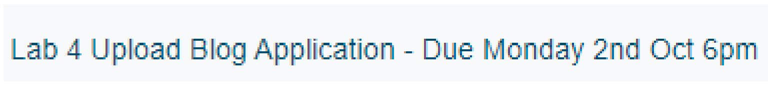
Lab 4 Part (1)

In this exercise we will build a Blog application that allows users to create, edit, and delete posts. The homepage will list all blog posts and there will be a dedicated blog details page for each individual post. We will also introduce CSS for styling and learn how Django works with static files.

Initial Set Up

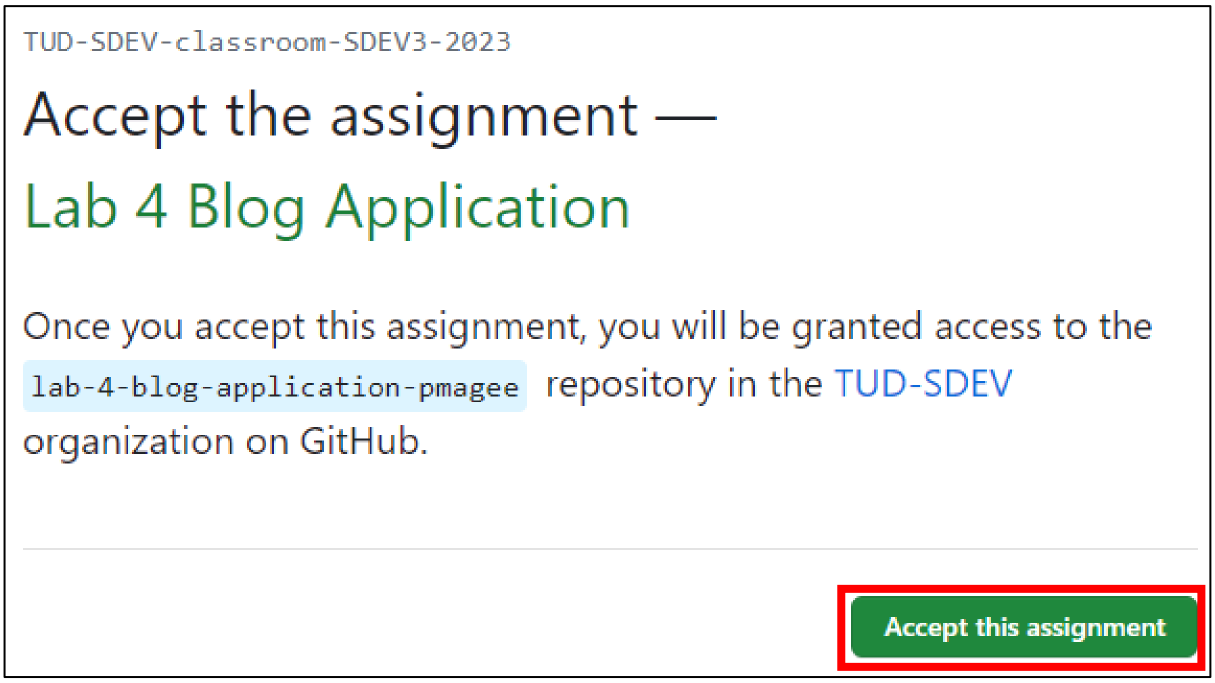
Go to www.github.com and log in to your GitHub account.

Go to Moodle and when you find the link shown below, click on it.



Lab 4 Upload Blog Application - Due Monday 2nd Oct 6pm

Next you should see a screen like the following asking you to accept the assignment. Click the green button to accept the assignment.

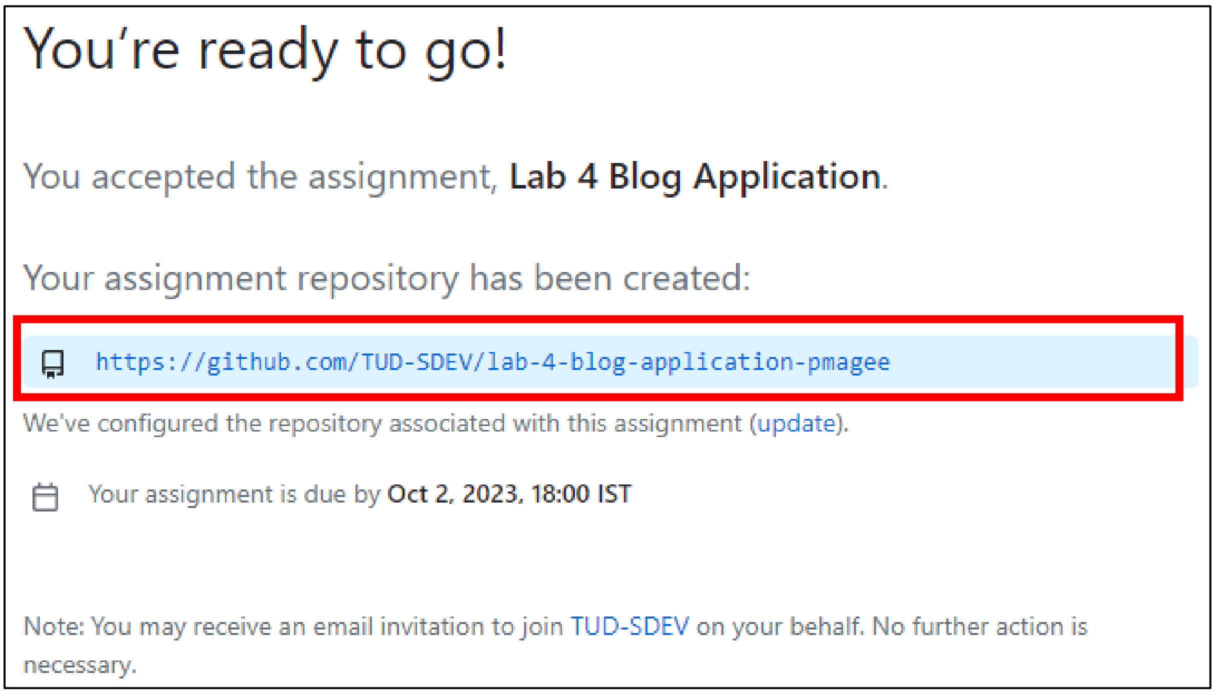


TUD-SDEV-classroom-SDEV3-2023

Accept the assignment — Lab 4 Blog Application

Once you accept this assignment, you will be granted access to the `lab-4-blog-application-pmagee` repository in the TUD-SDEV organization on GitHub.

Once you have accepted the assignment you are asked to refresh the page, and you are then presented with a link to your repository for lab 4 as shown below:


You're ready to go!

You accepted the assignment, **Lab 4 Blog Application**.

Your assignment repository has been created:

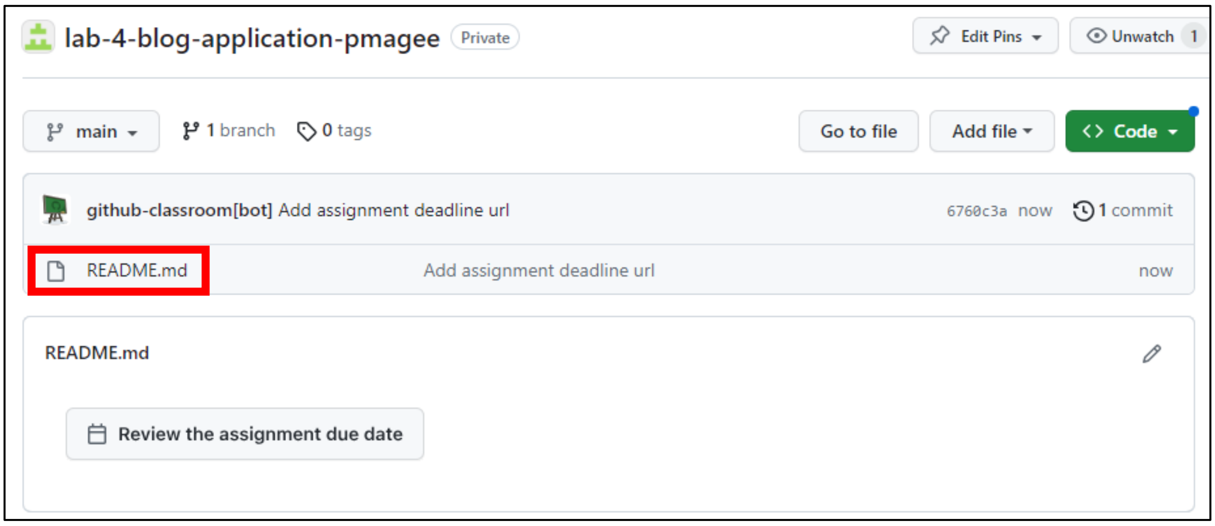
https://github.com/TUD-SDEV/lab-4-blog-application-pmagee

We've configured the repository associated with this assignment (update).

Your assignment is due by **Oct 2, 2023, 18:00 IST**

Note: You may receive an email invitation to join TUD-SDEV on your behalf. No further action is necessary.

When you click on the repository link, you are taken to the repository where you see a readme file has already been added:


lab-4-blog-application-pmagee Private

github-classroom[bot] Add assignment deadline url 6760c3a now 1 commit

README.md Add assignment deadline url now

README.md

Review the assignment due date

In Windows Command line make sure that you are in the **djangoprojects** folder and use the following command to activate the virtual environment:

---

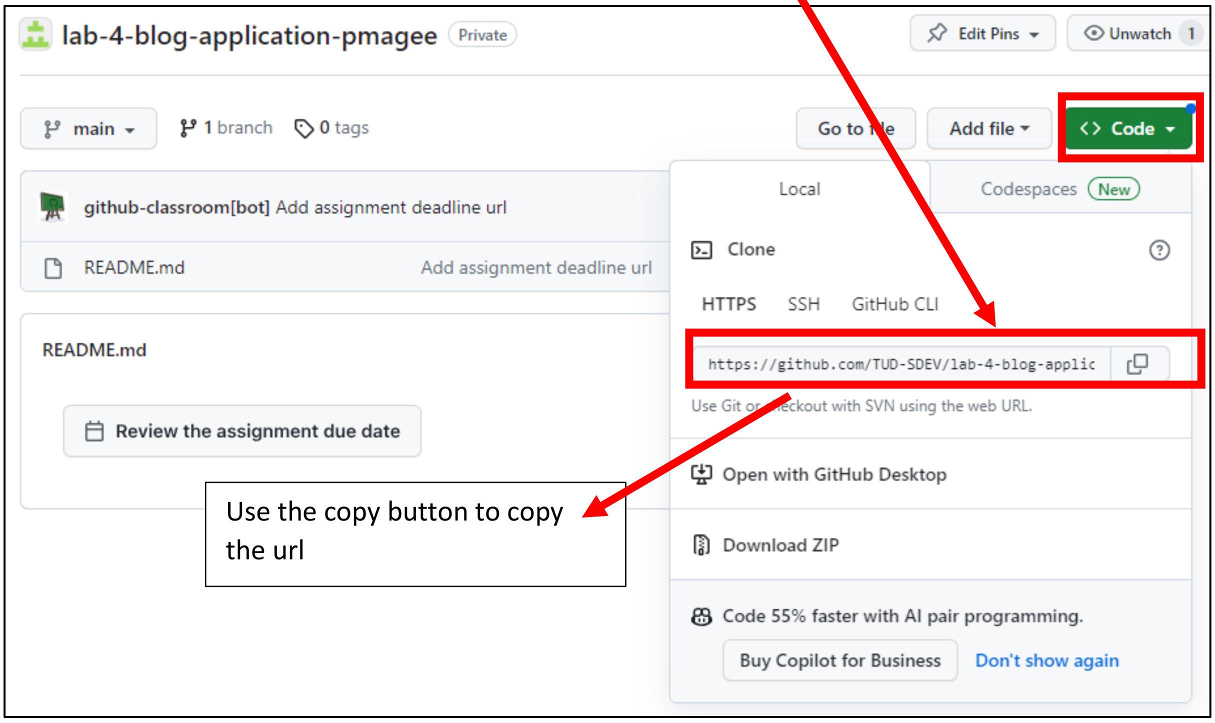
env\scripts\activate.bat

Use the following command to clone this repo to your local computer:

```
git clone <repo name>
```

Replace this part with your repo name.

You can get the repo details by clicking on the Code button as shown below:


lab-4-blog-application-pmagee
Private
Edit Pins
Unwatch 1
main
1 branch
0 tags
Go to file
Add file
Code
github-classroom[bot] Add assignment deadline url
README.md
Add assignment deadline url
README.md
Review the assignment due date
Local
Codespaces New
Clone
HTTPS
SSH
GitHub CLI
https://github.com/TUD-SDEV/lab-4-blog-applic
Use Git or checkout with SVN using the web URL.
Open with GitHub Desktop
Download ZIP
Code 55% faster with AI pair programming.
Buy Copilot for Business
Don't show again
Use the copy button to copy the url

After the clone command is finished you have a new folder called **lab-4-blogapplication-<username>** inside the **djangoprojects** folder.

Move into this new folder using the **cd** command.

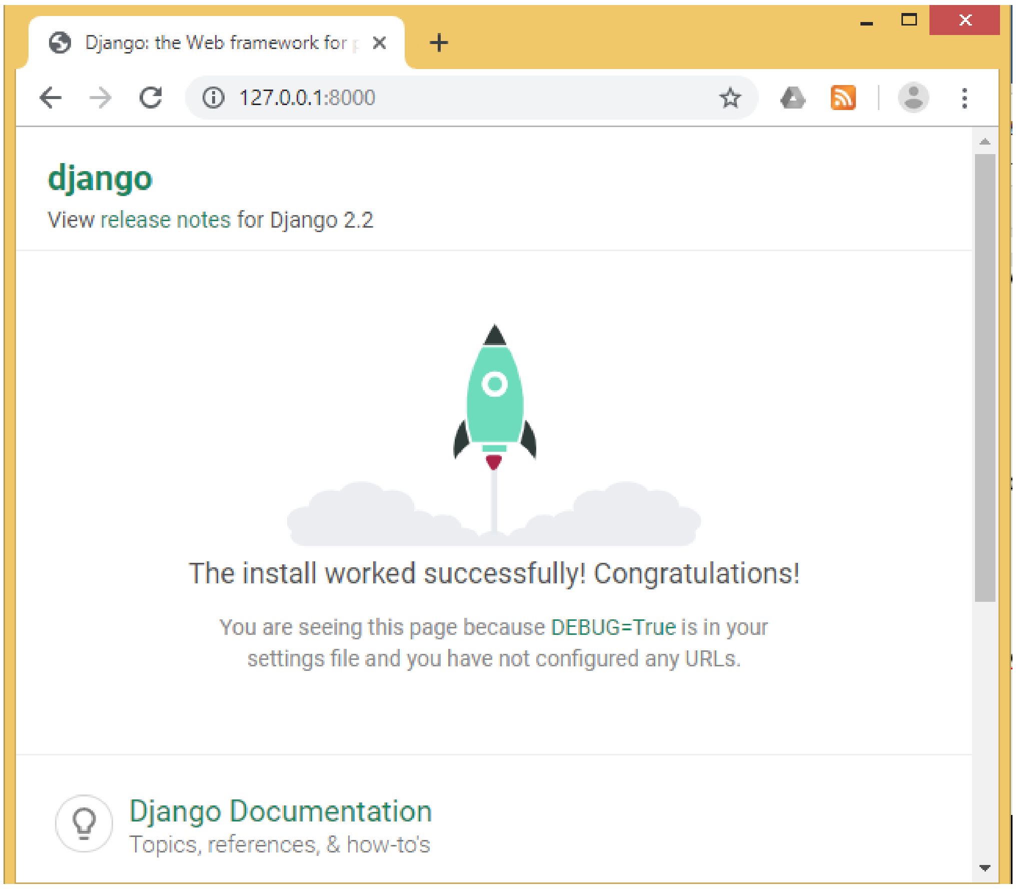
Create a new Django project called **blogproject** with the following command. Don't forget the period (fullstop) at the end:

```
django-admin startproject blogproject .
```

You can verify that the Django project works by typing in the following command:

```
python manage.py runserver 8080
```

If you visit http://127.0.0.1:8080/ you should see the familiar Django welcome page


Django: the Web framework for p
127.0.0.1:8000
django
View release notes for Django 2.2

The install worked successfully! Congratulations!

You are seeing this page because DEBUG=True is in your settings file and you have not configured any URLs.

Django Documentation
Topics, references, & how-to's

The output in the command line shows a warning about “18 unapplied migrations” although this warning has no effect on the project at this point. Django is letting us know that we have not yet “migrated” or configured our initial database. Since we don’t use a database in this exercise, the warning won’t affect the result.

You can remove the warning by running the migrate command as shown here:

```
python manage.py migrate
```

When you run this command, you will see in the output that 18 migrations are applied. We will look at the meaning of these migrations later. If you execute the **python manage.py runserver** again, the warning message is gone.

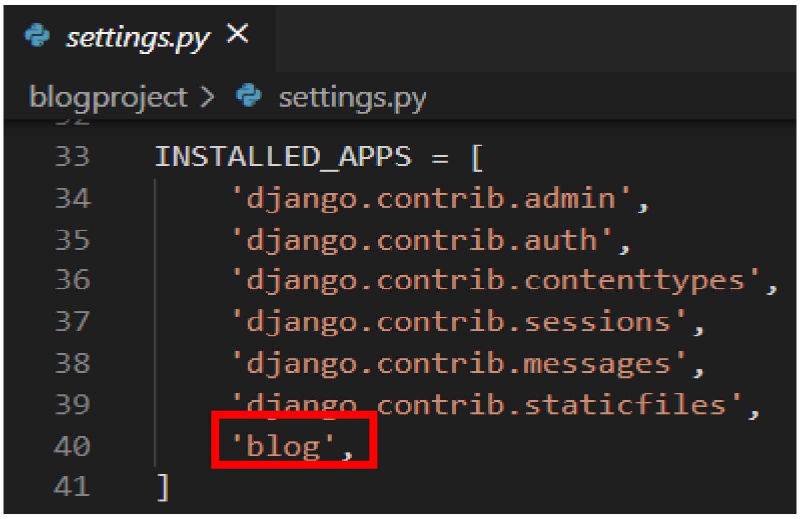
Create an app called **blog**. From the command line, quit the server with **Control+c**. Then use the **startapp** command as shown below:

python manage.py startapp blog

Settings.py

1. To ensure Django knows about our new app, open the project using VS Code and add the new app to **INSTALLED_APPS** in our **settings.py** file:

Code



settings.py ×

blogproject > settings.py

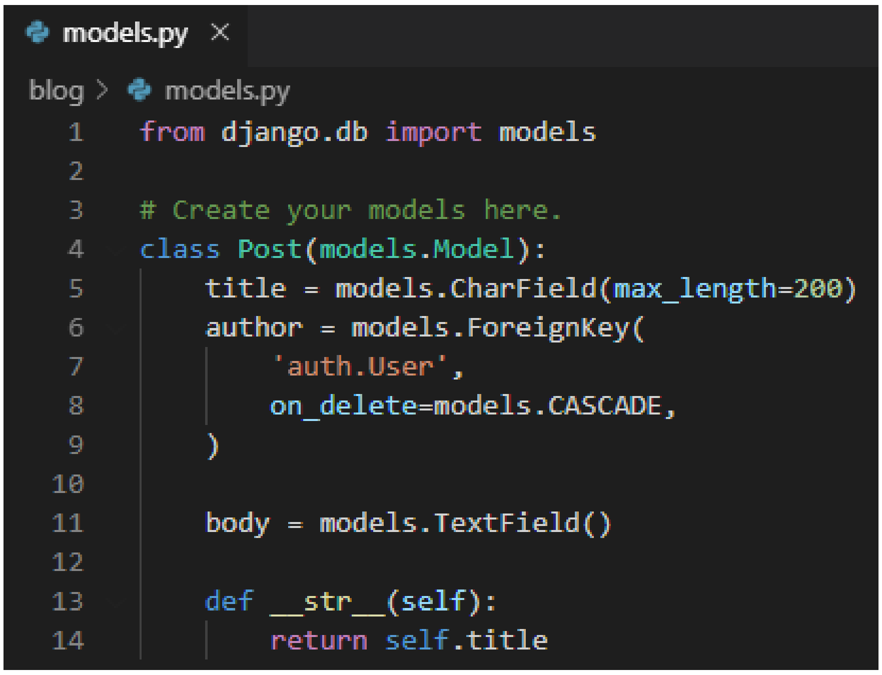
```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
]
```

Database Models

Next, we will create a **Post** class with a title, author, and body.

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

Code



models.py ×

blog > models.py

```
from django.db import models

# Create your models here.
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
```

At the top we import the class models and then create a subclass of models.Model called **Post**. Using this subclass functionality, we automatically have access to everything within django.db.models.Models and can add additional fields and methods as desired.

For title we limit the length to 200 characters and for body we use a TextField which will automatically expand as needed to fit the user's text.

For the author field we use a ForeignKey which allows for a many-to-one relationship. This means that a given user can be the author of many different blog posts but not the other way around. The reference 'auth.User' is to the built-in User model that Django provides for authentication. For all many-to-one relationships such as a ForeignKey we must also specify an on_delete option.

Now that our new database model is created, we need to create a new migration record for it and migrate the change into our database.

When we run the makemigrations command a new file called **0001_initial.py** is created in the **migrations** folder. The **makemigrations** command created the migration, but to make any changes in the database, you must apply it with the management command **migrate.** Stop the server with Control+c and type the two commands shown below:

```
python manage.py makemigrations blog
python manage.py migrate
```

Admin

We can now access our data using Django admin.

1. Create a superuser account by typing the command below and follow the prompts to set up an email and password. Note that when typing your password, it will not appear on the screen for security reasons.

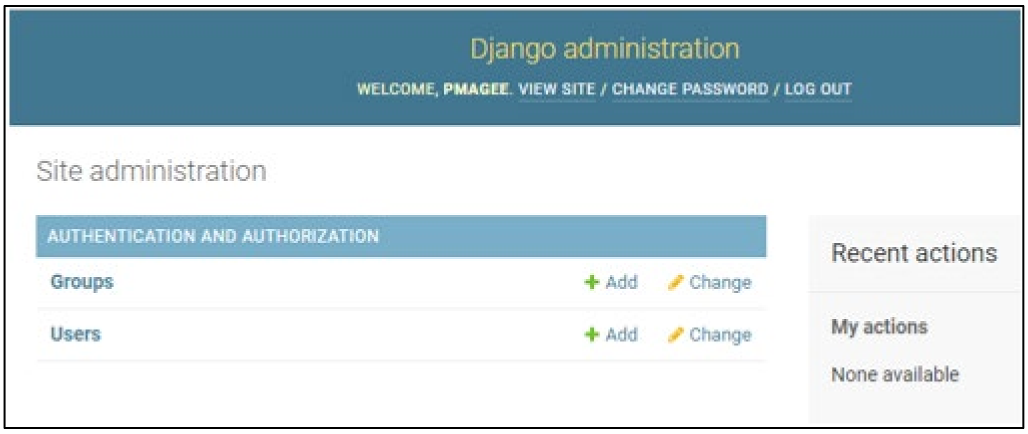
python manage.py createsuperuser

---

2. Start running the Django server again with the command **python manage.py runserver** and open the Django admin page at http://127.0.0.1:8080/admin/.

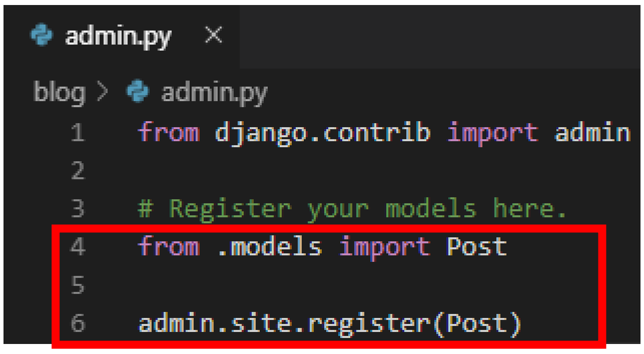
Log in with your new superuser account.

If you remember back in lab 3 when we logged into the admin superuser account, we couldn't see the Post model.



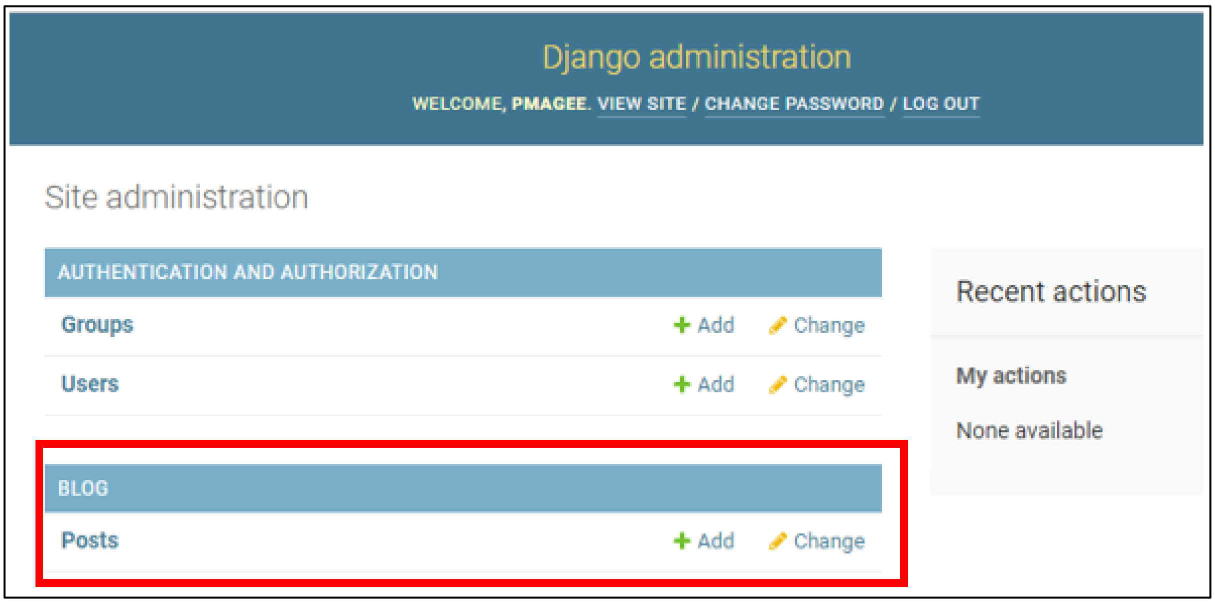
3. To see the **Post** model, we need to register it with admin. Open **blog/admin.py** & add the following code:

Code



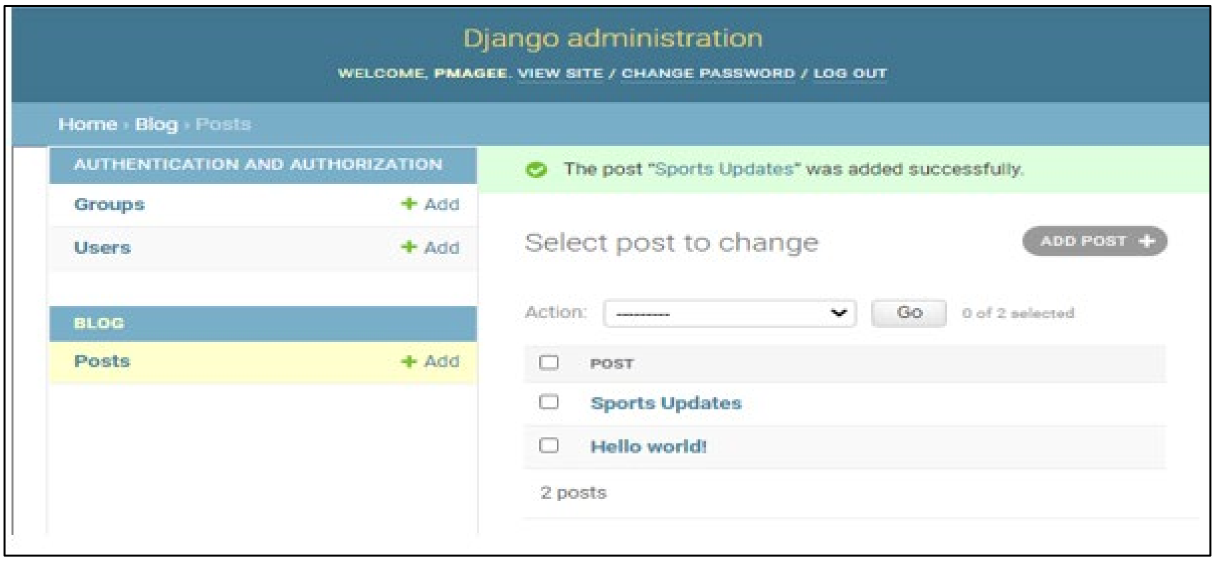
```python
from django.contrib import admin

# Register your models here.
from .models import Post

admin.site.register(Post)
```

4. Refresh the browser page & you will see the update as shown below



5. Add two blog posts so we have some sample data to work with. Click on the + Add button next to Posts to create a new entry. Make sure to add an "author" to each post too since by default all model fields are required. If you try to enter a post without an author, you will see an error.

Here is the admin homepage with two posts added:



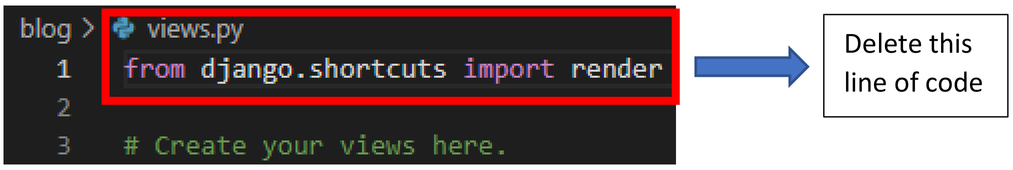
The post "Sports Updates" was added successfully.

To display the information on our web application, we need to create the necessary views, URLs, and templates.

Views

We will use the class-based view called **ListView** here.

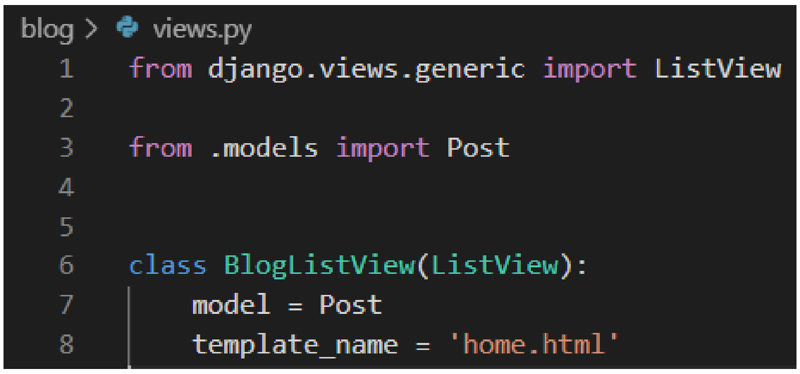
1. Open the file **blog/views.py** & delete the line of code at the top of the file.



```
blog > views.py
from django.shortcuts import render

# Create your views here.
```

2. Add the code below to display the contents of our Post model using **ListView**.

Code



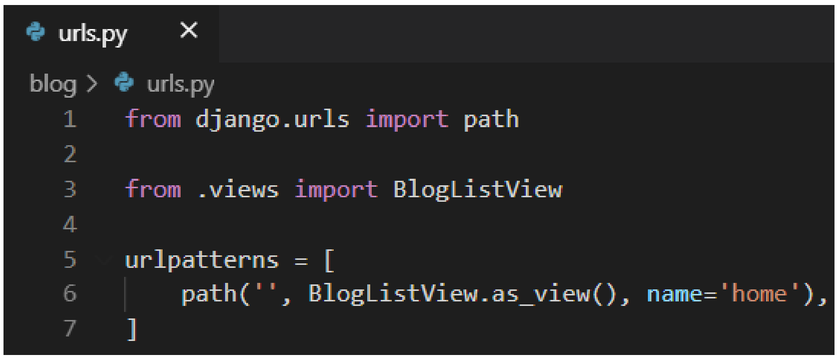
```
blog > views.py
from django.views.generic import ListView

from .models import Post


class BlogListView(ListView):
    model = Post
    template_name = 'home.html'
```

On the top two lines we import ListView and our database model Post. Then we subclass ListView and add links to our model and template.

URLs

1. In VS Code create a new **urls.py** inside the **blog** app & update it with the code below

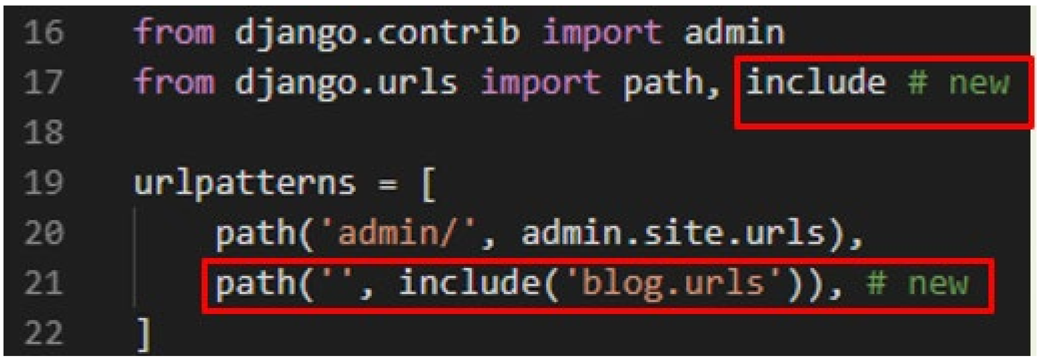
Code



```
urls.py ×
blog > urls.py
from django.urls import path

from .views import BlogListView

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

2. Add the following code to the **blogproject/urls.py** file so that it knows to forward all requests directly to the **blog** app.

Code



```
16    from django.contrib import admin
17    from django.urls import path, include # new
18
19    urlpatterns = [
20        path('admin/', admin.site.urls),
21        path('', include('blog.urls')), # new
22    ]
```

We have added **include** on line 17 indicating that URL requests should be redirected to **blogs/urls.py** for further instructions.

Templates

With our URLConfs and views now complete, we need to create our templates. As we already saw in the earlier lab, we can inherit from other templates to keep our code clean.

We will start off with a **base.html** file and a **home.html** file that inherits from it. Then later when we add templates for creating and editing blog posts, they too can inherit from **base.html**.

1. In Windows Command Line create a new folder called **templates:**

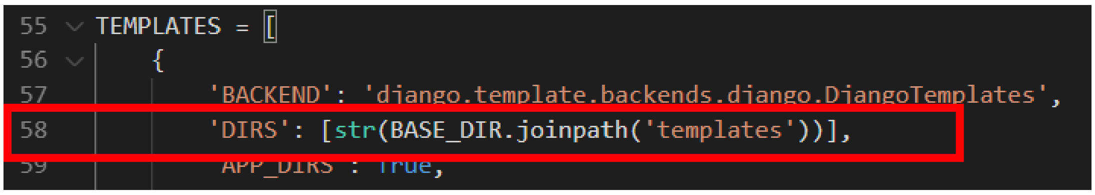
---

mkdir templates

In VS Code move into the **templates** folder and create two new files: **base.html** and **home.html**.

2. Update the **DIRS** field in our **settings.py** file so that Django knows to look in this **templates** directory.

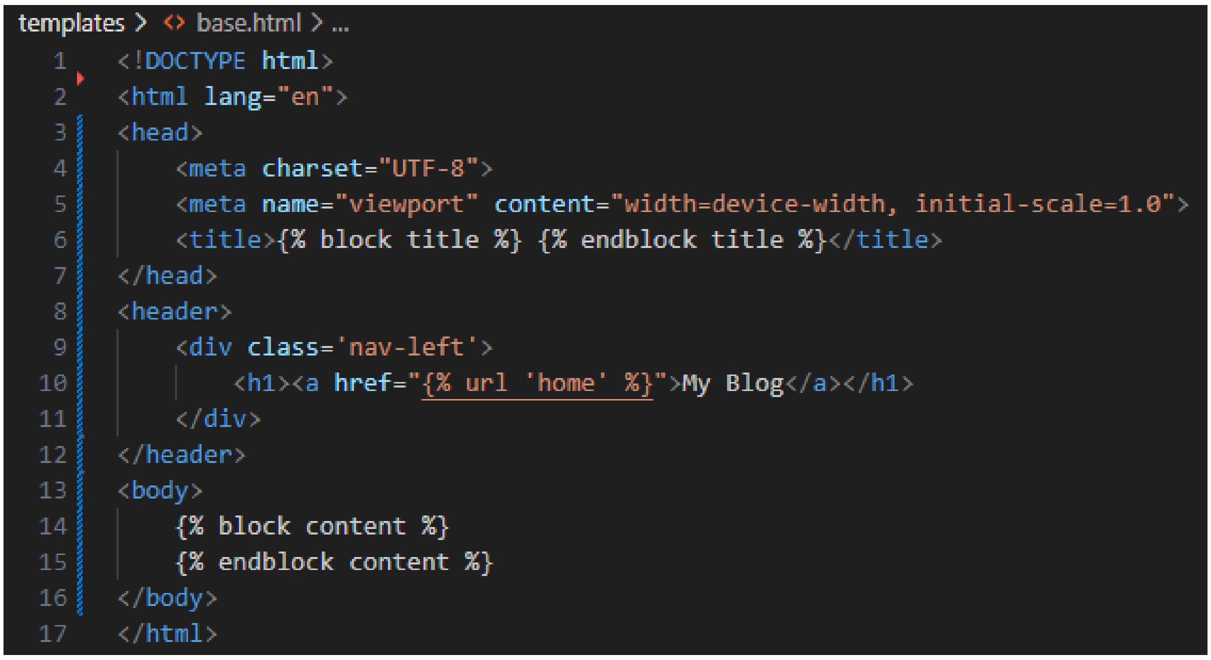
Code



```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [str(BASE_DIR.joinpath('templates'))],
        'APP_DIRS': True,
```

Add the following code to **base.html**

Code

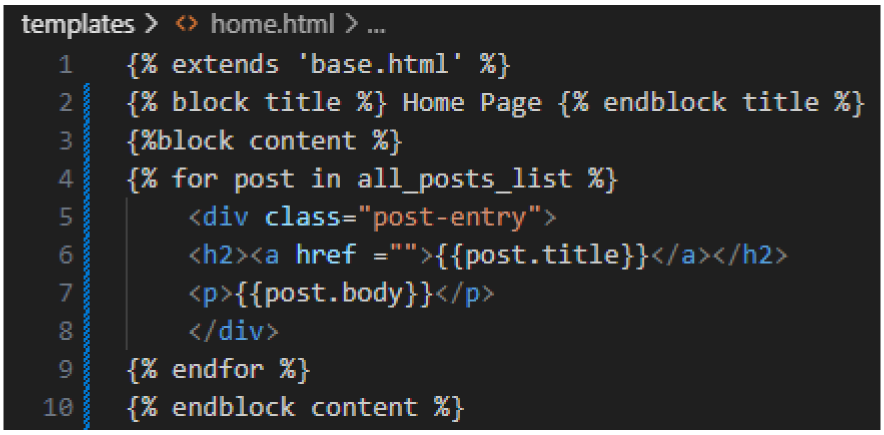


```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %} {% endblock title %}</title>
</head>
<header>
    <div class='nav-left'>
        <h1><a href="{% url 'home' %}">My Blog</a></h1>
    </div>
</header>
<body>
    {% block content %}
    {% endblock content %}
</body>
</html>
```

Note that the code between {% block content %} and {% endblock content %} can be filled by other templates.

1. Enter the following code into **home.html**

Code



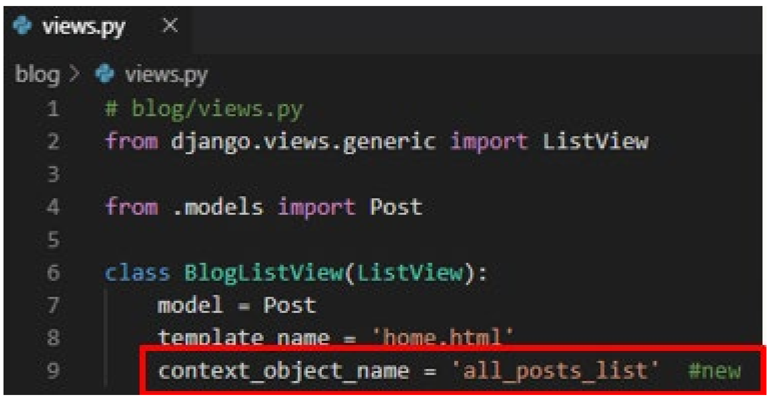
```
{% extends 'base.html' %}
{% block title %} Home Page {% endblock title %}
{%block content %}
{% for post in all_posts_list %}
    <div class="post-entry">
    <h2><a href ="">{{post.title}}</a></h2>
    <p>{{post.body}}</p>
    </div>
{% endfor %}
{% endblock content %}
```

At the top we note that this template extends **base.html** and then wraps our code with content blocks. We use the Django Templating Language to set up a simple for loop for each blog post. Note that **object_list** comes from **ListView** and contains all the objects in our view.

The name **object_list** that is provided to us is not very user friendly. We can use a more meaningful name by providing an explicit name in **views.py**

2. Add the line of code highlighted below to **views.py**.

Code

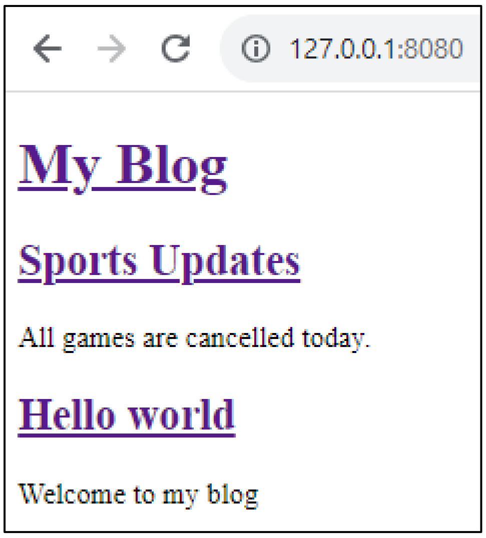


views.py

blog > views.py

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

from .models import Post

class BlogListView(ListView):
    model = Post
    template_name = 'home.html'
    context_object_name = 'all_posts_list'  #new
```

3. Change the name of the list in **home.html** from **object_list** to **all_posts_list**
4. Start the Django server again using the command: **python manage.py runserver 8080**
5. Refresh the web page & you can see that the application is working



127.0.0.1:8080

My Blog

Sports Updates

All games are cancelled today.

Hello world

Welcome to my blog

The next step is to add some CSS styling to improve the appearance of the web site.

Static files

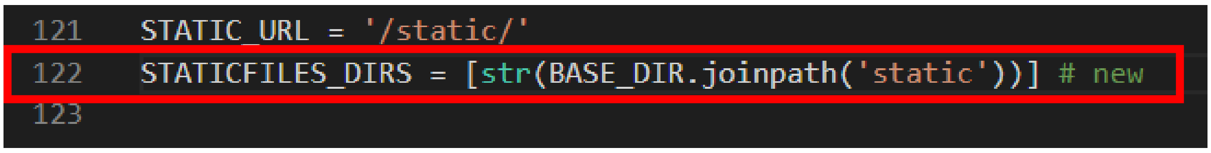
We need to add some CSS which is referred to as a static file because, unlike our dynamic database content, it doesn't change.

1. Stop the local server with Control+c and create a new directory called **static** inside the **lab4-blog-application** folder.

Just as we did with our **templates** directory, we need to update **settings.py** to tell Django where to look for these static files. We can update **settings.py** with a one-line change for **STATICFILES_DIRS**.

2. Scroll to the end of the file and add this line at the bottom of the file below the entry for **STATIC_URL**

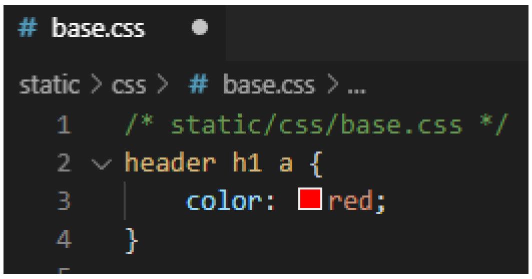
Code



```python
STATIC_URL = '/static/'
STATICFILES_DIRS = [str(BASE_DIR.joinpath('static'))] # new

```

3. Create a **css** directory within the **static** directory. You can do this in VS Code or on the command line.
4. In VS Code right click on the **css** folder and create a new **base.css** file within it
5. Add the following css code to the file

Code

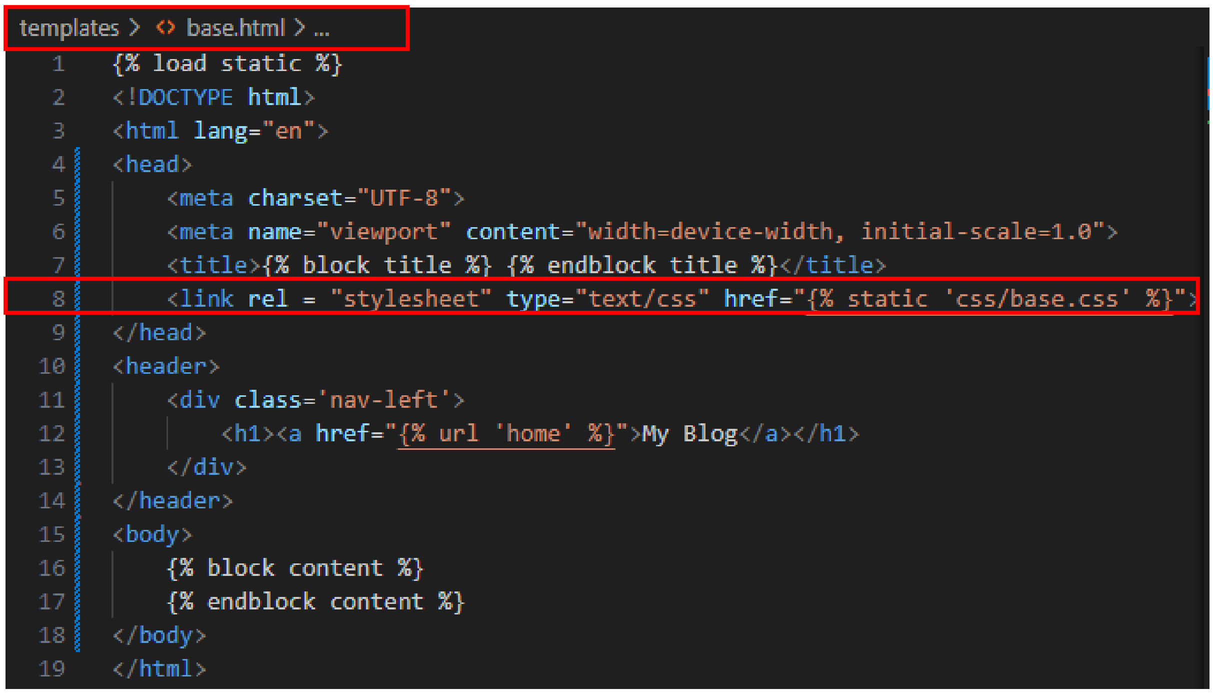


```css
/* static/css/base.css */
header h1 a {
    color: red;
}
```

We need to add the static files to our templates.

6. To do this, add {% load static %} to the top of base.html as shown in line 1 below
7. Include a new line as shown in line 5 below at the bottom of the <head></head> code that explicitly references our new `base.css` file.

Code

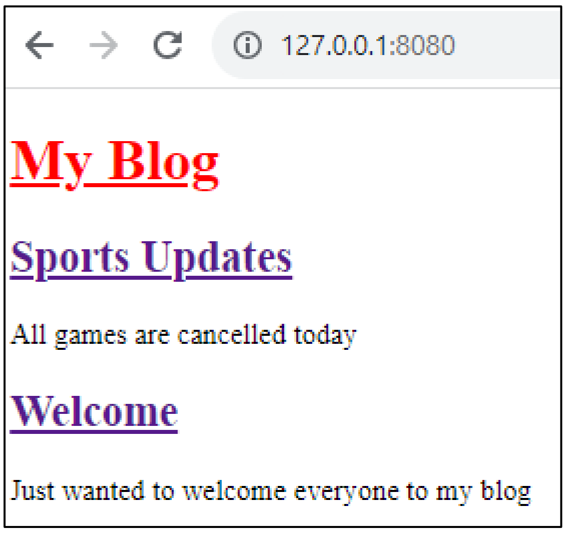


templates > <> base.html > ...

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>{% block title %} {% endblock title %}</title>
    <link rel = "stylesheet" type="text/css" href="{% static 'css/base.css' %}">
</head>
<header>
    <div class='nav-left'>
        <h1><a href="{% url 'home' %}">My Blog</a></h1>
    </div>
</header>
<body>
    {% block content %}
    {% endblock content %}
</body>
</html>
```

Now we can add static files to our static directory, and they will automatically appear in all our templates.

8. Start up the server again with the command **python manage.py runserver** and look at our updated homepage at http://127.0.0.1:8080/



127.0.0.1:8080

My Blog

Sports Updates

All games are cancelled today

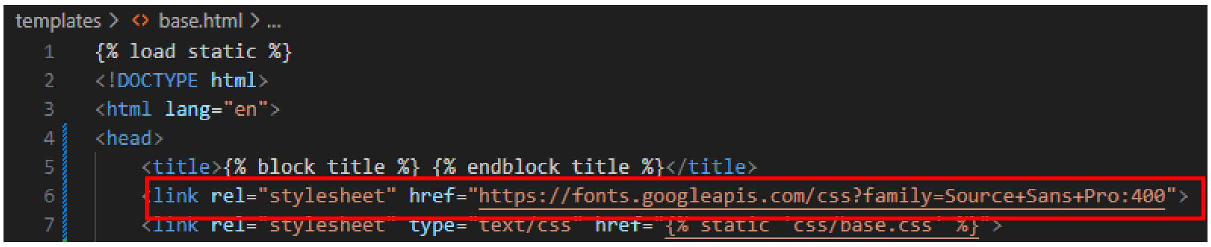
Welcome

Just wanted to welcome everyone to my blog

We will add some more styling to the page using the code provided below;

9. Add a custom font between the <head></head> tags to add Source Sans Pro, a free font from Google. To avoid typos, the code can be copied below and pasted into the page.

Code



```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <title>{% block title %} {% endblock title %}</title>
    <link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:400">
    <link rel="stylesheet" type="text/css" href="{% static 'css/base.css' %}">
```

Copy and paste this code:

```
<link rel="stylesheet" href="https://fonts.googleapis.com/css?family=Source+Sans+Pro:400">
```

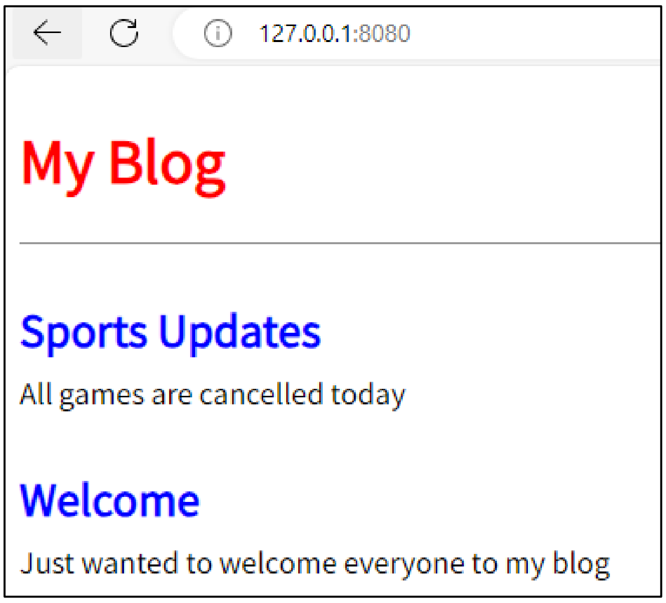
10. Overwrite the css file by copying and pasting the following code:

Code

```
body {
  font-family: 'Source Sans Pro', sans-serif;
  font-size: 18px;
}

header {
  border-bottom: 1px solid #999;
  margin-bottom: 2rem;
  display: flex;
}

header h1 a {
  color: red;
  text-decoration: none;
}

.post-entry {
  margin-bottom: 2rem;
}

.post-entry h2 {
  margin: 0.5rem 0;
}

.post-entry h2 a,
.post-entry h2 a:visited {
  color: blue;
  text-decoration: none;
}
```

```
.post-entry p {
  margin: 0;
  font-weight: 400;
}

.post-entry h2 a:hover {
  color: red;
}
```

11. Refresh the homepage at http://127.0.0.1:8080/ and you should see the following.



127.0.0.1:8080

My Blog

Sports Updates

All games are cancelled today

Welcome

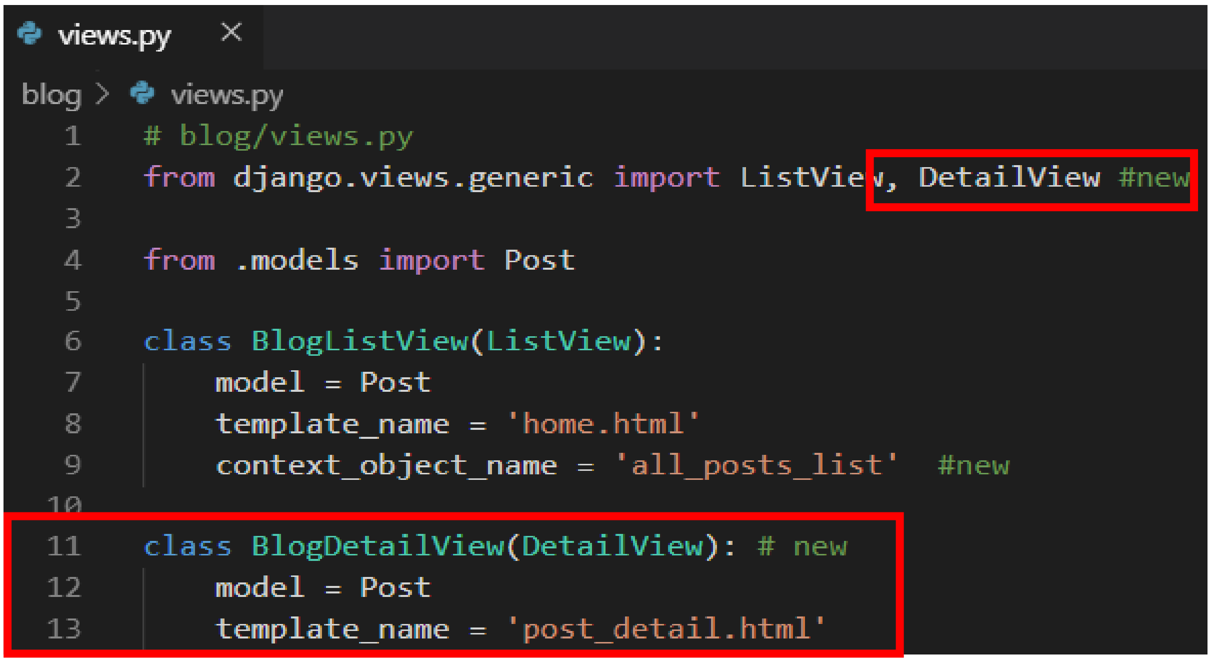
Just wanted to welcome everyone to my blog

Individual blog pages

Now we can add the functionality for individual blog pages. Starting with the view, we can use the generic class-based **`DetailView`** to simplify things

1. At the top of the **`views.py`** file add DetailView to the list of imports and then add the code from lines 11-13 to create our new view called **`BlogDetailView`**

Code



views.py

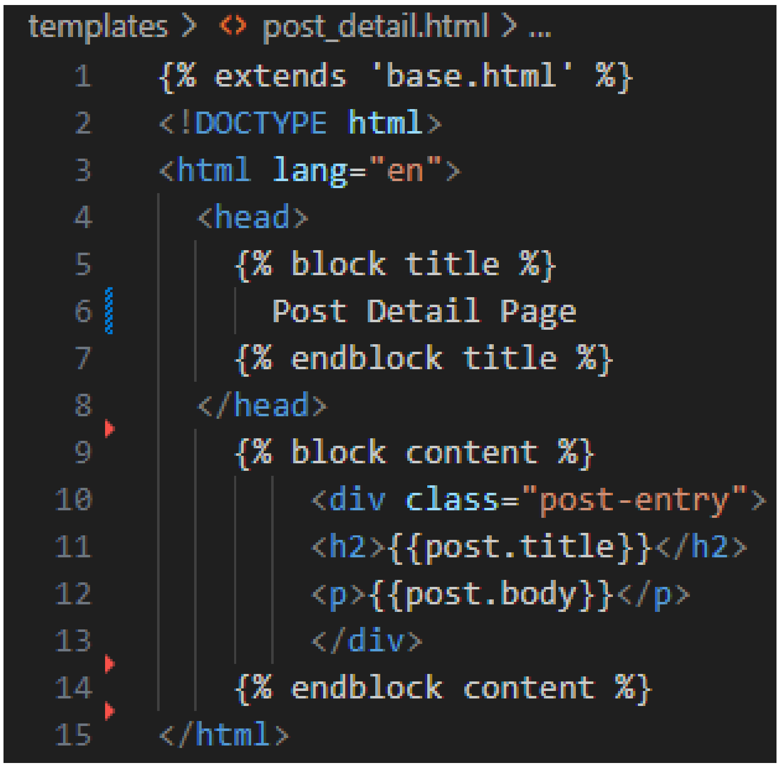
blog > views.py

```
1   # blog/views.py
2   from django.views.generic import ListView, DetailView #new
3
4   from .models import Post
5
6   class BlogListView(ListView):
7       model = Post
8       template_name = 'home.html'
9       context_object_name = 'all_posts_list'   #new
10
11  class BlogDetailView(DetailView): # new
12      model = Post
13      template_name = 'post_detail.html'
```

In this new view, we define the model we are using called **`Post`** and the template we want it associated with, **`post_detail.html`**.

2. In VS Code create a new template called **post_detail.html** inside the templates folder and add in the following code:

Code

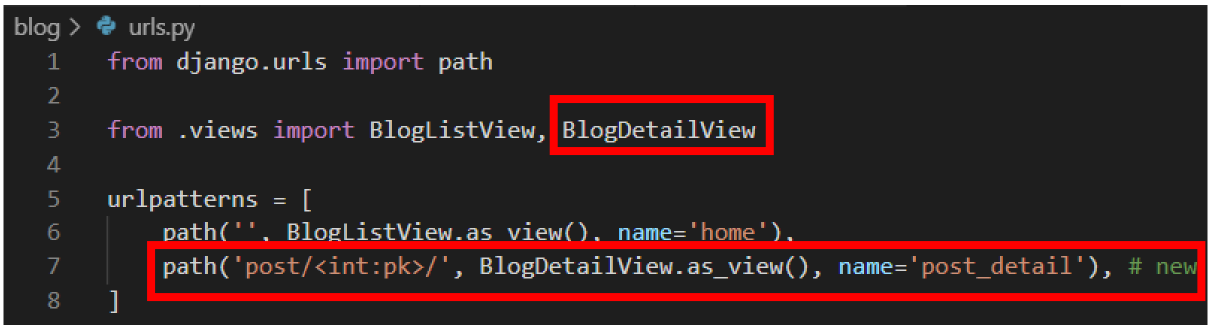


templates > post_detail.html > ...

```
{% extends 'base.html' %}
<!DOCTYPE html>
<html lang="en">
  <head>
    {% block title %}
      Post Detail Page
    {% endblock title %}
  </head>
    {% block content %}
        <div class="post-entry">
        <h2>{{post.title}}</h2>
        <p>{{post.body}}</p>
        </div>
    {% endblock content %}
</html>
```

3. Add a new URLConf for our view as follows:

Code



blog > urls.py

```
from django.urls import path

from .views import BlogListView, BlogDetailView

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

All blog post entries will start with post/. The next part is the primary key for our post entry which is represented as an integer <int:pk>. Django automatically adds an auto-incrementing primary key to our database models.

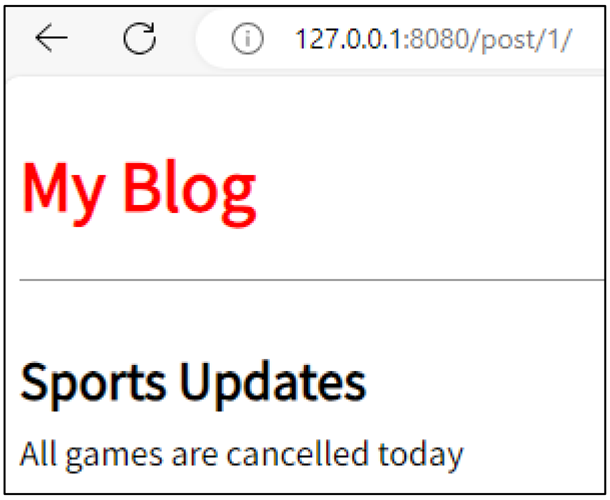
We only declared the fields title, author, and body on our Post model, but in the background, Django also added another field called id, which is our primary key

The pk for our first “Hello, World” post is 1. For the second post, it is 2. And so on.

When we go to the individual entry page for our first post, we can expect that the urlpattern will be post/1.

4. Start up the server using the command **python manage.py runserver** and go directly to http://127.0.0.1:8080/post/1/

You will see a dedicated page for our first blog post as shown below:


127.0.0.1:8080/post/1/

My Blog

Sports Updates

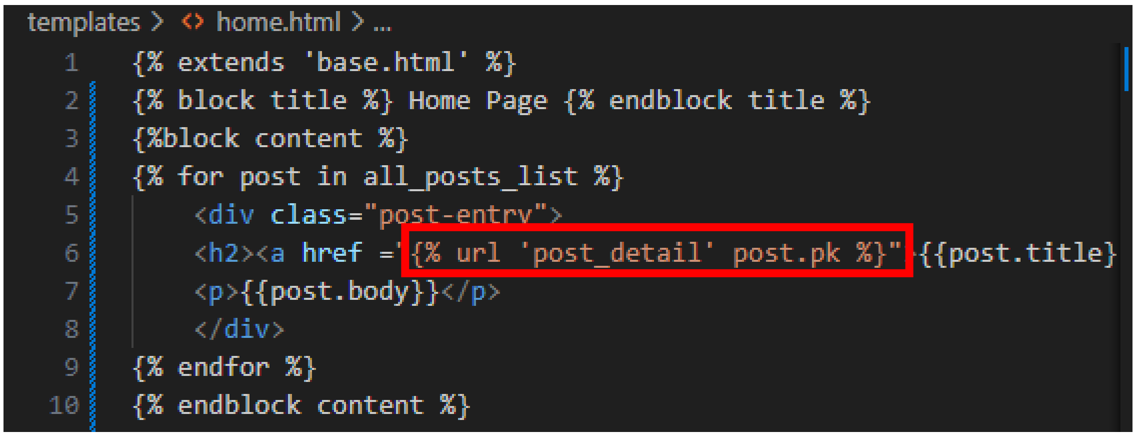
All games are cancelled today

5. Go to http://127.0.0.1:8000/post/2/ to see the second entry

To make our life easier, we should update the link on the homepage so we can directly access individual blog posts from there. Currently in **home.html** our link is empty: <a href="">.

6. Update it as shown in the code below:

Code


templates > home.html > ...

```
{% extends 'base.html' %}
{% block title %} Home Page {% endblock title %}
{%block content %}
{% for post in all_posts_list %}
    <div class="post-entry">
    <h2><a href ="{% url 'post_detail' post.pk %}">{{post.title}
    <p>{{post.body}}</p>
    </div>
{% endfor %}
{% endblock content %}
```

We start off by telling our Django template we want to reference a URLConf by using the code `{% url ... %}`. Which URL? The one named **post_detail**, which is the name we gave **BlogDetailView** in our URLConf just a moment ago. If we look at **post_detail** in our URLConf, we see that it expects to be passed an argument **`pk`** representing the primary key for the blog post. Fortunately, Django has already created and included this pk field on our post object. We pass it into the URLConf by adding it in the template as **post.pk**.

7. To confirm everything works, refresh the main page at http://127.0.0.1:8080/ and click on the title of each blog post to confirm the new links work

Open the **README.md** file in Vs Code and replace the contents with your name and id number.

Save a snapshot of the current project state with the following command:

git add -A

Commit the changes along with a suitable message:

git commit -m "lab 4 part 1"

Update the remote repository with the local commits:

git push -u origin main

Go to your GitHub page and refresh the page to see your local code now hosted online.

To deactivate the virtual environment type, deactivate as shown below:

deactivate

To exit out of Windows Command Line type exit:

exit