Simple Templates

For the time being, rename your original templates folder. It contains the templates for the final project along with the corresponding javascript files for the pages. At this moment we don't need them, but we don't want to get rid of them.

```
mv templates/ .templates/
```

We'll restore the folder later on.

Create an empty templates folder:

```
mkdir -p templates
```

and create two files:

```
touch ./templates/projects.html
touch ./templates/single-project.html
```

Using an editor or the command line, open each file and insert the corresponding text to each of them:

```
echo "<h1>Projects Template</h1>" > ./templates/projects.html
echo "<h1>Single-Project Template</h1>" > ./templates/single-project.html
```

Let Django know where to find these templates. Go to the project root folder and open the settings.py file:

```
TEMPLATES = [
    'BACKEND': 'django.templates.backends.django.DjangoTemplates',
    # add the templates folder to the base directory
    'DIRS': [
            str(BASE_DIR.joinpath('templates'))
        ],
    'APP_DIRS': True,
    'OPTIONS'; {
        'context_processors': [
            'django.template.context_processors.debug',
            'django.template.context_processors.request',
```

```
                ...
            ]
        }
    ]
```

Now, in order to render each template, we also need to modify the root project folder views.py file (devsite/views.py):

```
    from django.shortcuts import render
    from django.http import HttpResponse

    def projects(request):
        return HttpResponse('These are our current projects')

    def project(request, pk):
        return HttpResponse(f'This is project {pk}')
```

and change it to:

```
    from django.shortcuts import render
    from django.http import HttpResponse

    def projects(request):
        return render(request, 'projects.html')

    def project(request, pk):
        return render(request, 'single-project.html')
```

Next, open up the following link: Lorem Ipsum and copy a paragraph worth of content and place it in the templates/projects.html as a paragraph:

```
    <h1>Projects Template</h1>

    <p>Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do
eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit
esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat
cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est
laborum.</p>
```

run the server and test the templates:

```
    python manage.py runserver
```

in a browser, access the page:

```
    127.0.0.1:8000/projects
```

you should be able to open a page with the lorem ipsum text.

Complex Templates

It's possible to construct complex HTML templates from other templates using two main mechanisms: 1. Inclusion (include statement) 2. Template Inheritance

include statement

Let's include a simple navbar into the projects.html template:

Create the navbar.html and add the code:

```
<h1>LOGO</h1>
<hr>
```

At one point in the future, this navbar.html will include an actual icon. Let's continue and modify the projects template:

```
{% include 'navbar.html' %}     # Jinja template

<h1>Projects Template</h1>
<p>Lorem Ipsum ...

</p>
```

Access the projects page at 127.0.0.1/projects This is how the include statement works with Jinja

Template Extension

Template extension requires template inclusion (include statement). However, the main concept is more similar to inheritance in python. Let's see how this actually works.

For that, we need to create in our templates folder another template which we will call main.html. Open up a menu in your vscode IDE on your project files window and create a new file called main.html or using a command line type:

```
cd ./templates
touch main.html
```

Now type code into main.html. We will use all the styling in main.html to format all children html files that are extensions from main.html.

```
<html       # Wait for the indicator to show HTML in the cursor and then
click on it.
```

and press enter. It should populate the main.hml file with a snipet if we start typing html onto the first line of the new file that looks like this:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>Page Title</title>
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <link rel='stylesheet' type='text/css' media='screen' href='main.css'>
    <script src='main.js'></script>
</head>
<body>

</body>
</html>
```

and comment the <link rel='stylesheet ...> and <script src='main.js'...> lines and modify the main.html file as follows:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset='utf-8'>
    <meta http-equiv='X-UA-Compatible' content='IE=edge'>
    <title>DevSite</title>
```

```
    <meta name='viewport' content='width=device-width, initial-scale=1'>
    <!-- <link rel='stylesheet' type='text/css' media='screen'
href='main.css'> -->
    <!-- <script src='main.js'></script> -->
</head>
<body>
    <h1> This is the Title in the Main Template </h1>

    {% include 'navbar.html' %}

    <!-- This block content will place the content of the extending template
here  -->
    {% block content %}

    {% endblock content %}
    <p>FOOTER</p>
</body>
</html>
```

Now let's go back to our projects.html file and add the following code:

```
{% extends 'main.html' %}

{% block content %}         <!-- Don't copy this. Observe that the label
content is the same as in the main.html file -->

<h1>Projects Template</h1>
<p>Lorem Ipsum ...

</p>

{% endblock content %}
```

With these steps we have created an extensive template with main.html. The templates using the extensible template must use the statement:

```
{% extends '<extensible_template_name>.html' %}
```

The statements

```
{% block <label> %}
```

```
{% endblock <label> %}
```

indicate to the extending templates where their specific contents will be included into the extensible template.

Let's repeat the same procedure for the single project template:

```
{% extends 'main.html' %}

{% block content %}
<h1>Single Project Template</h1>
{% endblock content %}
```

Let's test the modifications by running the server and accessing the corresponding slugs in the site

```
python manage.py runserver
```

on the browser, access the corresponding slugs

```
127.0.0.1/projects
127.0.0.1/project/1
127.0.0.1/project/whatever
```

Summary

We showed the use of two important concepts: 1. Template inclusion by using the *include* statement in the templates. 2. Template extension, a very similar concept to inheritance, used to use the styling and properties of the parent templates.