**message\_board\_website 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 pip install virtualenv

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 ofdjango\_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 messageboard\_website\_project .

The period after the messageboard\_website\_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 messagebard\_website\_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 posts.

Add the posts app to the list of installed apps inside the messageboard\_website\_project directory settings

Create a database model inside the models.py file of the posts app & name is Post

We will have only one table called “Post” and a single field, “text,” containing the contents of a message.

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 posts/models.py file and write the following codes:

**from django.db import** models

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

text = models.TextField()

We’ve created a new database model called Post, which has the database field called text. We’ve also specified the *type of content* it will hold, TextField().

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 posts and python manage.py migrate

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 posts apps with the Post model

To use the Django admin, 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.

In your text editor, open up posts/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 posts 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 inside the admin dashboard

Let’s create our first message board 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 posts/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 posts app

**from django.db import** models

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

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 to work with in the next section. You can use the “Add Post +” button in the upper right corner

Create a function-based view

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

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

**from django.shortcuts import** render

**from .models import** Post

**def** post\_list(request):

posts = Post.objects.all()

**return** render(request, "post\_list.html", {"posts": posts})

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

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

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

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

Open the post\_list.html file and update the file with the following codes:

<**h1**>Message Board Homepage</**h1**>

<**ul**>

{% for post in posts %}

<**li**>{{ post.text }}</**li**>

{% endfor %}

</**ul**>

Create a urls.py file inside the posts app

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

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

**from django.urls import** path

**from .views import** post\_list

urlpatterns = [

path("", post\_list, name="post\_list"),

]

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

There is a built-in urls.py file inside the root directory messageboard\_website\_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("posts.urls")),

]

Start the development server with python manage.py runserver.

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

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

**from django.views.generic import** ListView

**from .models import** Post

**class PostList**(ListView):

model = Post

template\_name = "post\_list.html"

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

By default, ListView returns a context variable called <model>\_list, where <model> is our model name. Since our model is named post, we need to loop over post\_list with our for loop.

Note: our model name first letter will be in lowercase post

<**h1**>Message board homepage</**h1**>

<**ul**>

{% for post in post\_list %}

<**li**>{{ post.text }}</**li**>

{% endfor %}

</**ul**>

Create a urls.py file inside the posts app

**from django.urls import** path

**from .views import** PostList

urlpatterns = [

path("", PostList.as\_view(), name="home"),

]

.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