**Software and Environment**

Django Installation

* Check if Python installed > python --version
* Check if pip installed > pip --version
* virtual env:

The virtual environment is to isolate the project

|  |
| --- |
| * Using venv:   + Simple and built into Python.   + Suitable for basic virtual environment management.   + Requires manual activation and management.   + Command: python -m venv path\to\your\venv   + & .\path\to\your\venv\Scripts\activate |
| * Using virtualenvwrapper-win (use virtualenvwrapper without the –win for Mac and Linux) :   + Provides a higher level of convenience and functionality.   + Easier to manage multiple environments with simple commands.   + Requires installing the virtualenvwrapper-win package.   + Offers a more organised and streamlined workflow, especially for complex projects or when frequently switching between environments.   + Command: pip install virtualenvwrapper-win   + & mkvirtualenv myenv |

To be able to create virtual environments, install the env wrapper with

- > pip install virtualenvwrapper-win

- project specific (student\_env)

> mkvirtualenv student\_env

- Virtual envs are activated when you create them. If you want to activate an environment from previous creation, use command: > workon student\_env

- To exit a virtual env later, use command: > deactivate

- Now we want to install our django web framework inside our virtual environment/

> pip install django pip install django==5.0.6 for a specific version of django

- To verify the installation

> django-admin –version

***!!! Now we are finally ready to start our project.***

Start the Project

Now let’s take a step-by-step approach to create a Django project named student\_portal and an application named student\_app. We'll do this in stages, ensuring we can test each part using the development server.

**Stage 1: Project and Application Initialisation**

1.1 Create the Project

Open your terminal and create a new Django project named student\_portal.

* Make sure that you are inside the correct project env, ie. student\_env
* Move to the folder where you want your project files to be created in.
* Start the project initialisation with the below:

> django-admin startproject student\_portal

1.2 Navigate to the Project Directory

> cd student\_portal

* Open VS Code and add the project folder to your workspace. Let's investigate the folders:

\_\_init\_\_.py Just as with modules, this empty file indicates that this is not an ordinary folder, but the folder where this file is in is a Python Package folder.

asgi.py Used to configure the ASGI (Asynchronous Server Gateway Interface) server settings for the project. ASGI is the successor to WSGI (Web Server Gateway Interface) and is designed to handle asynchronous web requests. Asynchronous web requests allow a web server to handle multiple requests concurrently, rather than processing them one at a time in a sequential manner.

settings.py settings for your application environment  
DEBUG = True -> Here we only want true while we are developing.   
Set to False once the project goes live.

urls.py List route URLs to views.  
Maps the url of a page to the function that the url will be performing

wsgi.py It exposes the wsgi (web server gateway interface) callable as a module-level variable   
 named 'application'. In other words, wsgi defines how web servers communicate with the   
 web application.

manage.py This is used to interact with the project. Warning: Do not touch.

* To test if our installation was successful, Django made a development server available and we can access this server by *navigating to the project folder in the command prompt*.
* Now enter > python manage.py runserver
* *Enter localhost:8000 into your internet browser* window and the site should appear if the installation was successful.
* Ctrl + C to stop the server

1.3 Create the Application

Let’s create an application named courses. A Django **app** is a modular component of the project.

>python manage.py startapp courses

* The folder structure should now have added some files:

│── courses/ # The "courses" app

│ │── migrations/ # Database migrations

│ │── \_\_init\_\_.py # Every package folder gets an \_\_init\_\_.py

│ │── admin.py # Admin panel settings

│ │── apps.py

│ │── models.py # Database models

│ │── views.py # Logic for web pages

│ │── urls.py # Routes for this app

│ │── templates/ # HTML templates (to be created)

│ │── static/ # Static files (CSS, JS, images)

1.4 Register the Application

Open student\_portal/settings.py and add courses to the INSTALLED\_APPS list.

# student\_portal/settings.py

INSTALLED\_APPS = [

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

# register the courses app

    'courses',

]

**Stage 2: Basic URL Configuration and View**

2.1 Configure URLs

Create an urls.py file inside the courses directory with VS Code

create courses/urls.py

2.2 Create a View that Renders a Template

Django views return responses to users.

Open courses/views.py and create a function to render the home page:

# courses/views.py

from django.shortcuts import render

def home(request):

return render(request, 'courses/home.html')

2.3 Create a View That Sends Data to a Template

Now, let's create a view that passes data to the courses list template.

# courses/views.py

def courses\_list(request):

courses = ["Maths", "Science", "History", "Computer Science"]

return render(request, 'courses/courses\_list.html', {'courses': courses})

2.4 Link Views to URLs

Open courses/urls.py and add the following code:

# courses/urls.py

from django.urls import path

from . import views

urlpatterns = [

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

path('courses/', views.courses\_list, name='courses\_list'),

]

2.5 Include these app URLs in the Project URLs

Open student\_portal/urls.py and include the courses URLs.

This will copy the url pattens that are placed in the courses urls.py

from django.contrib import admin

from django.urls import path, include

urlpatterns = [

    path('admin/', admin.site.urls),

    path('', include('courses.urls')), # Connect 'courses' app URLs

]

**Stage 3: Create Templates**

3.1 Create a Base Template

Django uses template inheritance to keep layouts consistent.

Create a templates folder -> create courses/templates/courses folder

* mkdir -p courses/templates/courses

3.2 Create Template Base.html

Inside courses/templates/courses/, create a file called base.html:

Base.html acts as a parent template and all parameters mentioned in child templates will overwrite the parameter values of the inherited template, ie. base.html

<!-- templates/courses/base.html -->

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>{% block title %}Student Portal{% endblock %}</title>

<style>

body { font-family: Arial, sans-serif; margin: 40px; }

.container { max-width: 800px; margin: auto; }

</style>

</head>

<body>

<div class="container">

<h1>Student Portal</h1>

{% block content %}{% endblock %}

</div>

</body>

</html>

3.2 Create Template home.html - inheriting from Base.html

Now, let's create a home page that extends base.html.

Let’s create courses/templates/courses/home.html:

<!-- templates/courses/home.html -->

{% extends "courses/base.html" %}

{% block title %}Home - Student Portal{% endblock %}

{% block content %}

<h2>Welcome to the Student Portal</h2>

<p>Explore your courses and learning materials.</p>

{% endblock %}

3.3 Create a Template courses\_list.html - that receives data from a View

We'll create a courses page that displays a list of courses dynamically.

Let’s create courses/templates/courses/courses\_list.html:

<!-- templates/courses/courses\_list.html -->

{% extends "courses/base.html" %}

{% block title %}Courses - Student Portal{% endblock %}

{% block content %}

<h2>Available Courses</h2>

<ul>

{% for course in courses %}

<li>{{ course }}</li>

{% endfor %}

</ul>

<!-- If there are no courses, display a message -->

{% if not courses %}

<p>No courses available at the moment.</p>

{% endif %}

{% endblock %}

**Stage 4: Test and Run the Server**

4.1 First Check for Errors before running

> python manage.py check

NOTE: At this point in time, we have not implemented a database (Model) structure and therefor there is no need to run a migrate command that will normally create our database from our OOP model code in model.py

We will also at a later stage create a superuser for our Django Admin system.

4.2 Run the Development Server

Run the development server.

>python manage.py runserver

Navigate to <http://127.0.0.1:8000/> or localhost:8000, and you should see a list of blog posts.

**Stage 5: Make adjustments**

NOTE: There is no way to get from the home page to the courses\_list page.

We can alter the url in the browser address bar and to know what it is, we can have a look in the courses/urls.py file => the answer is <http://127.0.0.1:8000/courses/>

\*\* TO MAKE THIS MORE CONVENIENT, we can add a link to the homepage.

<!-- templates/courses/home.html -->

{% extends "courses/base.html" %}

{% block title %}Home - Student Portal{% endblock %}

{% block content %}

<h2>Welcome to the Student Portal</h2>

<p>Explore your courses and learning materials.</p>

<p><a href="{% url 'courses\_list' %}">View Available Courses</a></p>

{% endblock %}

### **Django Resolves** {% url 'courses\_list' %}

1. **Django looks for a URL pattern where** name='courses\_list'**.**
2. **It then finds the corresponding path and generates the correct URL.**

\*\* WE CAN ALSO ADD a link to courses\_list to get back to the homepage.

<!-- templates/courses/courses\_list.html -->

{% extends "courses/base.html" %}

{% block title %}Courses - Student Portal{% endblock %}

{% block content %}

<h2>Available Courses</h2>

<ul>

{% for course in courses %}

<li>{{ course }}</li>

{% endfor %}

</ul>

<!-- If there are no courses, display a message -->

{% if not courses %}

<p>No courses available at the moment.</p>

{% endif %}

{% endblock %}

\*\* PROVIDE BLANK COURSES LIST in courses/views.py and see what courses\_list.html displays.