Lab 9 Part 5

Search Functionality

Search is a fundamental feature of most websites and certainly anything e-commerce related like our Bookstore. In this exercise we will learn how to implement a basic search with forms and filters and then improve it with additional. We only have three books in our database now but the code here will scale to as many books as we'd like.

Search functionality consists of two parts: a form to pass along a user search query and then a results page that performs a filter based on that query. Determining "the right" type of filter is where search becomes interesting and hard. But first we need to create both a form and the search results page.

We could start with either one at this point, but we will configure the filtering first and then the form.

Search Results Page

We will start with the search results page. As with all Django pages that means adding a dedicated URL, view, and template.

Add the code for SearchResultsListView in **book/viewsy.py** as shown below:

```
16  class SearchResultsListView(ListView): # new
17      model = Book
18      context_object_name = 'book_list'
19      template_name = 'books/search_results.html'
```

Within **books/urls.py** add a search/ path that will take a view called SearchResultsListView and has a URL name of search_results.

books > urls.py

```
1  from django.urls import path
2  from .views import BookListView, BookDetailView, SearchResultsListView
3
4  urlpatterns = [
5      path('', BookListView.as_view(),name='book_list'),
6      path('<uuid:pk>', BookDetailView.as_view(), name='book_detail'),
7      path('search/', SearchResultsListView.as_view(), name='search_results'),
8  ]
```

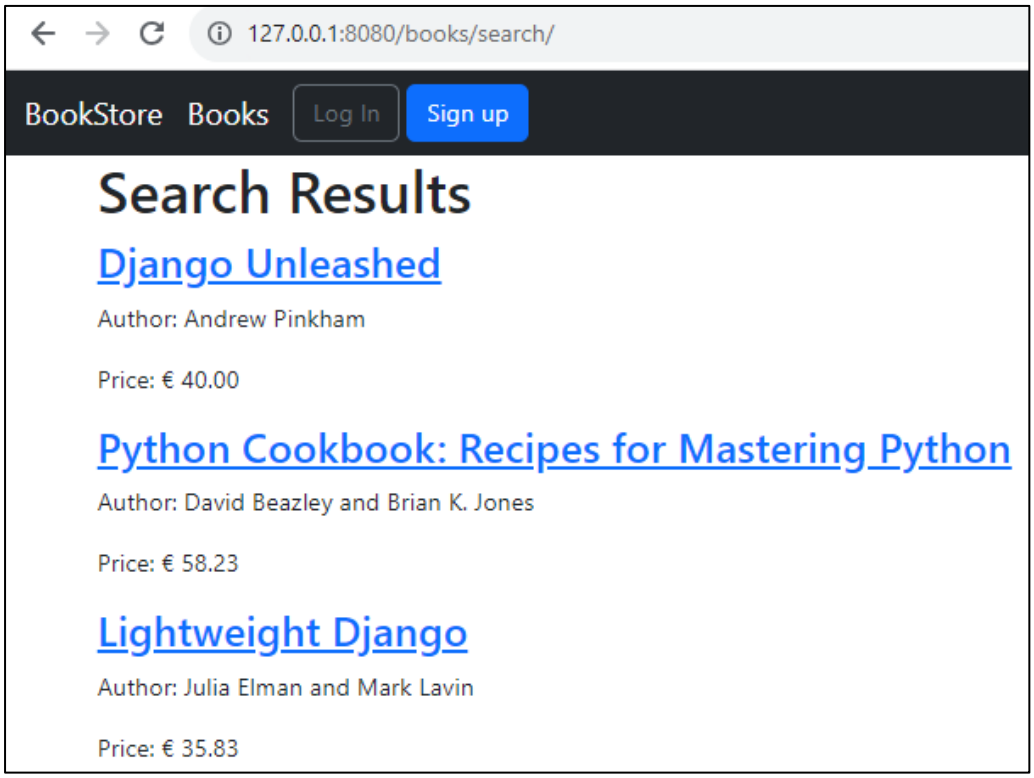
Create the template **search_results.html** inside the **templates/books** folder

Copy the following code into the file:

```
{% extends 'base.html' %}
{% block title %}Search{% endblock title %}
{% block content %}
<h1>Search Results</h1>
  {% for book in book_list %}
    <div>
      <h3><a href="{{ book.get_absolute_url }}">{{ book.title }}</a></h3>
      <p>Author: {{ book.author }}</p>
      <p>Price: € {{ book.price }}</p>
    </div>
  {% endfor %}
{% endblock content %}
```

The search results page is now available at:

http://127.0.0.1:8080/books/search/



Basic Filtering

In Django a QuerySet is used to filter the results from a database model. Currently our search results page doesn't feel like one because it is outputting all results from the Book model. Ultimately, we want to run the filter based on the user's search query, but first we'll work through multiple filtering options.

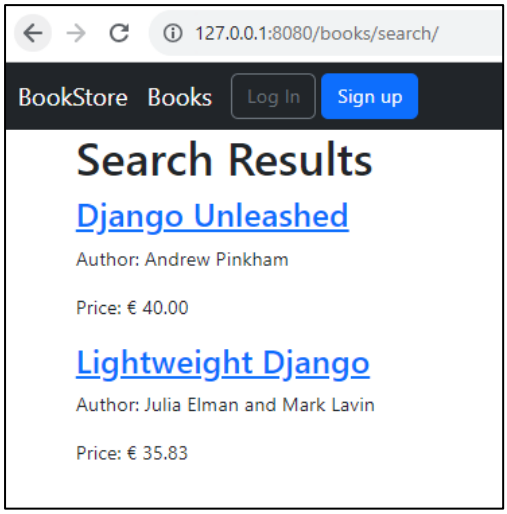
It turns out there are multiple ways to customize a queryset but to keep things simple, we can add a filter by adding just one line of code **to books/views.py**.

Add the line of code highlighted below to **books/views.py**

```
16  class SearchResultsListView(ListView): # new
17      model = Book
18      context_object_name = 'book_list'
19      template_name = 'books/search_results.html'
20      queryset = Book.objects.filter(title__icontains='Django') # new
```

We can override the default queryset attribute on ListView which by default shows all results. The queryset documentation is quite robust and detailed, but often using contains (which is case sensitive) or icontains (which is not case sensitive) are good starting points. We will implement the filter based on the title that "contains" the name "Django".

Refresh the search results page and now only the books with the title containing "Django" is displayed.



For basic filtering most of the time the built-in queryset methods of filter(), all(), get(), or exclude() will be sufficient.

Q Objects

Using filter() is powerful and it's even possible to chain filters together such as search for all titles that contain "beginners" and "django". However often you'll want more complex lookups that can use "OR" not just "AND"; that's when it is time to turn to Q objects. A Q object (django.db.models.Q) is an object used to encapsulate a collection of keyword arguments. These keyword arguments are specified as "Field lookups".

Here is an example where we set the filter to look for a result that matches a title of either "Django" or "Python". It's as simple as importing Q at the top of the file and then subtly tweaking our existing query. The | symbol represents the "or" operator. We can filter on any available field: not just title but also author or price as desired.

Delete the line of code shown at line 19 below from **views.py**

```
15  class SearchResultsListView(ListView):
16      model = Book
17      context_object_name = 'book_list'
18      template_name = 'search_results.html'
19      queryset = Book.objects.filter(title__icontains='Django')
```

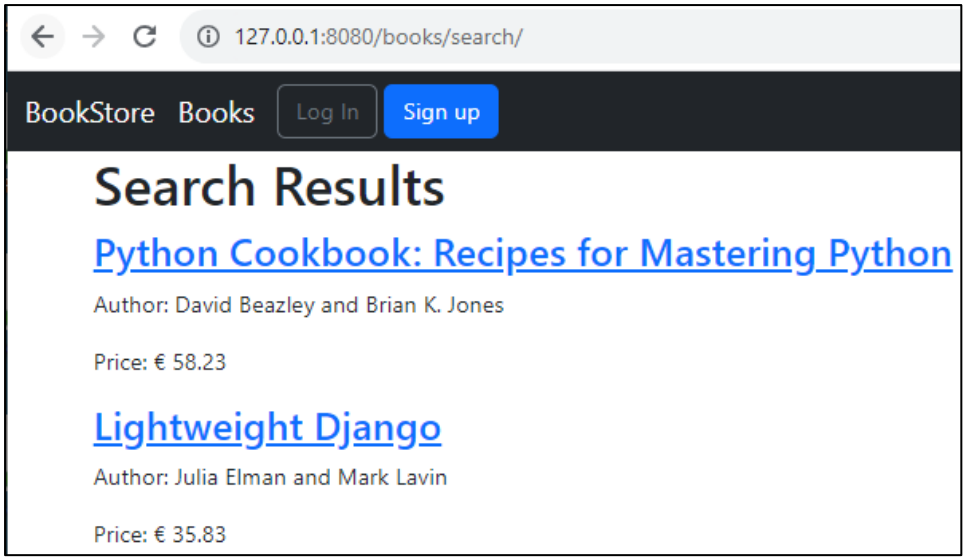
books > views.py > ...

```
1   from django.views.generic import ListView, DetailView
2   from .models import Book
3   from django.db.models import Q
4
5   # Create your views here.
6   class BookListView(ListView):
7       model = Book
8       context_object_name = 'book_list'
9       template_name = 'books/book_list.html'
10
11  class BookDetailView(DetailView):
12      model = Book
13      template_name = 'books/book_detail.html'
14
15  class SearchResultsListView(ListView):
16      model = Book
17      context_object_name = 'book_list'
18      template_name = 'books/search_results.html'
19
20
21      def get_queryset(self):
22          return Book.objects.filter(
23              Q(title__icontains='Lightweight') | Q(author__icontains='Jones'))
```

Add the import shown here on line 3 to books/viewy.py

Add the function shown here on lines 21-23 to books/views.py

Refresh the search results page to see the new result.



Now let's turn our attention to the corresponding search form so that rather than hardcode our filters in we can populate them based on the user's search query.

Forms

Fundamentally a web form is simple: it takes user input and sends it to a URL via either a GET or POST method. The first issue is sending the form data: where does the data go and how do we handle it once there? Not to mention there are numerous security concerns whenever we allow users to submit data to a website.

There are only two options for "how" a form is sent: either via GET or POST HTTP methods. A POST bundles up form data, encodes it for transmission, sends it to the server, and then receives a response. Any request that changes the state of the database–creates, edits, or deletes data–should use a POST.

A GET bundles form data into a string that is added to the destination URL. GET should only be used for requests that do not affect the state of the application, such as a search where nothing within the database is changing, basically we're just doing a filtered list view.

If you look at the URL after visiting Google.ie you'll see your search query in the actual search results page URL itself.

Search Form

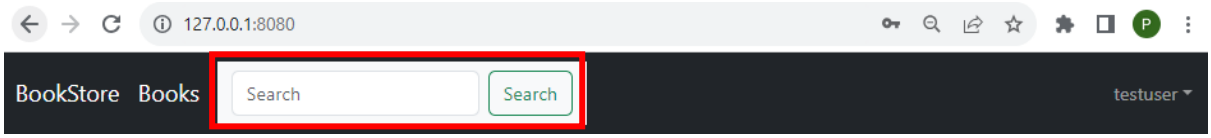
Let's add a basic search form to the navbar using bootstrap

Copy-paste the code below into **base.html** just after the closing button tag:

```
<nav class="navbar navbar-light bg-light">
   <div class="container-fluid">
      <form class="d-flex" action="{% url 'search_results' %}" method ="get">
        <input name = "q" class="form-control me-2" type="search"
         placeholder="Search" aria-label="Search">
        <button class="btn btn-outline-success" type="submit">Search</button>
      </form>
   </div>
</nav>
```

```
20          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-targ
21            <span class="navbar-toggler-icon"></span>
22          </button>
23          <nav class="navbar navbar-light bg-light">
24            <div class="container-fluid">
25              <form class="d-flex" action="{% url 'search_results' %}" method ="get">
26                <input name = "q" class="form-control me-2" type="search" placeholder="Searc
27                <button class="btn btn-outline-success" type="submit">Search</button>
28              </form>
29            </div>
30          </nav>
31          <div class="collapse navbar-collapse" id="navbarSupportedContent">
32            {% if user.is_authenticated %}
```

Refresh the home page and you should see the new search form in the navbar:



The second part of the form is the input which contains the user search query. We provide it with a variable name, q, which will be later visible in the URL and available in the views file.

Try inputting a search, for example for “hello.” Upon hitting Return you are redirected to the search results page. Note the URL contains the search query http://127.0.0.1:8080/books/search/?q=hello

However, the results haven’t changed! And that’s because our SearchResultsListView still has the hardcoded values from before. The last step is to take the user’s search query, represented by q in the URL, and pass it into the actual search filters.
Change the code for the function get_queryset() in **views.py**

```
21      def get_queryset(self):
22          query = self.request.GET.get('q')
23          return Book.objects.filter(
24              Q(title__icontains=query) | Q(author__icontains=query))
```

We added a query variable that takes the value of q from the form submission. Then updated our filter to use query on either a title or an author field.

Refresh the search results page–it still has the same URL with our query–and the result is expected: no results on either title or author for “hello”.

Go back to the homepage and try a new search such as for “django” to see the complete search functionality in action.

Run the following git commands to update the local and remote repositories:
git add -A
git commit -m “lab 9 part 5 commit”
git push -u origin main