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 organized 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

NOTE: In VS Code you can change terminal content to command prompt in '+' dropdown list

- project specific (auth_env)

  > mkvirtualenv auth_env

- Virtual envs are activated when you create them. If you want to activate an environment from previous creation, use command: > workon auth_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 is a complete example for setting up a Django project named auth_project with an app named auth_app. This example includes user authentication, a registration form, login and logout functionality, and an index page with a link to a protected page that requires authentication. Our application also includes messages for successes and errors.

We'll do this in stages, ensuring we can test each part using the development server.

Step 1: Project and Application Initialisation

1.1 Create the Project

Open your terminal and create a new Django project named auth_project.

- Make sure that you are inside the correct project env, ie. auth_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 auth_project

1.2 Navigate to the Project Directory

> cd auth_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 auth_app.

>python manage.py startapp auth_app

1.4 Register the Application

Open auth_project/settings.py and add auth_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',
    'auth_app',
]
```

Step 2: Basic URL Configuration

2.1 Configure URLs

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

create auth_app/urls.py

2.2 Update Project URLs

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

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

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

Step 3: Configuring Base Layout and Templates

3.1 Create the Templates Directory

Create a templates directory inside auth_app -> auth_app/templates

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

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

3.2 Create the Partials Directory

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

Use VS Code for this or command

>mkdir -p auth_app/templates/auth_app/partials

3.3 Create the templates in the partials directory

Create the file header.html

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

<div class="header">
    <nav>
        <a class="menu-button" href="{% url 'index' %}">Home</a>
        {% if user.is_authenticated %}
            <a class="menu-button" href="{% url 'protected' %}">Protected Page</a>
            <a class="menu-button" href="{% url 'logout' %}">Logout</a>
        {% else %}
            <a class="menu-button" href="{% url 'login' %}">Login</a>
            <a class="menu-button" href="{% url 'register' %}">Register</a>
        {% endif %}
    </nav>
</div>
```

Create the file footer.html

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

<div class="footer">
     <p>&copy; 2024 Django Auth</p>
</div>
```

3.4 Create the Base Template that also imports styling from a css static file.

Create auth_app/templates/auth_app/base.html

```
<!-- auth_app/templates/auth_app/base.html -->

<!DOCTYPE html>
<html>
<head>
    <title>Django Authentication</title>
   {% load static %}  <!-- styles.css is a static file -->
    <link rel="stylesheet" type="text/css" href="{% static 'auth_app/styles.css' %}">
    <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, .login-page, .logout-page, .protected-page, .register-page {
            margin-bottom: 2rem;
            color: #333;
            text-align: left;
        }
        .menu-button, .default-button {
            display: inline-block;
            margin: 1rem 0;
            padding: 0.5rem 1rem;
            background-color: #007bff;
            color: #fff;
            text-decoration: none;
            border-radius: 4px;
```

```
        }
        .default-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;
            padding: 0.5rem 1rem;
            background-color: #28a745;
            color: #fff;
            border: none;
            border-radius: 4px;
            cursor: pointer;
        }
    </style>
</head>
<body>
   {% include "auth_app/partials/header.html" %}
   {% if messages %}
   <ul class="messages">
       {% for message in messages %}
           <li{% if message.tags %} class="{{ message.tags }}"{% endif %}>{{ message }}</li>
       {% endfor %}
   </ul>
    {% endif %}
    <div class="container">
        {% block content %}
        {% endblock %}
    </div>
   {% include "auth_app/partials/footer.html" %}
</body>
</html>
```

NOTE: base.html is the parent template and child templates will now inherit from the parent template.

Only the parent template has the full html structure and this will be pulled into each child template.

Create auth_app/static/auth_app/styles.css

```
/* auth_app/static/auth_app/styles.css */

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

.messages {
    list-style-type: none;
    padding: 0;
    margin: 10px 0;
}

.messages li {
    padding: 10px;
    margin-bottom: 10px;
    border-radius: 5px;
}

.messages li.success {
    background-color: #d4edda;
    color: #155724;
}

.messages li.error {
    background-color: #f8d7da;
    color: #721c24;
}

.messages li.info {
    background-color: #d1ecf1;
    color: #0c5460;
}
```

Step 4: Implement User Registration

4.1 Create a Registration Form

Create auth_app/forms.py and add the Register Form Layout.

```
# auth_app/forms.py

from django import forms
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User  # This is a default table for the database

class UserRegisterForm(UserCreationForm):   # RegisterForm is inheriting from UserCreationForm
    email = forms.EmailField()

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']
```

4.2 Create a Registration Template

Create a templates/auth_app /register.html to display the registration page.

```
<!-- auth_app/templates/auth_app/register.html -->

{% extends 'auth_app/base.html' %}  <!-- We inherit all the html from base.html-->

{% block content %}
<div class="register-page">
    <h2>Register</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Register</button>
    </form>
    <p>Already have an account? <a href="{% url 'login' %}">Login</a></p>
</div>
{% endblock %}
```

NOTE: Just like methods in a class, the {% block content %} {% endblock %} in this child template will override the {% block content %} {% endblock %} inherited from the parent template base.html.

4.3 Create the Registration View

Open auth_app/views.py and add the registration view.

```
# auth_app/views.py

from django.shortcuts import render, redirect
from .forms import UserRegisterForm
from django.contrib import messages

# Create your views here.
def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('login')
    else:
        form = UserRegisterForm()
    return render(request, 'auth_app/register.html', {'form': form})
```

4.4 Set Up Registration URL

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

```
# auth_app/urls.py

from django.urls import path
from .views import register    # if this is ‘from . import views’ then below views.register

urlpatterns = [
    path('register/', register, name='register'),
]
```

Step 5: Implement User Login and Logout

5.1 Create a Login and Logout Template

Create a templates/auth_app /login.html

```
<!-- auth_app/templates/auth_app/login.html -->

{% extends 'auth_app/base.html' %}

{% block content %}
<div class="login-page">
    <h2>Login</h2>
    <form method="post">
        {% csrf_token %}
        {{ form.as_p }}
        <button type="submit">Login</button>
    </form>
    <p>Don't have an account? <a href="{% url 'register' %}">Register</a></p>
</div>
{% endblock %}
```

Create a templates/auth_app /logout.html

```
<!-- auth_app/templates/auth_app/logout.html -->

{% extends 'auth_app/base.html' %}

{% block content %}
<div class="logout-page">
    <h2>Logout</h2>
    <p>You have been logged out.</p>
    <a class="default-button" href="{% url 'login' %}">Login Again</a>
</div>
{% endblock %}
```

5.2 Create the Login and Logout Views

Open auth_app/views.py and add the login and logout views.

```
# auth_app/views.py

from django.shortcuts import render, redirect
from .forms import UserRegisterForm
from django.contrib import messages
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, authenticate, logout


def register(request):
    if request.method == 'POST':
        form = UserRegisterForm(request.POST)
        if form.is_valid():
            form.save()
            username = form.cleaned_data.get('username')
            messages.success(request, f'Account created for {username}!')
            return redirect('login')
    else:
        form = UserRegisterForm()
    return render(request, 'auth_app/register.html', {'form': form})

def login_view(request):
    if request.method == 'POST':
        form = AuthenticationForm(request, data=request.POST)
        if form.is_valid():
            username = form.cleaned_data.get('username')
            password = form.cleaned_data.get('password')
            user = authenticate(username=username, password=password)
            if user is not None:
                login(request, user)
                messages.info(request, f'You are now logged in as {username}.')
                return redirect('index')    # Go to Home page after login
            else:
                messages.error(request, 'Invalid username or password.')
        else:
            messages.error(request, 'Invalid username or password.')
    form = AuthenticationForm()
    return render(request, 'auth_app/login.html', {'form': form})

def logout_view(request):
    logout(request)
    messages.info(request, 'You have successfully logged out.')
    return redirect('index')
```

NOTE: The return redirect('index') in the login and logout views, allows for precise control over where the user goes after login and logout. Default settings can also be used in case this redirect is left out from a login or logout action, ie. In settings.py include LOGIN_REDIRECT_URL = 'index' and LOGOUT_REDIRECT_URL = 'index' where index refers to the relevant view name you want to go to.

5.3 Set Up Login and Logout URLs

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

```
# auth_app/urls.py

from django.urls import path
from .views import register, login_view, logout_view
# from . import views

urlpatterns = [
    path('register/', register, name='register'),
    path('login/', login_view, name='login'),
    path('logout/', logout_view, name='logout'),
]
```

5.4 Set Login and Logout Default Redirects

```
# Application definition

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

# Add redirect URLs for login and logout
LOGIN_REDIRECT_URL = 'index'
LOGOUT_REDIRECT_URL = 'index'
```

Step 6: Implement a page for no authentication and a page that needs authentication

6.1 Create an index and a protected template

Create a templates/auth_app /index.html

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

{% extends 'auth_app/base.html' %}

{% block content %}
<div class="index-page">
    <h1>Welcome to the Auth App</h1>
    {% if user.is_authenticated %}
        <p>You are logged in as {{ user.username }}.</p>
        <a class="default-button" href="{% url 'protected' %}">Go to Protected Page</a>
    {% else %}
        <p>You are not logged in.</p>
        <a class="default-button" href="{% url 'login' %}">Login</a>
        <a class="default-button" href="{% url 'register' %}">Register</a>
    {% endif %}
</div>
{% endblock %}
```

Create a templates/auth_app /protected.html

```
<!-- auth_app/templates/auth_app/protected.html -->

{% extends 'auth_app/base.html' %}

{% block content %}
<div class="protected-page">
    <h2>Protected Page</h2>
    <p>You are viewing a protected page.</p>
    <a class="default-button" href="{% url 'index' %}">Return to Home</a>
</div>
{% endblock %}
```

6.2 Create the Index and Protected Views

Open auth_app/views.py and add the index and protected views.

```
# auth_app/views.py

from django.shortcuts import render, redirect
from .forms import UserRegisterForm
from django.contrib import messages
from django.contrib.auth.forms import AuthenticationForm
from django.contrib.auth import login, authenticate, logout


def logout_view(request):
    logout(request)
    messages.info(request, 'You have successfully logged out.')
    return redirect('index')

def index(request):     # For each page that DOES NOT need authentication
    return render(request, 'auth_app/index.html')

def protected_view(request):   # For each page that DOES need authentication
    if not request.user.is_authenticated:
        return redirect('login')
    return render(request, 'auth_app/protected.html')
```

6.3 Set Up Index and Protected URLs

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

```
# auth_app/urls.py

from django.urls import path
from .views import register, login_view, logout_view, index, protected_view

# from . import views

urlpatterns = [
    path('register/', register, name='register'),
    path('login/', login_view, name='login'),
    path('logout/', logout_view, name='logout'),
    path('', index, name='index'),
    path('protected/', protected_view, name='protected'),
]
```

Step 7: Migrate the Database and Create a Superuser

Makemigrations do not have any migration files to create since no custom Models were created. Makemigrations do however do some compilation as well and might point out some initial errors that could be of assistance. To test this, in forms.py, change

```
class UserRegisterForm(UserCreationForm):  => class RegisterForm(UserCreationForm):
```

and see what makemigrations will show.

>python manage.py makemigrations auth_app

>python manage.py migrate

>python manage.py createsuperuser

Step 8: 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 test it out.