and passes the Headline objects to be shown in the web page.

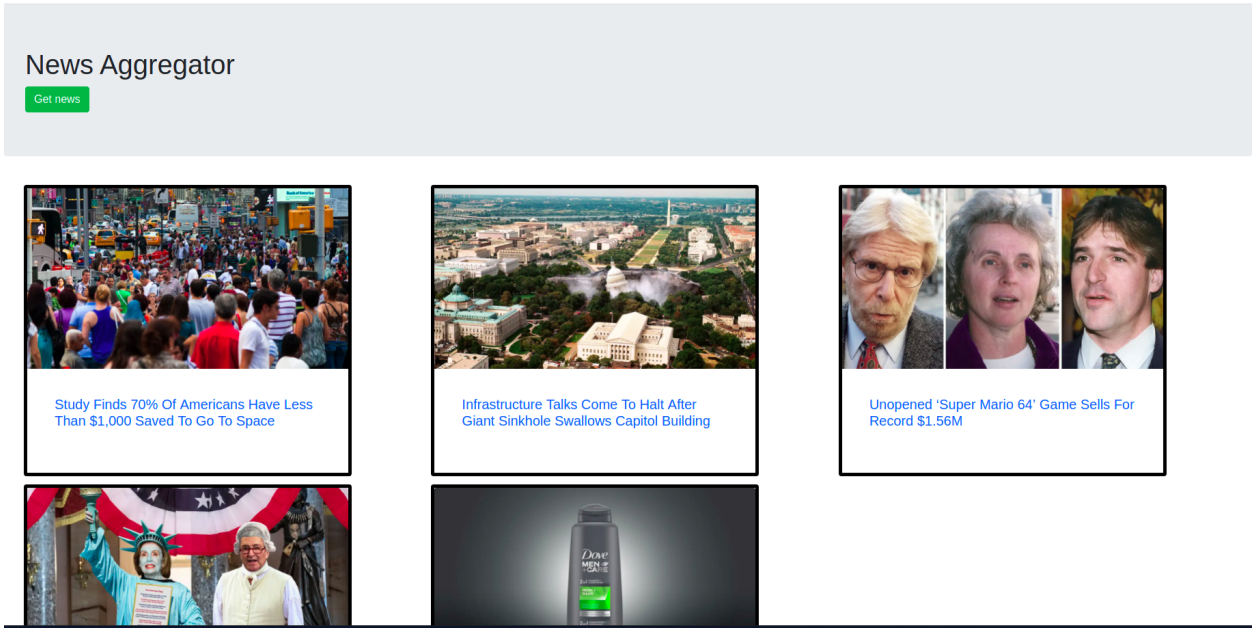
Web Template (HTML/CSS)

For this part of the project, I have copied the HTML code from the data-flair blog to check that my project is working.

The HTML needs to have a form submission button that calls the scrape function in "views.py". And we also need a part of HTML which will be rendered multiple times using a for loop (a 'div' element in our HTML code)

Besides above two mentioned parts, the HTML may use all kinds of CSS/Bootstrap for designing of the web page.

Screenshots



Overall 46 articles are scraped out of which 5 remain after filtering.

This is the result on pressing “Get news” button when the keywords list is:

```
keywords = ['student', 'study', 'democracy', 'infrastructure', 'game']
```

Conclusion

This has been a great learning experience for me. I tried a whole lot of new things with this project which includes web applications, Django, web scraping, and overall integration of Python programs with the web.

I have achieved all learning objectives that I set at the beginning of the project. However, I haven’t yet made this project into a form that I can use in my everyday life that I wished for. For that, I need to do 2 major changes, first is to make it look good using better CSS/HTML templates and second is to scrape more and better news websites. Both of these tasks are only time consuming and won’t take any more blog reading and stackoverflow searches.

Any queries/doubts regarding this project shall be conveyed to me and I will try my best to resolve it.

References

Django Tutorials:

https://www.geeksforgeeks.org/django-tutorial/

https://www.geeksforgeeks.org/django-basics/

https://www.geeksforgeeks.org/django-introduction-and-installation/

https://www.geeksforgeeks.org/django-project-mvt-structure/

https://www.geeksforgeeks.org/how-to-create-an-app-in-django/

https://www.geeksforgeeks.org/django-models/

https://www.geeksforgeeks.org/django-orm-inserting-updating-deleting-data/

https://www.geeksforgeeks.org/render-model-in-django-admin-interface/

Data-Flair blog (for web scraping):

https://data-flair.training/blogs/django-project-news-aggregator-app/

www.stackoverflow.com (obviously)