Lab 10 Part 4

Paginator & Search

Step 1 Paginator

In this first exercise we will implement the functionality for a paginator to allow just 6 products to be displayed per page and we will add a page number at the bottom of the page if the number of products is above 6.

As part of the Django's common Web application tools, Django offers several classes to manage paginated data. The paginator classes live in django.core.paginator. We will be working mostly with the Paginator, EmptyPage & InvalidPage classes.

Open **shop/views.py** and add the code at line 4 to import the 3 Paginator classes.

shop > views.py

```
1  from django.shortcuts import render, get_object_or_404
2  from .models import Category, Product
3  from django.core.paginator import Paginator, EmptyPage, InvalidPage
```

Next, add the following block of code to the function **prod_list**:

```
6  def prod_list(request, category_id=None):
7      category = None
8      products = Product.objects.all().filter(available = True)
9      if category_id:
10         category = get_object_or_404(Category, id = category_id)
11         products = Product.objects.filter(category = category, available = True)
12
13     paginator = Paginator(products, 6)
14     try:
15         page = int(request.GET.get('page','1'))
16     except:
17         page = 1
18     try:
19         products = paginator.page(page)
20     except (EmptyPage,InvalidPage):
21         products = paginator.page(paginator.num_pages)
22
23     return  render(request,'shop/category.html',{'category':category, 'prods':products})
```

Line 13: We are telling Paginator to paginate the products QuerySet in pages of 6. Given that we have 10 products in the database, this will create a 2 pages result. The first page with 6 products each and the second page with 4 products.

Line 15: The page query string value is fetched with the request.GET.get() function which returns the current page number.

Line 17: If there is no page number then it is set to the first page i.e. 1.

Line 19: The page() method is used to return a given page of the paginated results, which is an instance of Page. Below that we have two exception statements for EmptyPage and InvalidPage.

Line 21: If page is out of range deliver the last page of product results.

Step 2 Display Page Numbers in Template

Towards the end of **category.html** add a new div and create a class inside this div called mx-auto. Copy-paste the highlighted block of code to the location highlighted below:

```
                </div>
            {% endfor %}
        </div>
        <div class="mx-auto">
            {% if prods.paginator.num_pages > 1 %}
                <hr>
                <div class="text-center">
                    {% for pg in prods.paginator.page_range %}
                        <a href="?page={{pg}}" class="btn btn-light btn-sm {%
                        if products.number == pg %}active{% endif
                        %}">{{pg}}</a>
                    {% endfor %}
                </div>
            {% endif %}
        </div>
        <br>
    </div>
{% endblock %}
```

The code in the template uses an if statement to check that the number of pages is greater than 1. If so, then a for loop is used to iterate through the pages in the page range and a hyperlink to each page number is displayed with the currently selected page marked as active.

Refresh the server and you will see that pagination is working:

Step 3 Search Functionality

In this part of the exercise, we will build a basic search to allow us to look for the product's name or the product description.

Create a new app called **search_app**

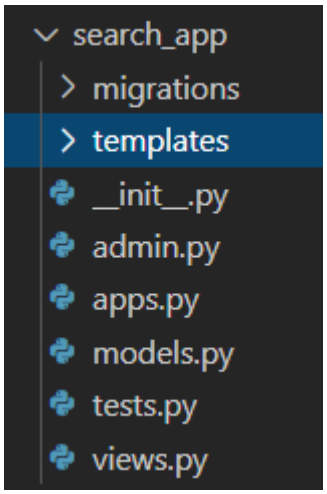
---

python manage.py startapp search_app

Open **settings.py** and register this app:

```
33  INSTALLED_APPS = [
34      'django.contrib.admin',
35      'django.contrib.auth',
36      'django.contrib.contenttypes',
37      'django.contrib.sessions',
38      'django.contrib.messages',
39      'django.contrib.staticfiles',
40      'accounts',
41      'shop',
42      'search_app',
43  ]
```

In VS Code right click on the **search_app** folder and create a **templates** folder:



The following code for the search box is already provided in **navbar.html**:

```
24          <form class="d-flex" action="" method="get">
25            <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search" name="q">
26            <button class="btn btn-secondary my-2 my-sm-0" type="submit">Search</button>
27          </form>
```

Open **search_app/views.py** and add the following code. We used code like this before when implementing a search in our Books project.

```
search_app > views.py > ...
1   from shop.models import Product
2   from django.views.generic import ListView
3   from django.db.models import Q
4
5
6   class SearchResultsListView(ListView):
7       model = Product
8       context_object_name = 'product_list'
9       template_name = 'search.html'
10
11      def get_queryset(self):
12          query = self.request.GET.get('q')
13          return Product.objects.filter(Q(name__icontains=query) | Q(description__icontains=query))
14
15      def get_context_data(self, **kwargs):
16          context = super(SearchResultsListView, self).get_context_data(**kwargs)
17          context['query'] = self.request.GET.get('q')
18          return context
```

We have added an additional method in this class-based view to pass the search term that the user enters at the keyboard into the template. We do this by overriding the get_context_data method. Often you need to present some extra information to the template beyond that provided by the generic view.

Inside the new app **search_app**, create a new **urls.py** file and add copy-paste the following code:

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

app_name='search_app'

urlpatterns = [
        path('', SearchResultsListView.as_view(), name='searchResult'),
]
```

Open **onlineshop/urls.py** and add in the following line of code:

```
16  from django.contrib import admin
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      path('search/', include('search_app.urls')),
25  ] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

Create the **search.html** template inside the templates folder of the **search_app** and copy-paste the following code. This template will display the product(s) that match the search term entered by the user:

```
{% extends "base.html" %}
{% load static %}
{% block metadescription %}
    We have a variety of stunning and comfy cushions. Look for the one that suits your needs.
{% endblock %}
{% block title %}
    Search - Perfect Cushion Store
{% endblock %}
{% block content %}
    <div>
        <p class="text-center my_search_text">You have searched for: <b>"{{ query }}"</b></p>
    </div>
    <div class="container">
        <div class="row mx-auto">
            {% for product in product_list %}
                <div class="my_bottom_margin col-9 col-sm-12 col-md-4 col-md-
                 12 col-lg-4">
                    <div class="card text-center" style="min-width: 18rem;">
                        <a href="{{product.get_absolute_url}}"><img
                        class="card-img-top my_image"
                            src="{{product.image.url}}"
                               alt="{{product.name}}"></a>
                        <div class="card-body">
                            <h4>{{product.name}}</h4>
                            <p>€{{product.price}}</p>
                        </div>
                    </div>
                </div>
                {% empty %}
```

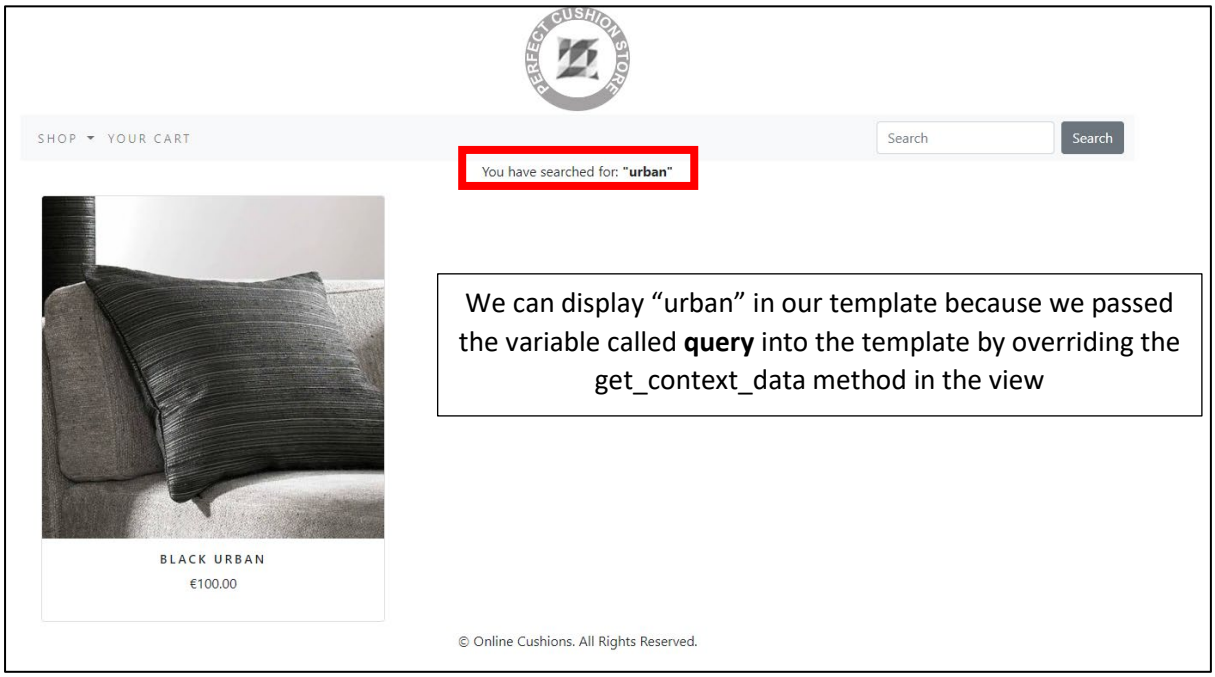
```
                <div class="row mx-auto">
                    <p class="text-center my_search_text">0 results found.</p>
                </div>
            {% endfor %}
        </div>
    </div>
{% endblock %}
```

Finally, we need to add the action value to the form of the **navbar.html** template. Add the following url at the location shown below:

```
24          <form class="d-flex" action="{% url 'search_app:searchResult' %}" method="get">
25            <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search" name="q">
26            <button class="btn btn-secondary my-2 my-sm-0" type="submit">Search</button>
27          </form>
```

Run the server and search for the words elephant or urban and note the results. When the product displays you can click on it to view the details.

In the output shown below we entered “urban” in the search bar which gave us one result.



Step 4: Commit the changes and push your code to the lab 10 repo.

git add -A

git commit -m “lab 10 part 4 commit”

git push -u origin main