Informe de Laboratorio 08

Tema: Django Rest Framework

Nota

Grupo B	Escuela	Asignatura
▪ Ccahuana Larota, Joshep Antony ▪ Christian Zapana Romero, Pedro Luis ▪ Mejia Ramos, Piero Douglas	Escuela Profesional de Ingeniería de Sistemas	Programación Web 2 Semestre: III Código: 1702122

Laboratorio	Tema	Duración
08	Django Rest Framework	04 horas

Semestre académico	Fecha de inicio	Fecha de entrega
2023 - A	Del 26 Julio 2023	Al 03 Julio 2023

1. Competencias del curso

- General: C.c. Diseña responsablemente aplicaciones web, sus componentes o procesos para satisfacer necesidades dentro de restricciones realistas: económicas, medioambientales, sociales, políticas, éticas, de salud, de seguridad, manufacturación y sostenibilidad.
- Específica: C.m. Construye responsablemente soluciones con tecnología web siguiendo un proceso adecuado llevando a cabo las pruebas ajustada a los recursos disponibles del cliente.
- C.p. Aplica de forma flexible técnicas, métodos, principios, normas, estándares y herramientas del desarrollo web necesarias para la construcción de aplicaciones web e implementación de estos sistemas en una organización.

2. Resultado del estudiante

- RE. 2 La capacidad de aplicar diseño de ingeniería para producir soluciones a problemas y diseñar sistemas, componentes o procesos para satisfacer necesidades específicas dentro de consideraciones realistas en los aspectos de salud pública, seguridad y bienestar; factores globales, culturales, sociales, económicos y ambientales.

- RE. 8 La capacidad de crear, seleccionar y utilizar técnicas, habilidades, recursos y herramientas modernas de ingeniería y tecnologías de la información, incluyendo la predicción y el modelado, con una comprensión de las limitaciones.

3. Equipos, materiales y temas

- Sistema Operativo (GNU/Linux de preferencia).
- GNU Vim.
- Python 3.
- Git.
- Cuenta en GitHub con el correo institucional.
- Entorno virtual.
- Django 4.
- Django REST framework.

4. Marco teórico

4.1. Django Rest Framerowk:

- Django REST framework es un conjunto de herramientas potente y flexible para crear API web.
- Algunas razones por las que podrías querer usar el marco REST:
  - La API navegable por la Web es una gran ganancia de usabilidad para sus desarrolladores.
  - Políticas de autenticación que incluyen paquetes para OAuth1 y OAuth2.
  - Serialización que admite fuentes de datos ORM y no ORM.
  - Personalizable hasta el final: solo use las vistas regulares basadas en funciones si no necesita las funciones más potentes.
  - Amplia documentación y gran apoyo de la comunidad.
  - Utilizado y confiado por empresas reconocidas internacionalmente, como Mozilla, Red Hat, Heroku y Eventbrite.

5. Tarea

- Elabore un servicio web que tenga un CRUD con el uso de este framework.
  - **Create** - *POST*
  - **Read** - *GET*
  - **Update** - *PUT*
  - **Delete** - *DELETE*
- Centrarse en el Core business de su aplicación web. Lo más importante y necesario debe estar disponible a través de un servicio web. A continuación, se presentan algunos ejemplos de servicios web:

- `https://reqbin.com/`
- `https://www.googleapis.com/youtube/v3/playlistItems`

- Muestre la funcionalidad consumiéndola desde el cliente REST de su preferencia. El método *GET* puede ser directamente consumido por un navegador web. Por ejemplo, en esta API se puede obtener la temperatura de Arequipa en un rango de fechas (la versión gratuita tiene un retraso de 7 días, por tanto, solo mostrará la temperatura en Arequipa desde el 01 de Julio hasta el 03 de Julio):
- `https://archive-api.open-meteo.com/v1/archive?latitude=-16.39889&longitude=-71.535&start_date=2023-07-01&end_date=2023-07-10&hourly=temperature_2m&daily=temperature_2m_max,temperature_2m_min&timezone=America%2FNew_York`

6. Desarrollo

6.1. Proyecto

Para está sección se presenta las configuraciones en los módulos settings.py y urls.py de la carpeta "library"dek proyecto "library":

6.1.1. settings.py

Se agrega la aplicación reservation, books, y el rest_framework.

Listing 1: settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'reservation',
    'rest_framework',
    'books',
]
```

Ademas de la modificación al directorio de busqueda para los templates:

Listing 2: settings.py

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR,'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
```

```
]
```

6.1.2. urls.py

En está sección se ve la implementación de la url api

Listing 3: urls.py

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', include('books.urls')),
    path('admin/', admin.site.urls),
    path('api-auth/', include('reservation.urls')),
]
urlpatterns = urlpatterns + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6.2. Aplicación reservation

En está sección se mostrara la configuración y la funcionalidad de cada módulo usado en la palicación.

6.2.1. admin.py

El código en el archivo admin.py de la aplicación reservationregistra los modelos "Libroz Üser"para que puedan ser administrados desde la interfaz de administración de Django. La función admin.site.register(Modelo) se utiliza para registrar un modelo en la interfaz de administración.

Una vez que los modelos están registrados, podrán ser gestionados (crear, leer, actualizar y eliminar objetos) desde la interfaz de administración en el navegador, siempre y cuando se tenga acceso como usuario con privilegios de administrador o superusuario. Recuerda que antes de desplegar la aplicación en producción, es esencial implementar la autenticación y autorización adecuadas para proteger el acceso a la interfaz de administración y los datos de la base de datos.

Listing 4: admin.py

```python
from django.contrib import admin
from .models import Libro,User

# Register your models here.
admin.site.register(Libro)
admin.site.register(User)
```

6.2.2. models.py

1. Libro
   - Tiene cuatro campos: "nombre", .ªuthor", çodigo", y ïmg"(imagen).
   - Los campos "nombrez .ªuthor"son cadenas de caracteres con una longitud máxima de 50 caracteres.

- El campo çodigo.[es] un entero que almacena un número.
- El campo ïmg.[es] una imagen que se almacena en el directorio "pics"dentro del sistema de archivos.
- También tiene un campo adicional "desc"(descripción), que es un campo de texto que puede almacenar una descripción del libro.
- El método __str__() se ha definido para que al imprimir un objeto "Libro", se muestre su nombre.

2. User

- Tiene cuatro campos: "nombre", .[em]ail", "deuda", y "libroPrest".
- Los campos "nombrez .[em]ail"son cadenas de caracteres con una longitud máxima de 50 caracteres.
- El campo "deuda.[es] un booleano que indica si el usuario tiene una deuda o no, con un valor predeterminado de "False".
- El campo "libroPrest.[es] una clave externa (ForeignKey) que establece una relación entre el modelo Üserz el modelo "Libro". Indica que un usuario puede tener varios préstamos de libros, y se define con el argumento .[o]n_delete=models.CASCADE", lo que significa que si se elimina un libro relacionado, también se eliminarán los préstamos asociados a ese libro.
- El método __str__() se ha definido para que al imprimir un objeto Üser", se muestre su nombre.

Listing 5: models.py

```python
from django.db import models

# Create your models here.

class Libro(models.Model):
    nombre = models.CharField(max_length=50)
    author = models.CharField(max_length=50)
    codigo = models.IntegerField()
    img = models.ImageField(upload_to='pics', default='')
    desc = models.TextField(default="")

    def __str__(self):
        return self.nombre

class User(models.Model):
    nombre = models.CharField(max_length=50)
    email = models.CharField(max_length=50)
    deuda = models.BooleanField(default=False)
    libroPrest = models.ForeignKey(Libro, on_delete=models.CASCADE)

    def __str__(self):
        return self.nombre
```

En resumen, estos modelos definen la estructura de datos para los libros y los usuarios en la aplicación, y permiten almacenar y gestionar información relacionada con los libros disponibles y los préstamos realizados por los usuarios.

6.2.3. serialiers.py

El código proporciona dos clases llamadas LibroSerializer y UserSerializer, que son utilizadas para convertir los objetos de los modelos Libro y User en formatos que se pueden enviar a través de la web o almacenar en una base de datos.

En términos sencillos, estas clases son "traductores"que toman los datos de los objetos de los modelos Libro y User y los convierten en una estructura que puede ser fácilmente entendida y procesada por aplicaciones que utilicen el framework Django REST.

Para ser más precisos:

- LibroSerializer se encarga de tomar un objeto de la clase Libro y convertirlo en un formato de datos que incluye los campos url, nombre, author, codigo, img y desc.
- UserSerializer toma un objeto de la clase User y lo convierte en un formato que contiene los campos url, nombre, email, deuda y libroPrest.

Estas clases son parte del módulo rest_framework de Django y son esenciales cuando se trabaja con API RESTful, ya que permiten serializar y deserializar datos entre las representaciones internas de los objetos y los formatos que se envían a través de la web, como JSON o XML. De esta manera, se facilita la comunicación entre el servidor y el cliente al procesar datos en diferentes formatos.

Listing 6: serializers.py

```python
from .models import Libro, User
from rest_framework import serializers


class LibroSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Libro
        fields = ['url', 'nombre', 'author', 'codigo', 'img', 'desc']


class UserSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = User
        fields = ['url', 'nombre', 'email', 'deuda', 'libroPrest']
```

6.2.4. urls.py

El código configura las URL para un servicio web utilizando Django REST Framework. Se importan las funciones y clases necesarias desde los módulos de Django, como `path` para definir rutas URL y `include` para incluir otras rutas URL desde otras aplicaciones. Se crea una instancia de `DefaultRouter` llamada `router`, que preconfigura las rutas comunes para un conjunto de vistas, facilitando la generación de URL para operaciones CRUD. Se registran las vistas `LibroViewSet` y `UserViewSet` con sus respectivas rutas en el `router`. La variable `urlpatterns` contiene las rutas URL que se utilizarán para acceder a las vistas de la API, incluyendo las rutas proporcionadas por el `router`.

En resumen, el código establece las rutas de acceso a diferentes vistas de la API mediante el uso de `DefaultRouter` de Django REST Framework.

Listing 7: urls.py

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('', include('books.urls')),
    path('admin/', admin.site.urls),
    path('api-auth/', include('reservation.urls')),
]
urlpatterns = urlpatterns + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

6.2.5. views.py

El código proporciona vistas (*views*) para un servicio web utilizando Django REST Framework. Se importan funciones y clases necesarias, incluyendo `render`, modelos `Libro` y `User`, y clases `viewsets` y `permissions` de Django REST Framework, junto con los serializadores `LibroSerializer` y `UserSerializer`.

Se definen dos clases de vistas: `LibroViewSet` y `UserViewSet`, que heredan de `viewsets.ModelViewSet`. Estas clases representan vistas basadas en modelos que brindan funcionalidades CRUD para los modelos `Libro` y `User`.

Dentro de cada clase de vista, se definen tres atributos:

- `queryset`: Especifica el conjunto de objetos que se utilizarán en la vista. Para la vista `LibroViewSet`, se trabaja con todos los objetos de la clase `Libro`.
- `serializer_class`: Indica el serializador que se utilizará para convertir los objetos del modelo en representaciones JSON y viceversa. Para la vista `LibroViewSet`, se utiliza `LibroSerializer`, y para la vista `UserViewSet`, se utiliza `UserSerializer`.
- `permission_classes`: Define las clases de permisos requeridas para acceder a la vista. Se requiere que el usuario esté autenticado para acceder a ambas vistas, por lo que se utiliza `permissions.IsAuthenticated`.

Listing 8: views.py

```python
from django.shortcuts import render
from .models import Libro, User
from rest_framework import viewsets
from rest_framework import permissions
from .serializers import LibroSerializer, UserSerializer
# Create your views here.

class LibroViewSet(viewsets.ModelViewSet):
    queryset = Libro.objects.all()
    serializer_class = LibroSerializer
    permission_classes = [permissions.IsAuthenticated]


class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
```

```
    serializer_class = UserSerializer
    permission_classes = [permissions.IsAuthenticated]
```

6.3. Base HTML

Listing 9: index.html

```
{% load static %}
{% static "images" as baseUrl %}
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<title>Book Store Template, Free CSS Template, CSS Website Layout</title>
<meta name="keywords" content="" />
<meta name="description" content="" />
<link href="{% static 'css/templatemo_style.css' %}" rel="stylesheet" type="text/css" />
<link href="{% static 'css/style.css' %}" rel="stylesheet" type="text/css" />
</head>
<body>
<!-- Free CSS Templates from www.templatemo.com -->
<div id="templatemo_container">
   <div id="templatemo_menu">
      <ul>
            <li><a href="index.html" class="current">Home</a></li>
            <li><a href="subpage.html">Search</a></li>
            <li><a href="subpage.html">Books</a></li>
            <li><a href="subpage.html">New Releases</a></li>
            <li><a href="#">Company</a></li>
            <li><a href="#">Contact</a></li>
      </ul>
    </div> <!-- end of menu -->

    <div id="templatemo_header" style="background-image: url({% static
        'images/templatemo_header_bg.jpg' %})">
      <div id="templatemo_special_offers">
         <p>
               <span>25%</span> discounts for
        purchase over $80
         </p>
        </div>


        <div id="templatemo_new_books">
         <ul>
               <li>Suspen disse</li>
               <li>Maece nas metus</li>
               <li>In sed risus ac feli</li>
            </ul>
        </div>
    </div> <!-- end of header -->

    <div id="templatemo_content">

        <div id="templatemo_content_left">
```

```
48          <div class="templatemo_content_left_section">
49               <h1>Categories</h1>
50                <ul>
51                    <li><a href="subpage.html">Donec accumsan urna</a></li>
52                    <li><a href="subpage.html">Proin vulputate justo</a></li>
53                    <li><a href="#">In sed risus ac feli</a></li>
54                    <li><a href="#">Aliquam tristique dolor</a></li>
55                    <li><a href="#">Maece nas metus</a></li>
56                    <li><a href="#">Sed pellentesque placerat</a></li>
57                    <li><a href="#">Suspen disse</a></li>
58                    <li><a href="#">Maece nas metus</a></li>
59                    <li><a href="#">In sed risus ac feli</a></li>
60               </ul>
61             </div>
62          <div class="templatemo_content_left_section">
63               <h1>Bestsellers</h1>
64                <ul>
65                    <li><a href="#">Vestibulum ullamcorper</a></li>
66                    <li><a href="#">Maece nas metus</a></li>
67                    <li><a href="#">In sed risus ac feli</a></li>
68                    <li><a href="#">Praesent mattis varius</a></li>
69                    <li><a href="#">Maece nas metus</a></li>
70                    <li><a href="#">In sed risus ac feli</a></li>
71                    <li><a href="#">Flash Templates</a></li>
72                    <li><a href="#">CSS Templates</a></li>
73                    <li><a href="#">Web Design</a></li>
74                    <li><a href="http://www.photovaco.com" target="_parent">Free
                          Photos</a></li>
75               </ul>
76             </div>
77
78             <div class="templatemo_content_left_section">
79                 <a href="http://validator.w3.org/check?uri=referer"><img
                       style="border:0;width:88px;height:31px"
                       src="http://www.w3.org/Icons/valid-xhtml10" alt="Valid XHTML 1.0
                       Transitional" width="88" height="31" vspace="8" border="0" /></a>
80 <a href="http://jigsaw.w3.org/css-validator/check/referer"><img
       style="border:0;width:88px;height:31px"
       src="http://jigsaw.w3.org/css-validator/images/vcss-blue" alt="Valid CSS!" vspace="8"
       border="0" /></a>
81          </div>
82        </div> <!-- end of content left -->
83
84        <div id="templatemo_content_right">
85
86             {% for libro in libros %}
87          <div class="templatemo_product_box">
88               <h1> {{libro.nombre}} <span>(by {{libro.author}})</span></h1>
89                 <img id="libros" src="{{libro.img.url}}" alt="image" />
90                <div class="product_info">
91                  <p>{{libro.desc}}</p>
92                    <div class="buy_now_button"><a href="subpage.html">Alquilar</a></div>
93                    <div class="detail_button"><a href="subpage.html">Detail</a></div>
94                </div>
95                <div class="cleaner"> </div>
96             </div>
```

```
            {% endfor %}

        </div> <!-- end of content right -->

      <div class="cleaner_with_height"> </div>
    </div> <!-- end of content -->

    <div id="templatemo_footer">

          <a href="subpage.html">Home</a> | <a href="subpage.html">Search</a> | <a
              href="subpage.html">Books</a> | <a href="#">New Releases</a> | <a
              href="#">FAQs</a> | <a href="#">Contact Us</a><br />
        Copyright   2024 <a href="#"><strong>Your Company Name</strong></a>
        <!-- Credit: www.templatemo.com --> </div>
    <!-- end of footer -->
<!-- Free CSS Template www.templatemo.com -->
</div> <!-- end of container -->
<!-- templatemo 086 book store -->
<!--
Book Store Template
http://www.templatemo.com/preview/templatemo_086_book_store
-->
</body>
</html>
```

7. Ejecución

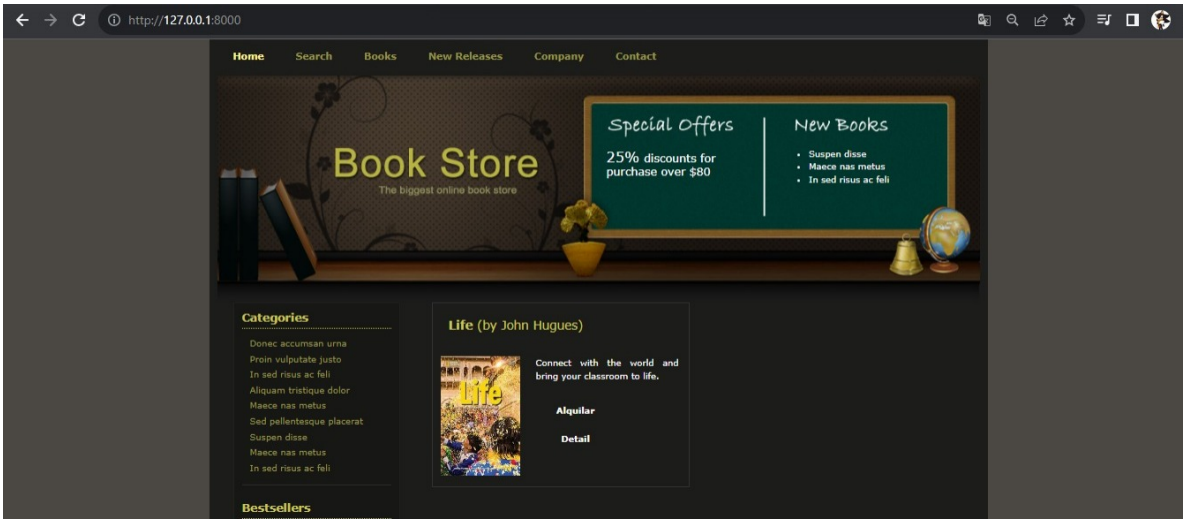

Figura 1: Html basico

8. Pregunta

- ¿Cuál fué la mayor dificultad del uso de este framework?
  La mayor dificultad sería útil izar el renderizador para poder utilizar los modelos.

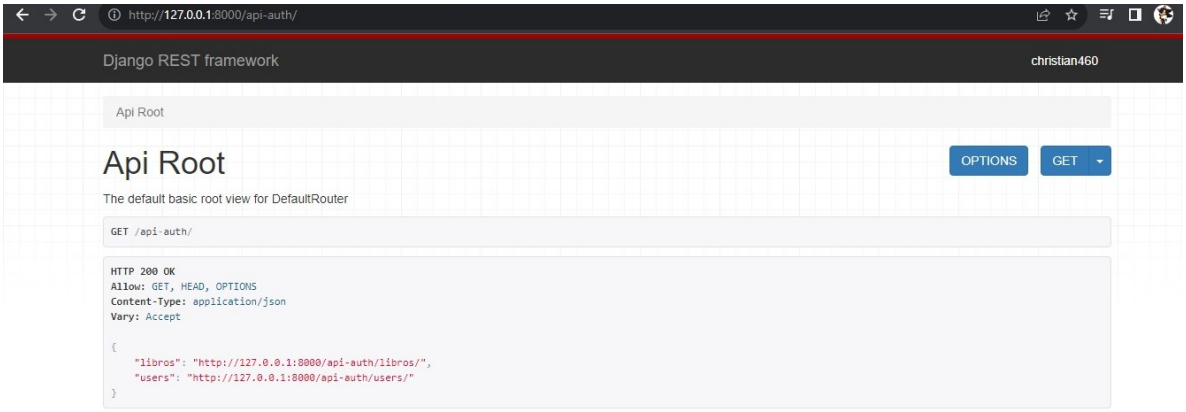

Figura 2: api-auth

9. URL de Repositorio Github

https://github.com/christian460/Lab_Grupal.git

10. URL de Referencias

- https://www.django-rest-framework.org/
- https://www.django-rest-framework.org/tutorial/quickstart/
- https://www.django-rest-framework.org/tutorial/1-serialization/
- https://www.django-rest-framework.org/tutorial/3-class-based-views/
- https://www.django-rest-framework.org/api-guide/authentication/
- https://www.bezkoder.com/django-rest-api/

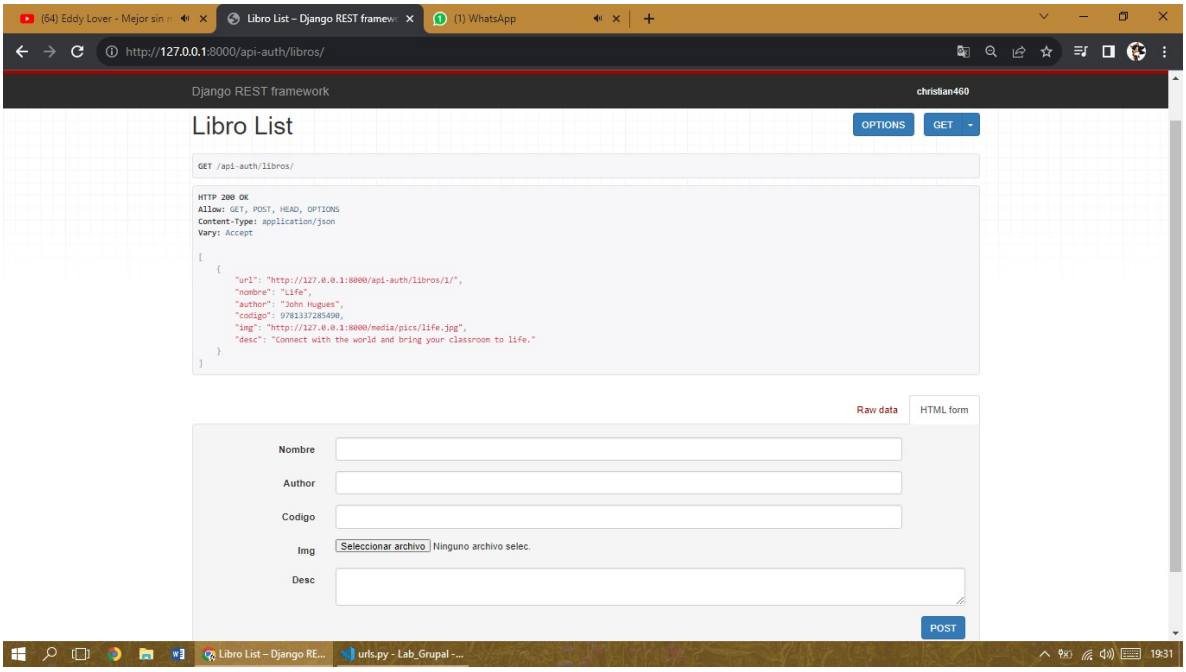

Figura 3: Lista de libros