Django- web framework

Django is a framework built with python

1. Create virtual environment.

To create the virtual environment, you need to install a package called virtualenv. Open a powershell terminal on your computer and type the code below to install the virtual environment package if you don't already have it installed.

pip install virtualenv

Once, you have this package installed, you can proceed to create a virtual environment for the project you are about to start.

To create an environment, type the command below into your terminal.

virtualenv venv

Where venv\_ is the name you are giving to your virtual environment and can be changed to anything of your choice. It's a good practice to have env as part of the name so you always know what and why you have that directory when created.

You will now have a new directory created called venv and have a few files and directories in it.

1. Activate virtual environment

Once the virtual environment is created, you will need to activate it. There is a file inside the directory that was created that you will use to activate. To do that, run the command below in a bash terminal:

source venv/bin/activate

Immediately after the activation, you will find that your default prompt on your terminal has a new portion at the beginning. The name of your virtual environment will be put in parenthesis and placed in front of your terminal's prompt.

(venv) $:

Having (venv) indicates that your virtual environment is active.

Make sure that it is active anytime that you are working on your project. Anytime that you find it not to be active, you just have to run the same code above to activate it.

Also, any point in time, you can deactivate the virtual environment by just running the deactivate command.

(venv) $: deactivate

1. Install Django

Now that we have our virtual environment set up and active, we can go ahead and install the Django framework. To install Django, run the command below: (venv) $: pip install django

This installs the latest stable Django version available.

Once you have Django installed, you get access to the django-admin tool that enables you to perform some important tasks like creating a new Django project.

Run the command below to check the list of packages installed.

(venv) $: pip list

Remember to deactivate the virtual environment with

(venv) $: deactivate

1. Create/Start a Django project.

After installing Django, you need to start a new Django project and you can do that by running the command below, remember to activate the virtual environment and insert period at the end of myblog (meaning create all the files inside the myblog directory).

(venv) $: django-admin startproject myblog .

Where myblog is the name of the project you are creating. This action will create a new directory called myblog. Inside this directory, you will find a file called manage.py. That file is very important and will be used for running various administrative tasks like initializing databases, creating databases, creating a superuser etc.

Other files inside myblog directory include; \_\_init.\_\_.py, settings.py, urls.py, wsgi.py, asgi.py and \_\_pycache\_\_

Use the command below to check

(venv) $: django\_class/myblog $: ls

1. Migrate your database.

Whenever you create a new Django project like we have just done, you automatically have an admin dashboard set up for you. However, we need to set up our database for it to work.

By default, Django is configured to use the sqlite database. You can always change this to any type of database that you want but for now let's keep using what's already available.

Since a site has already been created for us with it respective database details. We will need to put that database in order. We do that by migrating the database.

To migrate the database, we are going to make use of the manage.py. To use this file, you write a command starting with python followed by `[mange.py](http://mange.py/) and then the name of the actual command you want to execute.

So, let's open the new directory that was created for us before running those commands.

(venv) $: cd myblog

Once inside that directory, we can go ahead and run the manage.py commands.

The command for migrating is as follow:

(venv) $: django\_class/myblog $: python manage.py migrate

Right after successfully running that script, you will now find an additional file db.sqlite3 in your project directory. That happens to be your database file.

1. Create a superuser.

Now, that the database is set up, we need to create a superuser in order to access the admin site. We do that also with the manage.py commands. The command for creating a super user is as below:

(venv) $: django\_class/myblog $: python manage.py createsuperuser

You will then be prompted to type the details for the super user. These details include:

Username:

Email address:

Password:

Password (again):

Once you provide all of the above details, the superuser will be created successfully.

Great work so far. In order to see what we have created so far, we need to have a server up and running.

1. Run the development server.

To start the development server, run the following command.

(venv) $: django\_class/myblog $: python manage.py runserver

This command starts a local development server on port 8080 by default. This means that you can now access the website that you have created with the url https://127.0.0.0:8080/

You will have access to the website for as long as the server is on. As soon as the server is closed, you lose access to the site and you may have to start it again if you want to.

To quit the server, just use the keyboard shortcut, ctrl + c

1. View your website in a browser.

Once the server is up and running, click on the link or copy and paste the link https://127.0.0.0:8080/ into a browser and you will have the website showing as below:

1. Log into the admin dashboard.

I hope you still remember your super user login details that you used? Now you can access the admin dashboard by visiting the admin site and entering your superuser details. You can access the admin site with the site https://127.0.0.0:8080/admin.

After you successfully log in, you will be greeted with a dashboard that looks like the image shown below:

Congratulations! You have successfully set up your first Django website.

Summary of all the commands that were used to set up the project

1. pip install virtualenv

2. virtualenv myblog\_env

3. source myblog\_env/bin/activate

4. pip install django

5. django-admin startproject myblog

6. python manage.py migrate

7. python manage.py createsuperuser

8. python manage.py runserver

Install pylance extension in vscode, it helps with auto completion

A view is a function that takes in an http request and returns an http response.

Urls.py is the Django route handler

Inside the settings.py file under the project folder/directory, add the app folder to the list

Inside the urls.py under the project folder, add the url relative paths to the url patterns

**DJANGO TEMPLATING LANGUAGE**

Django templating language is a powerful tool for generating HTML dynamically in Django applications

Basic Syntax

Django templates use a combination of HTML and template tags. Template tags are enclosed in “{% %}”

and template variables are enclosed in “{{ }}”

Creating a Template

First, you need to create a template file. This file will have a .html extension and live in a special folder called templates. Here’s an example of what a simple template file might look like:

<!DOCTYPE html>

<html>

<head>

<title>My First Django Page</title>

</head>

<body>

<h1>Welcome to My Website!</h1>

<p>Hello, {{ name }}!</p>

</body>

</html>

In this file:

* {{ name }} is a placeholder. It’s like saying, "Put the person's name here." Django will replace {{ name }} with the actual name when it creates the web page.

### **Using Template Variables**

Templates use placeholders called **variables** to display dynamic content. Variables are wrapped in double curly braces, like {{ variable\_name }}. When Django processes the template, it replaces these placeholders with real data.

Using Template Variables

Templates use placeholders called **variables** to display dynamic content. Variables are wrapped in double curly braces, like {{ variable\_name }}. When Django processes the template, it replaces these placeholders with real data.

### **Template Tags**

Template tags are like special instructions in your recipe. They do things like loops and conditions. For example:

* **Loops**: If you want to list multiple items, you use a loop.

<ul>

{% for item in items %}

<li>{{ item }}</li>

{% endfor %}

</ul>

 Here, {% for item in items %} is a tag that loops through a list called items and displays each item.

 **Conditions**: If you want to show different things based on some condition, you use if statements.

{% if user.is\_authenticated %}

<p>Hello, {{ user.username }}!</p>

{% else %}

<p>Please log in.</p>

{% endif %}

* This will show a personalized message if the user is logged in, or a prompt to log in if not.

Filters

Filters modify the data that is displayed. They come after the variable and are separated by a pipe |. For example:

<p>The date is {{ my\_date|date:"F j, Y" }}</p>

This will format the date to look like "August 7, 2024" instead of just showing the raw date

Extending Templates

Sometimes you want to use a common layout for multiple pages, like having a header and footer on every page. You can create a base template and extend it in other templates:

**base.html**

<!DOCTYPE html>

<html>

<head>

<title>{% block title %}My Site{% endblock %}</title>

</head>

<body>

<header>

<h1>My Awesome Site</h1>

</header>

<main>

{% block content %}

{% endblock %}

</main>

<footer>

<p>&copy; 2024 My Site</p>

</footer>

</body>

</html>

**home.html**:

{% extends "base.html" %}

{% block title %}Home{% endblock %}

{% block content %}

<h2>Welcome to the Home Page!</h2>

<p>Enjoy your stay.</p>

{% endblock %}

**Including Other Templates**

If you have parts of a template that you want to reuse, you can include them using {% include "file\_name.html" %}.

**header.html**:

<header>

<h1>My Awesome Site</h1>

</header>

<header>

<h1>My Awesome Site</h1>

</header>

Then, in your main template:

{% include "header.html" %}

* **Templates**: Files that define how your web pages look.
* **Variables**: Placeholders for dynamic data (e.g., {{ name }}).
* **Tags**: Instructions for loops and conditions ({% for %}, {% if %}).
* **Filters**: Modify data display (e.g., {{ date|date:"F j, Y" }}).
* **Inheritance**: Reuse common layout ({% extends "base.html" %}).
* **Includes**: Reuse template parts ({% include "header.html" %}).

A model is the description of representation of the data base

1. makemigrations file contains/captures all the new changes made to the project model
2. migrate convert all the data in the makemigrations file and convert them to SQL commands and execute the SQL commands to create a table/database

To create a database, we need to model the database by describing the field, field data type, the records etc, the description of the table is called model

Migrate is the actual building of the database table.

Field types

String = CharField(requires length)

String with lots of content = TextField

Integer =IntegerField

Date= DateField

Field = column title

Records = row content

Models are python classes.

Use the command python manage.py makemigrations to create migration file

Run python manage.py migrate to create the database table

Register the model inside the admin.py