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 (blog_env)

  > mkvirtualenv blog_env
- Virtual envs are activated when you create them. If you want to activate an environment from previous creation, use command: > workon blog_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==5.0.6

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

Start the Project

Below are the step-by-step instructions to create a Django project named blog_project and an application named blog_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 blog_project.

- Make sure that you are inside the correct project env, ie. blog_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 blog_project

1.2 Navigate to the Project Directory

> cd blog_project

- 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.

1.3 Create the Application

Create an application named blog_app.

>python manage.py startapp blog_app

1.4 Register the Application

Open blog_project/settings.py and add blog_app to the INSTALLED_APPS list.

```
# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog_app',
]
```

Stage 2: Basic URL Configuration and View

2.1 Configure URLs

Create a urls.py file inside the blog_app directory with VS Code

create blog_app/urls.py

2.2 Update Project URLs

Open blog_project/urls.py and include the blog_app URLs. This will copy the url pattens that are placed in the blog_app urls.py

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('blog_app.urls')),
]
```

2.3 Create a Basic View

Open blog_app/views.py and add the index view.

```
# blog_app/views.py

from django.http import HttpResponse

# Create your views here.
def index(request):
    return HttpResponse("Hello, World!")
```

2.4 Define Basic URL Patterns

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

```
# blog_app/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
]
```

2.5 Run the Development Server

Run the server to test the basic setup.

>python manage.py runserver

Navigate to http://127.0.0.1:8000/ in your web browser, and you should see "Hello, World!".

Stage 3: Add Models and Migrations

3.1 Define the BlogPost Model

Open blog_app/models.py and define the BlogPost model.

```
#blog_app/models.py

from django.db import models

# Create your models here.
class BlogPost(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    author = models.CharField(max_length=30)

    # Below we want to create a display and this will be helpful when we
    # look at the data in the admin page
    def __str__(self):
        return self.title
```

3.2 Make Migrations and Migrate

Create and apply migrations to update the database schema.

NOTE: ***** Each time you add or update models, we need to do a migration to update the sqlite database

=> First we want to make this specific to the app

>python manage.py makemigrations blog_app

=> and then we want to make the migration for the project

>python manage.py migrate

NOTE: **** According to our current model and app name, the table name will be blog_app_blogpost

The model/class name will be made lowercase and this will be prefixed with the app name and an underscore.

Stage 4: Create Superuser and Admin Configuration

4.1 Create a Superuser

Create a superuser to access the Django admin.

>python manage.py createsuperuser

Follow the prompts to create your superuser.

User:riaan ; Pw:123

4.2 Register the Model with Admin

Open blog_app/admin.py and register the BlogPost model.

```
# blog_app/admin.py

from django.contrib import admin
from .models import BlogPost

# Register your models here.
admin.site.register(BlogPost)
```

4.3 Run the Server and Access Admin

Run the development server again.

>python manage.py runserver

Navigate to http://127.0.0.1:8000/admin and log in with your superuser credentials.

You should see the BlogPost model listed and be able to add blog posts from the admin interface.

Stage 5: Views and Templates for Blog Posts

5.1 Create Templates Directory

Create a templates directory inside blog_app -> blog_app/templates

Create a directory inside the templates directory that has the same name as the app, ie. blog_app

>mkdir -p blog_app/templates/blog_app or use VS Code to create the folder

5.2 Create Index Template

Create a templates/blog_app /index.html to display blog posts.

NOTE: Double Check if index.html is inside templates/blog_app and not in templates/

```
<!-- blog_app/templates/blog_app/index.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Blog</title>
</head>
<body>
    <h1>Blog Posts</h1>
    <ul>
        {% for post in posts %}
           <li>{{ post.title }} - {{ post.created_at }} by {{ post.author }}</li>
        {% endfor %}
    </ul>
</body>
</html>
```

5.3 Update the Index View

Modify blog_app/views.py to render the template with blog posts.

```
# blog_app/views.py

from django.shortcuts import render
from .models import BlogPost

# Create your views here.
def index(request):
    posts = BlogPost.objects.all()  # Grabbing all the records from the BlogPost table
    return render(request, 'blog_app/index.html', {'posts': posts})
```

5.4 Run the Server and Test

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 6: Add Blog Posts from the Webpage

6.1 Create a Form for Adding Blog Posts

Create a new template /blog_app/add_post.html for adding blog posts.

```
<!-- blog_app/templates/blog_app/add_post.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Add Blog Post</title>
</head>
<body>
    <h1>Add a New Blog Post</h1>
    <form method="post">
        {% csrf_token %}
        <!-- {% csrf_token %} to generate a Cross-Site Request Forgery (CSRF) token.
             in the form. Helps protect your web application from CSRF attacks. -->
        <label for="title">Title:</label>
        <input type="text" id="title" name="title"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content"></textarea><br>
        <label for="author">Author:</label>
        <input type="text" id="author" name="author"><br>
        <button type="submit">Add Post</button>
        <a href="{% url 'index' %}">Cancel</a>
    </form>
</body>
</html>
```

6.2 Create the View for Adding Posts

Add a view in blog_app/views.py to render the template for adding a post.

```
# blog_app/views.py

from django.shortcuts import render, redirect
from .models import BlogPost

# Create your views here.
def index(request):
#    return HttpResponse("Hello, World!")
    posts = BlogPost.objects.all()  # Grabbing all the records from the BlogPost table
    return render(request, 'blog_app/index.html', {'posts': posts})

def add_post(request):
    if request.method == 'POST':
        author = request.POST.get('author')
        title = request.POST.get('title')
        content = request.POST.get('content')
        # Below is the ORM to create a new post record in the BlogPost table
        BlogPost.objects.create(author=author, title=title, content=content)
        # When done, return to the index.html page
        return redirect('index')
    return render(request, 'blog_app/add_post.html')
```

6.3 Update URLs

Add a URL pattern for the add_post view.

```
# blog_app/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_post, name='add_post'),
]
```

6.4 Add a button to the index.html to interact with the add_post.html

```
<!-- blog_app/templates/blog_app/index.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Blog</title>
</head>
<body>
    <h1>Blog Posts</h1>
    <ul>
        {% for post in posts %}
           <li>{{ post.title }} - {{ post.created_at }} by {{ post.author }}</li>
        {% endfor %}
    </ul>
    <a href="{% url 'add_post' %}">Add New Post</a>
</body>
</html>
```

6.5 Run the Server and Test

Run the development server.

>python manage.py runserver

Navigate to http://127.0.0.1:8000/add/ and add a new blog post using the form.

After submission, you should be redirected to the index page, and the new post should be listed.

Stage 7: View and Edit Blog Posts from the Webpage

7.1 Create a template for Viewing a Blog Post

Create a new template /blog_app/view_post.html for viewing a specific blog post.

```
<!-- blog_app/templates/blog_app/view_post.html -->

<!DOCTYPE html>
<html>
<head>
    <title>View Post</title>
</head>
<body>
      <h1>{{ post.title }}</h1>
      <p>{{ post.content }}</p>
      <p><strong>Author:</strong> {{ post.author }}</p>
      <p><strong>Published on:</strong> {{ post.created_at }}</p>
      <a href="{% url 'edit_post' post.id %}">Edit Post</a>
      <!-- Add a link back to the list of posts -->
      <a href="{% url 'index' %}">Back to Posts List</a>
</body>
</html>
```

7.2 Create the View for Viewing Posts

Add a view in blog_app/views.py to render the template for viewing a post.

```
# blog_app/views.py

from django.shortcuts import render, redirect, get_object_or_404

from .models import BlogPost

# Create your views here.
def add_post(request):
    if request.method == 'POST':
        author = request.POST.get('author')
        title = request.POST.get('title')
        content = request.POST.get('content')
        # Below is the ORM to create a new post record in the BlogPost table
        BlogPost.objects.create(author=author, title=title, content=content)
        # When done, return to the index.html page
        return redirect('index')
    return render(request, 'blog_app/add_post.html')

def view_post(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    return render(request, 'blog_app/view_post.html', {'post': post})
```

7.3 Update URLs

Add a URL pattern for the view_post view.

```
# blog_app/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_post, name='add_post'),
    path('post/<int:post_id>/', views.view_post, name='view_post'),
]
```

7.4 Create a template for Editing a Blog Post

Create a new template /blog_app/edit_post.html for editing a specific blog post.

```
<!-- blog_app/templates/blog_app/edit_post.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Edit Post</title>
</head>
<body>
<h1>Edit Post</h1>
    <form method="post">
        {% csrf_token %}
        <label for="title">Title:</label>
        <input type="text" id="title" name="title" value="{{ post.title }}"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content">{{ post.content }}</textarea><br>
        <label for="author">Author:</label>
        <input type="text" id="author" name="author" value="{{ post.author }}"><br>
        <button type="submit">Save Changes</button>
        <a href="{% url 'view_post' post.id %}">Cancel</a>
    </form>
</body>
</html>
```

7.5 Create the View for Editing Posts

Add a view in blog_app/views.py to render the template for editing a post.

```
# blog_app/views.py

from django.shortcuts import render, redirect, get_object_or_404

from .models import BlogPost

# Create your views here.
def view_post(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    return render(request, 'blog_app/view_post.html', {'post': post})

def edit_post(request, post_id):
    post = get_object_or_404(BlogPost, id=post_id)
    if request.method == 'POST':
        author = request.POST.get('author')
        title = request.POST.get('title')
        content = request.POST.get('content')
        # Update the BlogPost record
        post.author = author
        post.title = title
        post.content = content
        post.save()
        # Redirect to the view_post page
        return redirect('view_post', post_id=post.id)
    return render(request, 'blog_app/edit_post.html', {'post': post})
```

7.6 Update URLs

Add a URL pattern for the edit_post view.

```
# blog_app/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('', views.index, name='index'),
    path('add/', views.add_post, name='add_post'),
    path('post/<int:post_id>/', views.view_post, name='view_post'),
    path('edit/<int:post_id>/', views.edit_post, name='edit_post'),
]
```

7.7 Add a view link to the index.html posts

```
<!-- blog_app/templates/blog_app/index.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Blog</title>
</head>
<body>
    <h1>Blog Posts</h1>
    <ul>
        {% for post in posts %}
           <li><a href="{% url 'view_post' post.id %}">{{ post.title }}</a> - {{
post.created_at }} by {{ post.author }}</li>
        {% endfor %}
    </ul>
    <a href="{% url 'add_post' %}">Add New Post</a>
</body>
</html>
```

7.8 Run the Server and Test

Run the development server.

>python manage.py runserver

Navigate to localhost:8000 and test it out.

Stage 8: Add a header.html and footer.html for modularisation and to use in multiple web pages.

8.1 Create the Partials Directory

Create a partials directory inside your blog_app/templates/blog_app directory.

Use VS Code for this or command

>mkdir -p blog_app/templates/blog_app/partials

8.2 Create the templates in the partials directory

Create the file header.html

```
<!-- blog_app/templates/blog_app/partials/header.html -->

<div>
    <h1>My Blog</h1>
</div>
```

Create the file footer.html

```
<!-- blog_app/templates/blog_app/partials/footer.html -->

<div>
    <p>&copy; 2024 My Blog</p>
</div>
```

8.3 Create a base.html template in the same folder as the index.html

If the common HTML structure, including the <head> section, is defined in the base.html file, then the individual templates (index.html, add_post.html, etc.) should not include their own <head> sections. Instead, they should extend base.html and define their content within the designated blocks.

```
<!-- blog_app/templates/blog_app/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 %}My Blog{% endblock %}</title>
</head>
<body>
    {% include "blog_app/partials/header.html" %}
    <div>
        {% block content %}{% endblock %}
    </div>
    {% include "blog_app/partials/footer.html" %}
</body>
</html>
```

8.4 Update the index.html

```
<!-- blog_app/templates/blog_app/index.html -->

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

{% block title %}Blog Posts{% endblock %}

{% block content %}
<div>
    <h1>Blog Posts</h1>
    <ul>
        {% for post in posts %}
            <li>
                <a href="{% url 'view_post' post.id %}">{{ post.title }}</a> - {{
post.created_at }} by {{ post.author }}
            </li>
        {% endfor %}
    </ul>
    <a href="{% url 'add_post' %}">Add New Post</a>
</div>
{% endblock %}
```

8.5 Update the add_post.html

```
<!-- blog_app/templates/blog_app/add_post.html -->

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

{% block title %}Add Blog Post{% endblock %}

{% block content %}
<div>
    <h1>Add a New Blog Post</h1>
    <form method="post">
        {% csrf_token %}
        <!-- {% csrf_token %} to generate a Cross-Site Request Forgery (CSRF) token.
             in the form. Helps protect your web application from CSRF attacks. -->
        <label for="author">Author:</label>
        <input type="text" id="author" name="author"><br>
        <label for="title">Title:</label>
        <input type="text" id="title" name="title"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content"></textarea><br>
        <button type="submit">Add Post</button>
        <a href="{% url 'index' %}">Cancel</a>
    </form>
</div>
{% endblock %}
```

8.6 Update the view_post.html

```
<!-- blog_app/templates/blog_app/view_post.html -->

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

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
<div">
    <h1>{{ post.title }}</h1>
    <p>{{ post.content }}</p>
    <p><strong>Author:</strong> {{ post.author }}</p>
    <p><strong>Published on:</strong> {{ post.created_at }}</p>
    <a href="{% url 'edit_post' post.id %}">Edit Post</a>
    <!-- Add a link back to the list of posts -->
    <a href="{% url 'index' %}">Back to Posts List</a>
</div>
{% endblock %}
```

8.7 Update the edit_post.html

```
<!-- blog_app/templates/blog_app/edit_post.html -->

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

{% block title %}Edit Post{% endblock %}

{% block content %}
<div>
    <h1>Edit Post</h1>
    <form method="post">
        {% csrf_token %}
        <label for="author">Author:</label>
        <input type="text" id="author" name="author" value="{{ post.author }}"><br>
        <label for="title">Title:</label>
        <input type="text" id="title" name="title" value="{{ post.title }}"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content">{{ post.content }}</textarea><br>
        <button type="submit">Save Changes</button>
        <a href="{% url 'view_post' post.id %}">Cancel</a>
    </form>
</div>
{% endblock %}
```

8.8 Run the Server and Test

Run the development server.

>python manage.py runserver

Navigate to localhost:8000 and test it out.

Stage 9: If everything is working, it is time to add some styling.

NOTE: Remember that styling is independent of Django and the HTML/CSS styling rules apply here.

Remember inline, internal, external and bootstrap styling options.

9.1 base.html

```
<!-- blog_app/templates/blog_app/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 %}My Blog{% endblock %}</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 0;
            background-color: #f4f4f4;
        }
        .header {
            background-color: #333;
            color: #fff;
            text-align: center;
            padding: 1rem 0;
        }
        .footer {
            background-color: #333;
            color: #fff;
            text-align: left;
            padding: 1rem;
            margin: 2rem 0 0 0; /* Top margin to separate from the content */
        }
        .container {
            width: 80%;
            margin: 2rem auto;
            padding: 1rem;
            background-color: #fff;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        .index-page, .add-post-page, .view-post-page, .edit-post-page {
            margin-bottom: 2rem;
        }
       .add-post-button, .edit-post-button, .back-button {
            display: inline-block;
            margin: 1rem 0;
            padding: 0.5rem 1rem;
            background-color: #007bff;
            color: #fff;
            border-radius: 4px;
```

```
            cursor: pointer;
            text-decoration: none;
        }

        .back-button {
            background-color: #6c757d;
        }
        form label {
            display: block;
            margin-top: 1rem;
        }
        form input, form textarea {
            width: 100%;
            padding: 0.5rem;
            margin-top: 0.5rem;
            border: 1px solid #ccc;
            border-radius: 4px;
            box-sizing: border-box; /* Ensures padding in the total width/height */
        }
        form button {
            margin-top: 1rem 0;
            padding: 0.6rem 1rem;
            background-color: #28a745;
            color: #fff;
            border-radius: 4px;
            cursor: pointer;
        }
    </style>
</head>
<body>
    {% include "blog_app/partials/header.html" %}
    <div class="container">
        {% block content %}{% endblock %}
    </div>
    {% include "blog_app/partials/footer.html" %}
</body>
</html>
```

9.2 header.html

```
<!-- blog_app/templates/blog_app/partials/header.html -->

<div class="header">
    <h1>My Blog</h1>
</div>
```

9.3 footer.html

```
<!-- blog_app/templates/blog_app/partials/footer.html -->

<div class="footer">
    <p>&copy; 2024 My Blog</p>
</div>
```

9.4 index.html

```
<!-- blog_app/templates/blog_app/index.html -->

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

{% block title %}Blog Posts{% endblock %}

{% block content %}
<div class="index-page">
    <h1>Blog Posts</h1>
    <ul>
        {% for post in posts %}
            <li>
                <a href="{% url 'view_post' post.id %}">{{ post.title }}</a> - {{
post.created_at }} by {{ post.author }}
            </li>
        {% endfor %}
    </ul>
    <a class="add-post-button" href="{% url 'add_post' %}">Add New Post</a>
</div>
{% endblock %}
```

9.5 add_post.html

```
<!-- blog_app/templates/blog_app/add_post.html -->

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

{% block title %}Add Blog Post{% endblock %}

{% block content %}
<div class="add-post-page">
    <h1>Add a New Blog Post</h1>
    <form method="post">
        {% csrf_token %}
        <!-- {% csrf_token %} to generate a Cross-Site Request Forgery (CSRF) token.
             in the form. Helps protect your web application from CSRF attacks. -->
        <label for="author">Author:</label>
        <input type="text" id="author" name="author"><br>
        <label for="title">Title:</label>
        <input type="text" id="title" name="title"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content"></textarea><br>
        <button type="submit">Add Post</button>
        <a class="back-button" href="{% url 'index' %}">Cancel</a>
    </form>
</div>
{% endblock %}
```

9.6 view_post.html

```
<!-- blog_app/templates/blog_app/view_post.html -->

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

{% block title %}{{ post.title }}{% endblock %}

{% block content %}
<div class="view-post-page">
    <h1>{{ post.title }}</h1>
    <p>{{ post.content }}</p>
    <p><strong>Author:</strong> {{ post.author }}</p>
    <p><strong>Published on:</strong> {{ post.created_at }}</p>
    <a class="edit-post-button" href="{% url 'edit_post' post.id %}">Edit Post</a>
    <!-- Add a link back to the list of posts -->
    <a class="back-button" href="{% url 'index' %}">Back to Posts List</a>
</div>
{% endblock %}
```

9.7 edit_post.html

```
<!-- blog_app/templates/blog_app/edit_post.html -->

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

{% block title %}Edit Post{% endblock %}

{% block content %}
<div class="edit-post-page">
    <h1>Edit Post</h1>
    <form method="post">
        {% csrf_token %}
        <label for="author">Author:</label>
        <input type="text" id="author" name="author" value="{{ post.author }}"><br>
        <label for="title">Title:</label>
        <input type="text" id="title" name="title" value="{{ post.title }}"><br>
        <label for="content">Content:</label>
        <textarea id="content" name="content">{{ post.content }}</textarea><br>
        <button type="submit">Save Changes</button>
        <a class="back-button" href="{% url 'view_post' post.id %}">Cancel</a>
    </form>
</div>
{% endblock %}
```

9.8 Run the Server and Test

Run the development server.

>python manage.py runserver

Navigate to localhost:8000 and test it out.