Lab 9 Part 2

UUIDs

In this exercise we will create a Book model in our system with fields for the id, title, author, and price. Using the pk field in the URL of our DetailView is quick and easy, but not ideal for a real-world project. Up to now the pk we have used is Django auto-incrementing id. Among other concerns, it tells a potential hacker exactly how many records you have in your database; it tells them exactly what the id is which can be used in a potential attack.

A better approach is to use a UUID (Universally Unique IDentifier) which Django supports via a dedicated UUIDField.

Create an app called **books**. From the command line, quit the server & use the startapp command as shown below:

---

python manage.py startapp books

Settings.py

1. Register the app in **settings.py**

Update the **books/models.py** file to include our new Book model. You can copy-paste the following code:

```
import uuid
from django.db import models
from django.urls import reverse


# Create your models here.
class Book(models.Model):
    id = models.UUIDField(primary_key=True,
        default=uuid.uuid4,
        editable=False)
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=200)
    price = models.DecimalField(max_digits=6, decimal_places=2)
    date_publication = models.DateField(blank=False, null=False)

    def __str__(self):
        return self.title

    def get_absolute_url(self):
        return reverse('book_detail', args=[str(self.id)])
```

Note the uuid import at the top and note the id field is a UUIDField that is now the primary key. We also use uuid4 for the encryption.

Migrations

Now that our new database model is created, we need to create a new migration record for it. Run the following commands to create the new database table:

python manage.py makemigrations books

python manage.py migrate

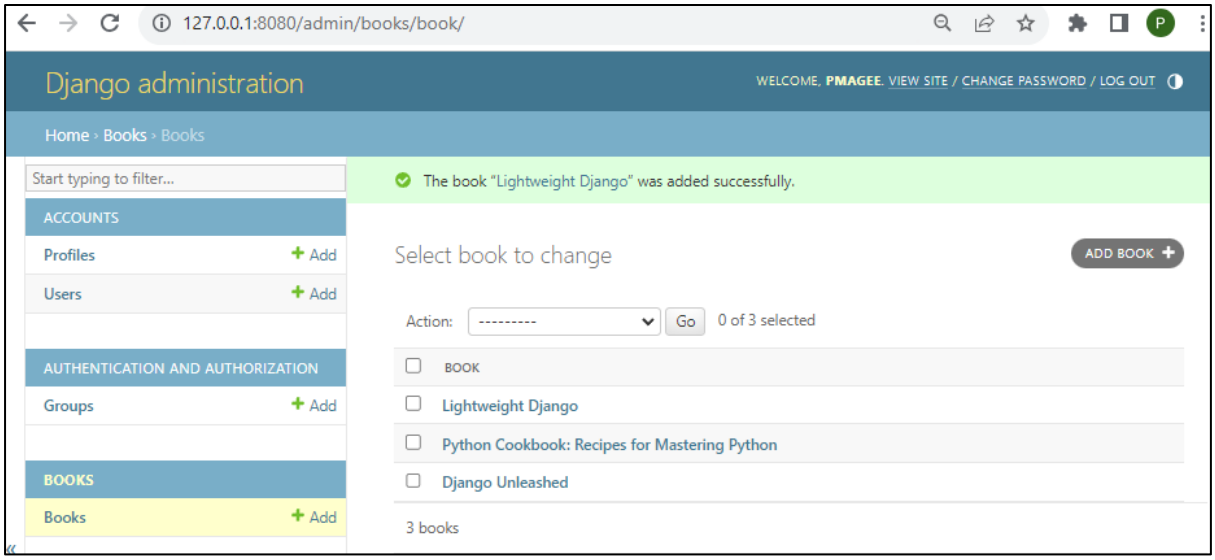
Admin

Copy-paste the following code to the **books/admin.py** file so that we can access our data in Django admin.

```
from .models import Book

admin.site.register(Book)
```

Run the server, log into Django admin, and add in three Book objects.

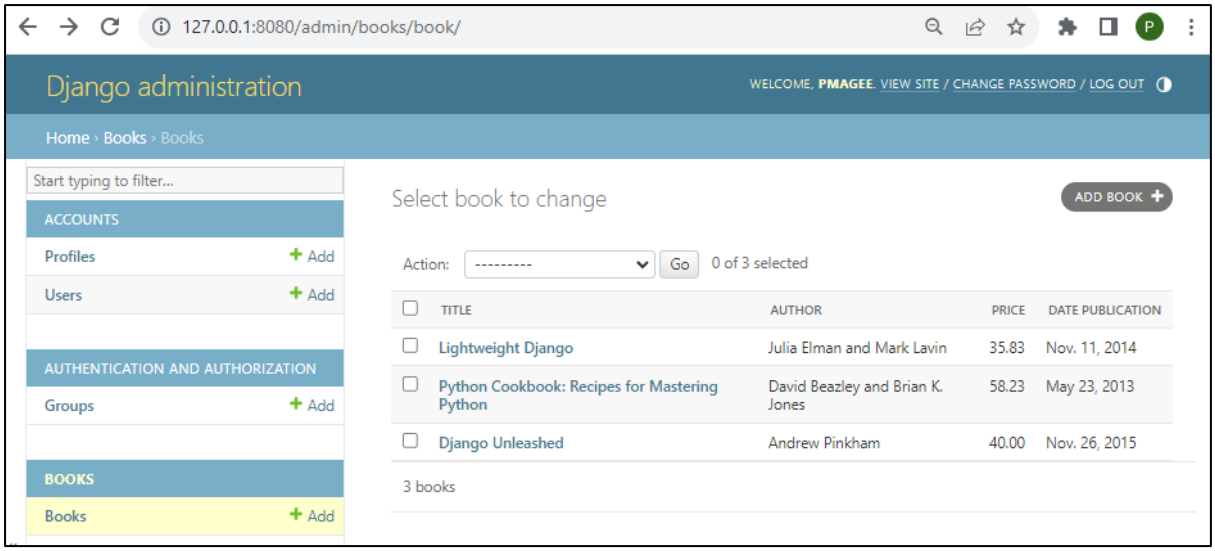
After clicking on the “Save” button we are redirected to the main Books page which only shows the book titles.



Next, we update the **books/admin.py** file to specify which fields we also want displayed. Add the code highlighted below:

```
books > admin.py > ...
1   from django.contrib import admin
2   from .models import Book
3
4   # Register your models here.
5
6   class BookAdmin(admin.ModelAdmin):
7       list_display = ("title", "author", "price", "date_publication")
8
9   admin.site.register(Book, BookAdmin)
```

Refresh the admin page and you will see the list of books with 4 fields displayed



Now that our database model is complete, we need to create the necessary views, URLs, and templates so we can display the information on our web application.

URLs

Create the **urls.py** file inside the **books** app and copy-paste the following code:

```
from django.urls import path
from .views import BookListView, BookDetailView

urlpatterns = [
    path('', BookListView.as_view(), name='book_list'),
    path('<uuid:pk>/', BookDetailView.as_view(), name='book_detail'),
]
```

Add the code highlighted below to **booksproject/urls.py**:

```
16  from django.contrib import admin
17  from django.urls import path, include
18
19
20  urlpatterns = [
21      path('admin/', admin.site.urls),
22      path('books/', include('books.urls')),
23      path('accounts/', include('accounts.urls')),
24      path('accounts/', include('django.contrib.auth.urls')),
25      path('', include('pages.urls')),
26  ]
```

Templates

Inside the **templates** folder create another folder called **books** to hold all templates related to the **books** app. Inside the **templates/books** folder create two files **book_list.html** and **book_detail.html..**

Copy the following code in to **book_list.html**

```
{% extends 'base.html' %}
{% block title %}Books{% endblock title %}
{% block content %}
   {% for book in object_list %}
      <div>
         <h2><a href="">{{ book.title }}</a></h2>
      </div>
   {% endfor %}
{% endblock content %}
```

Copy the following code in to **book_detail.html**

```
{% extends 'base.html' %}
{% block title %}{{ book.title }}{% endblock title %}
{% block content %}
  <div class="book-detail">
    <h2>{{ book.title }}</h2>
    <p>Author: {{ book.author }}</p>
    <p>Price: {{ book.price }}</p>
    <p>Date of Publication: {{ book.date_publication }}</p>
  </div>
{% endblock content %}
```

Views

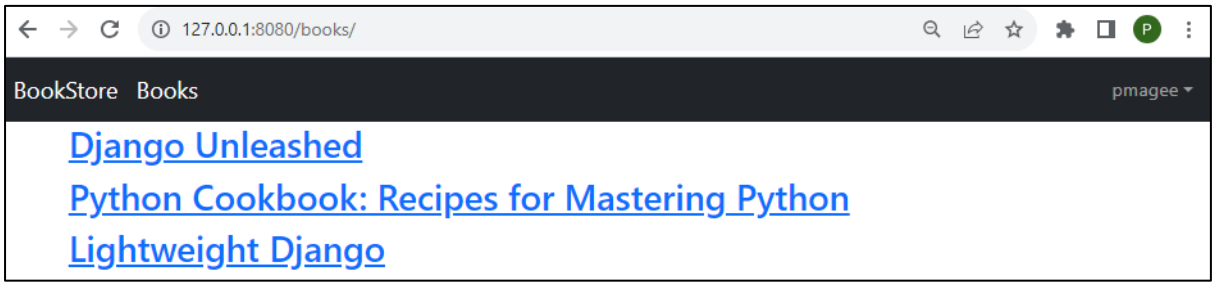
Open **views.py**, delete the import at the top of the file and copy-paste the following code:

```
from django.views.generic import ListView, DetailView
from .models import Book

# Create your views here.
class BookListView(ListView):
    model = Book
    context_object_name = 'book_list'
    template_name = 'books/book_list.html'

class BookDetailView(DetailView):
    model = Book
    template_name = 'books/book_detail.html'
```

Add the following two urls to **base.html**:

```
19          <a class="navbar-brand" href="{% url 'home' %}">BookStore</a>
20            <a class="navbar-brand" href="{% url 'book_list' %}">Books</a>
```

If you go to http://127.0.0.1:8080/ and click on the **Books** item in the nav bar, you will see the list of books displayed:



Individual Book Page

Add the following url to **book_list.html** to create a link for individual book pages:

templates > books > book_list.html > ...

```
{% extends 'base.html' %}
{% block title %}Books{% endblock title %}
{% block content %}
    {% for book in object_list %}
        <div>
            <h2><a href="{% url 'book_detail' book.pk %}">{{ book.title }}</a></h2>
        </div>
    {% endfor %}
{% endblock content %}
```

Refresh the book list page and the links are now all clickable and direct to the correct individual book page. Also notice the book detail page has a UUID in the URL.

get_absolute_url

The get_absolute_url() method in **models.py** sets a canonical URL for the model. It is also required when using the reverse() function which is commonly used.

Each of our model objects is displayed only on a single webpage, which means that each model object effectively has a single, canonical URL to access its full information. We can use the method get_absolute_url() to ask the model for its URL directly which results in nicer shorter code. Django not only recommends having a method on each model to create a canonical URL but expects us to name this method get_absolute_url()

We already added it to our **models.py** when we created the Book model. To use it we update the template **book_list.html**. Currently our a href link is using {% url 'book_detail' book.pk %}. However, we can instead use get_absolute_url directly which already has the pk passed in.

Replace the code you just entered on line 6 with the following:

templates > books > book_list.html > ...

```
1  {% extends 'base.html' %}
2  {% block title %}Books{% endblock title %}
3  {% block content %}
4      {% for book in object_list %}
5          <div>
6              <h2><a href="{{ book.get_absolute_url }}">{{ book.title }}</a></h2>
7          </div>
8      {% endfor %}
9  {% endblock content %}
```

There is no need to use the **url** template tag either, just one canonical reference that can be changed, if needed, in the **books/models.py** file and will propagate throughout the project from there. This is a cleaner approach and should be used whenever you need individual pages for an object.

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

git add -A

git commit -m “lab 9 part 2 commit”

git push -u origin main