

Lab 10 Part 2

Views & Templates & URLs

Step 1 Create a View to display Products & Categories

The first function-based view we write here is to display the products in each category if the user has selected a particular category. If the user has not selected a category, then all the products are displayed.

Open **shop/views.py** and add the following code:

```
shop > views.py > ...
1  from django.shortcuts import render, get_object_or_404
2  from .models import Category, Product
3
4  # Create your views here.
5  def prod_list(request, category_id=None):
6      category = None
7      products = Product.objects.filter(available=True)
8      if category_id:
9          category = get_object_or_404(Category, id=category_id)
10         products = Product.objects.filter(category=category, available=True)
11     return render(request, 'shop/category.html', {'category':category, 'prods':products})
```

Line 5: This view function has two parameters, the first one is always self which refers to the current class instance. The 2nd parameter is the category id which is optional in this case which is why it is set to None.

Line 6 we define a variable category to represent the category.

Line 7 A filter query filters the result, and all available products are returned in the query.

Lines 8-10: The If statement is used to check if the category_id parameter has a value. If it does, then the code on line 13 uses the get_object_or_404() function to return the category object. This get_object_or_404() function calls the Category model and gets the object from that. If that object or model does not exist, it raises the 404 error.

Line 11: Here we call the render function to render the template and we also pass in the data we need to display in the template (category & products).

Step 2 Templates

We will store our templates for this app inside the app itself. In VS Code create a folder inside the **shop** folder called **templates**. By default, the Django template loader will look within each app for a templates folder, so we do not need to specify the folder in settings.py. We only need to specify it in settings.py if we are using a templates folder at the project level.

Inside the **shop/templates** folder, create a new file called **base.html**. Copy-paste the following code into this file:

base.html

```
{% load static %}
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="description" content="{% block metadescription %}{% endblock %}">
    <title>{% block title %}{% endblock %}</title>
</head>
<body>
    {% include 'header.html' %}
    {% include 'navbar.html' %}
    {% block content %}
    {% endblock %}

    {% include 'footer.html' %}
</body>
</html>
```

Inside the **shop/templates** folder, create a new file called **header.html**. Copy-paste the following code into this file:

header.html

```
{% load static %}

<header>

    <div style="text-align: center;">

    </div>

</header>
```

The **header.html** file contains a link to an image called logo.png. In VS Code create a folder called **static** at the project level and inside this folder create a subfolder called **images**. Download the images folder from Moodle, extract it and copy the two images **banner.jpg** and **logo.png** into the **static/images** folder of your project.

Inside the **shop/templates** folder, create a new file called **navbar.html** and add the following code: The url is already mapped here but we will create it later.

navbar.html

```
<nav>
  <ul>
    <li><a href = "{% url 'shop:all_products' %}">All Products</a></li>
    <li>Your Cart</li>
  </ul>
</nav>
```

Inside the **shop/templates** folder, create a new file called **footer.html** and add the following code:

footer.html

```
<div>
  <p>&copy; Online Cushions. All Rights Reserved.</p>
</div>
```

Inside the **shop/templates** folder create a new folder called **shop** and create a new file called **category.html** inside this folder & copy-paste the following code into it:

category.html

```
{% extends "base.html" %}
{% load static %}
{% block metadescription %}
    {% if category %}
        {{ category.description|truncatewords:155 }}
    {% else %}
        Welcome to the cushion store where you can buy comfy and awesome cushions.
    {% endif %}
{% endblock %}
{% block title %}
    {% if category %}
        {{ category.name }} - Perfect Cushion Store
    {% else %}
        See Our Cushion Collection - Perfect Cushion Store
    {% endif %}
{% endblock %}
{% block content %}
    <!--Breadcrumb navigation-->
    {% if category %}
        <div>
            <div>
                <p><a href="{% url 'shop:all_products' %}">Our Product Collection</a> | {{category.name}}</p>
            </div>
        </div>
    {% endif %}
    <div>
        {% if category %}
            
        </div>
    <br>
```

```

    <div>
        <h1>{{category.name}}</h1>
        <p>{{category.description}}</p>
    </div>
{% else %}
    
</div>
<br>
<div>
    <h1>Our Products Collection</h1>
    <p>Finding the perfect cushion for your room can instantly add to the levels of comfort and
        sense of style throughout your home. They can transform the blandest of decors instantly
        by adding colour, softness and an air of cosiness. As well as being comfy and looking great
        our range of cushions will certainly embolden your décor with personalised charm.</p>
</div>
{% endif %}
<div>
    <div>
        {% for product in prods %}
            <div>
                <div>
                    <a class = "img" href=""></a>
                    <div>
                        <h4>{{product.name}}</h4>
                        <p>€{{product.price}}</p>
                    </div>
                </div>
            </div>
        {% endfor %}
    </div>
</div>

```

```
    <br>  
  </div>  
{% endblock %}
```

Step 3: URL Configuration

Create the **shop/urls.py** file and copy-paste the code provided below.


Note the new line where we create an application **namespace** for our app called shop with the `app_name` variable. This allows us to organize URLs by application and use the name when referring to them. The namespaces feature in Django helps us to avoid conflicts with third party apps or even multiple apps within your project.

```
from django.urls import path
from . import views

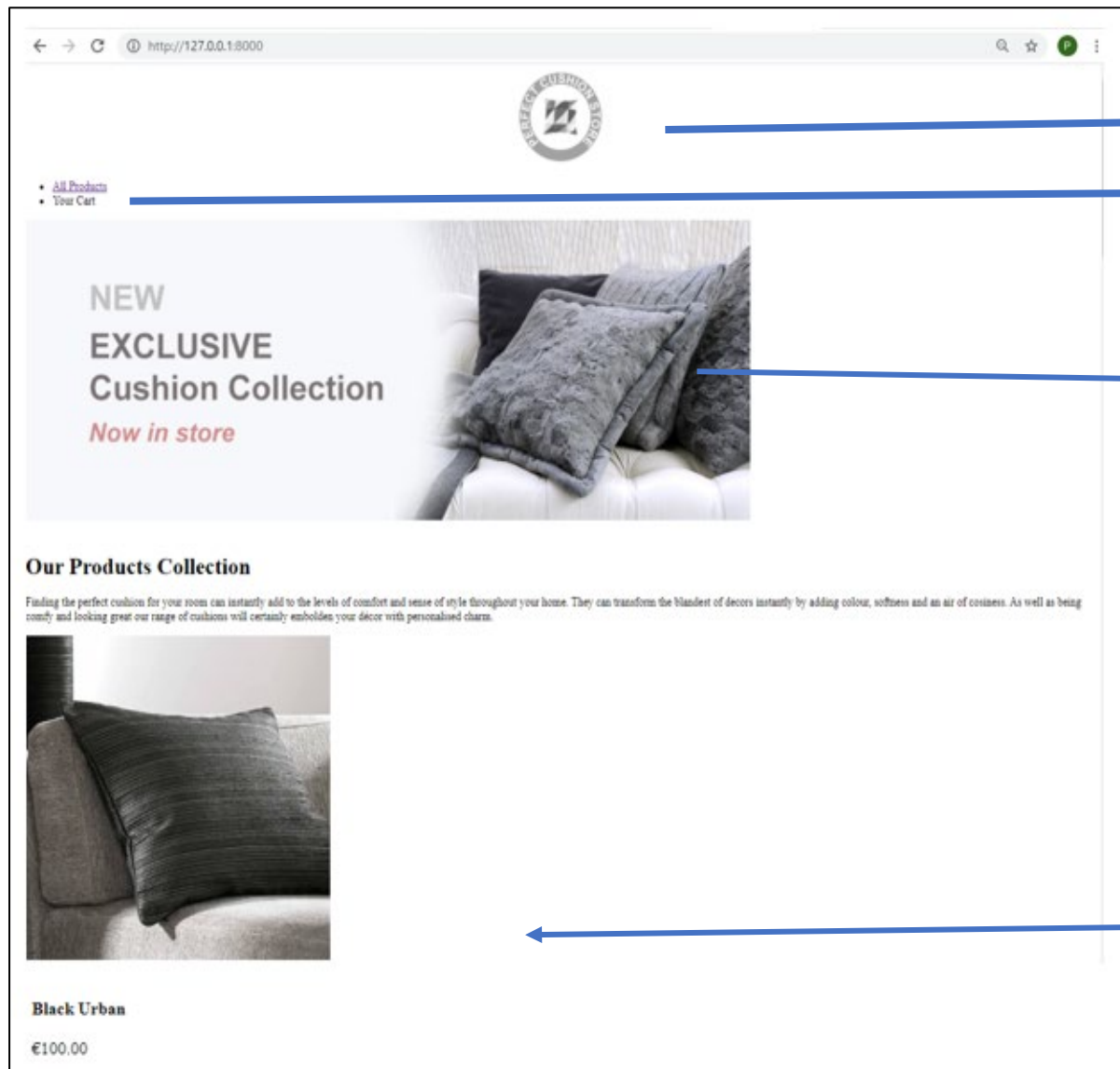
app_name = 'shop'

urlpatterns = [
    path('', views.prod_list, name = 'all_products'),
    path('<uuid:category_id>/', views.prod_list, name = 'products_by_category'
),
]
```

Open the **onlineshop/urls.py** file and add the following code:

```
shop_project >  urls.py
17  from django.urls import path, include
18  from django.conf import settings
19  from django.conf.urls.static import static
20
21  urlpatterns = [
22      path('admin/', admin.site.urls),
23      path('', include('shop.urls')),
24  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

If you run the server and access the website at <http://127.0.0.1:8080/> you should see the following screen that displays the logo and banner and the ten products:



header.html which includes logo.png

navbar.html

banner.jpg

category.html

Step 4: Context Processor

To add and enable the category links across the website we place them in the navbar but to make them work we need to create a new file called **context_processors.py**. Inside the **shop** app create a new file called **context_processors.py** and add in the following code:

```
shop > context_processors.py
1  from .models import Category
2
3  def menu_links(request):
4      links = Category.objects.all()
5      return {'links':links}
```

A **context processor** is a Python function that takes the request object as an argument and returns a dictionary that gets added to the request context. They come in handy when you need to make something available globally to all templates.

By default, when you create a new project using the startproject command, your project contains the following entries in the context_processors option inside the TEMPLATES setting:

```
63  'context_processors': [
64      'django.template.context_processors.debug',
65      'django.template.context_processors.request',
66      'django.contrib.auth.context_processors.auth',
67      'django.contrib.messages.context_processors.messages',
68  ],
```

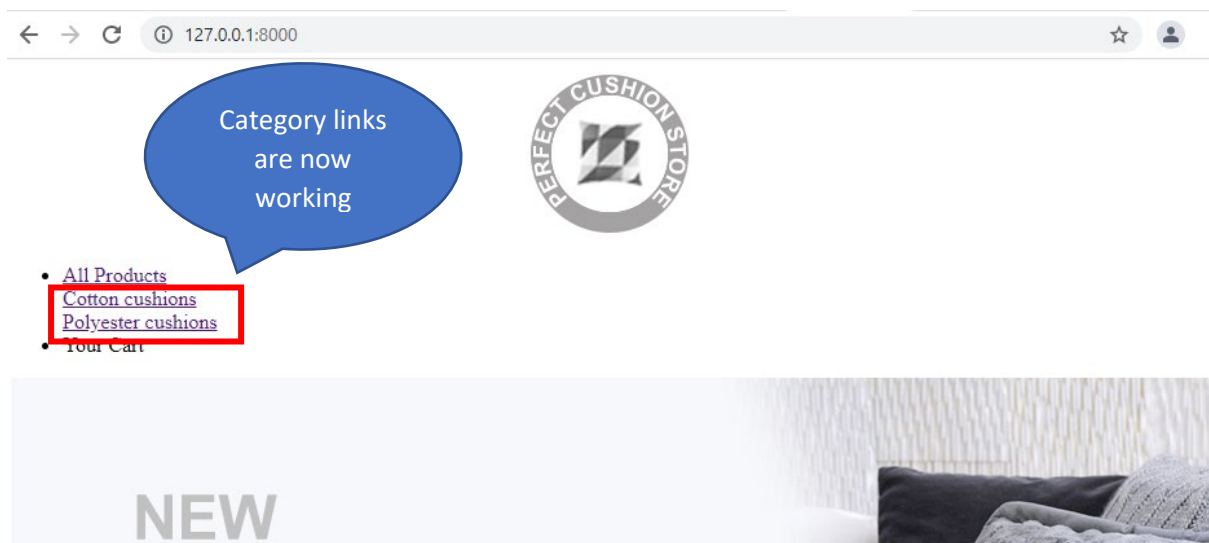
Add the following line of code to register this new context processor in **settings.py**:

```
63  'context_processors': [
64      'django.template.context_processors.debug',
65      'django.template.context_processors.request',
66      'django.contrib.auth.context_processors.auth',
67      'django.contrib.messages.context_processors.messages',
68      'shop.context_processors.menu_links',
69  ],
```

Open the **navbar.html** file and add in the following code:

```
1 <nav>
2   <ul>
3     <li><a href="{% url 'shop:all_products' %}">All Products</a></li>
4     {% for cat in links %}
5     <a href = "{{ cat.get_absolute_url }}"> {{cat.name}}</a>
6     <br>
7     {% endfor %}
8     <li>Your Cart</li>
9   </ul>
10 </nav>
```

Run the server and you will see that the category links in the navbar are now there and working. If you click on the link for Cotton cushions all the cushions in the cotton category are displayed. If you click on the link for Polyester cushions all the cushions in the polyester category are displayed.



Step 5: Create Product Template

Now we will create the Product template.

In the **shop/templates/shop** folder create a new template called **product.html** and copy-paste the following code:

```
{% extends "base.html" %}
{% load static %}
{% block metadescription %}
    {{ product.description|truncatewords:155 }}
{% endblock %}
{% block title %}
    {{ product.name }} - Perfect Cushion Store
{% endblock %}
{% block content %}
    <div>
        <div>
            <p><a href="{% url 'shop:all_products' %}">Home</a> | <a href="{{
product.category.get_url }}">{{product.category}}</a> |
{{product.name}}</p>
        </div>
        <div>
            <br>
            <div>
                <div>
                    <div>
                        
                    </div>
                </div>
                <div>
                    <div>
                        <h1>{{product.name}}</h1>
                        <p>€{{product.price}}</p>
                        <p>Product Description</p>
                        <p>{{product.description}}</p>
                        {% if product.stock == 0%}
                            <p><b>Out of Stock</b></p>
                        {% else %}
                            <a href="">Add to Cart</a>
                        {% endif %}
                    </div>
                </div>
            </div>
        </div>
    </div>
{% endblock %}
```

Step 6: Create View & URL for Product Detail

Open the **views.py** file and add in the following code to create a function-based view called **product_detail**.

```
13 def product_detail(request, category_id, product_id):
14     product = get_object_or_404(Product, category_id=category_id, id=product_id)
15     return render(request, 'shop/product.html', {'product':product})
```

Open the **shop/urls.py** file and add in the following path:

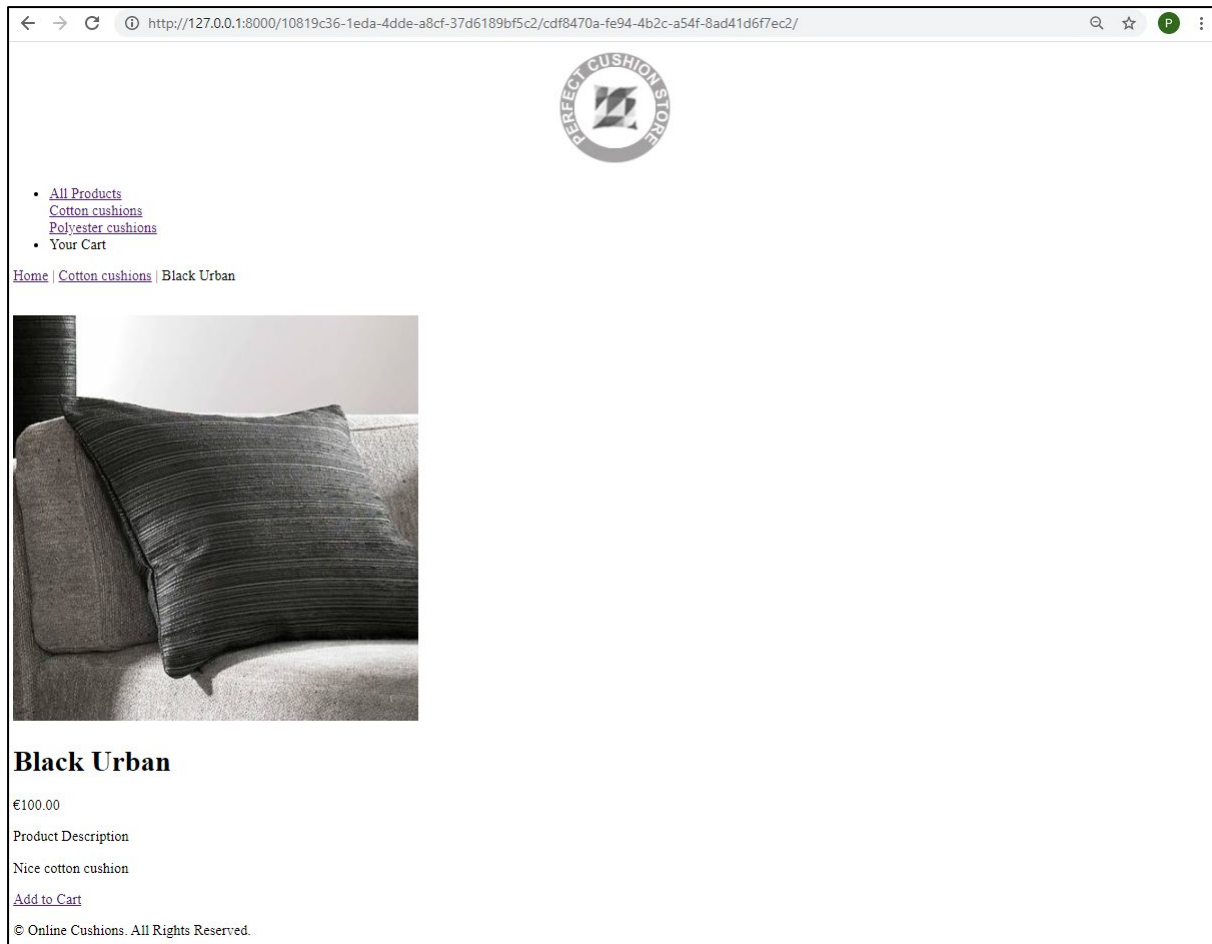
```
6 urlpatterns = [
7     path('', views.prod_list, name = 'all_products'),
8     path('<uuid:category_id>/', views.prod_list, name = 'products by category'),
9     path('<uuid:category_id>/<uuid:product_id>/', views.product_detail, name = 'product_detail'),
10 ]
```

Open **category.html** and add in the link to the product details page by calling the **get_absolute_url** function:

```
49     {% for product in prods %}
50         <div>
51             <div>
52                 <a class = "img" href="{{product.get_absolute_url}}"> img src="{{product.image.url}}" alt="{{product.name}}"></a>
53             <div>
54                 <h4>{{product.name}}</h4>
55                 <p>€{{product.price}}</p>
56             </div>
57         </div>
58     </div>
59     {% endfor %}
```

Run the server and access the home page. Now if you click on the image of a cushion you are taken to the product detail page for that product as shown below.

The header, navbar, product and footer are displayed.



Step 7: Commit the changes and push them to the lab10 repo

Stop the server and run the following git commands to update the local and remote repositories:

git add -A

git commit -m "lab 10 part 2 commit"

git push -u origin main