**Useful Commands:**

Pip freeze

Python –version

Python manage.py help

Make a virtual environment

python -m venv ./venv

Activate Environment:

venv\Scripts\activate.bat

Install:

Pip install django

Start a project with name btre (don’t forgot the fullstop!)

django-admin startproject btre .

Should set up my Git repo here:

python manage.py runserver

Create Django app called pages:

Python manage.py startapp pages

Next step is to add this app to the settings.py file of the project (btre)

Under the “Installed Apps” Section add ‘pages.apps.PagesConfig’ this ‘PagesConfig’ can be found in the pages ‘apps.py’ file

Now in my pages app create a file called ‘urls.py’:

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name='index')

]

We don’t have this ‘index’ member in the above code and it is flagging as an error because we do not have this method inside our views file. We must now add this to our views file.

Pages > views.py

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def index(request):

    return HttpResponse('<h1>Hello Django</h1>')

Now if we look at our webpage it is still not showing this “Hello Django” message as we still have to take this ‘urls.py’ file that we created and added it to the main ‘urls.py’ in the main project.

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('', include('pages.urls')),

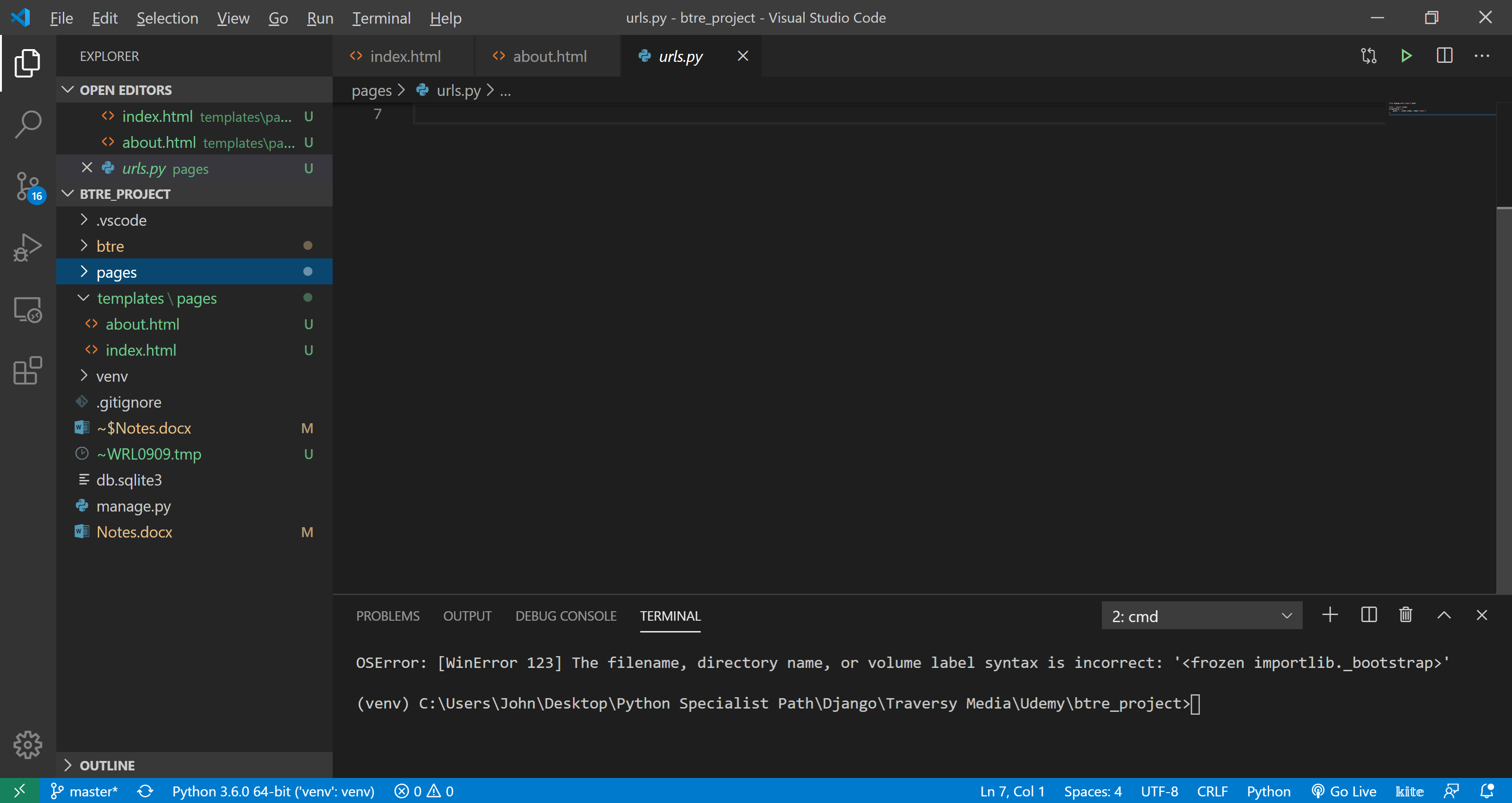
    path('admin/', admin.site.urls),

]

OK so now we have a basic webpage displaying the “Hello Django” message, we don’t really want to just insert html straight into our code like that. Instead we will make a template folder in our root directory to house these (front end) files. If we use this new file structure we have to tell the Django project where to find these templates, we can do that by updating the ‘DIRS’: line with the following in the projects ‘settings.py’ file under the templates section

'DIRS': [os.path.join(BASE\_DIR, 'templates')],

OK so now I have added a templates folder to the root, inside that folder I have another folder ‘pages’ inside pages I have ‘index.html’ and about.html’



Now that we have added these two new html files we need to add them to the pages ‘url.py’ file (note we already have the index added from before).

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name='index'),

    path('about', views.about, name='about')

]

Now we will get an error because we don’t have an about method in our views file so that line ‘views.about’ is a problem.

Pages ‘views.py’

from django.shortcuts import render

from django.http import HttpResponse

# Create your views here.

def index(request):

    return render(request, 'pages/index.html')

def about(request):

    return render(request, 'pages/about.html')

[Django documentation to render() method above](https://docs.djangoproject.com/en/3.0/topics/http/shortcuts/#django.shortcuts.render)

In the code above we are using the built in Django shortcut function ‘Render()’ to render the html page, it will return a HTMLResponse object with the rendered content. The request is a method such as ‘POST’, ‘GET’ etc …. In this case if we print request we get <WSGIRequest: GET '/'> . The second field here is the template name which we need to pass the location of the file. This is slightly confusing at first as you would think that you would pass ‘templates/pages/index.html’ but this would be incorrect as we have already told django to look for templates in the templates folder in one of the steps above.

Creating a ‘base.html’ template

There are some html components that we want in all of our pages, instead of copying and pasting the same code to multiple locations we can use a ‘base.html’ file. In our template folder we can all this ‘base.html’ file directly here.

Base.html

<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="UTF-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1.0" />

    <title>BT Real Estate</title>

  </head>

  <body>

    {% block content %} {% endblock %}

  </body>

</html>

Index.html

{% extends 'base.html' %} {% block content %}

<h1>Home</h1>

{% endblock %}

Note:

This uses jinja (this is a kind of programming language on its own?? )

{% extends 'base.html' %} {% block content %}

**Dealing with Static Files and paths:**

In the main project folder ‘btre’ create a new folder called static. In this static folder we will add any static files such as bootstrap files, css, webfonts. In this example I am following the Traversity media udemy course so he has given a list of files to add.

Now that we have added the static folder containing the static files we need to tell Django where to find them. We do this by modifying the main project ‘settings.py’ file with the following:

# Static files (CSS, JavaScript, Images)

# https://docs.djangoproject.com/en/3.0/howto/static-files/

STATIC\_ROOT = os.path.join(BASE\_DIR, 'static')

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

    os.path.join(BASE\_DIR, 'btre/static')

]

Now that we have set up the static structure we need to run:

python manage.py collectstatic

The above command goes into all of the applications and if it has a static folder it takes everything from that folder and puts it into a root static folder. This root static folder is automatically created whrn the collectstatic command is ran.

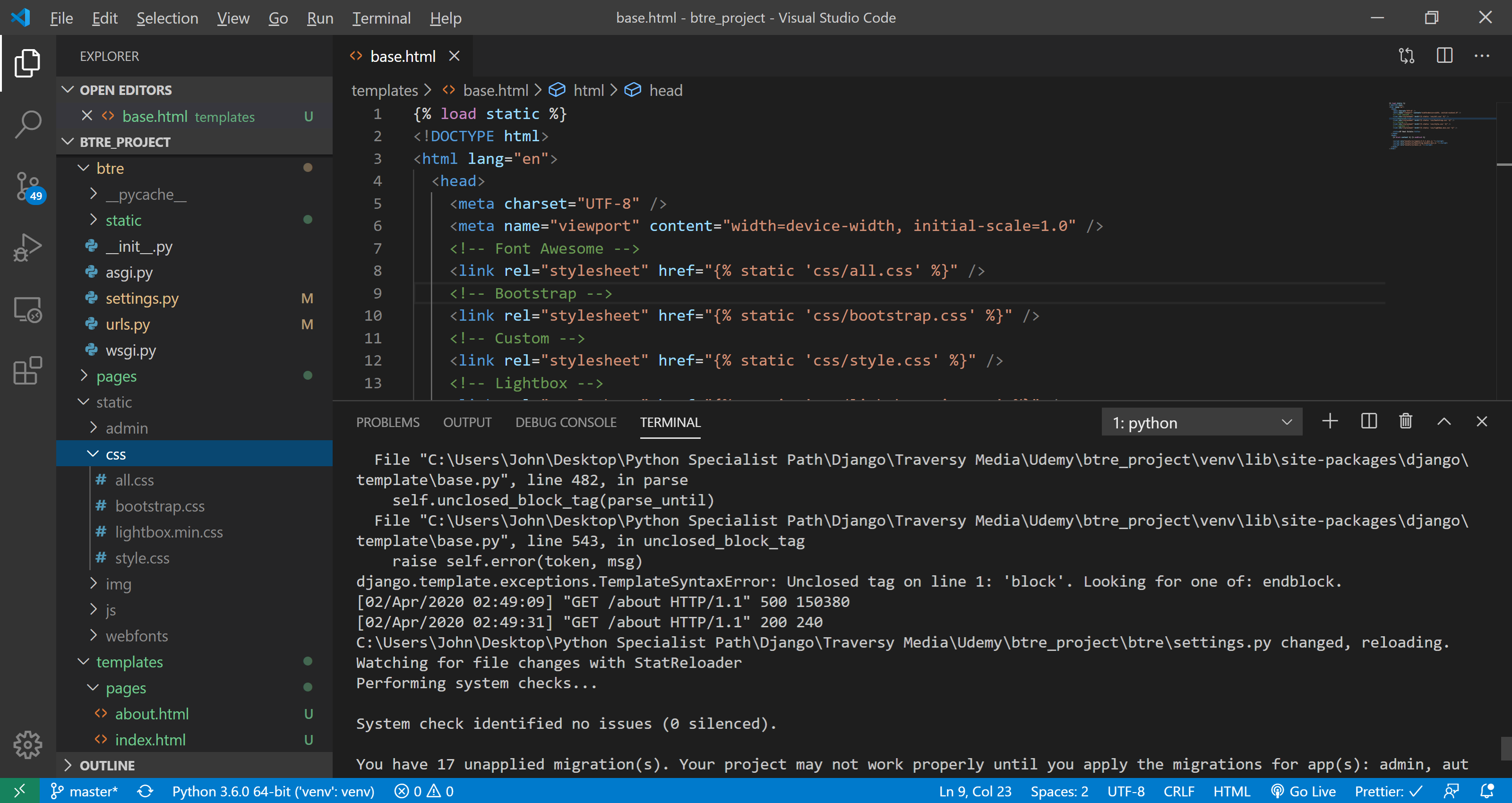
After this we want to update our .gitignore with:

/static

Now in our base.html file we need to modifiy a lot of things, first we need to load static at the very top of our file.

{% load static %}

Then we need to connect all of our links (note: this are actually in our static root folder)in the correct way for example



<!-- Font Awesome -->

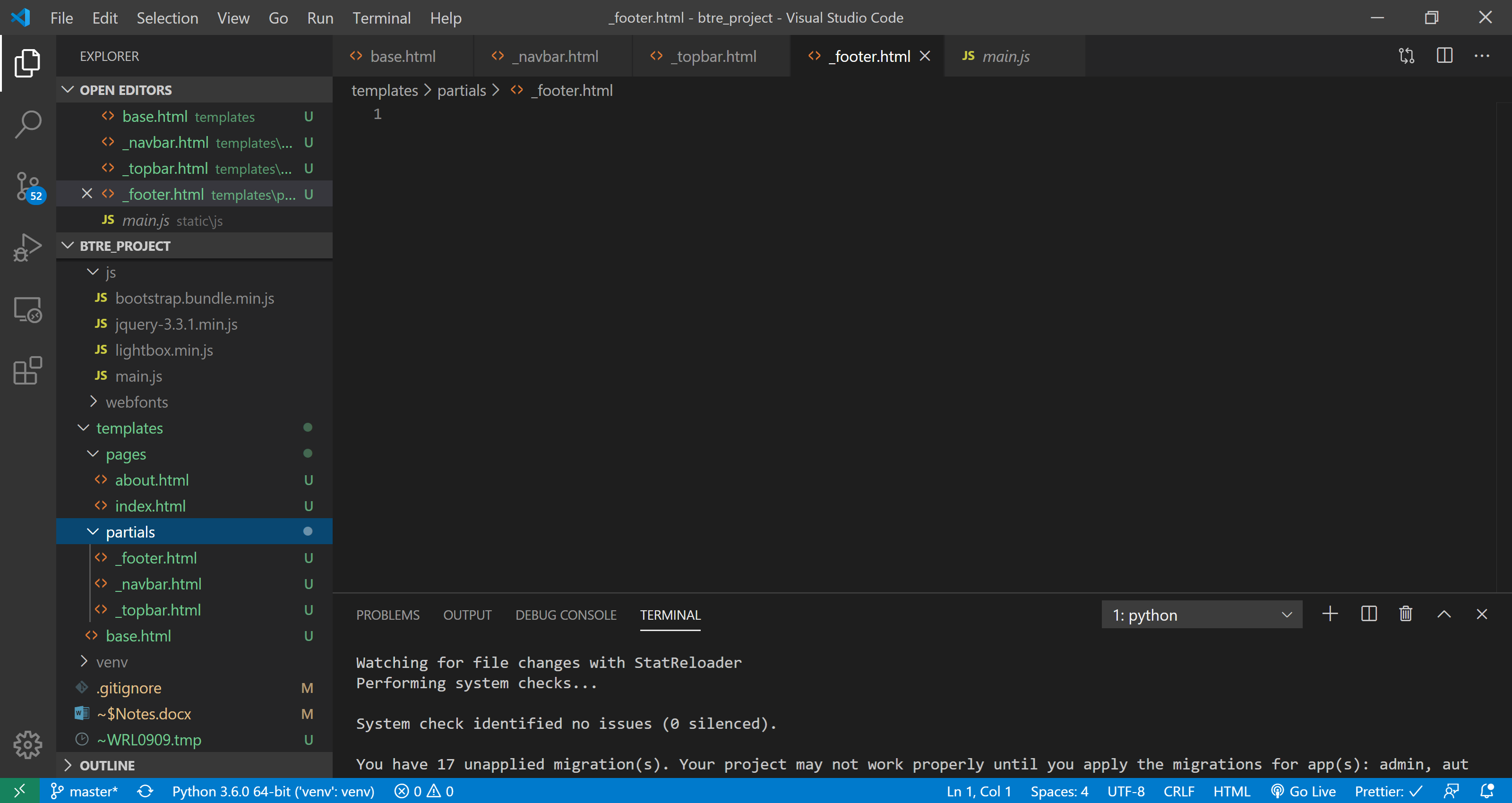
    <link rel="stylesheet" href="{% static 'css/all.css' %}" />

We need to do the same with the script tags:

<script src="{% static 'js/jquery-3.3.1.min.js' %} "></script>

Now we have the option of putting all of our html markup that is used on multiple pages into our ‘base.html’ this is ok to do but it is better if we use partials.

To use partials we need to create a ‘partial’ folder in our ‘templates’ folder. In this folder we can put in components of our pages such as the navbar, header, footer ect. The naming convention for partial files is ‘\_name.html’ the underscore is used.



To include these in the ‘base.html’ we just have to use

<!-- Nav Bar -->

    {% include 'partials/\_navbar.html' %}

The next step is to add our links within the site, so for example if we click ‘home’ on the navbar we want to be taken to the home page. To achieve this we use the following

<a class="nav-link" href="{% url 'index' %}">Home</a>

Note: we can use just ‘index’ instead of ‘index.html’ because in the pages app ‘urls.py’ file we have called it this:

path('', views.index, name='index'),

**ADDING A NEW APP:**

To add a new app called ‘listings’ to our project we can first run the ‘python manage.py startapp listings’ command. Then we are going to add our template files (first add a folder inside templates called listings: for this app inside the ‘listings’ folder we need: ‘listing.html’, ‘listings.html’ and ‘search.html’.

So now we need to add a ‘urls.py’ file inside our ‘listings’ app. Inside this file we will add our paths as shown below. This is a little bit confusing because inside of this app the path(‘ ’) is the home for this app and not the home for the project as a whole. The other path for listing uses an int to make it more dynamic i.e if we have a lot of different listings we can just pass the listing # or ID to display the required page. (note the reason that we don’t have for example ’listings/search’ for the paths is because we will link this to the main urls.py file so that django knows to look here for any path that has ‘listing/’ in it. Other notes: the methods index, listing and search will just be functions in our views file so we can call them anything we want.

‘urls.py’ for the ‘listings’ app

from django.urls import path

from . import views

urlpatterns = [

    path('', views.index, name='listings'),

    path('<int:listing\_id>', views.listing, name='listing'),

    path('search', views.search, name='search')

]

Now as mentioned above we need to update the main ‘urls.py’ file to tell Django where to find the paths for listings.

urlpatterns = [

    path('', include('pages.urls')),

    path('listings/', include('listings.url')),

    path('admin/', admin.site.urls),

]

Now we need to add our app to the main project ‘settings.py’ file

INSTALLED\_APPS = [

    'listings.apps.ListingsConfig',

We want to highlight (make active) the link on the navbar when we are on that specific page. To achieve this we need to go to the ‘\_navbar.html’ partial and add the following code:

<li

        {% if '/' == request.path  %}

        class="nav-item active mr-3"

        {% else %}

        class="nav-item mr-3"

        {% endif %}

        >

          <a class="nav-link" href="{% url 'index' %}">Home</a>

        </li>

<li

        {% if 'about' in request.path  %}

        class="nav-item active mr-3"

        {% else %}

        class="nav-item mr-3"

        {% endif %}

        >

Now this next section was slightly confusing but it involved installing postgress on my local machine.

I can then create a databse using the “pgAdmin4” web app that is include with postgress.

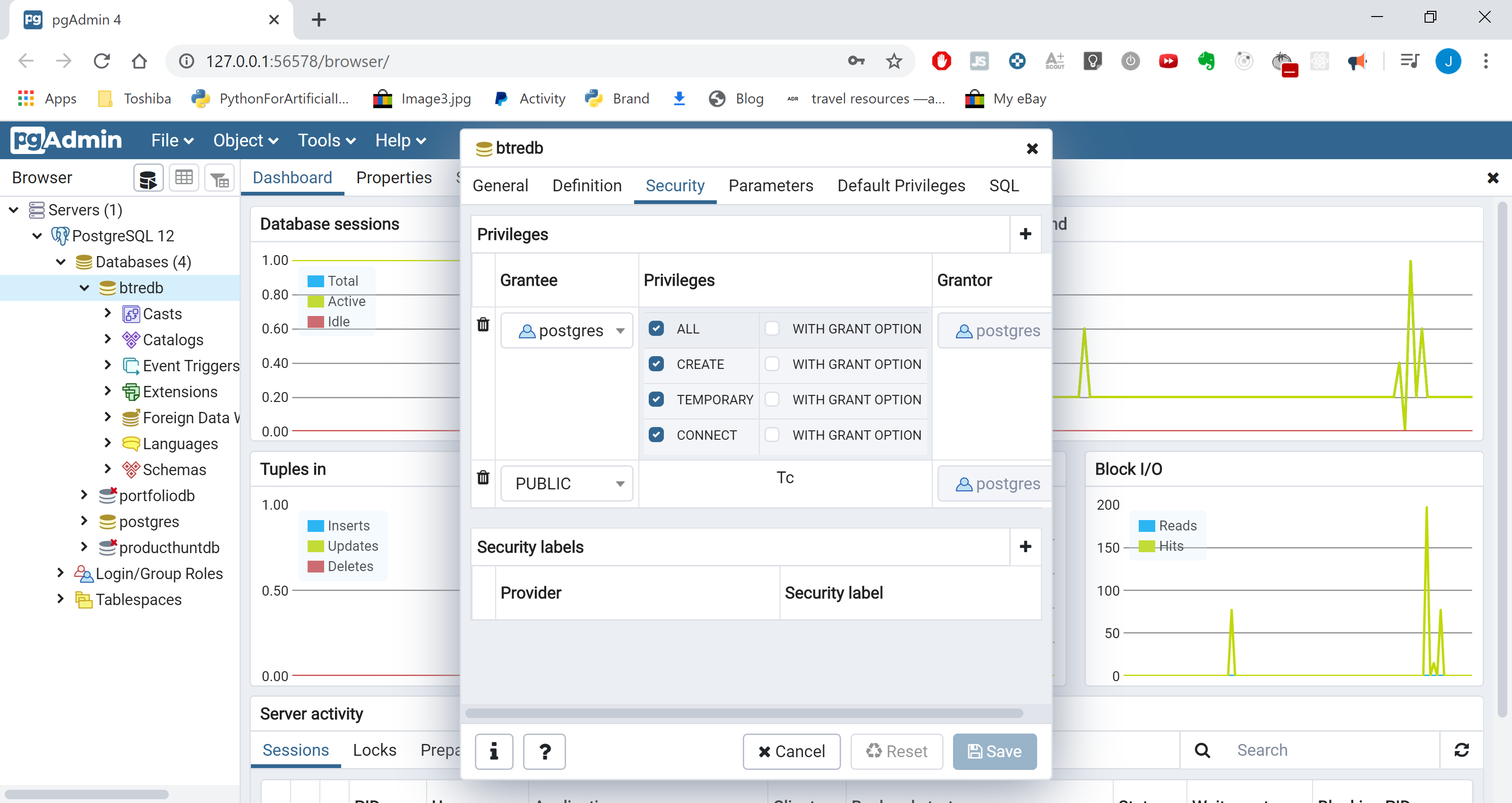
To do this open up the postgress ‘SQL Shell(psql)’ and press enter about 5 times until it prompts for a password. Enter password and then you should be able to enter the following SQL code (You will see an error about something too but can ignore).

run the following SQL:

CREATE DATABASE btredb OWNER postgress;

\l (this command will show all the DBs)

Now open “pgAdmin4”. You should see the data base that you made on the left had side. Right click and select properties, in the security tab add the postgress user and click the checkbox so they have access to everything. (not sure if this step is essential)



Now to use postgress with Djngo we need to install libraries in our virtual env.

Pip install psycopg2

pip install psycopg2-binary

Now that we have created this new DB we need to tell Django about it, to do this we must update the ‘settings.py’ file

DATABASES = {

    'default': {

        'ENGINE': 'django.db.backends.postgresql',

        'NAME': 'btredb',

        'USER': 'postgres',

        'PASSWORD': '123456',

        'HOST': 'localhost'

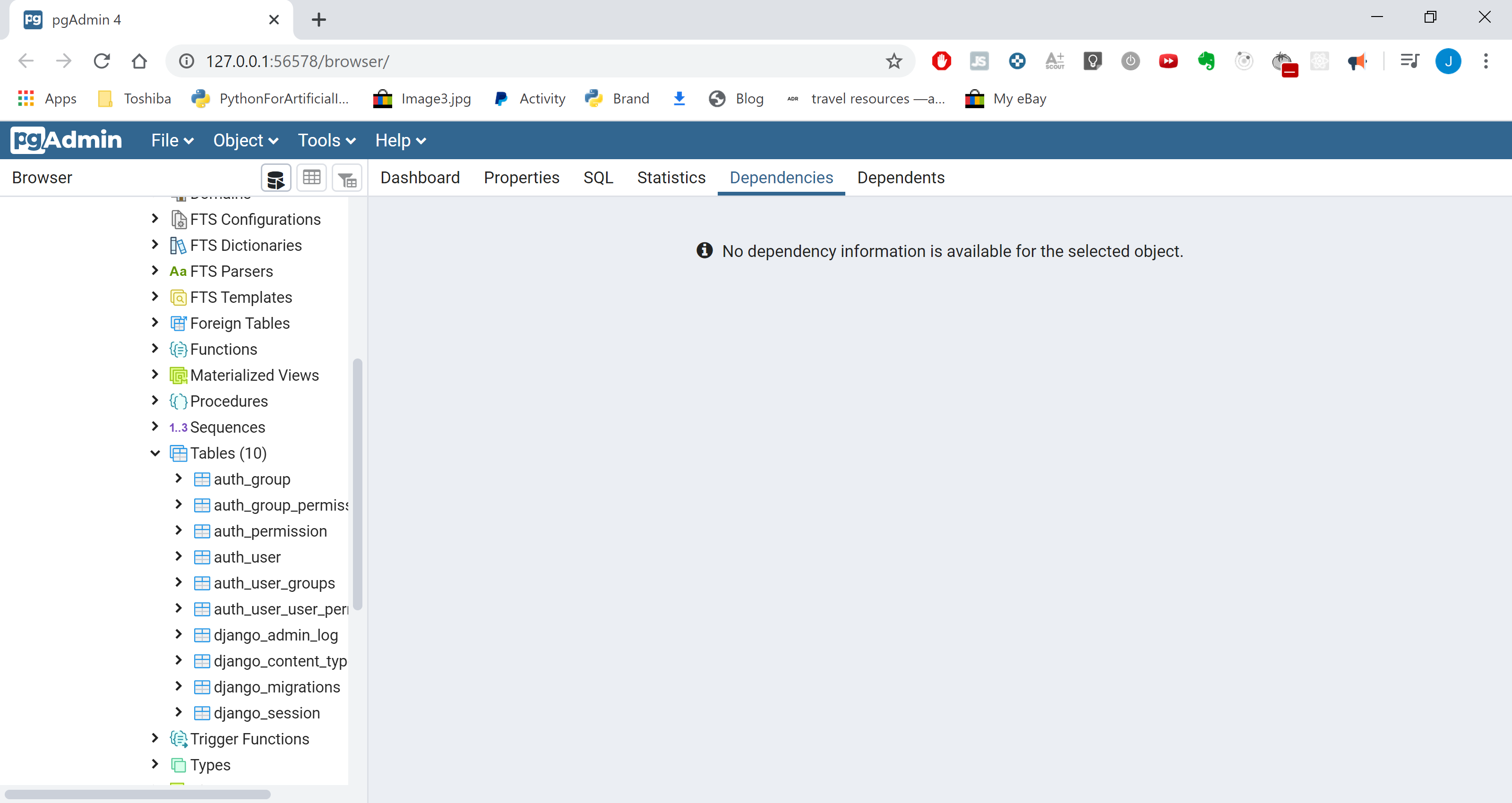
    }

}

Now run the migrations, the migrations will move everything to our database:

Python manage.py migrate

Now if we look into our ‘pgAdmin4’ we can see that all of the django tables are now in our database



Database design:

Her we need to think about what tables and fields we need in our database and they different tables are connected. For our real estate project an example layout is below:

Note: I am using [0], [true] to specify default values

### LISTING

Id: INT

Realtor INT (FOREGIN KEY [realtor])

Title: STR

Address: STR

City: STR

State: STR

Zipcode: STR

Description: TEXT

Price: INT

Bedrooms: INT

Bathrooms: INT

Garage: INT [0]

Sqft: INT

Lot\_size: FLOAT

Is\_published: BOOL [true]

List\_date: DATE

Photo\_main: STR (note: we are not actually storing an image here, we store only the location)

Photo\_1: STR

Photo\_2: STR

Photo\_3: STR

Photo\_4: STR

Photo\_5: STR

Photo\_6: STR

### REALTOR

Id: INT

Name: STR

Name: STR

Photo: STR

Description: TEXT

Email: STR

Phone: STR

Is\_mvp: BOOL [0]

Hire\_date: DATE

### CONTACT

Id: INT

User\_id: INT

Listing: INT

Listing\_id: INT

Name: STR

Email: STR

Phone: STR

Message: TEXT

Contact\_date: DATE

To start creating these tables in python we need to modify the models.py files

Listings > ‘models.py’

from django.db import models

from datetime import datetime

from realtors.models import Realtor

# Create your models here.

class Listing(models.Model):

    realtor = models.ForeignKey(Realtor, on\_delete=models.DO\_NOTHING)

    title = models.CharField(max\_length=200)

    address = models.CharField(max\_length=200)

    city = models.CharField(max\_length=100)

    state = models.CharField(max\_length=100)

    zipcode = models.CharField(max\_length=20)

    description = models.TextField(blank=True)

    price = models.IntegerField()

    bedrooms = models.IntegerField()

    bathrooms = models.DecimalField(max\_digits=2, decimal\_places=1)

    garage = models.IntegerField(default=0)

    sqft = models.IntegerField()

    lot\_size = models.DecimalField(max\_digits=5, decimal\_places=1)

    photo\_main = models.ImageField(upload\_to='photos/%Y/%m/%d/')

    photo\_1 = models.ImageField(upload\_to='photos/%Y/%m/%d/')

    photo\_2 = models.ImageField(upload\_to='photos/%Y/%m/%d/', blank=True)

    photo\_3 = models.ImageField(upload\_to='photos/%Y/%m/%d/', blank=True)

    photo\_4 = models.ImageField(upload\_to='photos/%Y/%m/%d/', blank=True)

    photo\_5 = models.ImageField(upload\_to='photos/%Y/%m/%d/', blank=True)

    photo\_6 = models.ImageField(upload\_to='photos/%Y/%m/%d/', blank=True)

    is\_published = models.BooleanField(default=True)

    list\_date = models.DateTimeField(default=datetime.now, blank=True)

    def \_\_str\_\_(self):

        return self.title

Because we used the ‘ImageField’ on our database we need to install Pillow to use them:

Pip install Pillow.

Now we have to make migrations for these new models and then migrate to the database:

Python manage.py makemigrations

Python manage.py sqlmigrate listings 0001 - This is optional to see the SQL code

Python manage.py migrate

**Django Admin:**

This is an amazin feature of Django….

To allow you access to this feature you need to create a superuser, run the command below and enter user details when prompted.

Python manage.py createsuperuser

Now we have access to the Admin section of Django. We can register our models with Admin so that we have access to the via the admin panel. To do this we have to go into the admin.py file for each individual app and add them:

from django.contrib import admin

from .models import Listing

# Register your models here.

admin.site.register(Listing)

note: in the code above the ‘Listing’ is coming from the models.py file (this is obvious but may as well make sure)

Adding a Media folder

In order to do this we first need to go to the settings.py file and add the following code:

# Media Folder Settings

MEDIA\_ROOT = os.path.join(BASE\_DIR, 'media')

MEDIA\_URL = '/media/'

We will also need to update the main project ‘urls.py’ file to look like the one below (not what was added was the imports and the ‘+ static’ line)

from django.contrib import admin

from django.urls import path, include

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

    path('', include('pages.urls')),

    path('listings/', include('listings.urls')),

    path('admin/', admin.site.urls),

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

**Customizing Django Admin:**

Django Admin is a really powerful feature of Django and if you are working as a freelancer or for a company it is a really usefull feature to give to your clients. For this reason it is important to customise it, the customisation includes adding logos titles and colour schemes.

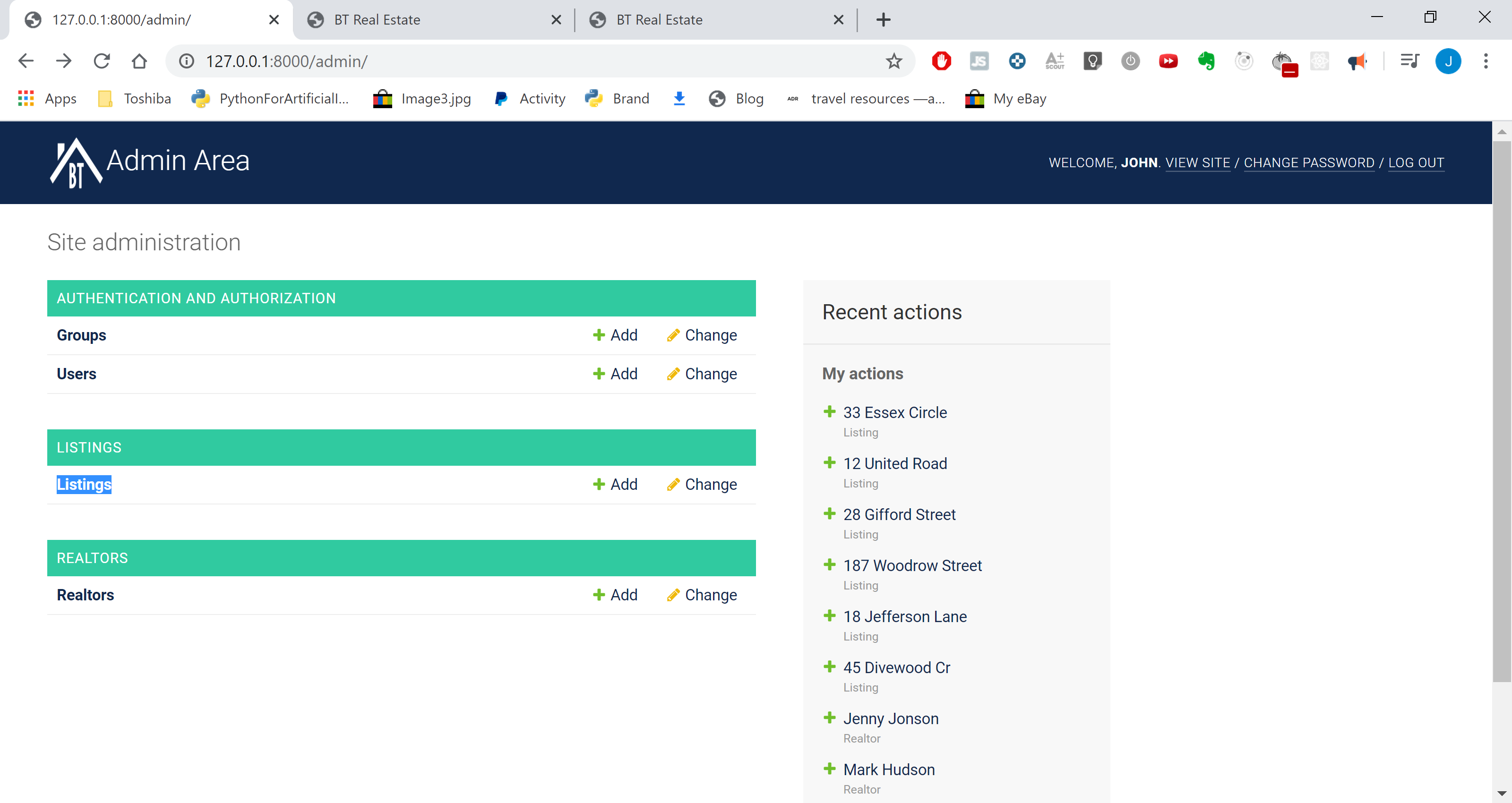
Side note:

In this part in the models.py files we are choosing what will be displayed in our Django Admin

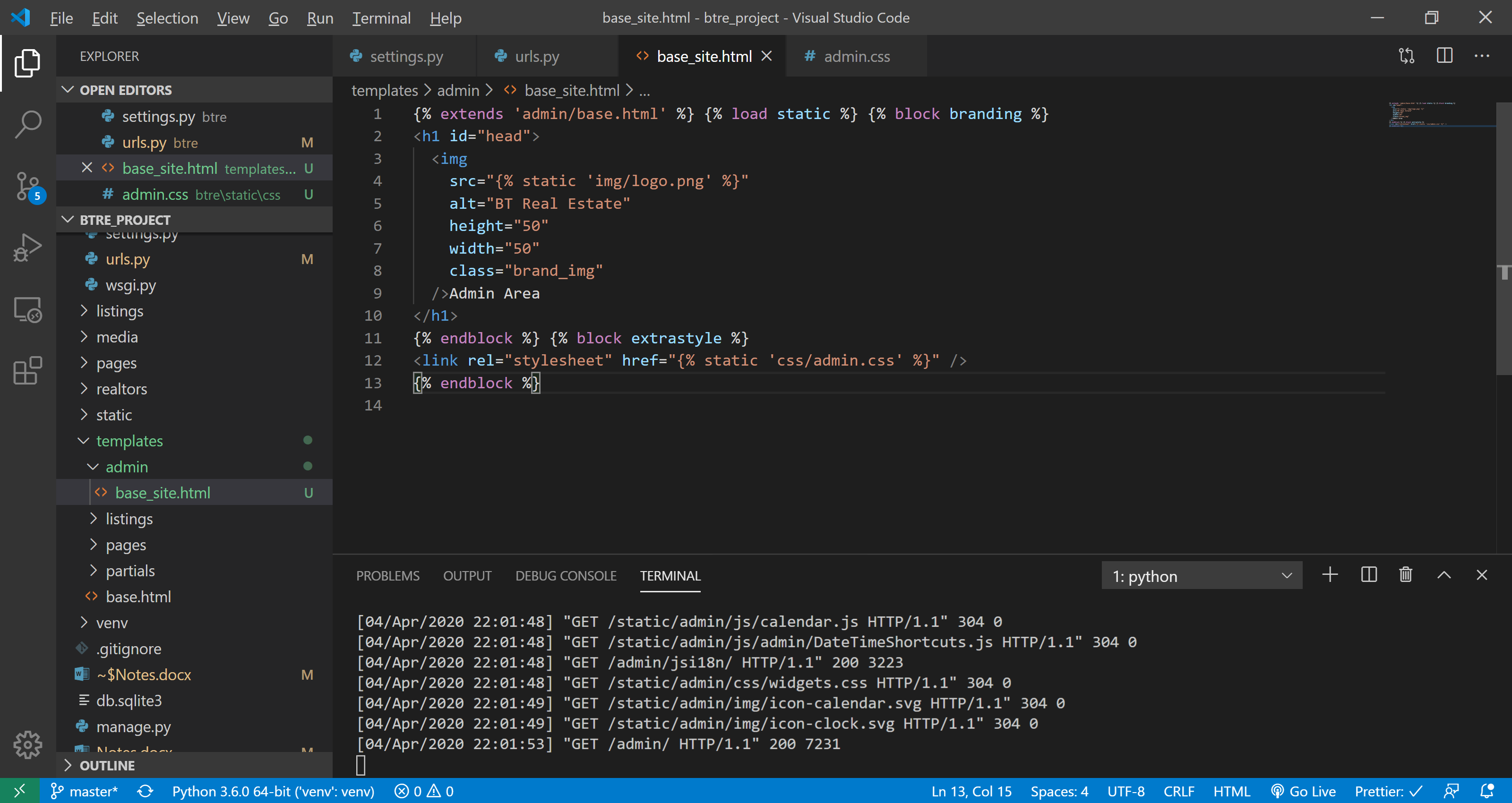
def \_\_str\_\_(self):

        return self.title

If we don’t use the code above django will use different names for ‘Listing’ maybe an ID number or something (not sure what the default display is but need to be aware of this)



For the customation we first need to add a new ‘admin’ folder inside of templates, inside this folder we need to add the file ‘base\_site.html’



‘base\_site.html’

{% extends 'admin/base.html' %} {% load static %} {% block branding %}

<h1 id="head">

  <img

    src="{% static 'img/logo.png' %}"

    alt="BT Real Estate"

    height="50"

    width="50"

    class="brand\_img"

  />Admin Area

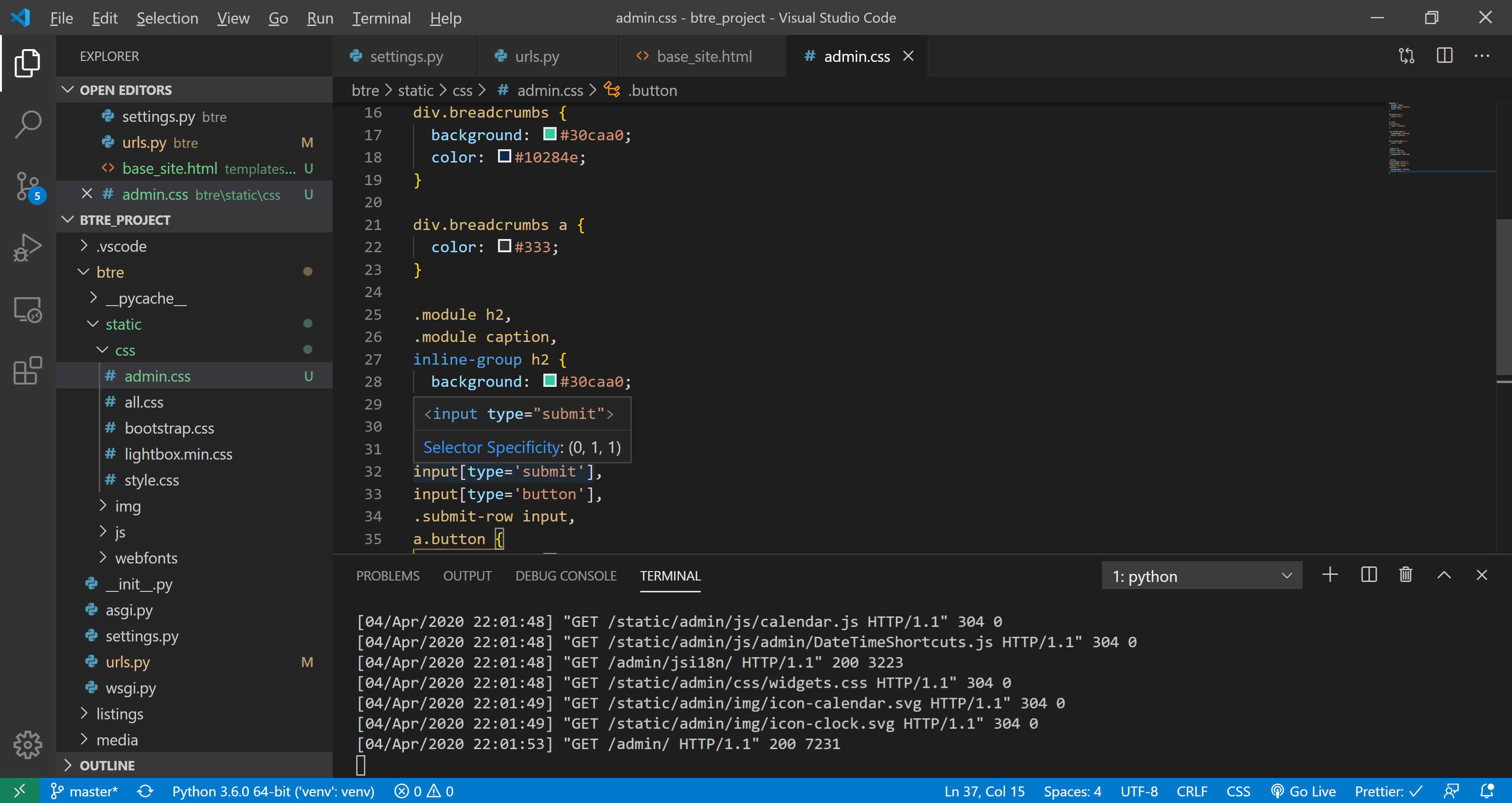
</h1>

{% endblock %} {% block extrastyle %}

<link rel="stylesheet" href="{% static 'css/admin.css' %}" />

{% endblock %}

In our ‘static’ folder ‘css’ we need to add ‘admin.css’



‘admin.css’

#header {

  height: 50px;

  background: #10284e;

  color: #fff;

}

#branding h1 {

  color: #fff;

}

a:link,

a:visited {

  color: #10284e;

}

div.breadcrumbs {

  background: #30caa0;

  color: #10284e;

}

div.breadcrumbs a {

  color: #333;

}

.module h2,

.module caption,

inline-group h2 {

  background: #30caa0;

}

.button,

input[type='submit'],

input[type='button'],

.submit-row input,

a.button {

  background: #10284e;

  color: #fff;

}