Making a website through django

1. Set up –

Create virtual environment through – conda create (--)name FullStackApp python=3

This setup the virtual environment and installs python3

Then we activate it by source activate FullStackApp

Ignore it we are making a Django project

So step 1 is install django so pip install django

django-admin startproject FullStackApp

Then we start the app

Django-admin startapp my\_app

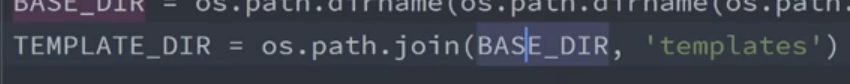
1. Creating website-

Create templates directory and in that base.html to have some code to render

We need to create a view and map a url to make this work. But for now we have to register our app we do this everytime we make a new app

To register we go to FullStackApp/settings and under INSTALLED\_APPS just add ‘my\_app’

For Django to know where templates is stored we need to add the below in settings



Then we add TEMPLATE\_DIR to

A screenshot of a cell phone

Description automatically generated

Then we add

A screenshot of a cell phone

Description automatically generated

These static files are used for creating graphics for the web app

Create a home view –

Under my\_app/views

A screenshot of a cell phone

Description automatically generated

Now we map urls to this

Under my\_app we make a python file called urls

A screenshot of a cell phone

Description automatically generated

Now we build a model and setup the fields

A screenshot of a cell phone

Description automatically generated

Now we want to migrate these changes to the python database

Python manage.py makemigrations

Python manage.py migrate

Then we check the out the page by pyhton manage.py runserver

Before that lets integrate search model to the admin page

Go to my\_app/admin and then

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

to remove the searchs from above we go to my\_app/models and add this class Meta

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

After searching for something we get this to replace the search object for the thing we searched we just add this to my\_app/models

A screenshot of a cell phone

Description automatically generated

So the home page isn’t working and the fix was under FullStackApp/urls

A screenshot of a cell phone

Description automatically generated

We import include and from my\_app import views and added this path with include to the urlpatterns

And now it works

A screenshot of a cell phone

Description automatically generated

So when we enter the home address it goes to my\_app/urls sees that it matches ‘ ‘ then it goes to run home function from views ( views.home ) and the home function renders the base.html file which gives us the above image

You want images to be contained in a flexbox cause its easier to manage when you lower the size of the browser the images change their shape

Flex box needs rows, and coloums represent the size of the images (s4 or s12)

A screenshot of a cell phone

Description automatically generated

We can use flash icon from materializecss.com

We want to use this base html everywhere as a template so we use block content to make it do it

A close up of a logo

Description automatically generated

So when we make the search page all the code in base.html is inherited and we can new code for the search page

**A screenshot of a cell phone screen with text

Description automatically generated**

Adding search page

We then make a new\_search.html under template/my\_app

Then under my\_app/views we will define a new\_search function and make it render to the new\_seatch.html file

Now we have to create a url for it so we go to my\_app/urls and add it to url patterns

Adding a button

A screenshot of a cell phone screen with text

Description automatically generated

Highlighted adds a button

A screenshot of a social media post

Description automatically generated

When we search we get this error and all it needs is a CSRF token (in Django security feature )

We change the position of the </div> so that the end block is included in the container

Leading to the following

A screenshot of a cell phone

Description automatically generated

Changed the submit button by looking at the documentation and placing this from materializecss.com

A screenshot of a cell phone

Description automatically generated

Request.POST returns a dictionary and then we use request.POST.get(‘search’) to get our searched item

We showed what we searched on the screen by placing the dictionary in the render function

A screenshot of a cell phone

Description automatically generated

A picture containing table

Description automatically generated

In html by writing {{ search }} we are basically printing out the variable on the site

We can automatically capitalize by adding a filter {{ search | title }} so now if we search py Thon ( avoid space ) it will print out Python

Return results in card format =

We also get this from cards in materializecss.com

To make them fit perfectly we set up a row

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

Now we are done with the front-end part let’s start adding the functionality

from requests.compat import quote\_plus

What this does is when we seach python tutor it will convert it to python%20tutor

I was getting error about the above that there is no quote\_plus in compat.pyi so instead I did from urllib.parse import quote.plus and it worked

A picture containing sitting, holding, display

Description automatically generated

When I searched for python tutor the final\_url printed out as A screenshot of a computer screen

Description automatically generated

A close up of a screen

Description automatically generated

Now that we get the html data as seen above we can use this for webscrapping for prices, pictures and all

beautifulSoup takes big files of html and makes it easily accessible by making it into an object

search = request.POST.get('search')

models.Search.objects.create(search=search)

so what the above does is creates an object of Search we made in models with the argument search which is stored in the admin site

A screenshot of a cell phone

Description automatically generated

Now we need to stuff with the data we are getting

We make a soup objects that parses all the html data in one object

post\_titles = soup.find\_all('a', {'class': 'result-title'})

the above line is basically find all the links that are a class of result-title

A screenshot of a cell phone

Description automatically generated

The post\_listing is important because we get all information from that

Now we loop through the information and send it to the front end

A screenshot of a cell phone

Description automatically generated

For services that don’t have a price we have to make an adjustment

A screenshot of a cell phone

Description automatically generated

We refer post to the front end

A screenshot of a cell phone

Description automatically generated

Post.1 = link

Post.2 = price

Post.0 = title

Now is the hard part the image scarping part

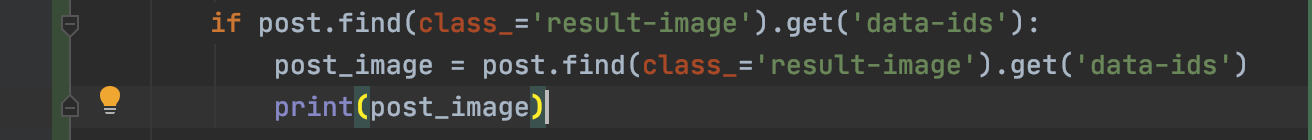
A screenshot of a cell phone

Description automatically generated

We notice there is no image url we can detect because there are multiple images which need data-ids and we notice the pattern with the data-id being used in the image url so we made a base image url

BASE\_IMAGE\_URL = 'https://images.craigslist.org/{}\_300x300.jpg'

Now we try to get the first data-id because we ignore the other images



Now this prints out multiple data-ids because the post has more than one picture so we need to split it to get the frist id

A picture containing food

Description automatically generated

So before we get a string and now after splitting it we will get a list of data-ids like below and we can access them by index

A screenshot of a cell phone

Description automatically generated

Now we need to remove the 1: prefix

We now split it again by ‘:’ to create another list but we want to the second item on the l list as the 1 before the ‘:’ will be in the list so now we get the below result



A close up of a logo

Description automatically generated