**Second\_project with Django**

Create the root directory.

Create the project root directory as a folder on the desktop, name is django\_chapter5, right click and open with vscode or open the root directory with the following command inside terminal: view-terminal-cd/desktop-mkdir django\_chapter5-cd/Django\_chapter5

Create the virtual environment.

Cd into django\_chapter5 directory and create the virtual environment.

Create a virtual environment using the venv module with the command python -m venv myvenv or virtualenv -p python myvenv

The name of the virtual environment directory is myvenv and the period that precedes the virtual environment directory indicates that the directory will be hidden.

Activate the virtual environment

Inside the django\_chapter5 directory, using the powershell terminal, use the following command to activate the virtual environment: myvenv/scripts/activate

Or using the gitbash terminal use the following command: source myvenv/scripts/activate

Install Django

With the virtual environment activated and at the root of django\_chapter5 directory, use the following command to install Django package: pip install Django

Check your installations

With the virtual environment activated and at the root of django\_chapter5 directory, use the following command to check the dependencies you have installed: pip freeze (click enter)

Start a new project

With the virtual environment activated and at the root of django\_chapter5 directory, use the following command to start a new project and name the new project as follows: django-admin startproject second\_project .

The period after the second\_project name is intentional; this will make Django to place all the files inside one single directory (django\_chapter5).

Apply migrations

When Django created the second\_project directory, several built-in apps were created along . similarly, some migrations files were created.

We need to apply migrations so that these changes can be applied to the database. we need to build the database

With the virtual environment activated, and at the root of django\_chapter5 directory, use the following command to apply migration: python manage.py migrate

Start the development server

With the virtual environment activated and at the root of django\_chapter5 directory, use the following command to start the development server: python manage.py runserver

Stop the development server with the following command: ctrl + c

Starting an app.

With the virtual environment activated and at the root of django\_chapter5, use the following command to start an app named pages: python manage.py startapp blog.

Add the blog app to the list of installed apps inside the second\_project directory settings.py

Create a database model inside the models.py file of the blog app & name it Post

We will have only one table called “Post” with three fields

Django imports a module, models, to help us build new database models that will “model” the characteristics of the data in our database.

Open the blog/models.py file and write the following codes:

**from django.db import** models

**class Post**(models.Model):

title = models.CharField(max\_length=30)

author = models.CharField(max\_length=30)

text = models.TextField()

We’ve created a new database model called Post, which has the database fields called title/author/text. We’ve also specified the *type of content* it will hold.

Activate the database model, Post

whenever we make or modify an existing model we need to activate it in two steps

1.First, we create a migrations file with the makemigrations command. Migration files record any changes to the database models, which means we can track changes over time and debug errors as necessary.

2. Second, we build the database with the migrate command, which executes the instructions in our migrations file.

Ensure the local server is stopped by typing Control+c on the command line and then run the commands, python manage.py makemigrations blog and python manage.py migrate blog

as a best practice, adopt the habit of always including the name of an app

when executing the makemigrations command!

Update the admin.py file of the blog app with the Post model

The administrator panel is a developer tool

In Django we use the admin panel to modify the database

We use the blog/admin.py to register the model and make it visible on the admin panel

To use the Django admin panel, we must first create a superuser who can log in.

With virtual environment activated, at the root of django\_chapter5 directory, use the following code to create the superuser: type python manage.py createsuperuser and respond to the prompts for a username, email, and password:

Username:admin

Email:olabowaleogunbanwo@gmail.com

Password:trevor4ivan

Restart the Django server with python manage.py runserver and, in your web browser, go to http://127.0.0.1:8000/admin/. You should see the login screen for the admin:

Log in by entering the username and password you just created. You will see the Django admin homepage next:

But where is our posts app since it is not displayed on the main admin page? Just as we must explicitly add new apps to the INSTALLED\_APPS config, we must also update an app’s admin.py file for it to appear in the admin panel.

In your text editor, open up blog/admin.py and add the following code to display the Post model

**from django.contrib import** admin

**from .models import** Post

admin.site.register(Post)

Django knows it should display our blog app and its database model Post on the admin page. If you refresh your browser, you’ll see that it appears.

Create our first message using the admin dashboard panel

Use the admin panel to add post to the blog

Let’s create our first post for our database. Click the + Add button inside the admin dashboard, opposite Posts and enter your content in the Text form field

Then click the “Save” button to redirect you to the main Post page. However, if you look closely, there’s a problem: our new entry is called “Post object (1)”, which isn’t very descriptive!

Let’s change that. Within the blog/models.py file, add a new method called \_\_str\_\_, which provides a human-readable representation of the model. In this case, we’ll have it display the first 50 characters of the text field.

Add the following code to the models.py file of the blog app

**from django.db import** models

**class Post**(models.Model):

title = models.CharField(max\_length=30)

author = models.CharField(max\_length=30)

text = models.TextField()

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

**return** self.text[:50]

If you refresh your Admin page in the browser, you’ll see that it now represents our database entry in a much more descriptive and helpful way.

It’s a best practice to add \_\_str\_\_() methods to all your models to improve their

readability.

Let’s add two more entries using the same method, so we have three total posts to work with in the next section. You can use the “Add Post +” button in the upper right corner

TemplateView renders an HTML files

ListView enable us to access the data of the database model

To access our database objects we need a ListView

To show our database objects we need the ListView and the ListView is inside generic

Create a class-based view

To display our posts on the homepage we have to wire up a view, template, and URL.

Open the blog/view.py file of blog app and write the following codes

**from django.shortcuts import** render

from django.views import generic

**from .models import** Post

class PostView(generic.ListView):

template\_name = 'blog/posts.html'

model = Post

Create the templates directory at the root of django\_chapter5 directory & update the content with posts.html file

At the root of django\_chapter5 with the virtual environment activated, create a directory named templates, select the templates directory and create another directory named blog, right click the blog directory and create an html file named posts.html file

Then, update the DIRS field in our second\_project/settings.py file so Django can look in this file for the new templates directory.

"DIRS": [BASE\_DIR / "templates"],

Open the posts.html file and update the file with the following codes:

1. <h1>Posts</h1>
3. {% for post in object\_list %}
4. <hr>
5. <h2>{‌{ post.title }}</h2>
6. <b>{‌{ post.author }}</b>
7. <br>
8. {‌{ post.text }}
9. {% endfor %}

Create a urls.py file inside the blog app

we need to match a url path to this view function.

create a urls.py file inside the blog app and write the following code

**from django.urls import** path

from . import views

urlpatterns = [

path("posts/", views.PostView.as\_view(), name='posts'),

]

update the second\_project /urls.py (gateway to other urls) file

There is a built-in urls.py file inside the root directory second\_project

Open the urls.py file and import the following code:

**from django.contrib import** admin

**from django.urls import** path, include

urlpatterns = [

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

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

]

Start the development server with python manage.py runserver.

If you navigate to the Django admin and add or delete blog posts, the homepage will be updated to reflect the changes.

For viewing the individual posts

Create a class-based view inside the blog/views.py

Open blog/views.py and add the following codes

From django.shortcuts import render

from django.views import generic

**from .models import** Post

class PostDetailView(generic.DetailView):

template\_name = "blog/post.html"

model = Post

Note: with ListView we have an object\_list in templates, but we don’t have object\_list in DetailedView, but we now have access to object.title/object.author/object.text because this is a DetailedView

Detailed view gives usaccess to a single object

Listview gives us access to all the objects

Create the templates directory at the root of django\_chapter5 directory & update the content with post.html file

Note: our model name first, letters will be in lowercase post

At the root of django\_chapter5 with the virtual environment activated, create a directory named templates, select the templates directory and create another directory named blog, right click the blog directory and create an html file named post.html file

1. <h2>{‌{ object.title }}</h2>
2. <b>{‌{ object.author }}</b>
3. <br>
4. {‌{ object.text }}

Open blog/urls.py file inside the blog app and update the code

**from django.urls import** path

from . import views

urlpatterns = [

path("posts/", views.PostView.as\_view(), name='posts'),

path(‘post/’(?P<pk>[0-9]+)/', views.PostDetailView.as\_view(), name='post'),

]

To access the individual blog post, log into the admin panel and go to the route for single post by indicating their primary keys, 127.0.0.1:8000/post/1 to see the first post or 127.0.0.1:8000/post/2 to see the second post etc

To see all the post together, log into the admin panel and go to the route for all posts, 127.0.0.1:8000/posts to see all the posts together

Adding static files

At the root of django\_chapter5 with the virtual environment activated, create a directory named static, select the static directory and create another directory named blog, right click the blog directory and create a css file named style.css

Then, update the DIRS field in our second\_project/settings.py file so Django can look in this new templates directory.

"DIRS": [BASE\_DIR / "static"],

Also add os.path.join(BASE\_DIR, ‘static’),

Views.py will usually load the static files

Add the following code to style.css

1. body {
2. background: blue;
3. color: white;
4. }

Add the following codes to the top of blog/posts.html

1. {% load static %}
2. <link rel="stylesheet"
3. type="text/css"
4. href="{% static 'blog/style.css' %}" />
5. <h1>Posts</h1>
7. {% for post in object\_list %}
8. <hr>
9. <h2>{‌{ post.title }}</h2>
10. <b>{‌{ post.author }}</b>
11. <br>
12. {‌{ post.text }}
13. {% endfor %}

Open the url 127.0.0.1:8000/posts to confirm that the css has been applied

Add the following codes to the top of blog/post.html

1. {% load static %}
2. <link rel="stylesheet"
3. type="text/css"
4. href="{% static 'blog/style.css' %}" />
5. <h1>Posts</h1>
6. <h2>{‌{ object.title }}</h2>
7. <b>{‌{ object.author }}</b>
8. <br>
9. {‌{ object.text }}

Create a navbar.html for navigation

At the root of django\_chapter5 with the virtual environment activated, create a directory named templates, select the templates directory and create another directory named blog, right click the blog directory and create an html file named navbar.html file

Update the navbar.html with the following codes

1. <div class="topbar">
2. <div class="container">
3. <div class="row">
4. <div class="left">
5. <h1>Django Blog App</h1>
6. </div>
8. <div class="right">
9. <ul>
10. <li><a href="/posts">See All Posts</a></li>
11. <li><a href="/post/1">See 1st Posts</a></li>
12. </ul>
13. </div>
14. </div>
15. </div>
16. </div>

Use this code to include navbar.html in the post.html and posts.html:

{% include 'blog/navbar.html' %}

Add more codes to the style.css

Start the development server and see the css applied

Django User login functionality

Django comes with a login system that can have one or more users.

Open second\_project/urls.py and add the following codes as another installed apps:   url('accounts/', include('django.contrib.auth.urls')),

Login templates

Login functionality is installed by default by Django, represented inside second\_project/settings.py file as django.contrib.auth

At the root of django\_chapter5 with the virtual environment activated, create a directory named templates, select the templates directory and create another directory named registration, right click the registration directory and create an html file named login.html file

Add the following codes to login.html file

1. <!-- templates/registration/login.html -->
2. <h2>Login</h2>
3. <form method="post">
4. {% csrf\_token %}
5. {‌{ form.as\_p }}
6. <button type="submit">Login</button>
7. </form>

Start the development server and login form will be automatically be opened

Update the second\_project/setting.py file as follows:

Where do people go or redirected after clicking login, this settings below will determine, add these to second\_project/settings.py

1. LOGIN\_REDIRECT\_URL = '/posts/'
2. LOGOUT\_REDIRECT\_URL = '/posts/'

Update posts.html with the following codes

Adding login message

1. {% block content %}
2. {% if user.is\_authenticated %}
3. Welcome {‌{ user.username }}! <a href="{% url 'logout' %}">logout</a>
4. {% else %}
5. You are not logged in <a href="{% url 'login' %}">login</a>
6. {% endif %}
7. {% endblock %}

Signing up or register account

At the root of second\_project, We need to create a new app called accounts, using python manage.py startapp accounts.

Visit the second\_project/settings.py and add the new ‘accounts. apps.AccountsConfig’. This will initialize our account app.

Update the second\_project/urls.py with the url(‘accounts/’ , include(‘accounts.urls’)

Create accounts/urls.py file and update it with

from django.conf.urls import url

From . import views

Urlpatterns = [

url(‘signup/’ , views.SignUp.as\_view(), name = ‘signup’]

update the accounts/view.py file with the following codes

1. from django.shortcuts import render
2. from django.views import generic
3. from django.contrib.auth.forms import UserCreationForm
4. from django.urls import reverse\_lazy
6. # Create your views here.
8. class SignUp(generic.CreateView):
9. template\_name = 'registration/signup.html'
10. success\_url = reverse\_lazy('login')
11. form\_class = UserCreationForm

update template/registration/signup.html file with the following codes;

1. <!-- templates/signup.html -->
3. {% block content %}
4. <h2>Sign up</h2>
5. <form method="post">
6. {% csrf\_token %}
7. {‌{ form.as\_p }}
8. <button type="submit">Sign up</button>
9. </form>
10. {% endblock %}

Restart the sever, open accounts and sign up.

Register a new user and login

Password change configuration

**open accounts/urls.py and add the following codes**

1. from django.conf.urls import url
2. from . import views
3. from django.contrib.auth.views import (
4. PasswordChangeView,
5. PasswordChangeDoneView,
6. )
8. urlpatterns = [
9. url('signup/', views.SignUp.as\_view(), name='signup'),
10. url(r'^password\_change/$',
11. PasswordChangeView.as\_view(template\_name='registration/password\_change\
12. .html'),
13. name='password\_change'),
14. ]

Create template/registration/password\_change.html file and add the following codes

1. <form method="post">
2. {% csrf\_token %}
3. {‌{ form }}
4. <button type="submit">Save changes</button>
5. </form>

Create template/registration/password\_change\_done.html file and add the following codes; Passwords\_Changed

Update account/urls.py file with the following codes

1. url(r'password\_change/done',
2. PasswordChangeView.as\_view(template\_name='registration/password\_change\_done.html'),
3. name='password\_change\_done'),

update the second\_project/urls.py file with the following codes

1. from django.conf.urls import url
2. from django.contrib import admin
3. from django.urls import path, include
5. urlpatterns = [
6. url(r'^admin/', admin.site.urls),
7. url('', include('blog.urls')),
8. url('accounts/', include('accounts.urls')),
9. url('accounts/', include('django.contrib.auth.urls')),

Restart sever, open account and change password

Password reset

**Open second\_project/settings.py and set email backend, as follows** EMAIL\_BACKEND = 'django.core.mail.backends.console.EmailBackend'

Add template for password reset as follows account/registration/password\_reset\_form.html and add the following codes

1. {% block content %}
2. <h1>Forgot your password?</h1>
3. <p>Enter your email address below, and we'll email instructions for setting a new one.</p>
5. <form method="POST">
6. {% csrf\_token %}
7. {‌{ form.as\_p }}
8. <input type="submit" value="Send me instructions!">
9. </form>
10. </div>
11. {% endblock %}

Create password\_reset\_done.html and add the following codes

1. {% block content %}
2. <h1>Check your inbox.</h1>
3. <p>We've emailed you instructions for setting your password. You should receive the email shortly!</p>
4. {% endblock %}

Create template/password\_reset\_confirm.html

1. {% block content %}
2. <h1>Set a new password!</h1>
3. <form method="POST">
4. {% csrf\_token %}
5. {‌{ form.as\_p }}
6. <input type="submit" value="Change my password">
7. </form>
8. {% endblock %}

Create template password\_reset\_complete.html

1. {% block content %}
2. <h1>Password reset complete</h1>
3. <p>Your new password has been set. You can log in now on the <a href="{% url 'login' %}">log in page</a>.</p>
4. {% endblock %}

**Update accounts/urls.py file**

1. from django.conf.urls import url
2. from . import views
3. from django.contrib.auth.views import (
4. LoginView,
5. LogoutView,
6. PasswordResetView,
7. PasswordResetDoneView,
8. PasswordChangeView,
9. PasswordChangeDoneView,
10. PasswordResetConfirmView,
11. PasswordResetCompleteView
12. )
14. urlpatterns = [
15. url('signup/', views.SignUp.as\_view(), name='signup'),
16. url(r'^password\_change/$',
17. PasswordChangeView.as\_view(template\_name='registration/password\_change.html'),
18. name='password\_change'),
19. url(r'^password\_reset/$',
20. PasswordResetView.as\_view(template\_name='registration/password\_reset\_form.html'),
21. name='password\_reset'),
22. url(r'^password\_reset\_done/$',
23. PasswordResetDoneView.as\_view(template\_name='registration/password\_reset\_done.html'),
24. name='password\_reset\_done'),
25. url(r'reset/<uidb64>/<token>/',
26. PasswordResetConfirmView.as\_view(template\_name='registration/password\_reset\_confirm.html'),
27. name='password\_reset\_confirm'),
28. url(r'^reset/done/$',
29. PasswordResetCompleteView.as\_view(template\_name='registration/password\_reset\_complete.html'),
30. name='password\_reset\_complete'),
31. ]

**Start server, login, open account/change password/reset password/**

Add objects to database dynamically

Update blog/urls.py and add new route

url(r'^write/$', views.WriteView.as\_view(success\_url="/posts/"), name='write')

open blog/views.py and add the following

1. class WriteView(generic.CreateView):
2. template\_name = 'blog/post\_new.html'
3. model = Post
4. fields = ['title', 'author', 'text']
6. def get\_initial(self,):
7. initial = super().get\_initial()
8. initial['author'] = self.request.user.get\_username()
9. return initial

create template blog/post\_new.html and add the following

1. <div class="container">
3. {% if user.is\_authenticated %}
5. {% block content %}
6. <h2>New post</h2>
7. <form method="POST" class="post-form">{% csrf\_token %}
8. {‌{ form.as\_p }}
9. <button type="submit" class="save btn btn-default">Save</button>
10. </form>
11. {% endblock %}
13. {% else %}
14. You are not logged in <a href="{% url 'login' %}">login</a>
15. {% endif %}
16. </div>

Start the server, login, write and add your post

.GITIGNORE FILE

At the root of django\_chapter5, create a file and name it .gitignore.

This will be used to store the virtual environment, to prevent it from been pushed to github.

Open the .gitignore file and add myvenv

\_\_pycache\_\_/

db.sqlite3

testing the database model Post

Because database is involved in our project, we will import TestCase at the top of the file, which will let us create a test database.

All test methods must start with the phrase test so Django knows to test them!

Open the test.py file of the posts app and add the following codes.

**from django.test import** TestCase

**from .models import** Post

**class PostTests**(TestCase):

@classmethod

**def** setUpTestData(cls):

cls.post = Post.objects.create(text="This is a test!")

**def** test\_model\_content(self):

self.assertEqual(self.post.text, "This is a test!")

Stop the development server with ctrl + c

Use the following command to test python manage.py test

It is time to check our URLs, views, and templates. We want to check the following four things

for our message board page:

• URL exists at / and returns a 200 HTTP status code

• URL is available by its name of “home”

• Correct template is used called “post\_list.html”

• Homepage content matches what we expect in the database

Since this project has only one webpage, we can include all of these tests in our existing

PostTests class. Make sure to import reverse at the top of the page and add the four tests

as follows:

**from django.test import** TestCase

**from django.urls import** reverse

**from .models import** Post

**class PostTests**(TestCase):

@classmethod

**def** setUpTestData(cls):

cls.post = Post.objects.create(text="This is a test!")

**def** test\_model\_content(self):

self.assertEqual(self.post.text, "This is a test!")

**def** test\_url\_exists\_at\_correct\_location(self):

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

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

**def** test\_url\_available\_by\_name(self):

response = self.client.get(reverse("home"))

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

**def** test\_template\_name\_correct(self):

response = self.client.get(reverse("home"))

self.assertTemplateUsed(response, "post\_list.html")

**def** test\_template\_content(self):

response = self.client.get(reverse("home"))

self.assertContains(response, "This is a test!")

**def** test\_homepage(self):

response = self.client.get(reverse("home"))

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

self.assertTemplateUsed(response, "post\_list.html")

self.assertContains(response, "This is a test!")

Storing the installed packages of the virtual environment

Record of installed packages inside virtual environment

With virtual environment activated at the root of django\_chapter5, use the following command to store the packages inside a requiremens.txt file.

pip freeze > requirement.txt

pushing to github account

stop the development server with ctrl + c and at the root of django\_chapter5 directory, deactivate the virtual environment with the following command: deactivate

go to github account and create a repository named django\_chapter5

At the root of django\_chapter5 directory, use the following command to initialize git and push the code to github

git init

git add -A

git commit -m “messageboard”

git branch -M main

git remote add origin https://github.com/trevor4ivan/django\_chapter5.git

git push -u origin main