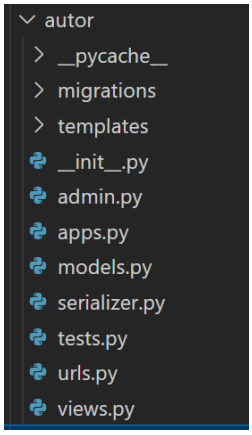
Valeria Ivaniuk
Servidor Básico con Django

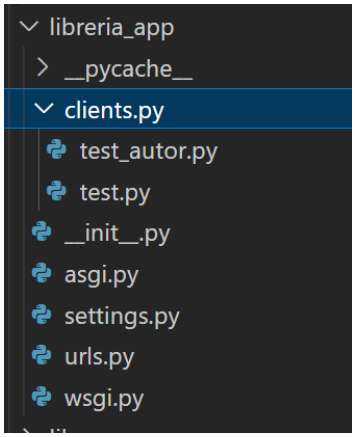
1. En este proyecto he creado dos módulos: uno de libros y otro de autores que están enlazados entre si.

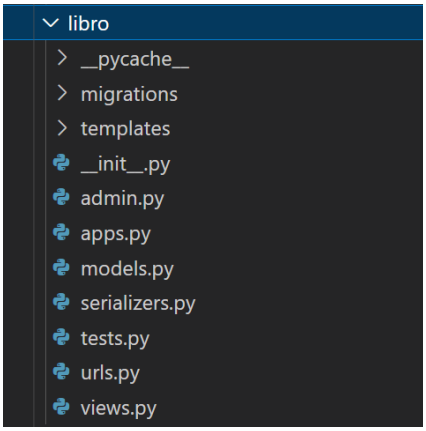
   **Posible mejora:** *como ya hemos visto el modulo de libros en la clase, sería interesante crear dos módulos completamente distintos de lo que vimos antes. Como, por ejemplo, clientes y pedidos.*

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

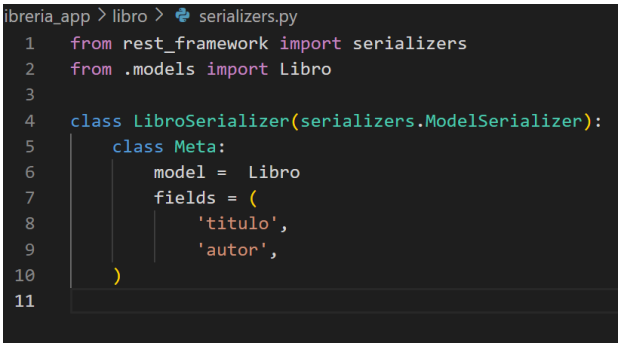
   **Posible mejora**: *se puede ver que cada modulo contiene un archivo que se llama "Templates" y contiene un archivo html. Ha sido mi intento de agregar un poco de interacción a través del front para Libros y Autores, sin embargo, esta parte se quedó sin finalizar.*




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:

Valeria Ivaniuk
# Servidor Básico con Django

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

```python
from django.db import models

# Create your models here.
class Autor(models.Model):
    nombre = models.CharField(max_length=100)
    apellido = models.CharField(max_length=100)

```

libreria_app > libro > models.py > Libro > autor

```python
from django.db import models
from autor import models as m

# Create your models here.
class Libro(models.Model):
    titulo = models.CharField(max_length=100)
    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**: *aregar más views de la documentación de generics.*

libreria_app > autor > views.py > ...

```python
from django.shortcuts import render
from django.http import HttpResponse
from .models import Autor
from .serializer import AutorSerializer
from rest_framework import generics
# Create your views here.

class AutorRetrieveAPIView(generics.RetrieveAPIView):
    queryset = Autor.objects.all()
    serializer_class = AutorSerializer

class AutorListCreateAPIView(generics.ListCreateAPIView):
    queryset = Autor.objects.all()
    serializer_class = AutorSerializer

class AutorUpdateAPIView(generics.UpdateAPIView):
    queryset = Autor.objects.all()
    serializer_class = AutorSerializer

class AutorDestroyAPIView(generics.DestroyAPIView):
    queryset = Autor.objects.all()
    serializer_class = AutorSerializer

```

libreria_app > libro > views.py > api_libros

```python
from django.shortcuts import render
from rest_framework import generics
from .models import Libro
from .serializers import LibroSerializer
from autor.models import Autor
from rest_framework.response import Response
from rest_framework.decorators import api_view


class LibroRetrieveAPIView(generics.RetrieveAPIView):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

class LibroListCreateAPIView(generics.ListCreateAPIView):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

class LibroUpdateAPIView(generics.UpdateAPIView):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

class LibroDestroyAPIView(generics.DestroyAPIView):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer

@api_view(['GET'])
def api_libros(request):
    instanceAutor = Autor.objects.all().order_by('?').first()
    instanceLibro = Libro.objects.filter(autor=instanceAutor)
    data = {}
    if instanceLibro:
        data = LibroSerializer(instanceLibro[0]).data
        return Response(data)
```

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

libreria_app > autor > urls.py > ...

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


urlpatterns = [
path('<int:pk>', views.AutorRetrieveAPIView.as_view()),
path('create', views.AutorListCreateAPIView.as_view()),
path('update/<int:pk>', views.AutorUpdateAPIView.as_view()),
path('delete/<int:pk>', views.AutorDestroyAPIView.as_view()),
 ]
```

libreria_app > libro > urls.py > urlpatterns

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

urlpatterns = [
    path('', views.api_libros),
    path('<int:pk>', views.LibroRetrieveAPIView.as_view()),
    path('create', views.LibroListCreateAPIView.as_view()),
    path('update/<int:pk>', views.LibroUpdateAPIView.as_view()),
    path('delete/<int:pk>', views.LibroDestroyAPIView.as_view()),

]
```

libreria_app > libreria_app > urls.py > ...

```python

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('libro/', include('libro.urls')),
    path('autor/', include('autor.urls')),
]

```

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 librería_app. Nota bene: descomente los test para ejecutar uno de ellos:

libreria_app > libreria_app > clients.py > test_autor.py > response

```python
import requests

endpoint = 'http://127.0.0.1:8000/autor/3'
endpoint_delete = 'http://127.0.0.1:8000/autor/delete/2'
endpoint_add = 'http://127.0.0.1:8000/autor/create'
endpoint_update = 'http://127.0.0.1:8000/autor/update/3'
endpoint_api_view = 'http://127.0.0.1:8000/libro'


#response = requests.get(endpoint)
#response = requests.post(endpoint_add, json={'nombre': 'J.K.', 'apellido':'Rowling
#response = requests.delete(endpoint_delete)
#response = requests.put(endpoint_update, json={'nombre': 'Miguel', 'apellido':'de
response = requests.get(endpoint_api_view)

print(response.json())
```

Valeria Ivaniuk
Servidor Básico con Django

libreria_app > libreria_app > clients.py > test.py > ...

```python
import requests

endpoint = 'http://127.0.0.1:8000/libro/1'
endpoint_delete = 'http://127.0.0.1:8000/libro/delete/3'
endpoint_add = 'http://127.0.0.1:8000/libro/create'
endpoint_update = 'http://127.0.0.1:8000/libro/update/3'

response = requests.post(endpoint_add, json={'titulo': 'Don Quijote', 'autor': 3})
#response = requests.get(endpoint)
#response = requests.delete(endpoint_delete)
#response = requests.put(endpoint_update, json={'titulo': 'Don Quijote de la Mancha
💡
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.