Project Name: News Aggregator Web App

What is a News Aggregator?

It is a web application which aggregates data (news articles) from multiple websites. Then presents the data in one location.

News aggregator service is a very important start of the day.

There are various publications and news sites online. They publish their content on multiple platforms. Now, imagine when you open 10-20 news sites every day. The time you waste to gain information. Information gain is everything in today's world.

A news aggregator makes this task easier. In a news aggregator, you can select the websites you want to follow. Then the news aggregator collects the articles for you. And, you are just a click away to get information from various websites.

Quick Notes

Our news aggregator will work in 3 steps:

1. It scrapes the web for the articles. (In this Django project, we are scraping a website called theonion)
2. Then it stores the article's images, links, and title.
3. The stored objects in the database are served to the client. The client gets information in a nice template.

So, that's how our web app will work.

Steps:

Before starting, we will need to install some of the libraries.

We will install the requests and BeautifulSoup libraries.

You can install them using pip.

Pip install bs4

Pip install requests

Now, we will make a new Python Django project named DF_NewsAggregator.

`django-admin startproject DF_NewsAggregator`

Then we will make new application news.

`python manage.py startapp news`

We will be storing the urls and articles in our database. For that, we will need the model.

In news/models.py, create these models.

```
from django.db import models
class Headline(models.Model):
  title = models.CharField(max_length=200)
  image = models.URLField(null=True, blank=True)
  url = models.TextField()

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

Our models will be able to store three things:

1. Title of the article
2. URL of the origin or source
3. URL of the article image

We are using simple model fields for that purpose. Also, the image field can be blank.
The __str__() method will return the string representation of the object.
Now, let's start with the steps for web crawlers.

Step 1 – Scrape the website

We will be scraping the website for getting articles. Web-Scraping means extracting data from the websites. We extract meaningful data from the websites. In this case, we will be extracting the articles from the "theonion" website.
To scrape the website, we will use beautifulsoup and requests module**.** These libraries are the bs4 and requests and modules are used for web crawling.
Open news/views.py file.

```
import requests
from django.shortcuts import render, redirect
from bs4 import BeautifulSoup as BSoup
from news.models import Headline
```

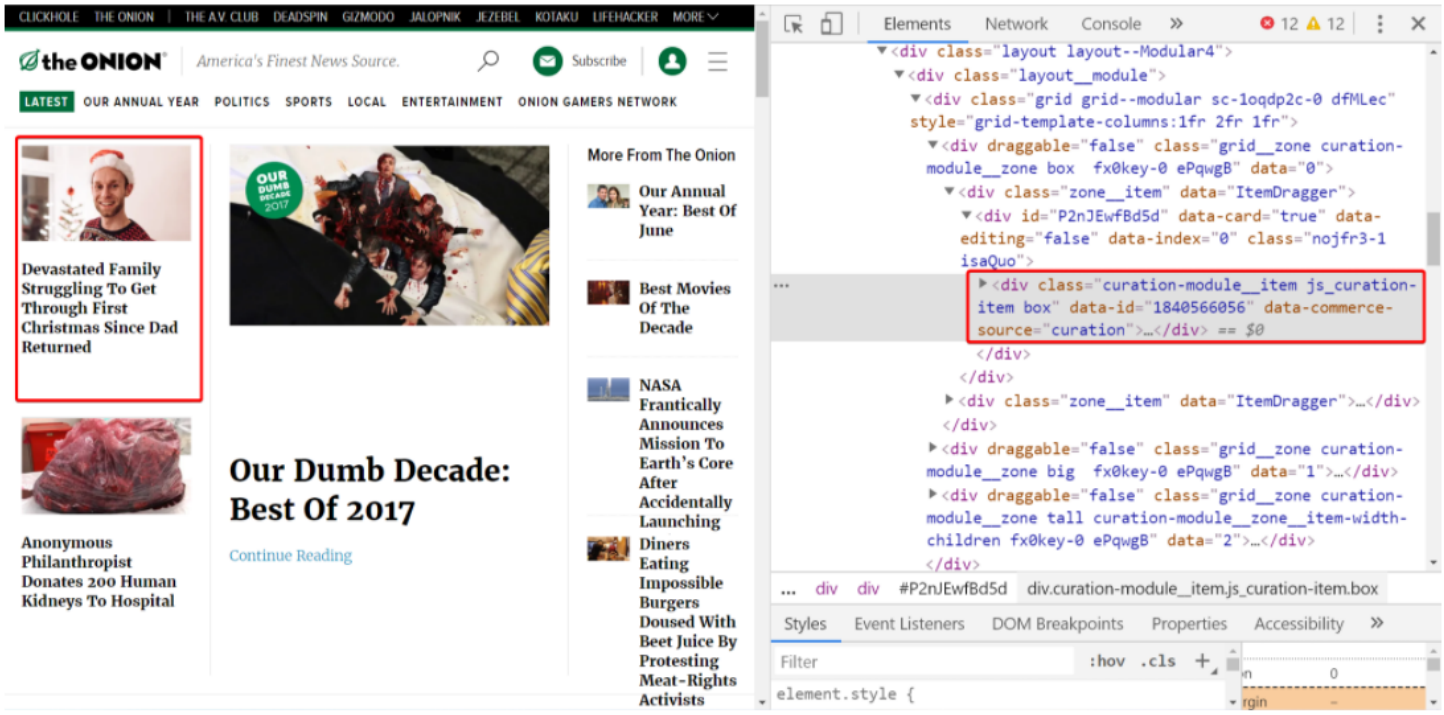
We will be making the first view function as **"**scrape()**".**

```
def scrape(request):
  session = requests.Session()
  session.headers = {"User-Agent": "Googlebot/2.1 (+http://www.google.com/bot.html)"}
  url = "https://www.theonion.com/"

  content = session.get(url, verify=False).content
  soup = BSoup(content, "html.parser")
  News = soup.find_all('div', {"class":"curation-module__item"})
  for artcile in News:
    main = artcile.find_all('a')[0]
    link = main['href']
    image_src = str(main.find('img')['srcset']).split(" ")[-4]
    title = main['title']
    new_headline = Headline()
    new_headline.title = title
    new_headline.url = link
    new_headline.image = image_src
    new_headline.save()
  return redirect("../")
```

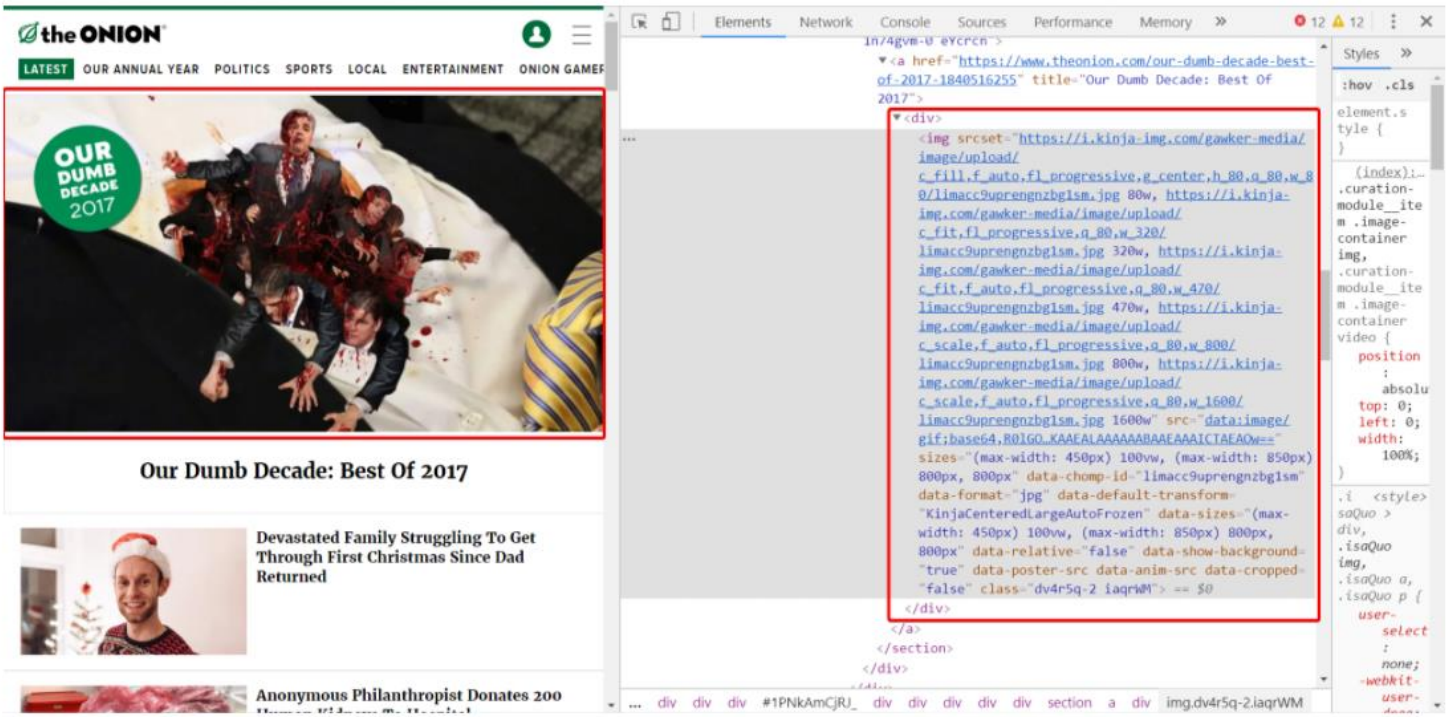
This view function uses modules like requests, bs4 and Django's shortcuts.
We have imported the model Headline from news.models. Also, we have other libraries.
The first line of the function is a setting for requests framework. These settings are necessary. They will prevent the errors to stop the execution of the program.
Then we write our view function scrape().
The scrape() method will scrape the news articles from the URL "theonion.com".
The first variable is the session object of the requests module. These are essential to make a connection to the server. This is the abstraction provided by requests framework.

The session variables have headers as HTTP headers. These headers are used by our function to request the webpage. The scrapper acts like a normal http client to the news site. The User-Agent key is important here.
This HTTP header will tell the server information about the client. We are using Google bot for that purpose. When our client requests anything on the server, the server sees our request coming as a Google bot. You can configure it look like a browser User-Agent.
That won't affect our use-case though.
After that, we introduce the content variable. We store the webpage or response given by the server in content. Now, the beautifulsoup comes in.
The beautiful soup is a library that can extract data from HTML web pages. We create a soup object where we pass the HTML page. Alongside the HTML page, we also pass HTML parser as a parameter.
The HTML parser will parse the HTML as a BeautifulSoup object. In this object, we can access HTML elements and their texts.
In the News object, we return the of a particular class. We selected this class from the webpage inspection. We inspected the webpage of the website 'theonion'. Now, we select the elements which have the information we need.



As you can see from this image, by inspecting the element, we find a common class. The rest is just extracting information from that element.
Now we get 3 elements of this class. That means that the three articles are present in this class. These articles have a very general structure.
Now, we will extract the information which we need.
In this case, we have to extract the title, link, and image link.
Using a for loop, we can iterate over soup objects. In the for loop, the main variable will hold the link to the origin webpage. The main attribute gets the anchor tag. Since, the s returned only have one <a>tag, we get most of our work done here.
The <a> tag contains title and href of the original link.

We can access the href in <a> tag by writing main['href'].
Similarly, we can extract the title by main['title']. Remember the main is the <a> tag beautifulsoup object.
Then we find the image URL. To get the image_src, we find the image in the main. This is all according to the webpage layout. We are not doing this because of syntax.
These are how the website has made its webpage. We are simply finding the elements and accessing them appropriately. You need to have some basics clear of beautiful soup and HTML.
So, once we get the image, we extract the srcset attribute from the same.



The srcset attribute contains various sizes of images, as we can see in the image. There we have to extract the size of the image which is big enough for us. We select the one with 800 width.
We get a string that has the source of the image and its width. And, we can travel over that list using Python indexing. As you can see in the code, we use the split() on the string to get a list. There we use index [-4]. That will give us the URL of 80 width image. That is stored as string in the image_src variable.

Step 2 – Store the data in the database

We have made our model Headline for this purpose. Now we will be performing the standard storing procedure. We create a new Headline() object. There we fill the corresponding fields. This the standard code for storing in the database.

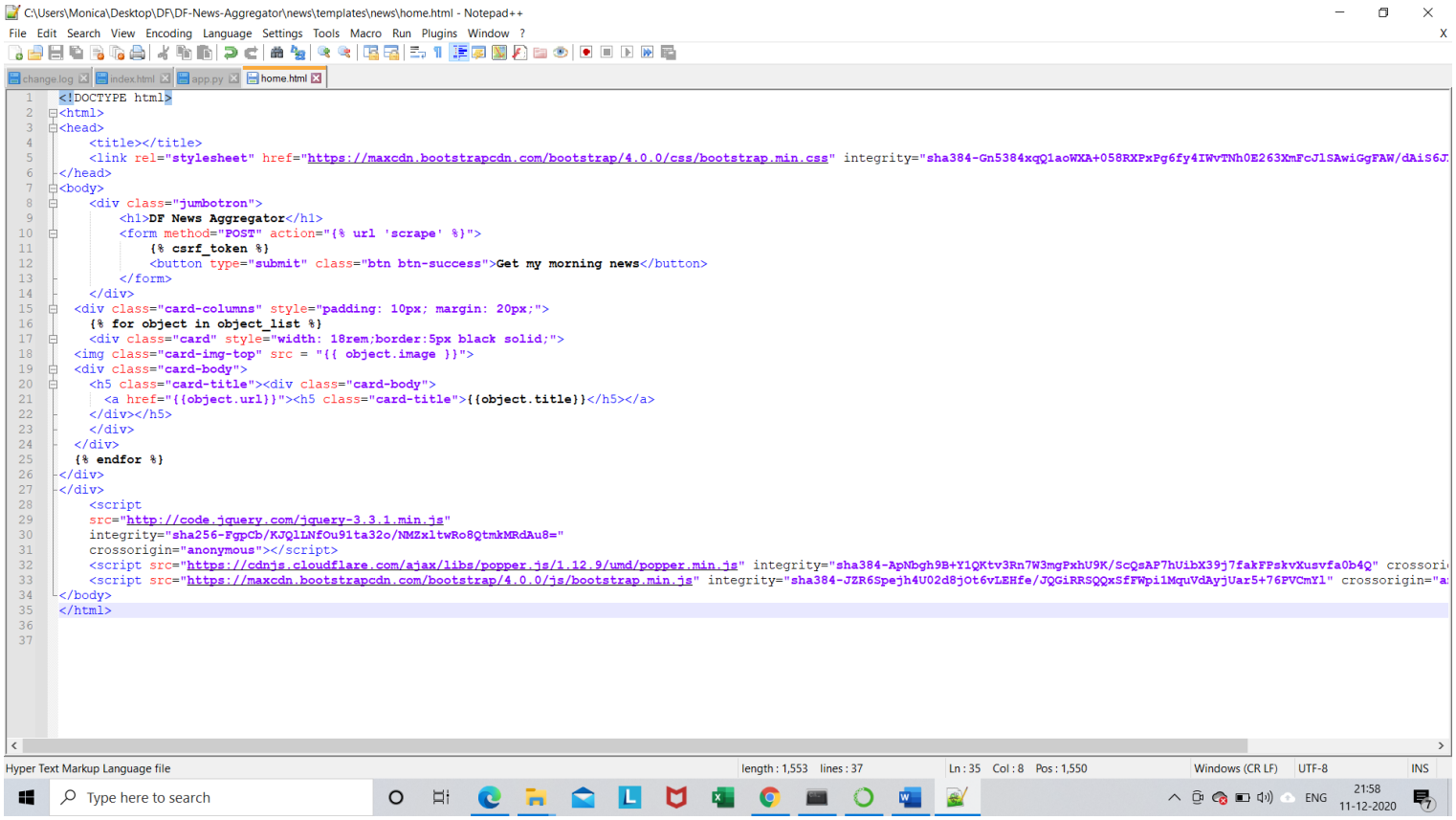
```
new_headline = Headline()
new_headline.title = title
new_headline.url = link
new_headline.image = image_src
new_headline.save()
```

Step 3 – Serve the stored database objects

This step is very easy too. We create a new view function for this purpose. That is news_list() method. The code lies in the file news/views.py file.

```
def news_list(request):
    headlines = Headline.objects.all()[::-1]
    context = {
        'object_list': headlines,
    }
    return render(request, "news/home.html", context)
```

We simply extract all the elements from the database. Since we want the latest info on top, we reverse the list. Then we simply pass the list in a context. The context is then given to home.html in folder news/template/news. In home.html we are using bootstrap and html.



Configuring urls.py

Last, we configure our urls.py file. Make a file news/urls.py. Paste this code inside the urls.py

```
from django.urls import path
from news.views import scrape, news_list
urlpatterns = [
    path('scrape/', scrape, name="scrape"),
    path('', news_list, name="home"),
]
```

Then we also need to connect this to main urls.py. Open DF_NewsAggregator/urls.py file and paste this code inside that or update it.

```
from django.contrib import admin
from django.urls import path, include

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

`To get output write, python manage.py runserver`