**Django Front End**

* Static files
* Base HTML
* Add style

**Static files in Django**

It is time to think about our project's front end. Our templates are too simple with just HTML on them, so let's add some colors. The first thing we need to do is create a folder where we're going to store all the static files, such as the CSS and JavaScript files, images, videos, etc.

Graphical user interface

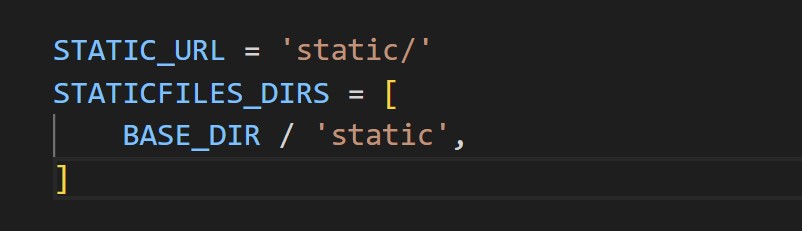
Description automatically generated

Now we need to tell Django that this is the folder it needs to look into when searching for static files. To do that, let's go to the smartnotes->settings. After STATIC\_URL add:

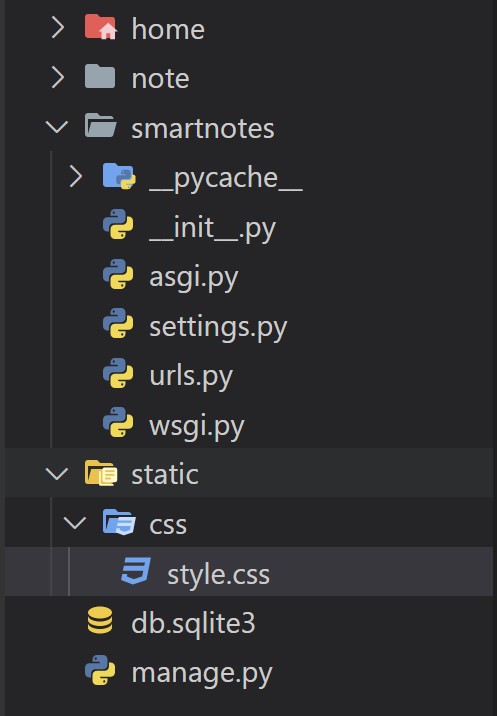
STATICFILES\_DIRS = [

BASE\_DIR / 'static',

]

This will lead Django to the folder that we just created.

Now we can go back to the static and create a new folder ‘css’ just for the CSS files and one CSS file. Let's call it style.css.



Now we can create a simple CSS file. Type ->

.note-li {

color: red;

}



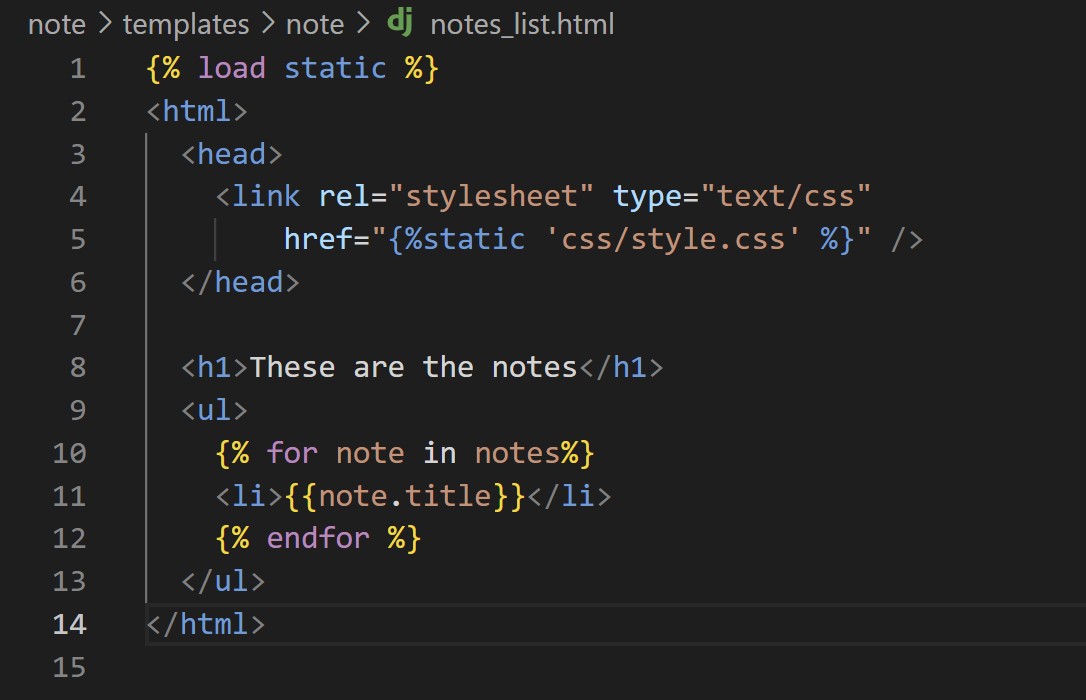
What we need to do now is make sure that our template and Django recognize this file. So let's go to the note/ templates/ notes\_list file. The first thing we need to do is actually tell Django that this HTML is going to use the static files. So, in the first line type -> {% load static %}

Now what we need here is to add a CSS file as we would in any HTML file, which is ->

<head>

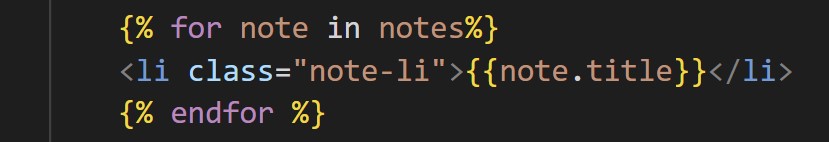
<link rel="stylesheet" type="text/css" href="{%static 'css/style.css' %}"/>

</head>

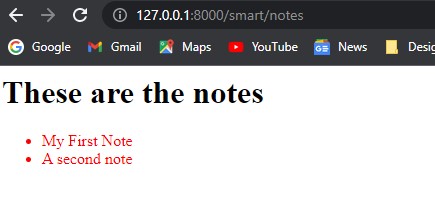


That's it. That's all we need to do to have the CSS being rendered on this file.

So let's go here on the li and add a class ‘note-li’ and save it.



Okay, let's try it out.



There you go! The notes are red because the CSS is being rendered and used in this file. So, now you can use CSS in all your templates.

**How to set up a base HTML for every Django template**

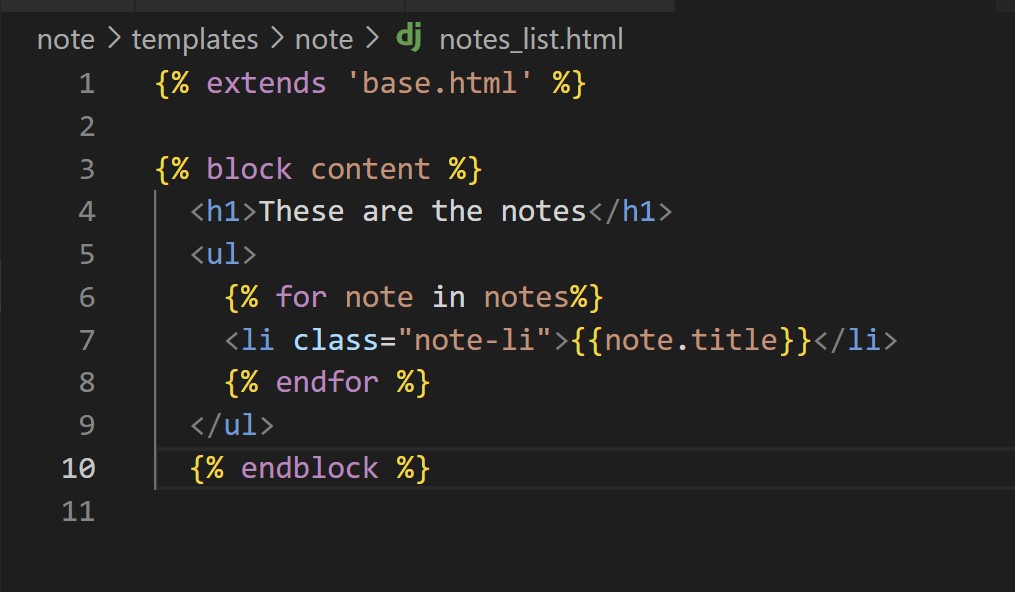
As we've seen, it's pretty easy to add CSS files into Django template. But it would be quite exhaustive to need to always remember to add the CSS link to all templates we have in all our apps. There is a better way of ddoing it.

What we need is a base template. Let's create a templates folder in the static folder, and a base.html template in it and in there, create a normal HTML file. In html->body add the following command:  {% block content %}{% endblock %}

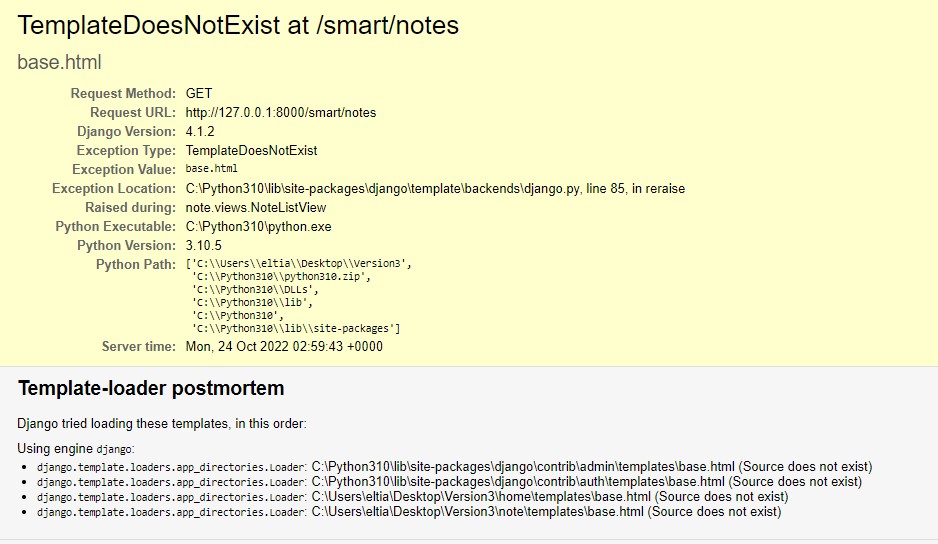


We called this ‘content’, but you can call it whatever you like. The important thing here is to know that this is a block area where we can inject things. Let's try it out.

Let's go back to the note/notes\_list template. In here, what we can do is extends base.html. Now, what we can do is get rid of all this basic HTML here and use this block content here.



What we're doing here, we're taking only the important part of our template, and wrapping it on the ‘block content’ command so this can be injected on the ‘base’ template. Let's try it out.

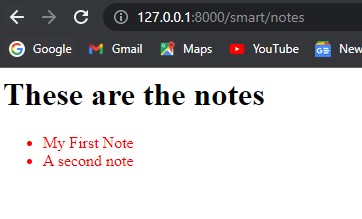


So we have a problem here. The template is showing us non-existent. What happens here? You can see here that Django is trying to search for a base.html template. You can see that it tried in multiple places, including the two templates folder in home and note app. But as you can see, the static folder template is not being looked for. So what we need to do is tell Django what to look for.

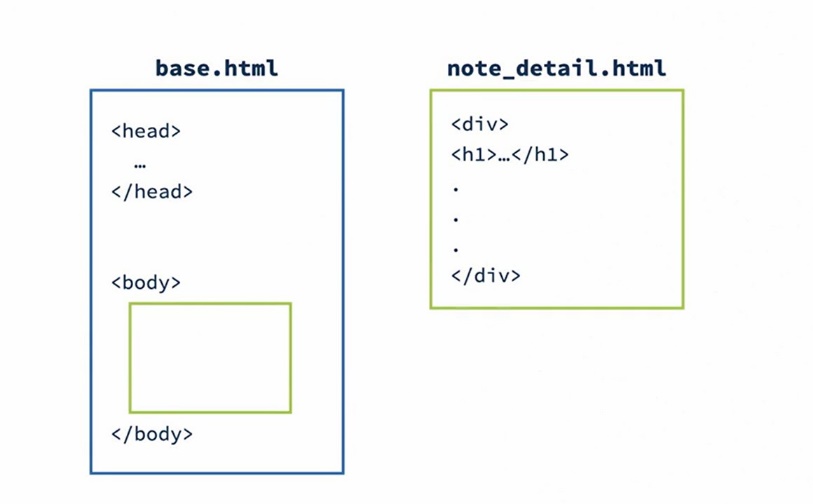
Let's go to settings file-> TEMPLATES and you can see that there is a list of directories that we can add here. So similar to what we did on the static files, we're going to add that particular folder in here. In DIRS add -> BASE\_DIR / 'static/templates',



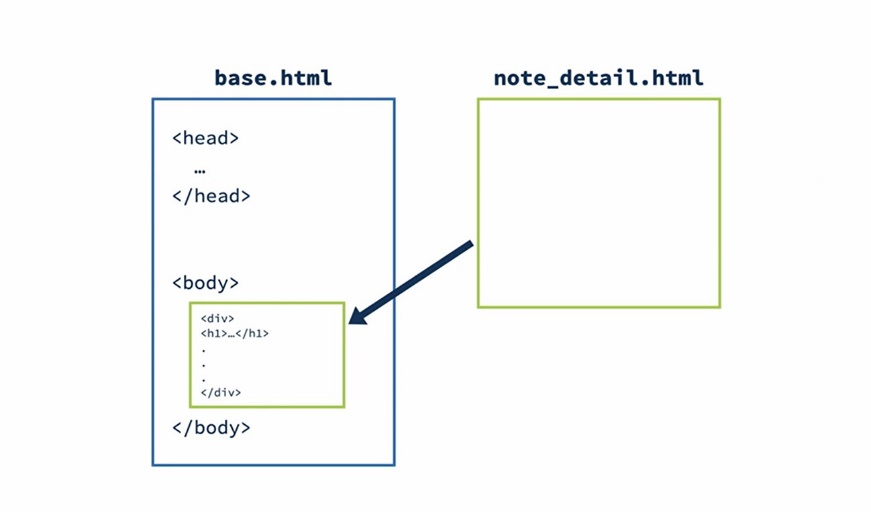
Let's try it again. And there you go!



Let's take a minute to understand what's going on here.



What's happening here is that with this syntax we can define the basics of our HTML in our base.html template, and then we create each web page as a separate template that extends the base. So we will build each template separately (just the small parts but then we'll inject it to the base template where we can have all our default configurations, such as the CSS files and the JavaScript.



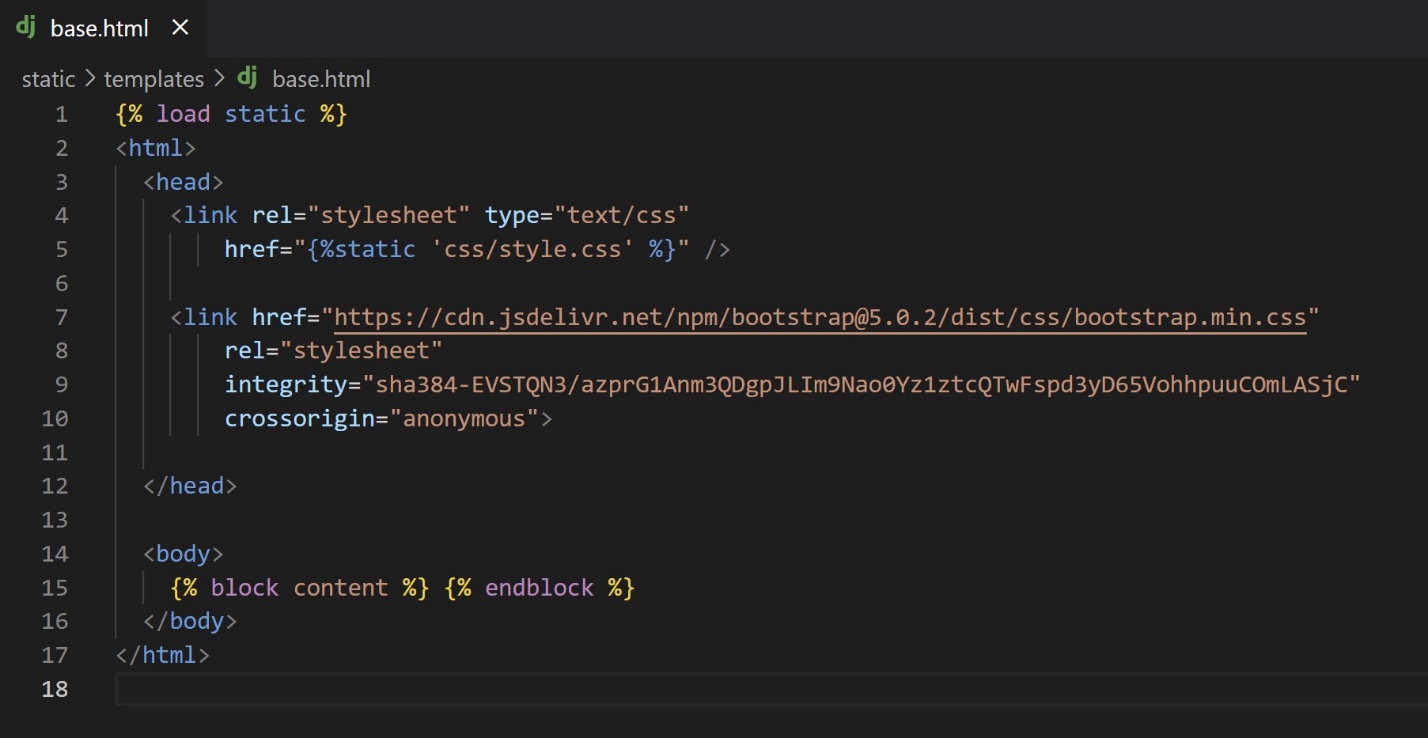
This will allow us to keep each web page template as simple as we can while keeping all the configuration in a single place. That's another power of the Django template language.

**Let's add some style**

Now, instead of defining all the CSS, we want to speed up our front end a little bit, so let's use the CSS framework. We're going to use Bootstrap for now.

What we need to do is on the static/templates/base.html change CSS we just created with the link to the Bootstrap framework version five.

<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC" crossorigin="anonymous">



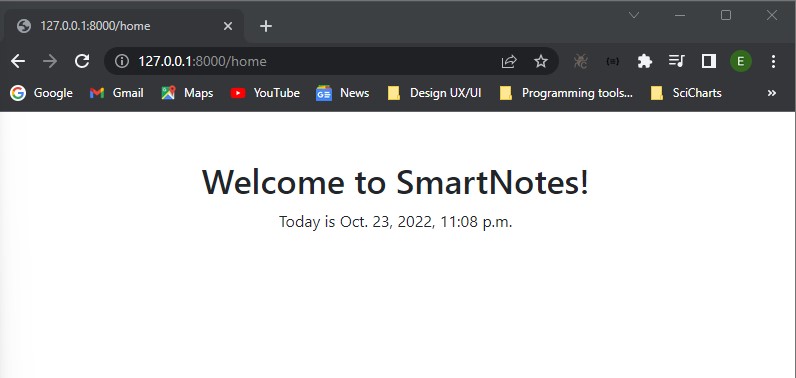
Once we define this, we can start using it. The first thing we can do is:

<div class="my-5 text-center container">

{% block content %} {% endblock %}

</div>

Now we can go to the home page and check the changes that the Bootstrap's already made.

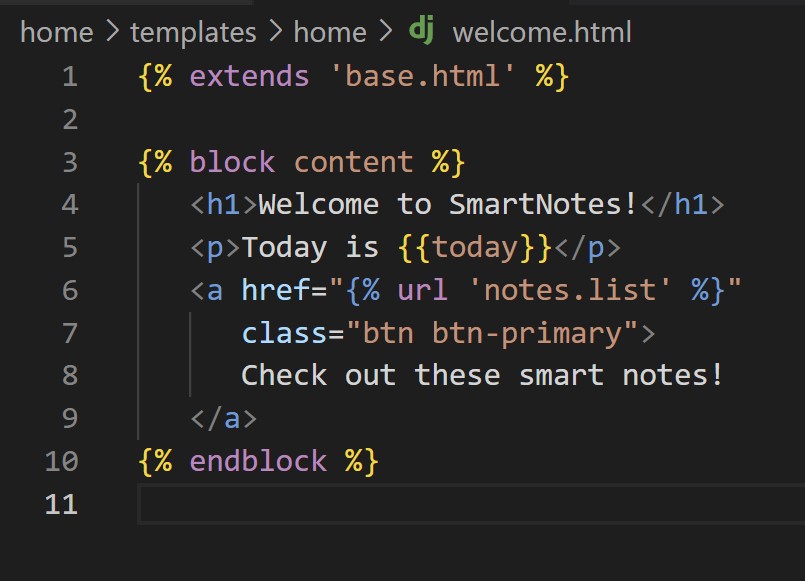


You can see here now that the style's a little bit better. We have more spacing, the text is on center, et cetera.

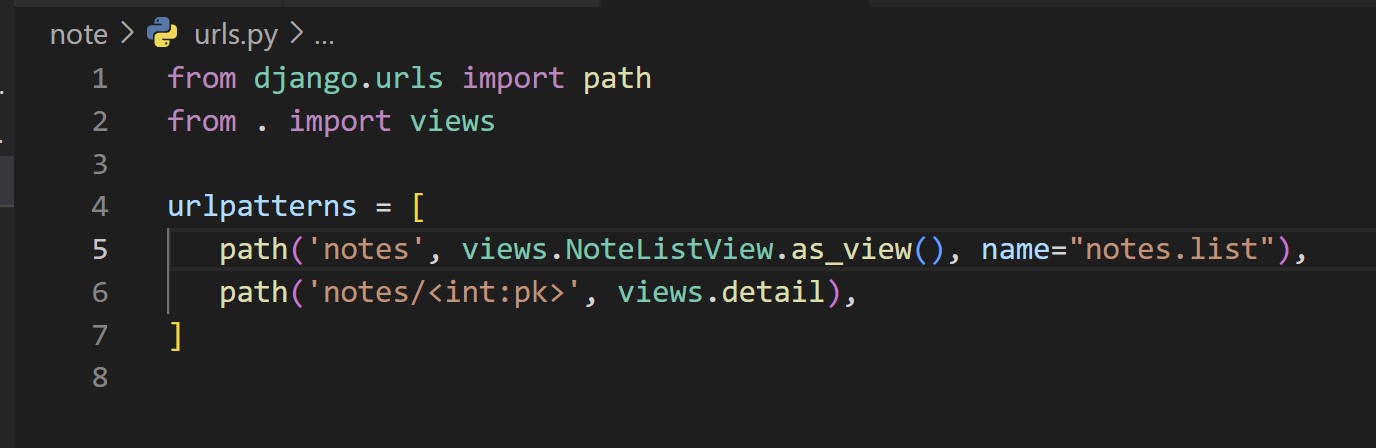
Now let's add a button on the home page that will lead us to the list of notes. In home/home/welcome.html type ->

<a href="" class="btn btn-primary">Check out these smart notes!</a>

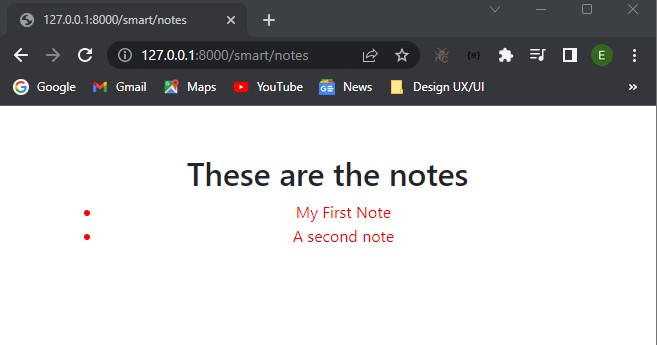
Now we have a button but it doesn't do anything. So how do we deal with links here? We could hard code the link to our localhost but imagine that when we deploy our website, we need to remember to come back and change everything. Not so good. Thankfully, the Django template language has a function for that. What we need to do is the following. Type {% url 'notes.list' %} in href.



You might be wondering, okay, how Django knows which endpoint to link? It doesn't, and we need to tell it. So let's go back to note/ urls.py and in here, what we're going to do is add the name ‘notes.list’.



That's all we need for Django dynamically define each endpoint we are pointing to, no matter if you're on localhost or production. You can test it out. The button should redirect to smart/notes.



That also needs some styling. We'll get there. For now, let's go back to home page and try to style it up a little bit.

Go to notes\_list and add some styling. In ‘block content’ type ->

<h1 class="my-5">These are the notes</h1>

<div class="row row-cols3 g-2">

{% for note in notes %}

<div class="col">

<div class="p-3 border">

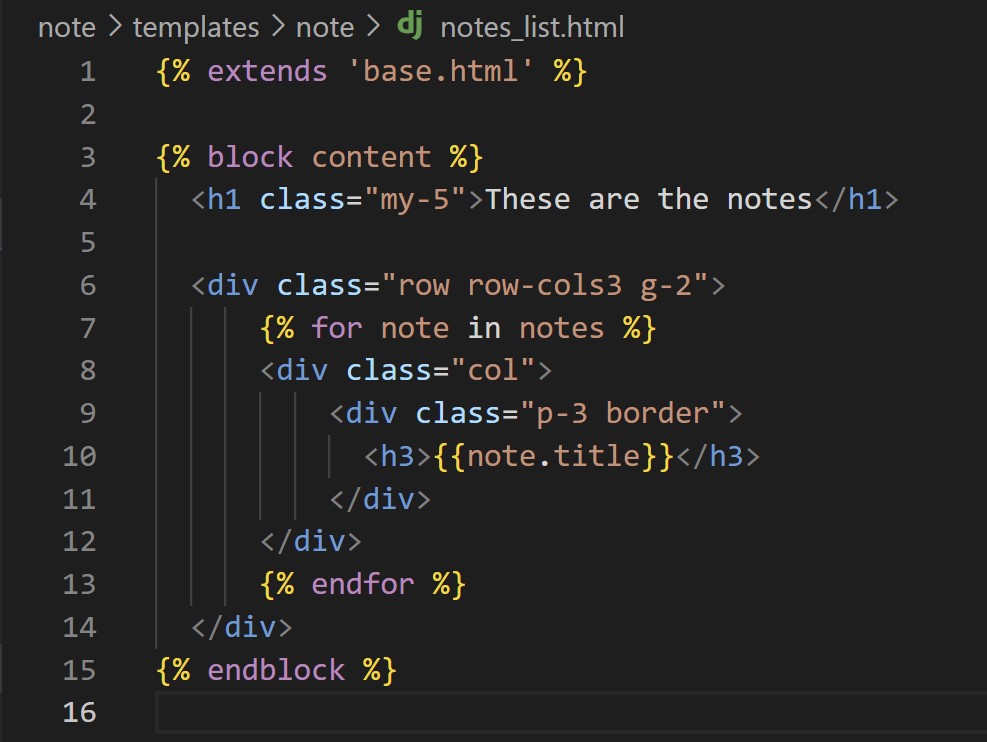
<h3>{{note.title}}</h3>

</div>

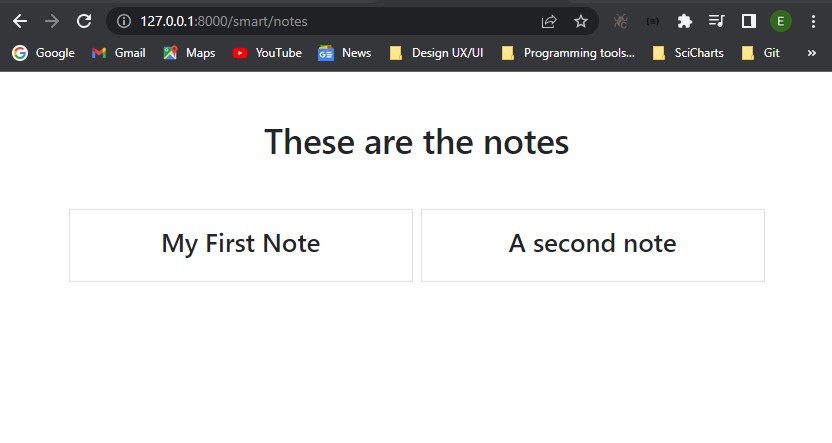
</div>

{% endfor %}

</div>



Let's check out.



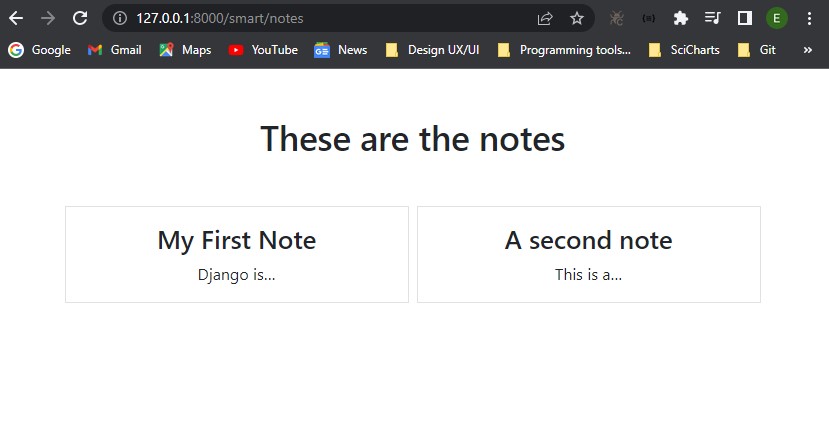
Amazing! This would look a little bit better if we could display some of the text of a note, but not all of it. We can use the truncatechars function to do this. Let's try it out.

After ‘h3’ type ->

{{note.text|truncatechars:10}}



There you go!



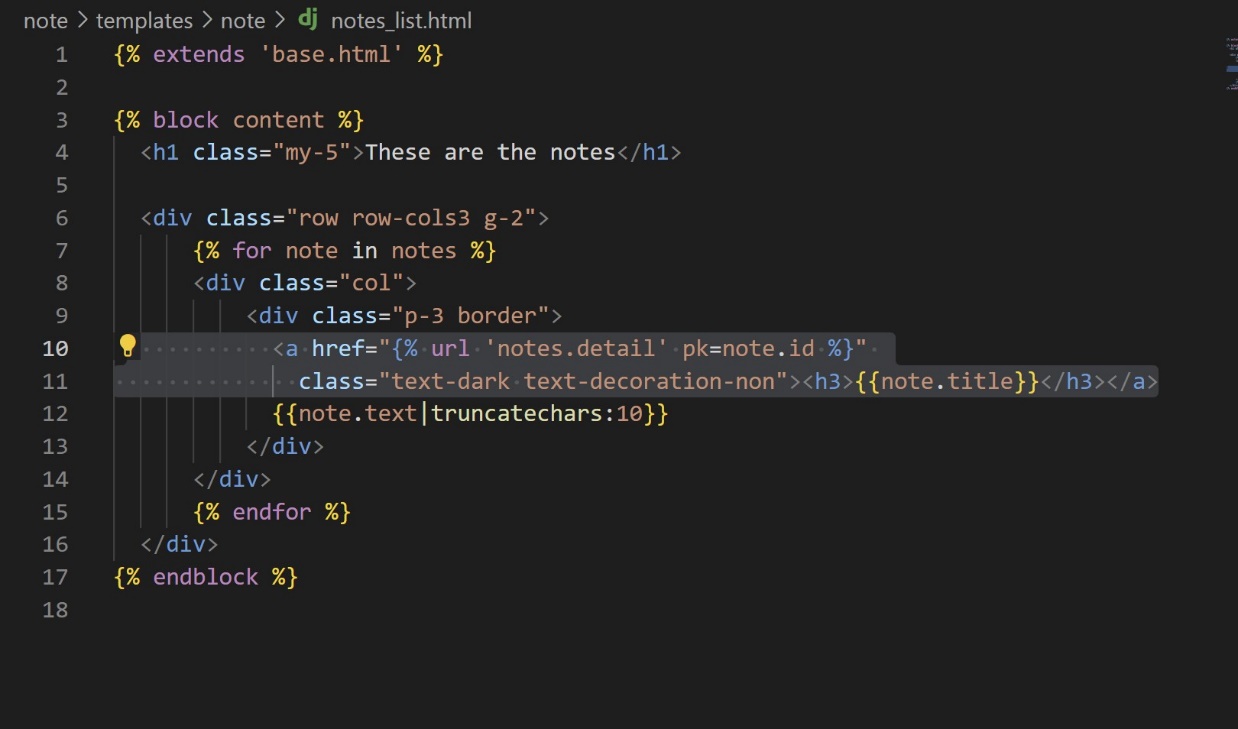
So what's happening here is that Django is taking the text and just displaying the first 10 characters plus the three dots. It looks a little bit better, doesn't it?

It's still missing a couple of things. We can't really access all the details of that particular note.  First, let's give a name to the detail URLs as well. Go to note/urls.py and in detail path add name="notes.detail".

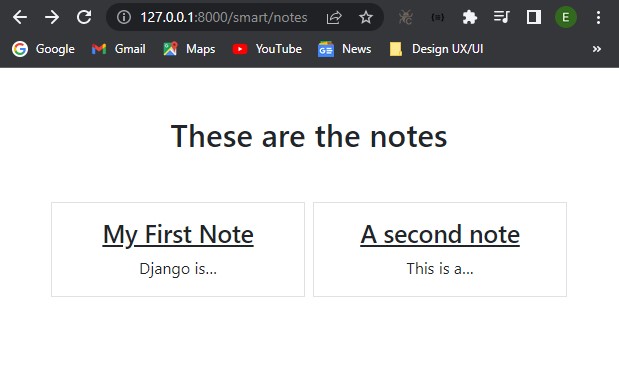
Now we can go back, and in the title, we can add the link and some style. Type ->

<a href="{% url 'notes.detail' pk=note.id %}"

class="text-dark text-decoration-non"><h3>{{note.title}}</h3></a>



Refresh.



Now in the note/notes.detail delete everything and add this ->

{% extends "base.html" %}

{% block content %}

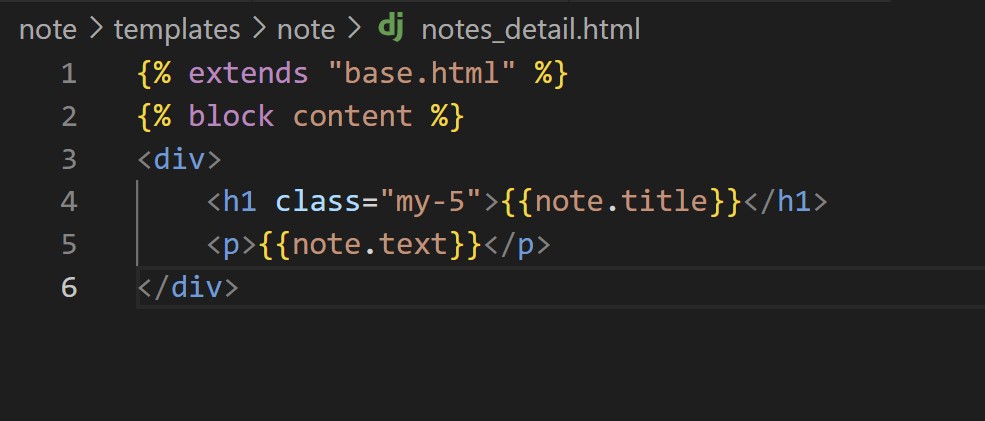
<div>

<h1 class="my-5">{{note.title}}</h1>

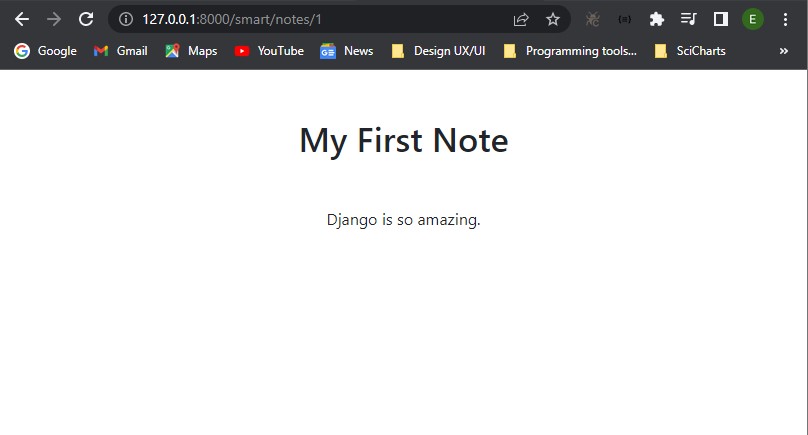
<p>{{note.text}}</p>

</div>

{% endblock %}



Let's go back. Refresh.



All done! Now you have style and dynamically generated links.