Blog Application:

Create a folder and open it with VS Code

```
PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> python -m venv env
PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> env/Scripts/Activate.ps1
(env) PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> pip install django
```

```
(env) PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> django-admin startproject django_project .
(env) PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> python manage.py startapp blog
(env) PS C:\Users\tasawerk\OneDrive - Whitecliffe College\SSDW\Week 2\session 2> python manage.py migrate
```

To ensure Django knows about our new app, open your text editor, and add the new app to INSTALLED_APPS in the django_project/settings.py file:

```
# django_project/settings.py
INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "blog.apps.BlogConfig",  # new
]
```

Open the **blog/models.py** file and enter the code below:

```python
# blog/models.py
from django.db import models
from django.urls import reverse


class Post(models.Model):
    title = models.CharField(max_length=200)
    author = models.ForeignKey(
        "auth.User",
        on_delete=models.CASCADE,
    )
    body = models.TextField()

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse("post_detail", kwargs={"pk": self.pk})
```

Get_absolute_url:
https://docs.djangoproject.com/en/4.0/ref/models/instances/#django.db.models.Model.get_absolute_url

Stop the server with **Control+C.** Two-step process to update the database can be completed with the commands below:

```
(.venv) > python manage.py makemigrations blog
(.venv) > python manage.py migrate
```

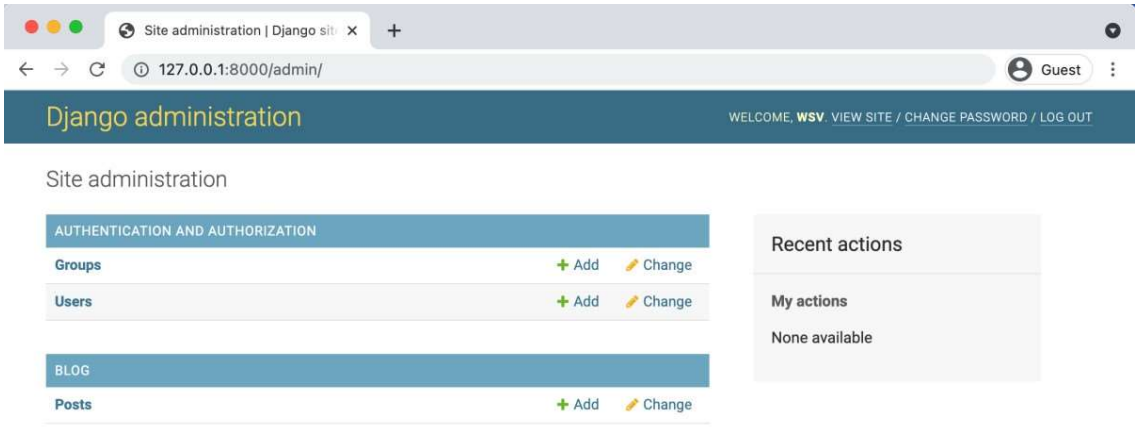
Create super user:

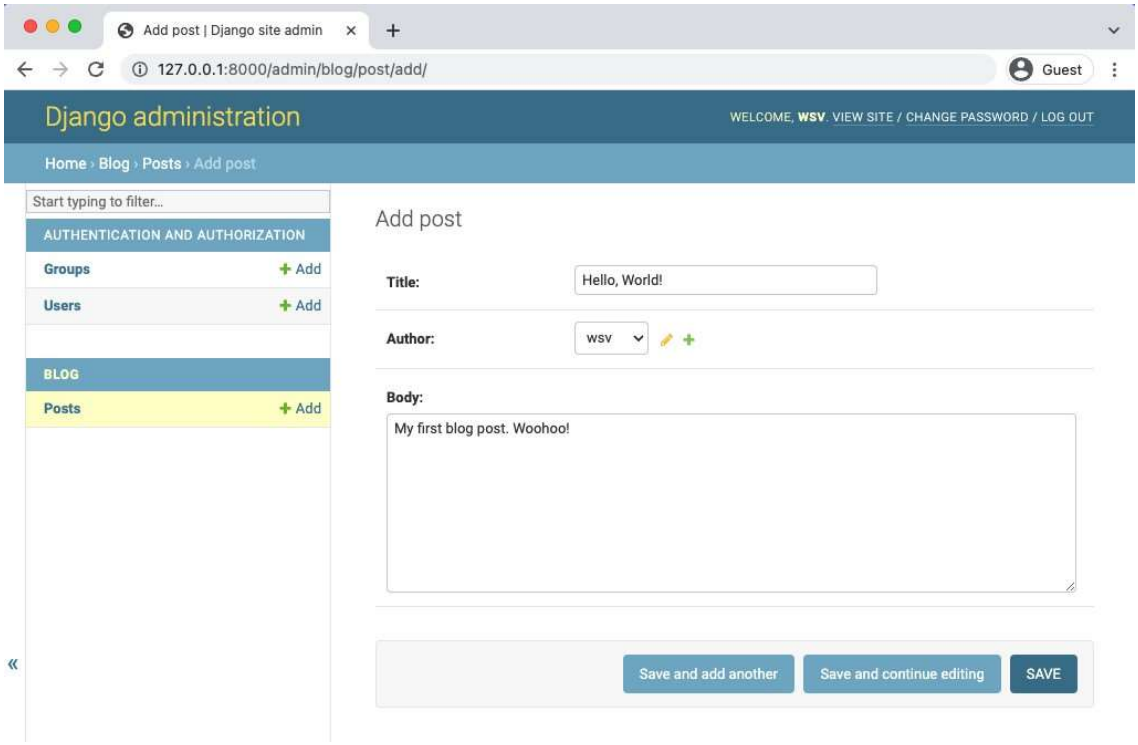
```
(.venv) > python manage.py createsuperuser
Username (leave blank to use 'wsv'): wsv
Email:
Password:
Password (again):
Superuser created successfully.
```

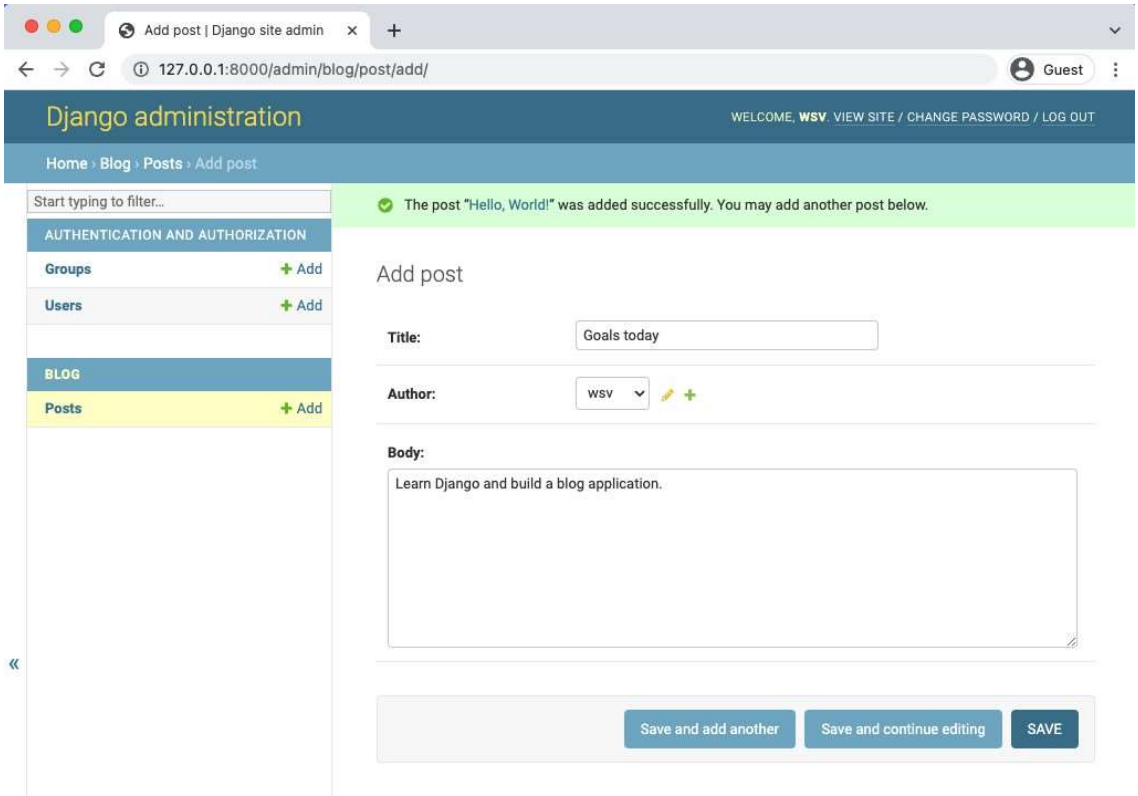
update blog/admin.py

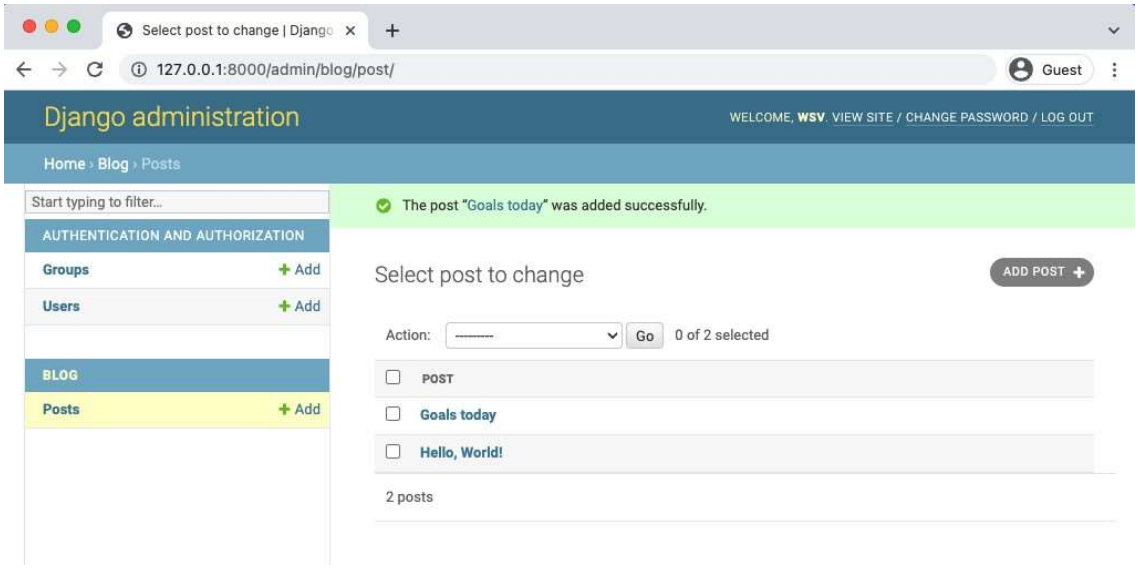
```
# blog/admin.py
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

If you launch the server, and login as admin/super user, you will see:




Add post | Django site admin
127.0.0.1:8000/admin/blog/post/add/
Guest
Django administration
WELCOME, WSV. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Blog › Posts › Add post
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BLOG
Posts + Add
Add post
Title: Hello, World!
Author: wsv
Body:
My first blog post. Woohoo!
Save and add another
Save and continue editing
SAVE


Add post | Django site admin
127.0.0.1:8000/admin/blog/post/add/
Guest
Django administration
WELCOME, WSV. VIEW SITE / CHANGE PASSWORD / LOG OUT
Home › Blog › Posts › Add post
Start typing to filter...
AUTHENTICATION AND AUTHORIZATION
Groups + Add
Users + Add
BLOG
Posts + Add
The post "Hello, World!" was added successfully. You may add another post below.
Add post
Title: Goals today
Author: wsv
Body:
Learn Django and build a blog application.
Save and add another
Save and continue editing
SAVE



Build ‘urls’

```python
# blog/urls.py
from django.urls import path
from .views import BlogListView

urlpatterns = [
    path("", BlogListView.as_view(), name="home"),
]
```

```python
# django_project/urls.py
from django.contrib import admin
from django.urls import path, include  # new

urlpatterns = [
    path("admin/", admin.site.urls),
    path("", include("blog.urls")),  # new
]
```

Build views:

```
# blog/views.py
from django.views.generic import ListView
from .models import Post


class BlogListView(ListView):
    model = Post
    template_name = "blog/home.html"
```

Base Template

A base template in Django is a master HTML template that defines the overall structure and common elements (like headers, footers, and navigation) for your website. Other templates extend or inherit from the base template, allowing you to create consistent design and layout across multiple pages. This approach improves code reusability, simplifies maintenance, and ensures a uniform look and feel throughout your site.

As we are going to use this as base for whole project, create a directory named templates in the base directory of the project, same directory where manage.py is. Inside the template's directory, create a new HTML file named base.html. This will be your base template.

Edit the base.html file and define the basic structure that you want to use across all pages. You can include the common HTML structure, CSS links, navigation bar, etc.

```
<!-- templates/base.html -->
<html>
  <head>
    <title>Django blog</title>
  </head>
  <body>
    <header>
      <h1><a href="{% url 'home' %}">Django blog</a></h1>
    </header>
    <div>
      {% block content %}
      {% endblock content %}
    </div>
  </body>
</html>
```

You will need to add this directory in settings to make sure Django know where to find templates. You can do that by making following change in settings.py file

```
# django_project/settings.py
TEMPLATES = [
    {
        ...
        "DIRS": [BASE_DIR / "templates"],  # new
        ...
    },
]
```

App Templates

Build app level templates folder and then a folder named blog (same as app name) inside it. Create 'home.html' inside:

```
{% extends "base.html" %}

{% block content %}
{% for post in post_list %}
  <div class="post-entry">
    <h2><a href="">{{ post.title }}</a></h2>
    <p>{{ post.body }}</p>
  </div>
{% endfor %}
{% endblock content %}
```

Adding static folders

Static files in Django are files like CSS, JavaScript, images, and other assets that are used to style and enhance the presentation of your website. Unlike dynamic content generated by views and templates, static files remain constant and are directly served to the user's browser. Django provides a way to manage and serve these static files efficiently. Let's add static files to the project

Create project 'static' folder in the same folder where manage.py file is. It is the base directory of the project. Then add following settings at the end of settings.py file.

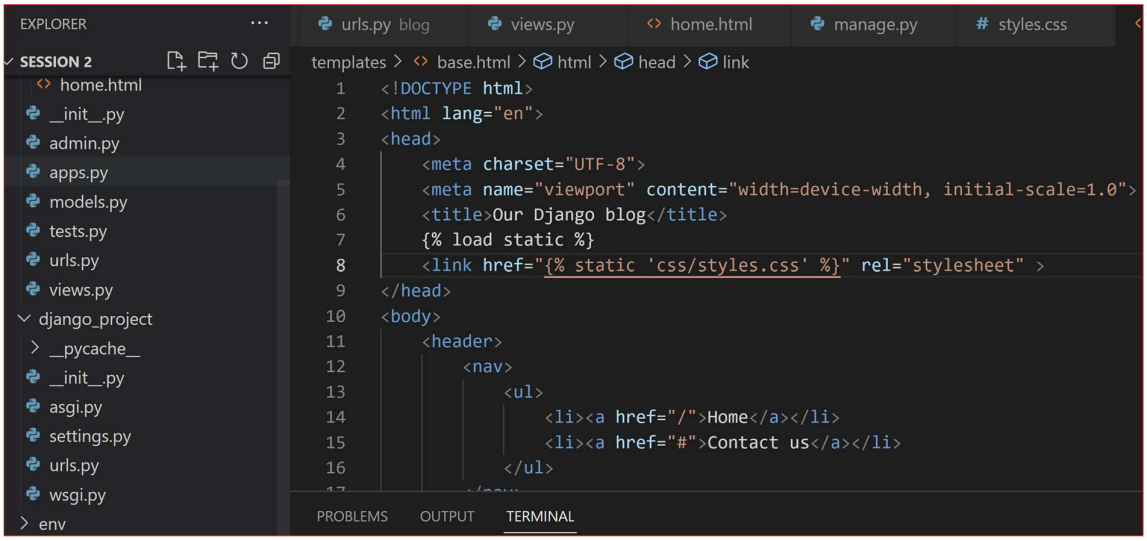
```
# django_project/settings.py
STATIC_URL = " static/"
STATICFILES_DIRS = [BASE_DIR / "static"]  # new
```

Create 'css' folder inside 'static' and then create 'styles.css' inside 'css':

static > css > # styles.css > nav a

```css
nav{
    padding: 10px;
}
nav ul{
    display: flex;
    list-style-type: none;
}
nav a{
    padding: 10px;
    padding: 10px;
    color: white;
    background-color: cadetblue;
    margin: 2px;
    border-radius: 12px;
}
```

Add static files to our templates



Build views of the app:

We are using class-based view here with built in generic views. generic views are pre-built views that provide common functionality and patterns for performing tasks like displaying lists of objects, displaying details of an object, creating, updating, and deleting objects. They are designed to save you time and effort by abstracting away repetitive code and following best practices. **ListView** is a generic view that displays a list of objects from a specified model. It's useful for scenarios where you want to show multiple instances of a model, such as a list of blog posts, products, or users. **DetailView** is used to display the details of a single object from a model. It's commonly used for pages that show a complete view of a single item, like a detailed blog post.

```python
# blog/views.py
from django.views.generic import ListView, DetailView  # new
from .models import Post


class BlogListView(ListView):
    model = Post
    template_name = "blog/home.html"


class BlogDetailView(DetailView):  # new
    model = Post
    template_name = "blog/posts_detail.html"
```

Put below code into ‘post_detail.html’

```
{% extends "base.html" %}

{% block content %}
<div class="post-entry">
  <h2>{{ post.title }}</h2>
  <p>{{ post.body }}</p>
</div>
{% endblock content %}
```

Update blog's URLs. Note how the view part is different than what we did in function-based views.

```
# blog/urls.py
from django.urls import path
from .views import BlogListView, BlogDetailView  # new

urlpatterns = [
    path("post/<int:pk>/", BlogDetailView.as_view(), name="post_detail"),  # new
    path("", BlogListView.as_view(), name="home"),
]
```

Congratulations! Now run project, view the blog post list and a single blog post, take screenshots and please share with me.