**Lab 4 Part (2)**

In this exercise we will continue to work on our blog application from lab 4 Part (1) by adding forms so a user can create, edit, or delete any of their blog entries.

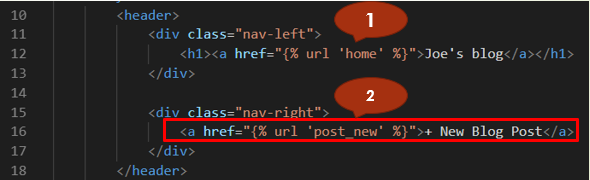
**Forms**

Forms are very common in web applications and can be difficult to implement correctly. When accepting data input from a user there are always security concerns in terms of XSS attacks and because of this, proper error handling is required. We also need to alert the user to problems with the form in terms of validation. Fortunately for us Django’s built-in Forms abstract away much of the difficulty and provide a rich set of tools to handle common challenges we face when working with forms.

To start we will need to update our base template to display a link to a page for entering new blog posts. It will take the form <a href="{% url 'post\_new' %}"></a> where post\_new is the name for our URL.

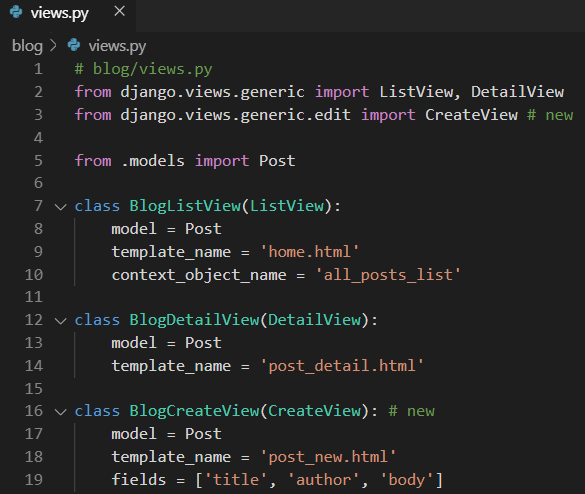
1. Open the **blogproject** in VS Code and open the **base.html** file. Update the code inside the header to look like the file shown below. Make sure to add the code to define the div classes **nav-left** and **nav-right** for both URLs in the header to enable CSS styling to be applied from **base.css**.

**Code**



1. Open the **views.py** file and create a new view using the code provided below. Here we are importing a new generic class called **CreateView** at the top and then subclassing it to create a new view called **BlogCreateView**.

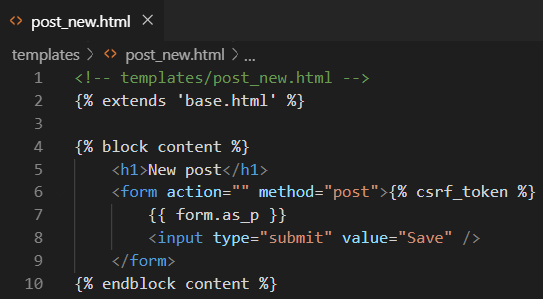
**Code**



Within **BlogCreateView** we specify our database model **Post**, the name of our template **post\_new.html**. For fields we explicitly set the database fields we want to use which are title, author, and body.

1. Create a new template in VS Code called **post\_new.html** and type in the following code:

**Code**



Let’s break down what we have done:

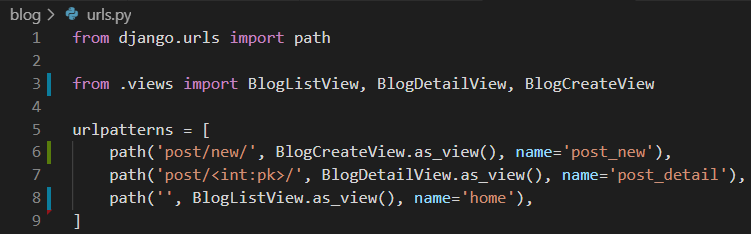
* Line 2: we inherit from our base template
* Line 6 we use the HTML <form> tags with the POST method since we are sending data. We also add a {% csrf\_token %} which Django provides to protect our form cross site scripting attacks. This should be used for all your Django forms
* Line 7 we use {{ form.as\_p }} which renders the data within paragraph <p> tags
* Line 8 we specify an input type of submit and assign it the value “Save”

**URLConf**

Let’s add a new URLConf for **post\_new**.

1. Open the file **blog/urls.py** and update it with the code shown below.

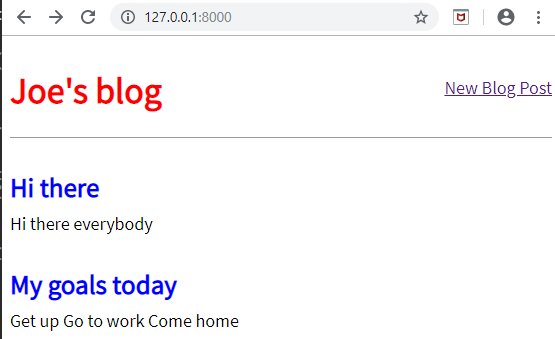
**Code**



We just imported our view called **BlogCreateView** at line 3 and then we added the path to the new URL which will start with **post/new/** and is called **post\_new**.

1. Start the server with the command **python manage.py runserver** and go to the

homepage at <http://127.0.0.1:8000/>.

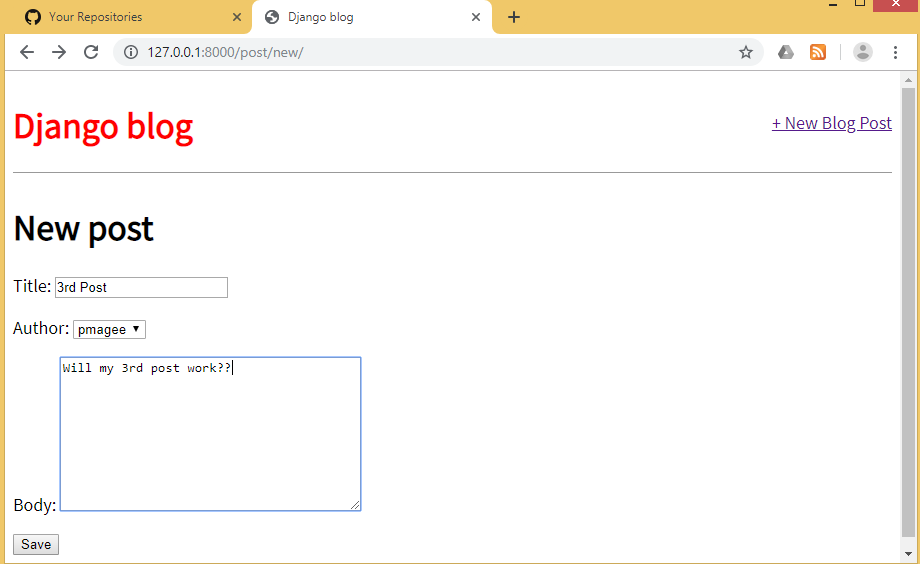


1. Click on our link for “+ New Blog Post” which will redirect you to:

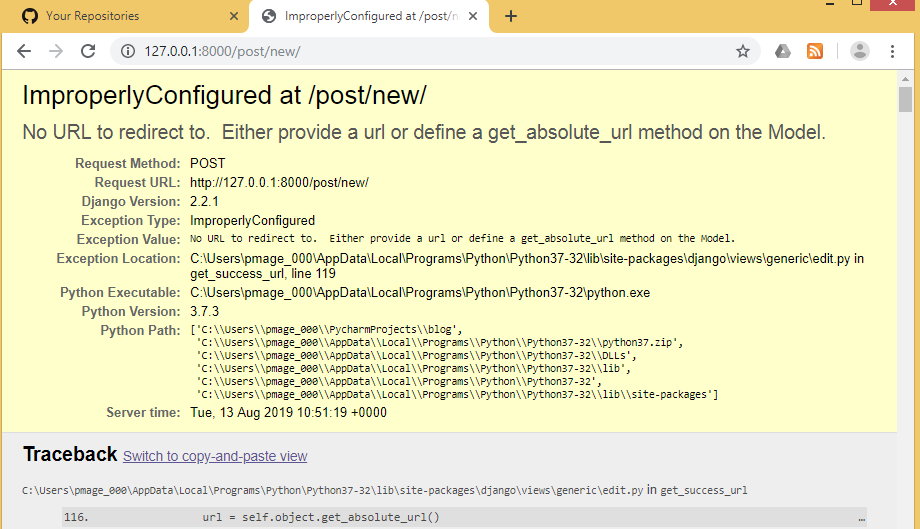
<http://127.0.0.1:8000/post/new/>.



1. Go ahead and try to create a new blog post and submit it.



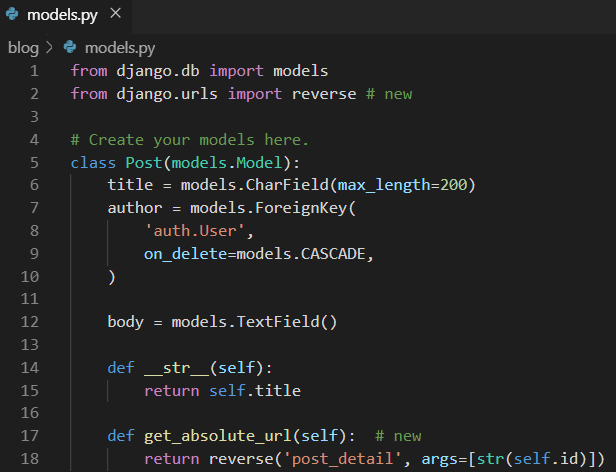
Something has gone wrong!



Django’s error message is telling us that we did not specify where to send the user after successfully submitting the form. Let’s send a user to the new post detail page so that they can see their new post.

1. Open the **models.py** file. Add an import on the second line for **reverse** and a new **get\_absolute\_url** method.

Code



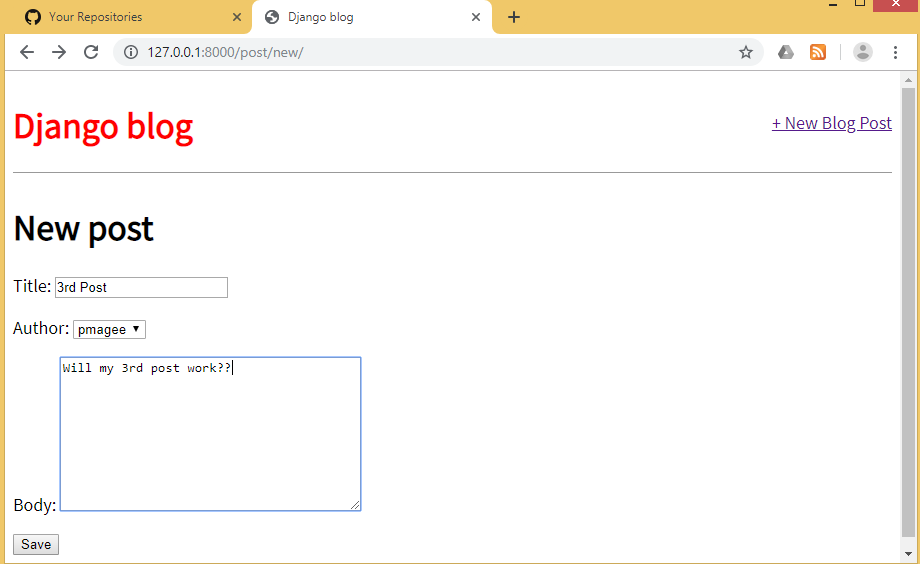
**reverse** is a very handy utility function Django provides us to reference an object by its URL template name, in this case **post\_detail**. If you recall our URL pattern is the following:

**Code**



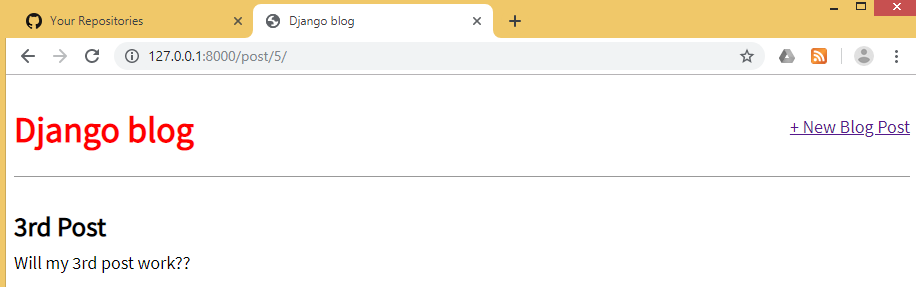
In order for this route to work we must also pass in an argument with the pk or primary key of the object. Confusingly pk and id are interchangeable in Django but the Django docs recommend using self.id with **get\_absolute\_url**. So, we are telling Django that the ultimate location of a **Post** entry is it’s **post\_detail** view which is **posts/<int:pk>/** so the route for the first entry we have made will be at **posts/1**.

1. Try to create a new blog post again at <http://127.0.0.1:8000/post/new/>

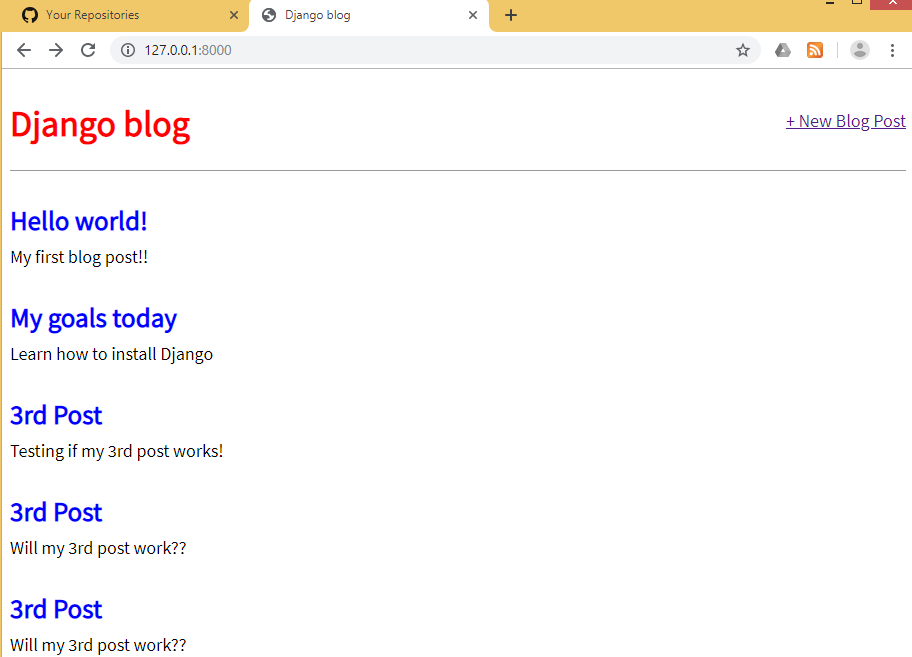


Upon clicking the “Save” button you are new redirected to the detailed view page

where the post appears.



1. Access the homepage at http://127.0.0.1:8000/ and you will also notice that our earlier blog post is also there. It was successfully sent to the database, but Django didn’t know how to redirect us after that.

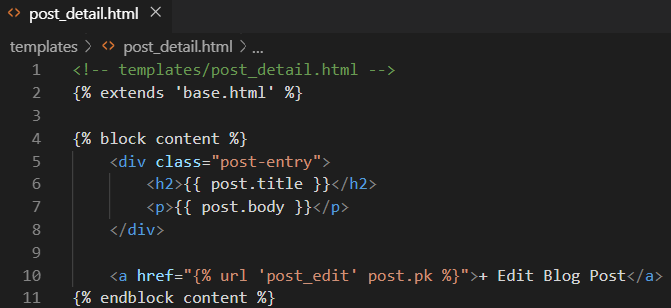


**Update Form**

In order to create an update form so users can edit blog posts we will use a built-in Django class-based generic view, **UpdateView**, and create the template, url, and view.

1. To start, add a new link shown at line 10 below to **post\_detail.html** so that the option to edit a blog post appears on an individual blog page

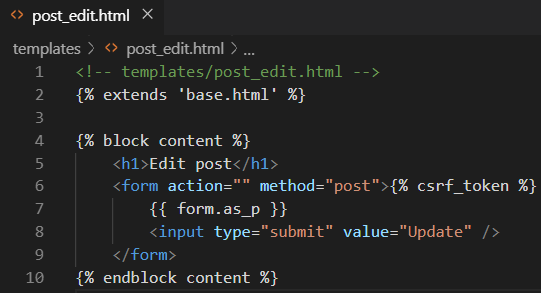
**Code**



We have just added a link using **<a href>...</a>** and the Django template engine’s **{% url ... %}** tag. Within it we have specified the target name of our url, which will be called **post\_edit** and we also passed the parameter needed, which is the primary key of the post post.pk.

1. Create the template in VS Code for the edit page called **post\_edit.html** & add in the following code.

**Code**

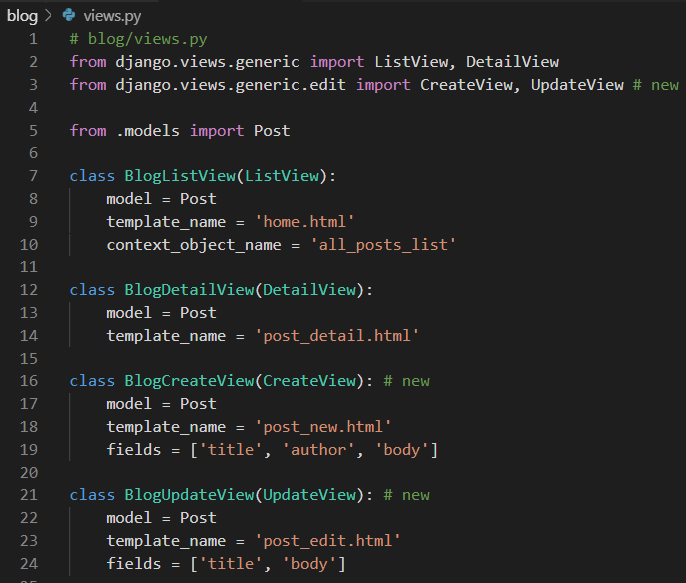


We again use HTML **<form></form>** tags, Django’s **csrf\_token** for security, **form.as\_p** to display our form fields with paragraph tags, and finally give it the value “Update” on the submit button.

Now to our view. We need to import **UpdateView** on the second-from-the-top line and

then subclass it in our new view **BlogUpdateView**. Add the code highlighted below to the **views.py** file.

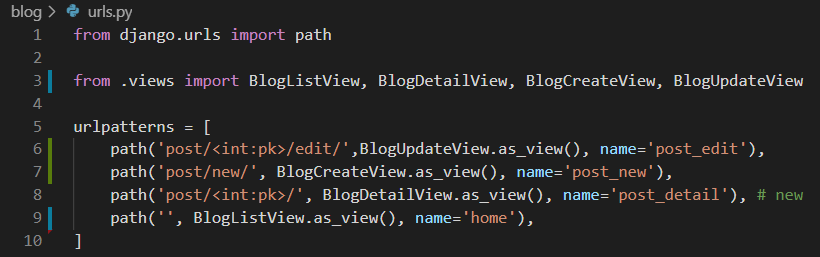
**Code**



Notice that in **BlogUpdateView** we are explicitly listing the fields we want to use ['title', 'body']. This is because we assume that the author of the post is not changing; we only want the title and text to be editable.

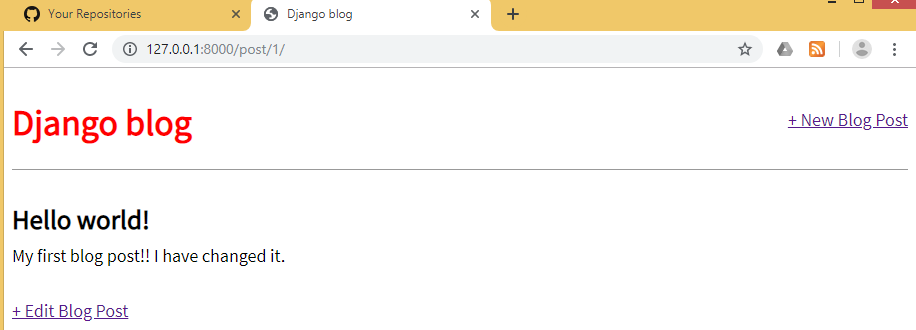
1. Update the **urls.py** file as follows. Add the **BlogUpdateView** at the end of line 1 and then add the new route at line 10.

**Code**

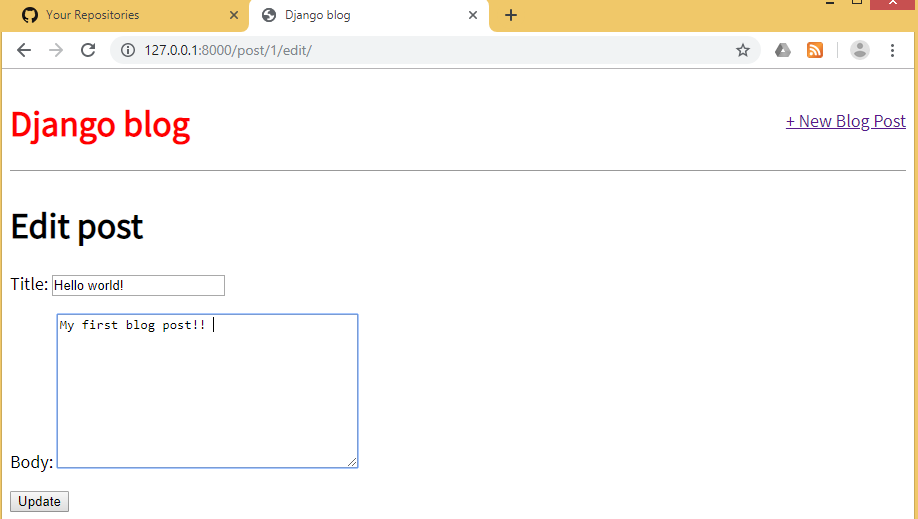


At the top we added our view **BlogUpdateView** to the list of imported views, then created a new url pattern for /post/pk/edit and gave it the name **post\_edit**.

1. If you click on a blog entry you will see the new Edit button.

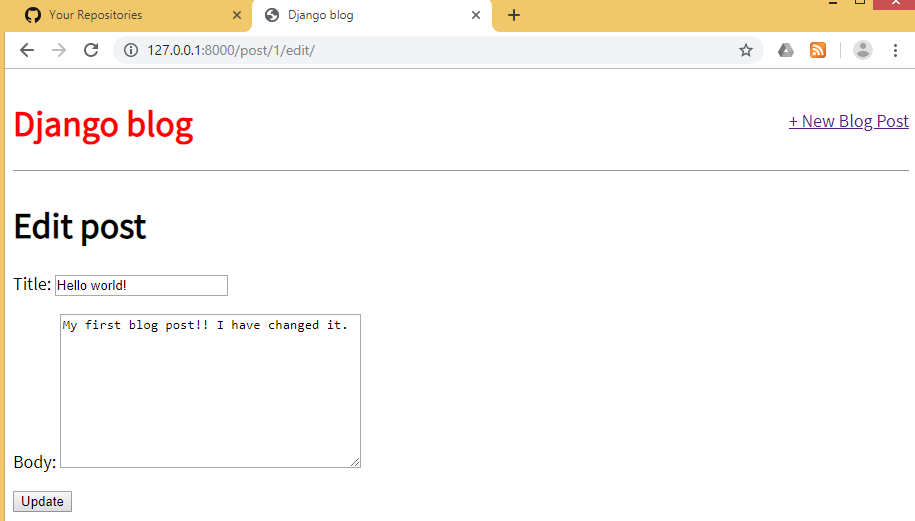


1. Click on “+ Edit Blog Post” and you will be redirected to the following URL <http://127.0.0.1:8000/post/1/edit/> if you selected your first blog post



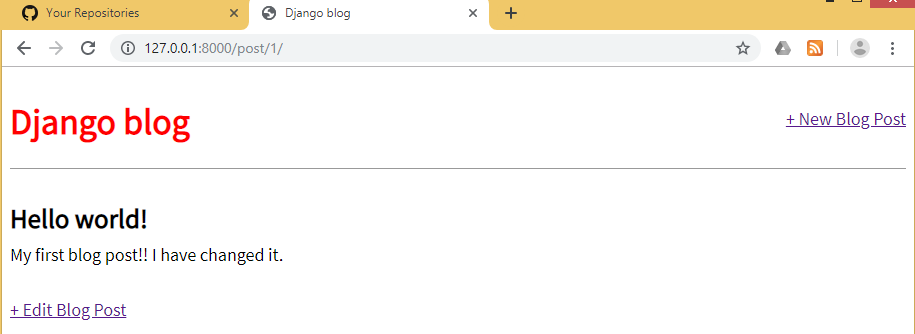
Note that the form is pre-filled with our database’s existing data for the post.

1. Change the text in the blog post and click **Update**



And after clicking the “Update” button we are redirected to the detail view of the post where you can see the change. This is because of our get\_absolute\_url setting.

1. Navigate to the homepage and you can see the change next to all the other entries.



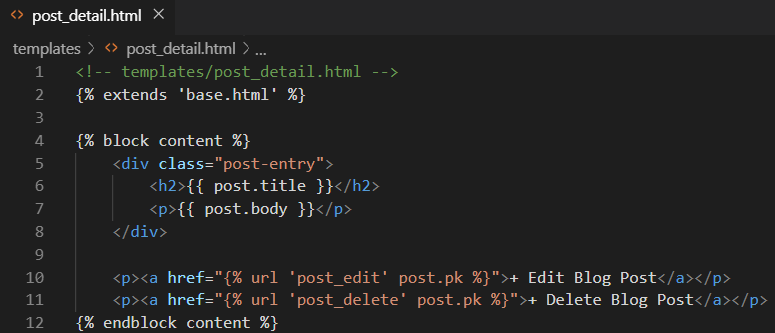
**Delete View**

The process for creating a form to delete blog posts is very similar to that for updating

a post. We will use yet another generic class-based view, **DeleteView**, to help and we will then create a view, url, and template for the functionality.

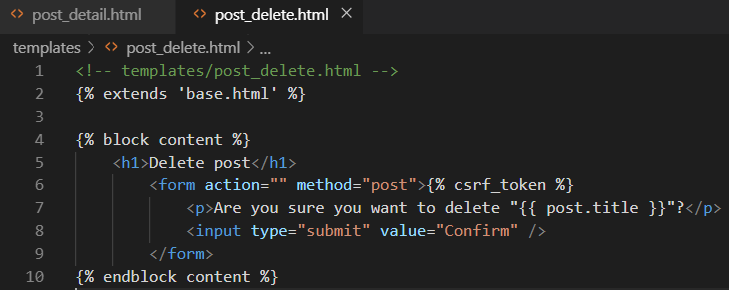
1. Add a new link (line 11) to **post\_detail.html** to delete blog posts. This time we have surrounded the link by **<p>** tags to create a bit of space.
2. Add <p> tags to the Edit Blog Post link on line 10 as well.

**Code**



1. In VS Code, create a new template file called **post\_delete.html** and enter the following code:

**Code**



1. Update the **views.py** file using the code provided below, by importing **DeleteView** and **reverse\_lazy** at the top,then create a new view that subclasses **DeleteView**.

**Code**

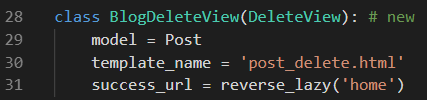
Import **DeleteView**



Add this new import **reverse\_lazy**



Add this new class

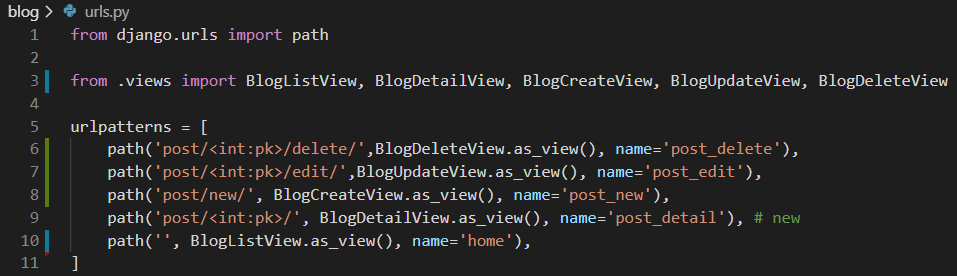


We use reverse\_lazy as opposed to just reverse so that it won’t execute the URL

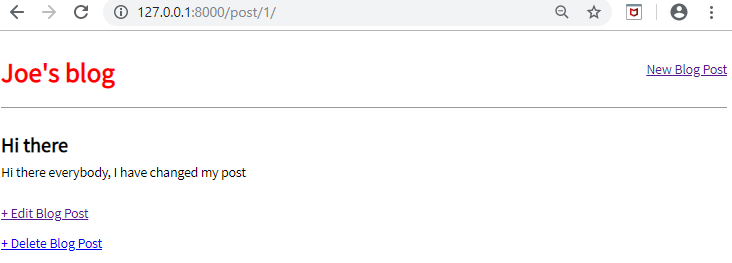
redirect until our view has finished deleting the blog post.

**URLs**

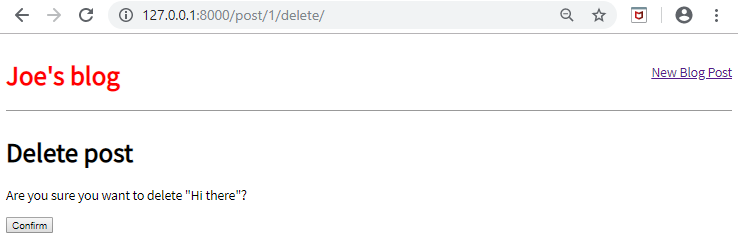
1. Open blog/urls.py and add the following code to create a URL

**Code**

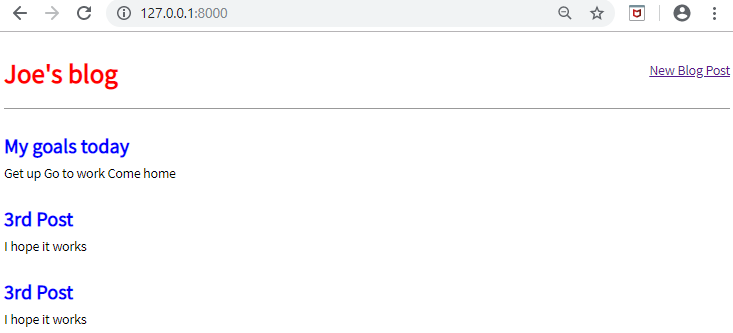
1. Start the server again using **python manage.py runserver** and refresh the individual post page you will see our “Delete Blog Post” link.



1. Click the link and it takes you to the delete page for the blog post, which displays the name of the blog post.



1. Click on the “Confirm” button and you are redirected you to the homepage where the blog post has been deleted!



Run the following git commands to update the local and remote repositories:

(env) djangoprojects\lab4>git add -A

(env) djangoprojects\lab4>git commit -m “lab 4 part 2 commit”

(env) djangoprojects\lab4>git push -u origin master