**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**

Open **PowerShell**, move into the directory called **djangoprojects** and create a new Django project called **lab4**

mkdir lab4

Move into the **lab4** directory and run the following command to create a virtual environment

virtualenv env

You should see output indicating that the virtual environment has been created. We now need to activate the virtual environment by typing the following command:

env\scripts\activate

You should now see parentheses around the name of your current directory on your command line which indicates the virtual environment is activated:

(env) djangoprojects\lab4>

Install Django using the following command:

(env) djangoprojects\lab4>pip install django

When Django is installed, you may get a warning suggesting that you upgrade to the latest version of pip. If so, type the following command to upgrade pip:

(env) djangoprojects\ lab4>python -m pip install –upgrade pip

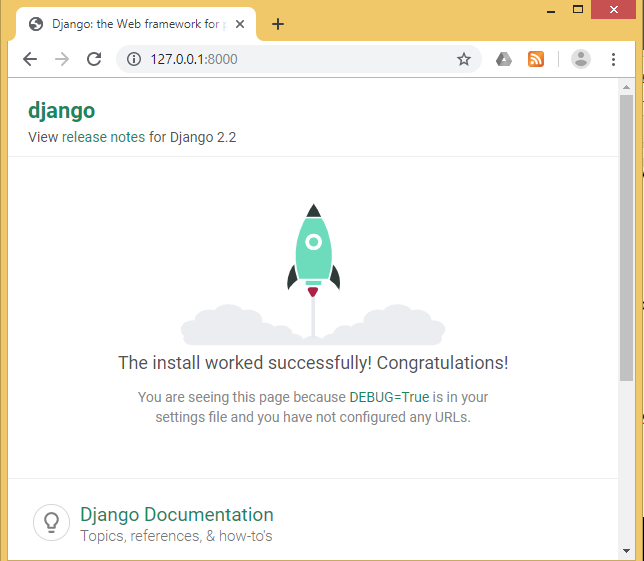
Create a new Django project called **blogproject** with the following command. Don’t forget the period (fullstop) at the end:

(env) djangoprojects\ lab4>django-admin startproject blogproject .

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

(env) djangoprojects\ lab4 >python manage.py runserver

If you visit <http://127.0.0.1:8000/> you should see the familiar Django welcome page



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:

(env) djangoprojects\ lab4 >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 at a later time.

If you execute the python manage.py runserver again, the warning message is gone.

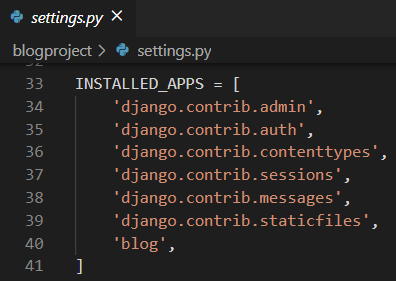
Create an app called posts. From the command line, quit the server with Control+BREAK. Then use the startapp command as shown below:

(env) djangoprojects\ lab4 >python manage.py startapp blog

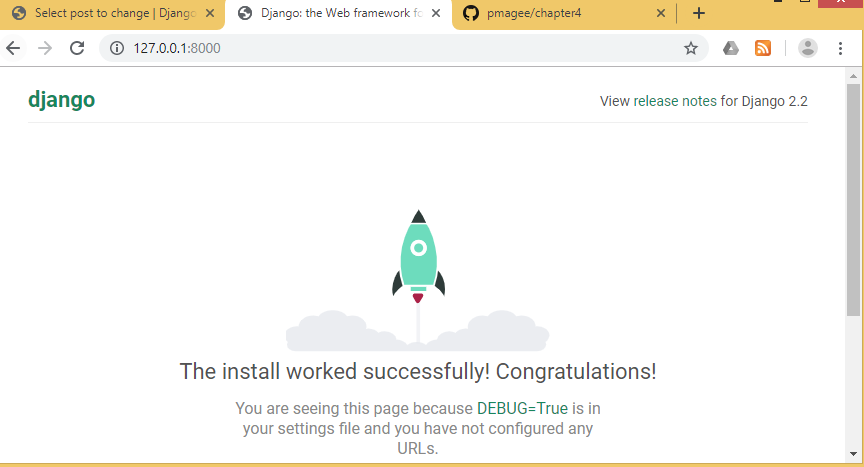
**Settings.py**

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**



1. Run the local server and navigate to http://127.0.0.1:8000/ in your browser & you should see the following page

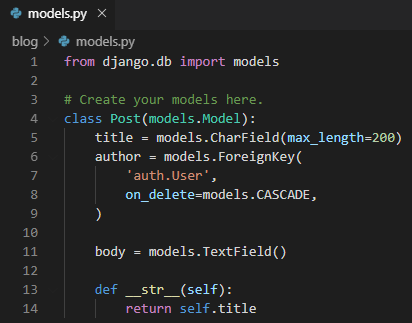


**Database Models**

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**



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.

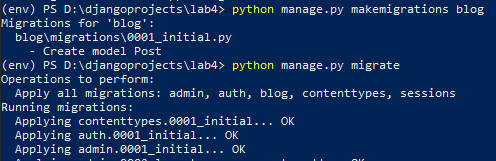
For title we limit the length to 200 characters and for body we use a TextField which will automatically expand as needed to fit the user’s text.

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 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.

1. Stop the server with Control+BREAK. This two-step process can be completed with the commands below:

**Command Line**

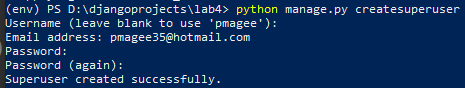


**Admin**

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

**Command Line**

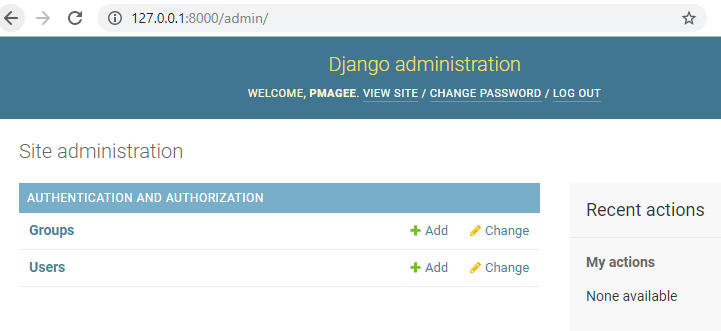


1. Start running the Django server again with the command **python manage.py**

**runserver** and open the Django admin page at <http://127.0.0.1:8000/admin/>.

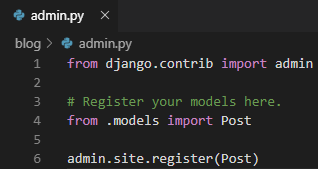
Log in with your new superuser account.

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

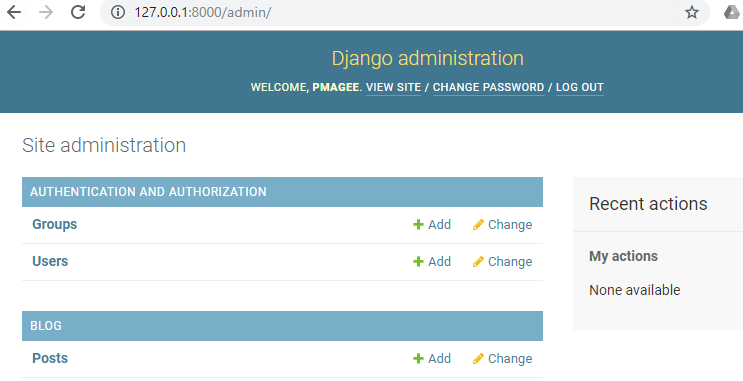


1. In order to see the **Post** model, we need to register it with admin. Open **blog/admin.py** & add the following code:

**Code**

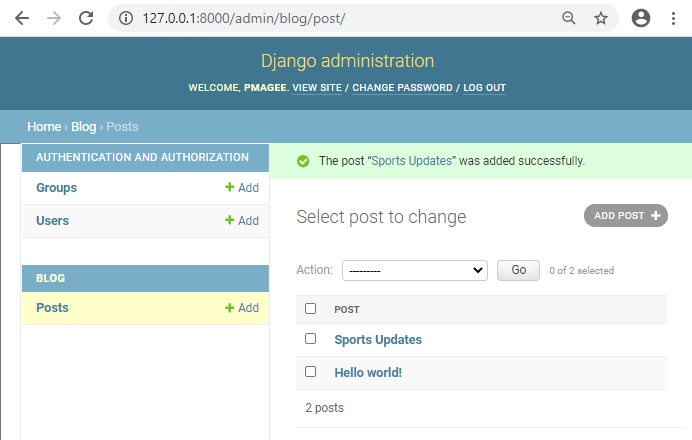


1. Refresh the browser page & you will see the update as shown below



1. 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:

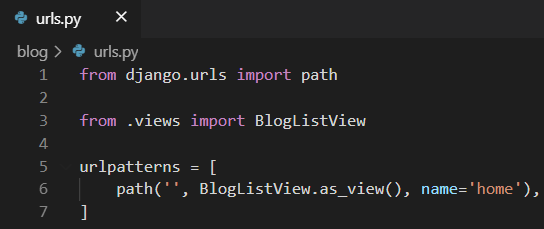


In order to display the information on our web application, we need to create the necessary views, URLs, and templates.

**URLs**

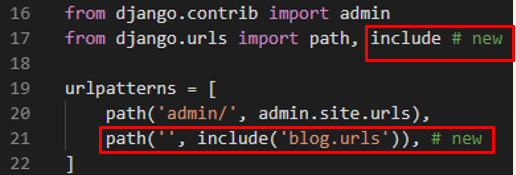
1. In VS Code create a new **urls.py** inside the **blog** app & update it with the code below

**Code**



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

**Code**



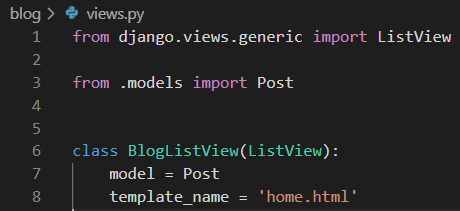
We have added **include** on line 17 indicating that URL requests should be redirected to **blogs/urls.py** for further instructions.

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

**Code**



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.

**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 PowerShell create a new folder called **templates:**

(env) djangoprojects\ lab4 >mkdir templates

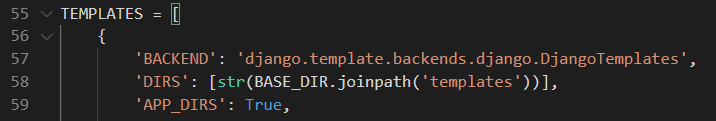
Move into the templates folder and create two new files: **base.html** and **home.html**:

(env) djangoprojects\ lab4\templates >new-item base.html

(env) djangoprojects\ lab4\templates >new-item home.html

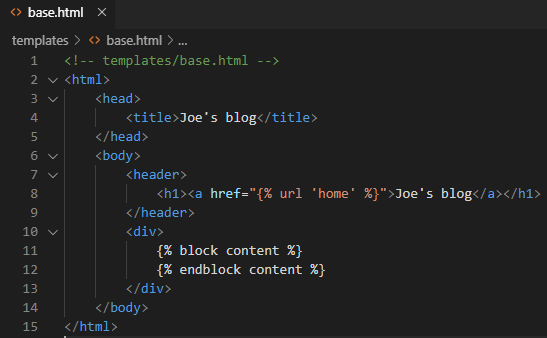
1. Update the **DIRS** field in our **settings.py** file so that Django knows to look in this templates directory.

**Code**



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

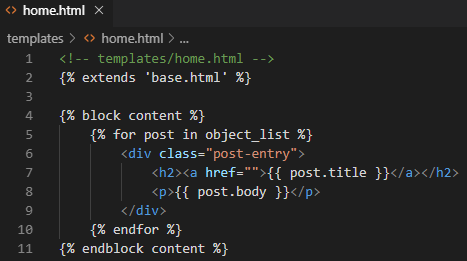
**Code**



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**

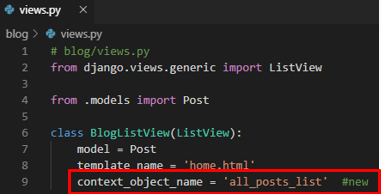


At the top we note that this template extends **base.html** and then wraps our code with content blocks. We use the Django Templating Language to set up a simple for loop for each blog post. Note that **object\_list** comes from **ListView** and contains all the objects in our view.

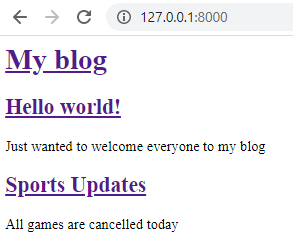
The name **object\_list** that is provided to us is not very user friendly. We can use a more meaningful name by providing an explicit name in **views.py**

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

**Code**



1. Change the name of the list in home.html from **object\_list** to **all\_posts\_list**
2. Start the Django server again using the command: **python manage.py runserver**
3. Refresh the web page & can see that the application is working



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.

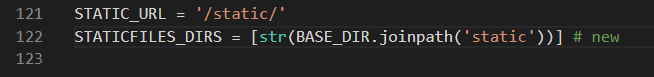
1. Stop the local server with Control+BREAK and create a new directory called **static**.

(env) djangoprojects\ lab4 >mkdir static

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**.

1. Scroll to the end of the file and add this line at the bottom of the file below the entry for **STATIC\_URL**

**Code**



1. Create a **css** directory within the static directory. You can do this in VS Code or on the command line.

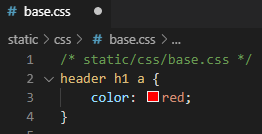
**Command Line**

(env) djangoprojects\ lab4 >cd static

(env) djangoprojects\ lab4 \static>mkdir css

1. In VS Code right click on the **css** folder and create a new **base.css** file within it
2. Add the following css code to the file

**Code**



We need to add the static files to our templates.

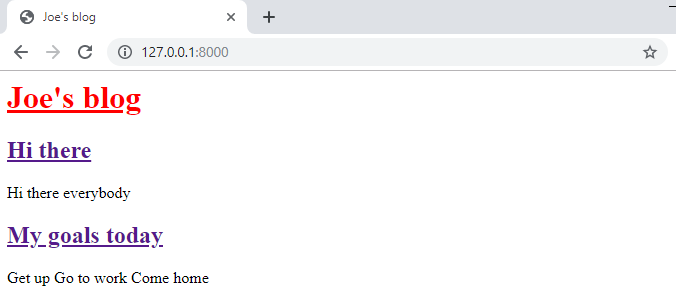
1. In order to do this, add {% load static %} to the top of base.html as shown in line 1 below
2. Include a new line as shown in line 5 below at the bottom of the <head></head> code that explicitly references our new **base.css** file.

**Code**



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

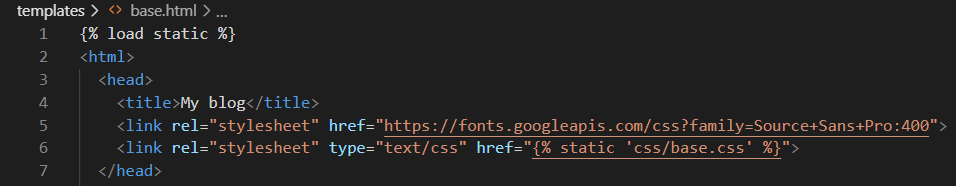
1. Start up the server again with the command **python manage.py runserver** and look at our updated homepage at <http://127.0.0.1:8000/>



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

1. 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**



Copy and paste this code:

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

1. 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;

}

.nav-left {

margin-right: auto;

}

.nav-right {

display: flex;

padding-top: 2rem;

}

.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;

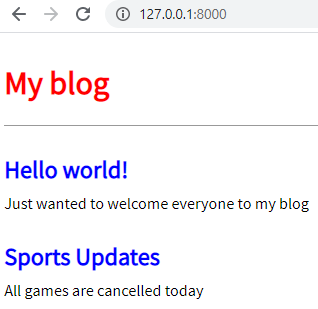
}

.post-entry h2 a:hover {

color: red;

}

1. Refresh the homepage at http://127.0.0.1:8000/ and you should see the following.

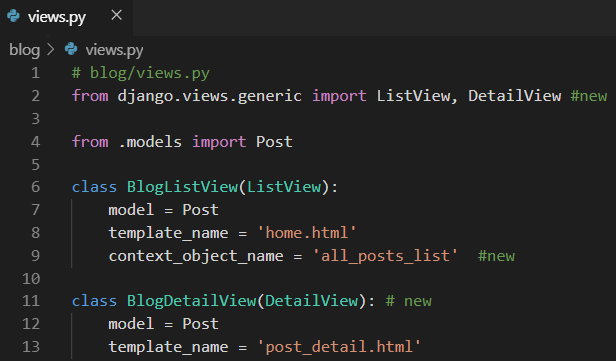


**Individual blog pages**

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**

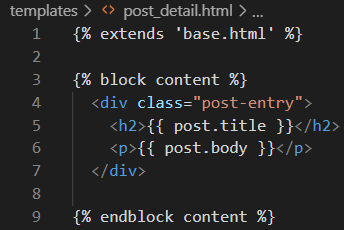


In this new view, we define the model we are using called **Post** and the template we want it associated with, **post\_detail.html**.

1. Stop the local server and in VS Code create a new template called **post\_detail.html** using the code below.

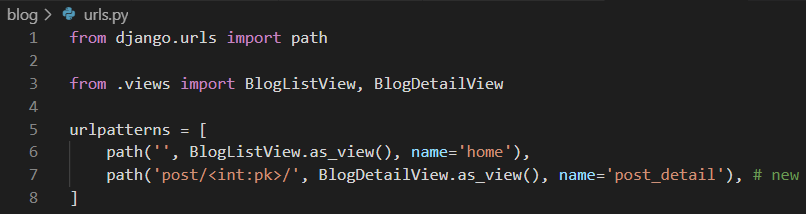
(env) djangoprojects\ lab4\templates >new-item post\_detail.html

**Code**



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

**Code**



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.

1. Start up the server using the command **python manage.py runserver** and go directly to <http://127.0.0.1:8000/post/1/>

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



1. Go to <http://127.0.0.1:8000/post/2/> to see the second entry

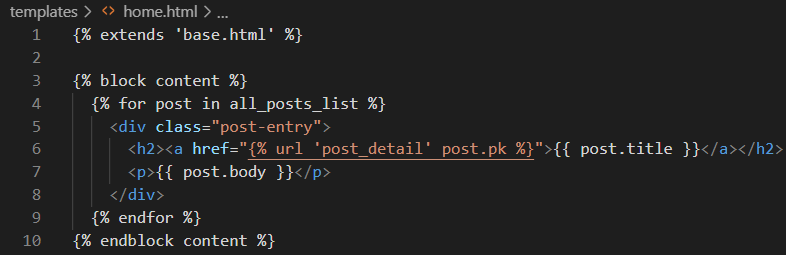
To make our life easier, we should update the link on the homepage so we can directly

access individual blog posts from there. Currently in **home.html** our link is empty: <a

href="">.

1. Update it as shown in the code below:

**Code**



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**.

1. To confirm everything works, refresh the main page at <http://127.0.0.1:8000/> and

click on the title of each blog post to confirm the new links work

Create and initialise a local repo for your lab 4 project and follow the steps set out in the previous lab sheets to push your code to the Classroom GitHub repository – see link on Moodle

