

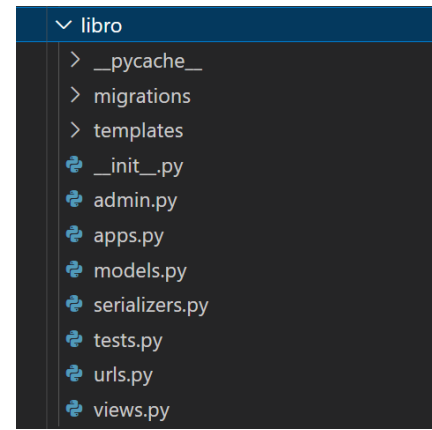
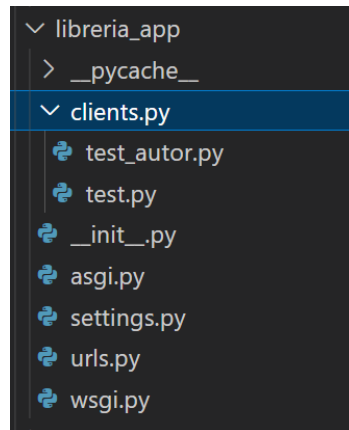
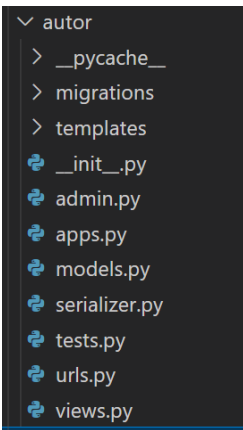
Valeria Ivaniuk
Servidor Básico con Django
https://github.com/valeria-ivaniuk/servidor_basico_django.git

1. En este proyecto he creado dos módulos: uno de libros y otro de autores que están enlazados entre si.

Posible mejora: como ya hemos visto el modulo de libros en la clase, sería interesante crear dos módulos completamente distintos de lo que vimos antes. Como, por ejemplo, clientes y pedidos.

2. Principalmente, he trabajado con 4 archivos dentro de cada módulo, que serían views, serializer, modules y urls. Aparte de eso, en libreria_app he creado una carpeta clients.py donde había agregado los tests para ambos módulos.

Posible mejora: se puede ver que cada modulo contiene un archivo que se llama “Templates” y contiene un archivo html. Ha sido mi intento de agregar un poco de interacción a través del front para Libros y Autores, sin embargo, esta parte se quedó sin finalizar.



3. Los archivos tipo Serializer se usan para la fácil creación y modificación de datos para las API. En este caso mis serializers tienen la siguiente estructura:

```
libreria_app > autor > serializer.py > ...
1  from rest_framework import serializers
2  from .models import Autor
3
4  class AutorSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Autor
7          fields = (
8              'nombre',
9              'apellido',
10         )
11
```

```
libreria_app > libro > serializers.py
1  from rest_framework import serializers
2  from .models import Libro
3
4  class LibroSerializer(serializers.ModelSerializer):
5      class Meta:
6          model = Libro
7          fields = (
8              'titulo',
9              'autor',
10         )
11
```

Valeria Ivaniuk
Servidor Básico con Django
https://github.com/valeria-ivaniuk/servidor_basico_django.git

4. Los siguientes archivos de “Models.py” son la estructura de datos recogidos en la db.sqlite, i.e. mi base de datos.

Posible mejora: tener más campos para ambos módulos y ampliar la cantidad de datos recogidos. Por ejemplo, agregar el campo año, descripción, género para los Libros y año de nacimiento para el Autor, agregar mas libros para un autor etc.

```
libreria_app > autor > models.py > Autor > apellido
1 from django.db import models
2
3 # Create your models here.
4 class Autor(models.Model):
5     nombre = models.CharField(max_length=100)
6     apellido = models.CharField(max_length=100)
7
8
```

```
libreria_app > libro > models.py > Libro > autor
1 from django.db import models
2 from autor import models as m
3
4 # Create your models here.
5 class Libro(models.Model):
6     titulo = models.CharField(max_length=100)
7     autor = models.ForeignKey(m.Autor, on_delete=models.CASCADE)
```

5. Otros dos archivos son Views.py. Estos me permiten definir mis generic views importados de rest_framework además de una view especial dentro de Libro que me permite obtener todos los libros de un autor que es la @api_view.

Posible mejora: agregar más views de la documentación de generics.

```
libreria_app > autor > views.py > ...
1 from django.shortcuts import render
2 from django.http import HttpResponse
3 from .models import Autor
4 from .serializer import AutorSerializer
5 from rest_framework import generics
6 # Create your views here.
7
8 class AutorRetrieveAPIView(generics.RetrieveAPIView):
9     queryset = Autor.objects.all()
10    serializer_class = AutorSerializer
11
12 class AutorListCreateAPIView(generics.ListCreateAPIView):
13    queryset = Autor.objects.all()
14    serializer_class = AutorSerializer
15
16 class AutorUpdateAPIView(generics.UpdateAPIView):
17    queryset = Autor.objects.all()
18    serializer_class = AutorSerializer
19
20 class AutorDestroyAPIView(generics.DestroyAPIView):
21    queryset = Autor.objects.all()
22    serializer_class = AutorSerializer
23
```

```
libreria_app > libro > views.py > api_libros
1 from django.shortcuts import render
2 from rest_framework import generics
3 from .models import Libro
4 from .serializers import LibroSerializer
5 from autor.models import Autor
6 from rest_framework.response import Response
7 from rest_framework.decorators import api_view
8
9
10 class LibroRetrieveAPIView(generics.RetrieveAPIView):
11    queryset = Libro.objects.all()
12    serializer_class = LibroSerializer
13
14 class LibroListCreateAPIView(generics.ListCreateAPIView):
15    queryset = Libro.objects.all()
16    serializer_class = LibroSerializer
17
18 class LibroUpdateAPIView(generics.UpdateAPIView):
19    queryset = Libro.objects.all()
20    serializer_class = LibroSerializer
21
22 class LibroDestroyAPIView(generics.DestroyAPIView):
23    queryset = Libro.objects.all()
24    serializer_class = LibroSerializer
25
26 @api_view(['GET'])
27 def api_libros(request):
28     instanceAutor = Autor.objects.all().order_by('?').first()
29     instanceLibro = Libro.objects.filter(autor=instanceAutor)
30     data = {}
31     if instanceLibro:
32         data = LibroSerializer(instanceLibro[0]).data
33     return Response(data)
```

Valeria Ivaniuk
Servidor Básico con Django
https://github.com/valeria-ivaniuk/servidor_basico_django.git

6. Se agregan todas las urls necesarias para cada vista en Libro, Autor y en libreria_app usando urlpatterns:

```
libreria_app > autor > urls.py > ...
1  from django.urls import path
2  from . import views
3
4
5  urlpatterns = [
6      path('<int:pk>', views.AutorRetrieveAPIView.as_view()),
7      path('create', views.AutorListCreateAPIView.as_view()),
8      path('update/<int:pk>', views.AutorUpdateAPIView.as_view()),
9      path('delete/<int:pk>', views.AutorDestroyAPIView.as_view()),
10 ]

libreria_app > libro > urls.py > [?] urlpatterns
1  from django.urls import path
2  from . import views
3
4  urlpatterns = [
5      path('', views.api_libros),
6      path('<int:pk>', views.LibroRetrieveAPIView.as_view()),
7      path('create', views.LibroListCreateAPIView.as_view()),
8      path('update/<int:pk>', views.LibroUpdateAPIView.as_view()),
9      path('delete/<int:pk>', views.LibroDestroyAPIView.as_view()),
10 ]

libreria_app > libreria_app > urls.py > ...
1
2  from django.contrib import admin
3  from django.urls import path, include
4
5  urlpatterns = [
6      path('admin/', admin.site.urls),
7      path('libro/', include('libro.urls')),
8      path('autor/', include('autor.urls')),
9  ]
10
```

7. Una vez hecho esto, podemos hacer los tests de nuestras views. Todos los test, tanto para el Libro como para el Autor, se encuentran en la carpeta de Clients dentro de libreria_app. Nota bene: descomente los test para ejecutar uno de ellos:

```
libreria_app > libreria_app > clients.py > test_autor.py > [?] response
1  import requests
2
3  endpoint = 'http://127.0.0.1:8000/autor/3'
4  endpoint_delete = 'http://127.0.0.1:8000/autor/delete/2'
5  endpoint_add = 'http://127.0.0.1:8000/autor/create'
6  endpoint_update = 'http://127.0.0.1:8000/autor/update/3'
7  endpoint_api_view = 'http://127.0.0.1:8000/libro'
8
9
10 #response = requests.get(endpoint)
11 #response = requests.post(endpoint_add, json={'nombre': 'J.K.', 'apellido': 'Rowling'})
12 #response = requests.delete(endpoint_delete)
13 #response = requests.put(endpoint_update, json={'nombre': 'Miguel', 'apellido': 'de C'})
14 response = requests.get(endpoint_api_view)
15
16 print(response.json())
```

Valeria Ivaniuk
Servidor Básico con Django

https://github.com/valeria-ivaniuk/servidor_basico_django.git

```
libreria_app > libreria_app > clients.py > test.py > ...
1  import requests
2
3  endpoint = 'http://127.0.0.1:8000/libro/1'
4  endpoint_delete = 'http://127.0.0.1:8000/libro/delete/3'
5  endpoint_add = 'http://127.0.0.1:8000/libro/create'
6  endpoint_update = 'http://127.0.0.1:8000/libro/update/3'
7
8  response = requests.post(endpoint_add, json={'titulo': 'Don Quijote', 'autor': 3})
9  #response = requests.get(endpoint)
10 #response = requests.delete(endpoint_delete)
11 #response = requests.put(endpoint_update, json={'titulo': 'Don Quijote de la Mancha'})
12
13 print(response.json())
```

Conclusiones:

Usar generics de `rest_framework` me pareció bastante útil y fácil. Sin embargo, me gustaría entender más a través de los ejemplos para qué exactamente necesitamos los serializers cuando realmente tenemos la misma información en los archivos de Models. También me gustaría poder desarrollar más la tarea y renderizar el archivo de HTML que había creado para interacción simple con el front.

De todas las generic views, `@api_view` me ha parecido la más difícil. Como tuvimos que enlazar dos modelos en esta view, me gustaría haber tenido un ejemplo parecido previo de como se podría hacer y entender mejor la estructura de esta petición.