**SETTINGS.PY**

**This file controls our project’s settings**

import os

# Build paths inside the project like this: os.path.join(BASE\_DIR, ...)

BASE\_DIR = os.path.dirname(os.path.dirname(os.path.abspath(\_\_file\_\_)))

TEMPLATE\_DIR = os.path.join(BASE\_DIR, 'shoes/templates/shoes')

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

INSTALLED\_APPS = [

'apps.shoes',

'django.contrib.admin',

'django.contrib.auth',…

TEMPLATES = [

{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [TEMPLATE\_DIR,],

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [

STATIC\_DIR,

]

**URLS.PY**

**This file tells Django which pages to build in response to a browser or url request**

from django.urls import path, include

from django.contrib import admin

urlpatterns = [

path('admin/', admin.site.urls),

path('', include("apps.blog.urls")),

]

**CREATE .gitignore under the main project directory**

**Tells Git to ignore these files so they cannot be viewed**

\*.pyc

.vs

\_\_pycache\_\_/

venv/

**Create a base.html page to extend your header to other pages**

<header>

<a href="{% url 'home' %}">Home</a> | <a href="{% url 'about' %}">About</a>

</header>

{% block conntent %}

{% endblock %}

**Add the extends expression on your html pages**

<body>

{% extends 'blog/base.html' %}

{% block conntent %}

<h1>Hello This is My Blog Project!</h1>

{% endblock %}

</body>

**Using Our Test.py file. Open this file under your project apps directory and add the following:**

from django.test import TestCase

class SimpleTests(SimpleTestCase):

def test\_home\_page\_status\_code(self):

response = self.client.get('/')

self.assertEqual(response.status\_code, 200)

def test\_about\_page\_status\_code(self):

response = self.client.get('/about/')

self.assertEqual(response.status\_code, 200)

**Ctl C out of your server and type python manage.py test**

**Let’s execute the migrate command to create an initial database using Django’s default settings.**

**Go to your main project’s directory and type:**

**python manage.py migrate**

**BUILDING MODELS FOR OUR DATABASE**

**Django’s Doc found at:**[**https://docs.djangoproject.com/en/2.0/ref/models/fields/**](https://docs.djangoproject.com/en/2.0/ref/models/fields/)

**In order to display our database content on our homepage, we have to wire up our**

**views, templates, and URLConfs. This pattern should start to feel familiar now.**

**1. Open up your models.py file under your blog app and add:**

from django.db import models

class Post(models.Model):

text = models.TextField()

**2. Go to command line in main project directory and type:**

**python manage.py makemigrations blog**

**python manage.py migrate**

**3. Create a superuser for your admin to access your database via GUI. Run:**

**python manage.py createsuperuser**

**Enter required info.**

**4. Tell Django what to display in the Admin console:**

**Open admin.py and add:**

from django.contrib import admin

from .models import Post #imports the Post class in your model to display in admin GUI

admin.site.register(Post)

**5. Go to models.py and add:**

**In order to display our database content on our homepage, we have to wire up our**

**views, templates, and URLConfs. This pattern should start to feel familiar now.**

**6. Let’s edit our model’s view.py file**

**7. Let’s configure our URLConfs and make our post template.  
Go to your main project urls.py file and include our post path**

**Go to the app url and make the following changes:**

**Querying sqlite database in command line:**

Faker Library

**STEP 1**

**--------------------------------------------------------------------------------------------------------------------------------**

Go into your first\_app directory under apps and open up the **models.py** file.

Import the following:

**from django.db import models**

Create classes for your fake data.

class Topic(models.Model):

top\_name = models.CharField(max\_length=264, unique=True)

def \_\_str\_\_(self):

return self.top\_name

class Webpage(models.Model):

topic = models.ForeignKey('Topic', on\_delete=models.CASCADE)

name = models.CharField(max\_length=264, unique=True)

url = models.URLField(unique=True)

def \_\_str\_\_(self):

return [self.name](http://self.name/)

class AccessRecord(models.Model):

name = models.ForeignKey('Webpage', on\_delete=models.CASCADE)

date = models.DateField()

def \_\_str\_\_(self):

return str(self.date)

Save the models.py file.

**STEP 2**

**--------------------------------------------------------------------------------------------------------------------------------**

Open up your django environment.  Go to your main project directory and enter:

**python manage.py migrate**

Once migration is complete enter:

**python manage.py makemigrations first\_app**

reenter

**python manage.py migrate**

Your model should be ready for sql integration.

Run a test to see if the migration worked by entering the following:

**python manage.py shell**

>>>**from first\_app.models import Topic**

>>>**print(Topic.objects.all())**

run by hitting enter again

it should return an empty set

**>>> QuerySet [ ]**

Now let's create a new object using our models

**>>> t = Topic(top\_name = "Social Network")**

**>>> t.save()**

**>>> print(Topic.objects.all())**

it should print out:

**>>> <QuerySet [<Topic: Social Network>]>**

To exit the shell enter:

quit()

**STEP 3**

**--------------------------------------------------------------------------------------------------------------------------------**

Now let's open the **admin.py** file under your first\_app directory and add the following:

**from first\_app.models import AccessRecord, Topic, Webpage**

now let's register them by adding:

**admin.site.register(AccessRecord)**

**admin.site.register(Topic)**

**admin.site.register(Webpage)**

**STEP 3**

**--------------------------------------------------------------------------------------------------------------------------------**

**Now the fun begins!!**

Let's create a super user to keep our files secure and only accessible by the developers working on the project

Go into your django environment and make sure you are in your main project directory. Enter:

**python manage.py createsuperuser**

It will prompt you to enter a username, email address then password.

Your admin interface is now active.  Woohoo!!

Okay so I might have been too quick on the draw.  Let's check to be sure.

Enter:

**python manage.py runserver**

Go to your first\_app's url and at the end enter the sub-domain:

**/admin**

Your beautiful interface should be anxiously waiting for you to enter a username and password.  Once entered...... voila, you're in!

**STEP 4**

**--------------------------------------------------------------------------------------------------------------------------------**

**Now we are ready for the Faker!**

Go to your command line and enter:

**pip install Faker**

Go to your main project directory and create a new file and let's call it populate\_first\_app.py

Open the file and enter the following:

import os

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'first\_project.settings')

import django

django.setup()

import random

from first\_app.models import AccessRecord, Webpage, Topic

from faker import Faker

fakegen = Faker()

topics = ['Search', 'Social', 'Marketplace', 'News', 'Games']

def add\_topic():

    t = Topic.objects.get\_or\_create(top\_name=random.choice(topics))[0]

    t.save()

    return t

def populate(N = 5):

    for entry in range(N):

        # get the topic for the entry

        top = add\_topic()

        # create the fake data for that entry

        fake\_url = fakegen.url()

        fake\_date = fakegen.date()

        fake\_name = fakegen.company()

        # create the new webpage entry

        webpg = Webpage.objects.get\_or\_create(topic=top, url=fake\_url, name=fake\_name)[0]

        # create a fake access record for that webpage

        acc\_rec = AccessRecord.objects.get\_or\_create(name=webpg, date=fake\_date)[0]

if \_\_name\_\_ == '\_\_main\_\_':

    print("populating script!")

    populate(40)

    print("populating complete!")

Go to your django environment and enter:

**python populate\_first\_app.py**

If populate script is effective you will see:

populating script!

populating complete!

Now run the server:

**python manage.py runserver**

Go to your admin section and your populated info should be accessible.