

# Informe de Laboratorio 08

## Tema: Django Rest Framework

Nota

Estudiante	Escuela	Asignatura
Arles Carrasco Choque, Jean Carlo Chara Condori, Reyser Zapata Butron, Daniela Choquecondo Aspilcueta	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 18 julio del 2023	Al 26 julio del 2023

### 1. Ejercicio

- Practique desarrollando el ejercicio de iniciación:
- <https://www.django-rest-framework.org/tutorial/quickstart/>
- Para consumir el web-service puede usar el cliente SOAP UI Community: <https://www.soapui.org/downloads/soapui/>

**Solución:** Después de seguir los pasos dados en la documentación de Django Rest Framework, se logra obtener una api gráfica que se puede acceder haciendo login como super usuario, obteniendo así la siguiente imagen en nuestra pantalla.

Para poder ejecutar este proyecto, debemos seguir los siguientes comandos en nuestra terminal:

Listing 1: Ingresango a la carpeta Tutorial

```
$ cd Tutorial
```

Ahora debemos abrir el servidor manage.py

Listing 2: Abriendo el servidor del ejercicio

```
$ python .\manage runserver
```

Luego de hacer estos pasos, podemos ver el proyecto del ejercicio en ejecución

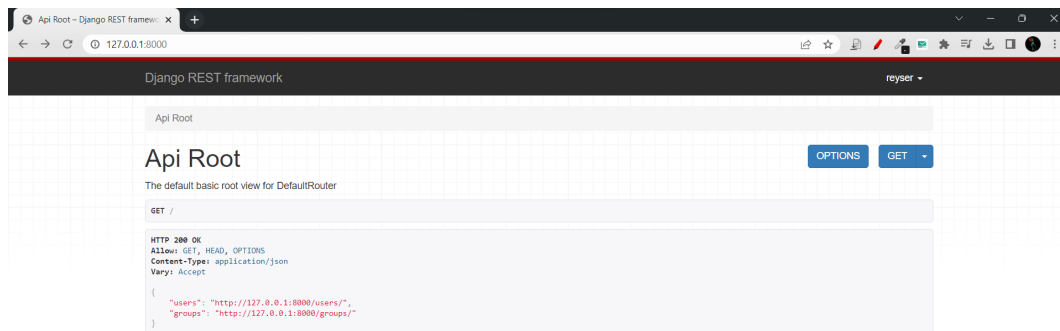


Figura 1: Captura de pantalla del manage.py de tutorial

Si accedemos a Users, nos devolverá los datos de la API, que en este caso serían todos los usuarios registrados en el panel admin de Django:

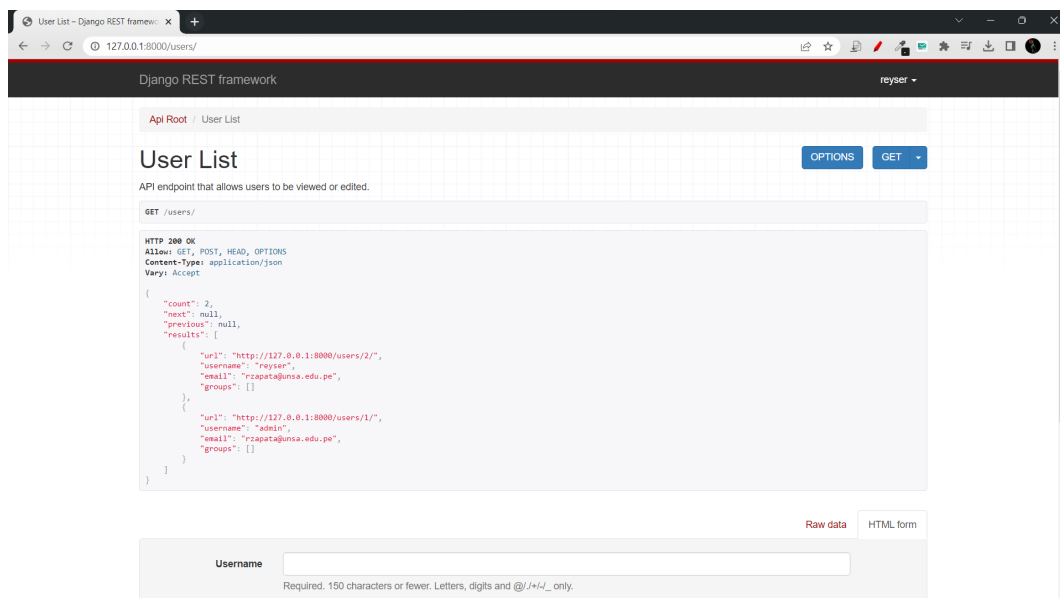


Figura 2: Captura de pantalla de Users API

Se logró una explicación eficiente y clara del funcionamiento de Django Rest Framework, un requisito esencial para la tarea asignada. Gracias a esta explicación, se pudo comprender a fondo cómo funcionan las APIs y cómo interactúan con la plataforma.

## 2. Tarea

- En sus grupos de trabajo correspondientes. Elabore un servicio web que tenga un CRUD con el uso de este framework
- Create - POST
- Read - GET
- Update - PUT
- Delete - DELETE
- Centrarce en el Core business de su aplicacion web. Los mas importante y necesario que este disponible a traves de un servicio web.
- Ejemplos: <https://reqbin.com/>, <https://www.googleapis.com/youtube/v3/playlistItems>
- Muestre la funcionalidad consumiendola desde el cliente Rest de su preferencia.
- El metodo 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 version 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&amp>

### 2.1. Crear una API con respuesta en JSON

#### Solución:

Para completar todos los requerimientos de este laboratorio, se hizo 2 proyectos django, "DjangoApiRestz" "dniAPI", en el que podremos apreciar como funciona, una API y un servicio web consumiendo una API de validación.

Primero explicaremos el proyecto Principal que resuelve la tarea, "DjangoApiRest", es nuestra API de contactos. El Core Business de nuestra API de contactos sería gestionar y proporcionar acceso a la información relacionada con los contactos almacenados en nuestro sistema. Esto incluiría operaciones para listar todos los contactos, buscar un contacto específico por su ID, agregar nuevos contactos, actualizar la información de un contacto existente y eliminar contactos.

Ahora mostraremos los codigos de los archivos más relevantes de este proyecto y posteriormente de la app:

#### 2.1.1. Proyecto DjangoApiRest

Listing 3: DjangoApiRest/urls.py

```
1 from django.contrib import admin
2 from django.urls import path, include
3 from django.shortcuts import redirect
4
5 urlpatterns = [
6     path('admin/', admin.site.urls),
7     path('', lambda request: redirect('api/get/', permanent=False)),
```

```
8 path('api/', include('djangoApiRest.api.urls')),
9 ]
```

Este es el `Urls.py` del proyecto, en el que definimos nuestra url principal, que sería `.api/get/`, para que así al abrir nuestro servidor, se nos muestre todos los contactos.

Listing 4: `DjangoApiRest/middleware.py`

```
1 class JsonResponseMiddleware:
2     def __init__(self, get_response):
3         self.get_response = get_response
4
5     def __call__(self, request):
6         response = self.get_response(request)
7         response['Content-Type'] = 'application/json'
8         return response
```

Este archivo es un Middleware, con la clase `JsonResponseMiddleware`, encargado de que todo lo que nos responda nuestro servidor, sea en formato JSON, así como se pide en la tarea

### 2.1.2. App api

En la app de nuestro proyecto, está lo esencial de que nuestra Api funcione y haga todo eficiente:

Listing 5: `DjangoApiRest/urlsApi.py`

```
1 from django.urls import path
2 from djangoApiRest.api import views
3
4 urlpatterns = [
5     path('get/', views.getContacto),
6     path('get/<int:id>', views.getContactoById),
7     path('post/', views.postContacto),
8     path('put/<int:pk>', views.putContacto),
9     path('delete/<int:pk>', views.deleteContacto),
10 ]
```

En esta urls de nuestra app api, definimos las urls con las diferentes opciones de un CRUD (GET, POST, PUT, DELETE), cada uno teniendo una url y función diferente

Listing 6: `DjangoApiRest/views.py`

```
1 from django.http import JsonResponse
2 from rest_framework.decorators import api_view
3 from djangoApiRest.api.models import Contacto
4 from djangoApiRest.api.serializers import ContactoSerializer
5
6 @api_view(['GET'])
7 def getContacto(request):
8     contactos = Contacto.objects.all()
9     serializer = ContactoSerializer(contactos, many=True)
10    return JsonResponse(serializer.data, safe=False)
11
12 @api_view(['GET'])
13 def getContactoById(request, id):
14     try:
15         contacto = Contacto.objects.get(id=id)
```

```
16     serializer = ContactoSerializer(contacto)
17     return JsonResponse(serializer.data)
18 except Contacto.DoesNotExist:
19     return JsonResponse({"message": "Contacto no encontrado"}, status=404)
20
21 @api_view(['POST'])
22 def postContacto(request):
23     serializer = ContactoSerializer(data=request.data)
24     if serializer.is_valid():
25         serializer.save()
26         return JsonResponse(serializer.data)
27     return JsonResponse(serializer.errors, status=400)
28
29 @api_view(['PUT'])
30 def putContacto(request, pk):
31     try:
32         contacto = Contacto.objects.get(id=pk)
33     except Contacto.DoesNotExist:
34         return JsonResponse({"message": "Contacto no encontrado"}, status=404)
35
36     serializer = ContactoSerializer(instance=contacto, data=request.data)
37     if serializer.is_valid():
38         serializer.save()
39         return JsonResponse(serializer.data)
40     return JsonResponse(serializer.errors, status=400)
41
42 @api_view(['DELETE'])
43 def deleteContacto(request, pk):
44     try:
45         contacto = Contacto.objects.get(id=pk)
46         contacto.delete()
47         return JsonResponse({"message": "Contacto eliminado"})
48     except Contacto.DoesNotExist:
49         return JsonResponse({"message": "Contacto no encontrado"}, status=404)
```

Este es el código que nos permitiera recibir un request y devolver un JSON en pantalla, cada uno tiene su propia funcionalidad

Listing 7: DjangoApiRest/models.py

```
1 from django.db import models
2
3 class Contacto(models.Model):
4     nombre = models.CharField(max_length=100)
5     direccion = models.CharField(max_length=200)
6     telefono = models.CharField(max_length=20)
7     correo_electronico = models.EmailField()
8
9     def __str__(self):
10         return self.nombre
```

Este es el modelo de los Contactos, cada uno con su propósito

Listing 8: DjangoApiRest/serializers.py

```
1 from rest_framework.serializers import ModelSerializer
2 from djangoApiRest.api.models import Contacto
```

```

3
4 class ContactoSerializer(ModelSerializer):
5     class Meta:
6         model = Contacto
7         fields = '__all__'

```

Finalmente, el serializer que nos permitirá darle formato a nuestra API, convirtiendola en 100x100 API

### 2.1.3. Ejecución

Como es un API CRUD, tendremos que ver las diferentes acciones, con "Thunder Cliente" de Visual Studio Code, que nos permitira saber si nuestra api está recibiendo y devolviendo lo que deseamos y tambien consumirla desde el navegador web, demostrando su eficiencia.

Primero debemos ejecutar el servidor, nuestro manage.py es el que está en el directorio principal.

Listing 9: Abriendo servidor

```
$ python .\manage.py runserver
```

**GET** - Este GET, es general, devuelve todos los contactos de la base de datos.

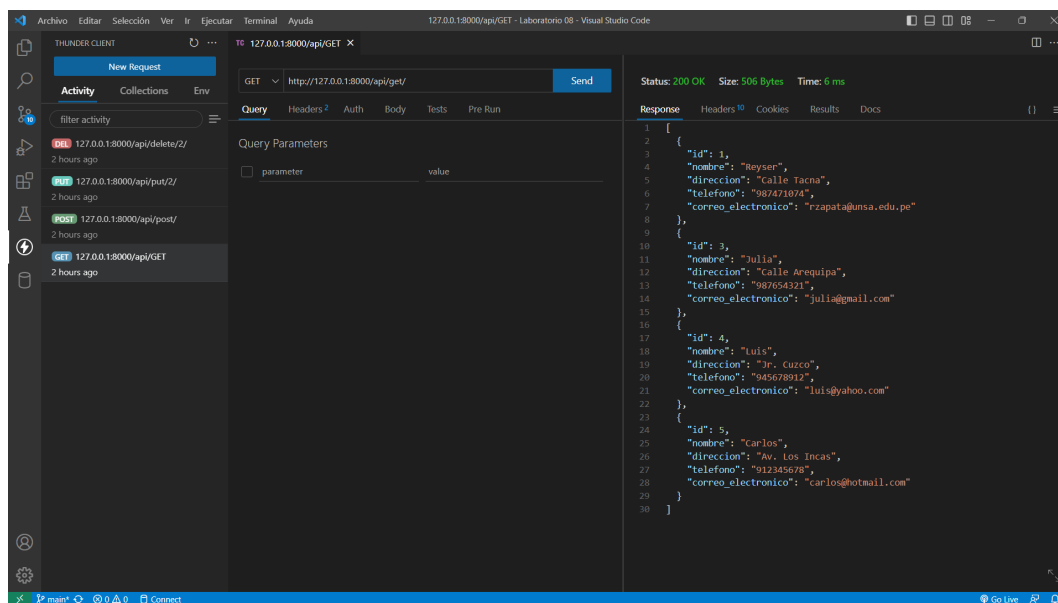
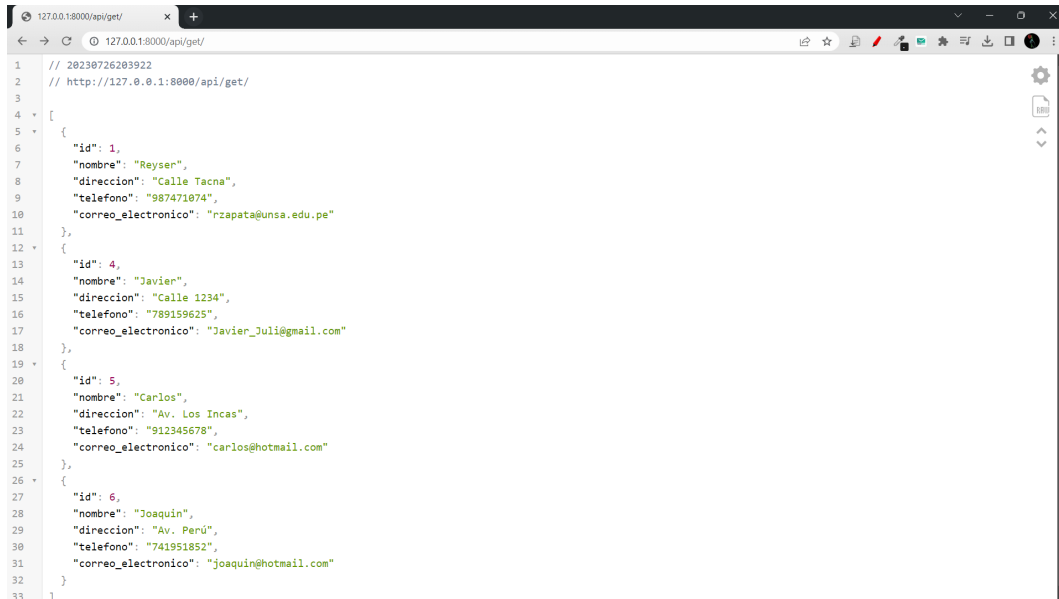


Figura 3: GET de todos los Contactos desde Thunder Client

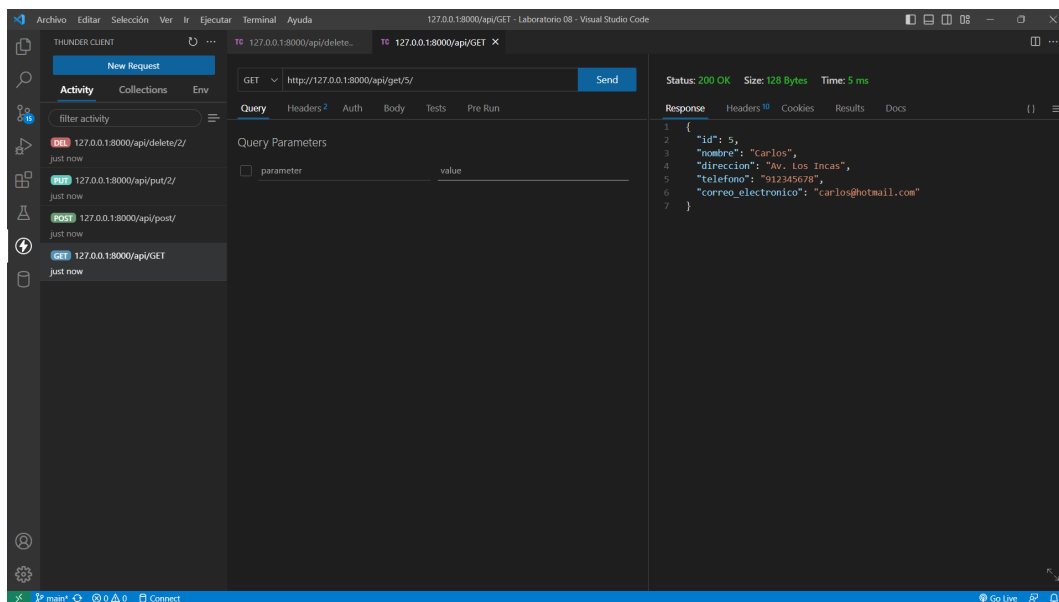


```

1 // 20230726203922
2 // http://127.0.0.1:8000/api/get/
3
4 *
5 {
6   "id": 1,
7   "nombre": "Reyser",
8   "direccion": "Calle Tacna",
9   "telefono": "987471074",
10  "correo_electronico": "rzapata@unsa.edu.pe"
11 },
12 {
13   "id": 4,
14   "nombre": "Javier",
15   "direccion": "Calle 1234",
16   "telefono": "789159625",
17   "correo_electronico": "Javier_Juli@gmail.com"
18 },
19 {
20   "id": 5,
21   "nombre": "Carlos",
22   "direccion": "Av. Los Incas",
23   "telefono": "912345678",
24   "correo_electronico": "carlos@hotmail.com"
25 },
26 {
27   "id": 6,
28   "nombre": "Joaquin",
29   "direccion": "Av. Perú",
30   "telefono": "741951851",
31   "correo_electronico": "joaquin@hotmail.com"
32 }
33 ]
  
```

Figura 4: GET de todos los Contactos desde la web

**GET/id/** - Este GET, es específico, devuelve solo 1 contacto de acuerdo al id ingresado en la url.



Thunder Client interface showing a GET request to `127.0.0.1:8000/api/get/5/`. The response is a JSON object for the contact with id 5.

```

1 {
2   "id": 5,
3   "nombre": "Carlos",
4   "direccion": "Av. Los Incas",
5   "telefono": "912345678",
6   "correo_electronico": "carlos@hotmail.com"
7 }
  
```

Figura 5: GET de un contacto desde Thunder Client

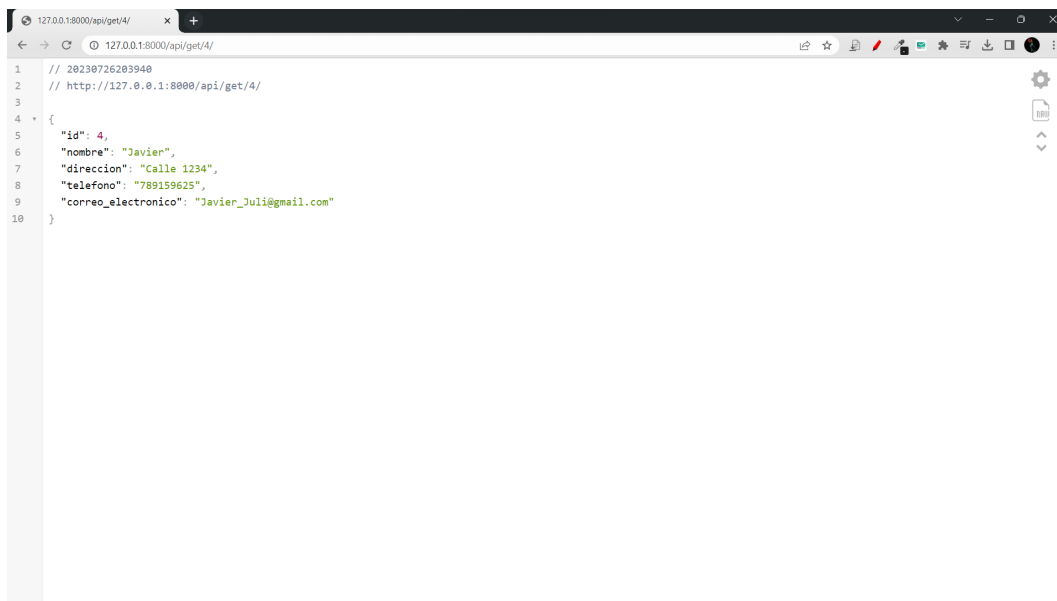


Figura 6: GET de un contacto desde la web

**POST** - Para el post necesitamos enviar todos los datos en formato JSON dentro del Body, y así obtenemos una respuesta correcta y añadimos un nuevo contacto desde nuestra API

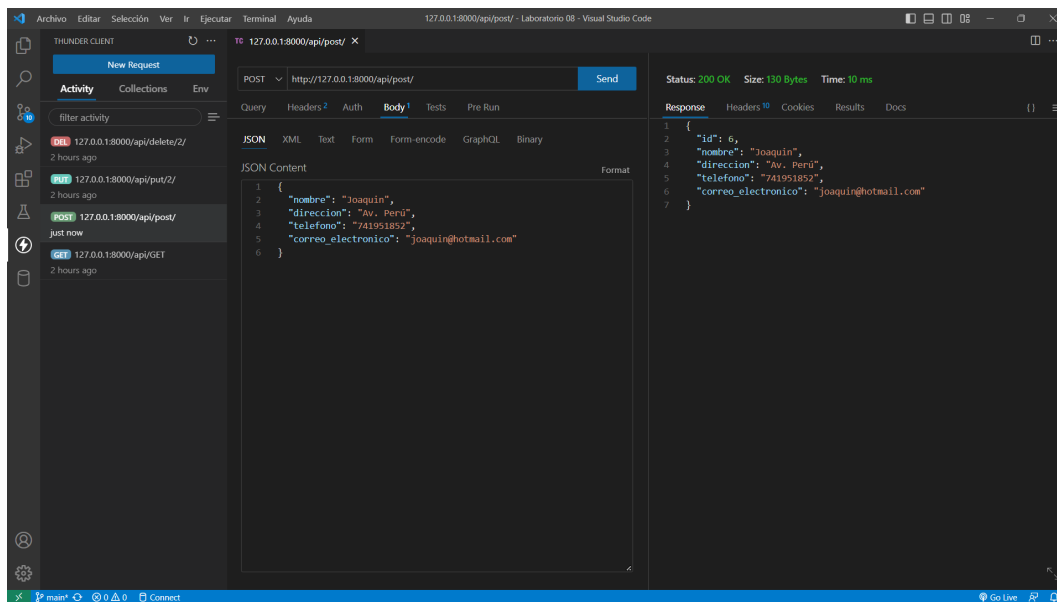


Figura 7: POST de un contacto desde Thunder Client

**PUT** - Para el put, debemos ingresar el id en la url y los demás datos dentro del Body, así hacemos una modificación de todos los datos, de lo contrario, se mandará mensaje de error "Contacto no encontrado"



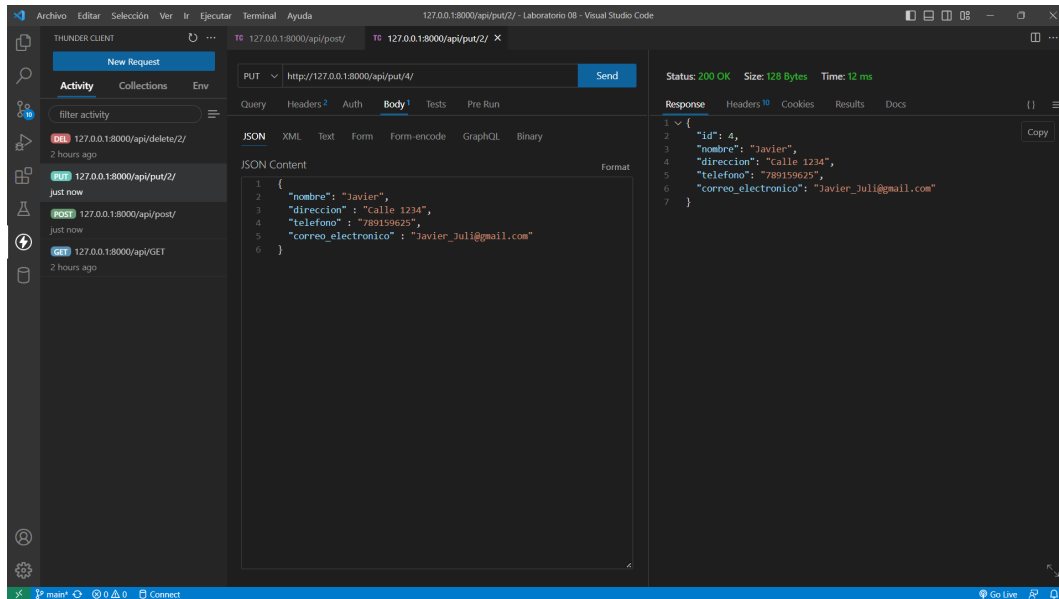


Figura 8: PUT de un contacto desde Thunder Client

**DELETE** - Para el delete, debemos de solo ingresar el id del contacto dentro de la url, con esto conseguimos eliminar el contacto deseado, de lo contrario, se mandará mensaje de error “Contacto no encontrado”

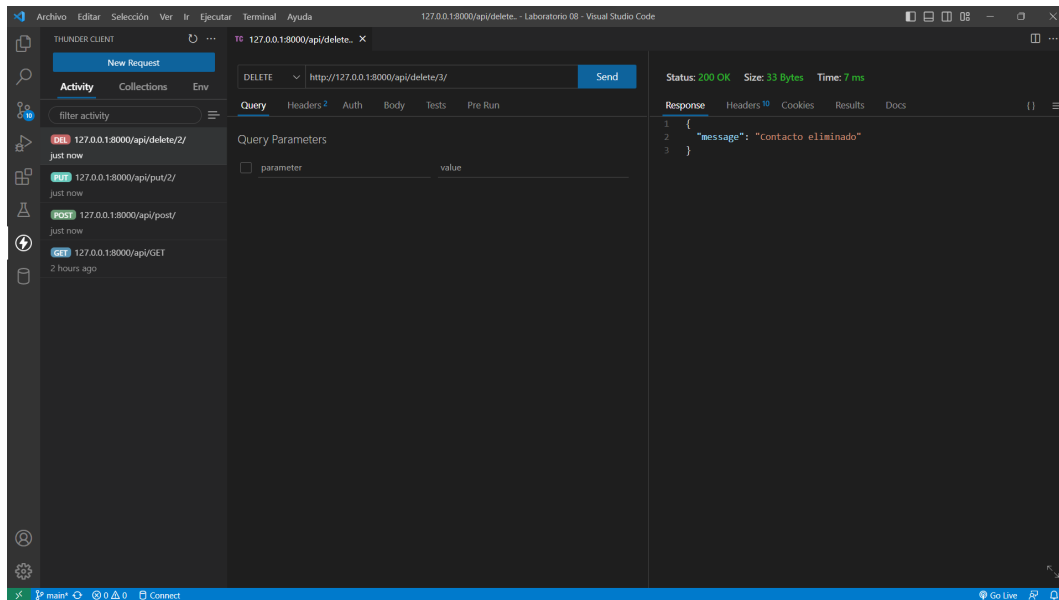


Figura 9: DELETE de un contacto desde Thunder Client

## 2.2. Como consumir una API de DNI para Validaciones

Como segundo proyecto presentado, es el como consumir una API de la WEB encargada de devolver JSON con información de un DNI, para así validar por ejemplo el registro de cuentas para un

Casino Virtual.

Para abrir este servidor debemos ejecutar los siguientes comandos desde nuestra terminal:

Listing 10: Ingresango a la carpeta dniAPI

```
$ cd dniAPI
```

Ahora debemos abrir el servidor manage.py

Listing 11: Abriendo el servidor de este proyecto de consumir API DNI

```
$ python .\manage runserver
```

A continuación se presentarán los códigos más importantes de este proyecto con su explicación:

### 2.2.1. Proyecto dniAPI

Primero se definieron las url con las que se trabajará esta simulación de Register y Login de Casino Virtual

Listing 12: dniAPI/urls.py

```
1 from django.contrib import admin
2 from django.urls import path
3 from dni import views
4 from django.contrib.auth import views as auth_views
5
6 urlpatterns = [
7     path('admin/', admin.site.urls),
8     path('', views.homeView, name='home'),
9     path('login/', views.loginView, name='login'),
10    path('logout/', auth_views.LogoutView.as_view(next_page='home'), name='logout'),
11    path('register/', views.registerView, name='register'),
12 ]
```

### 2.2.2. App dni

Ahora, haremos nuestras modelo para la base de datos, en la que usaremos un modelo y un AUTH personalizado, hecho para nuestra validación:

Listing 13: dniAPI/dni/models.py

```
1 from django.db import models
2 from django.contrib.auth.models import AbstractBaseUser, BaseUserManager, PermissionsMixin
3
4 class UserManager(BaseUserManager):
5     def create_user(self, dni, nombres, apellido_paterno, apellido_materno, password=None,
6                     **extra_fields):
7         extra_fields.setdefault('is_staff', False)
8
9         if not dni:
10             raise ValueError('El DNI debe ser proporcionado.')
11
12         user = self.model(
13             dni=dni,
```

```
13         nombres=nombres,
14         apellido_paterno=apellido_paterno,
15         apellido_materno=apellido_materno,
16         **extra_fields
17     )
18     user.set_password(password)
19     user.save(using=self._db)
20     return user
21
22     def create_superuser(self, dni, nombres, apellido_paterno, apellido_materno,
23                          password=None, **extra_fields):
24         extra_fields.setdefault('is_staff', True)
25         extra_fields.setdefault('is_superuser', True)
26
27         if extra_fields.get('is_staff') is not True:
28             raise ValueError('Superuser debe tener is_staff=True.')
29         if extra_fields.get('is_superuser') is not True:
30             raise ValueError('Superuser debe tener is_superuser=True.')
31
32         return self.create_user(dni, nombres, apellido_paterno, apellido_materno, password,
33                                **extra_fields)
34
35     def authenticate_user(self, dni=None, password=None):
36         try:
37             user = self.get(dni=dni)
38             if user.password == password:
39                 return user
40             return None
41         except self.model.DoesNotExist:
42             return None
43
44 class userData(AbstractBaseUser, PermissionsMixin):
45     dni = models.CharField(max_length=8, unique=True)
46     nombres = models.CharField(max_length=100)
47     apellido_paterno = models.CharField(max_length=100)
48     apellido_materno = models.CharField(max_length=100)
49     password = models.CharField(max_length=100, null=True)
50
51     objects = UserManager()
52
53     USERNAME_FIELD = 'dni'
54     REQUIRED_FIELDS = ['nombres', 'apellido_paterno', 'apellido_materno']
55
56     def __str__(self):
57         return self.dni
```

Posteriormente, se definieron los forms para el login y register de nuestra aplicación:

Listing 14: dniAPI/dni/forms.py

```
1 from django import forms
2
3 class BaseForm(forms.Form):
4     def __init__(self, *args, **kwargs):
5         super().__init__(*args, **kwargs)
6         for field_name, field in self.fields.items():
7             field.widget.attrs['class'] = 'form-control'
```

```
8         field.widget.attrs.setdefault('autocomplete', 'off')
9
10 class LoginForm(BaseForm):
11     dni = forms.CharField(label="DNI", max_length=8, widget=forms.TextInput(attrs={
12         'placeholder': 'Ingresa tu DNI',
13         'data-lpignore': 'true',
14     }))
15     contrasea = forms.CharField(label="Contrasea", widget=forms.PasswordInput(attrs={
16         'placeholder': 'Introduce tu contrasea',
17         'data-lpignore': 'true',
18     }))
19
20 class RegisterForm(BaseForm):
21     dni = forms.CharField(label="DNI", max_length=8, widget=forms.TextInput(attrs={
22         'placeholder': 'Ingresa tu DNI',
23     }))
24     nombres = forms.CharField(label="Nombres", max_length=100, widget=forms.TextInput(attrs={
25         'placeholder': 'Ingresa tus nombres',
26     }))
27     apellido_paterno = forms.CharField(label="Apellido Paterno", max_length=100,
28         widget=forms.TextInput(attrs={
29         'placeholder': 'Ingresa tu apellido paterno',
30     }))
31     apellido_materno = forms.CharField(label="Apellido Materno", max_length=100,
32         widget=forms.TextInput(attrs={
33         'placeholder': 'Ingresa tu apellido materno',
34     }))
35     contrasea = forms.CharField(label="Contrasea", widget=forms.PasswordInput(attrs={
36         'placeholder': 'Crea tu contrasea',
37     })))
```

Luego, hacemos nuestras views, cada uno con sus validaciones respectivas para lograr una correcta ejecución, mostrando mensaje si la validación no fué exitosa:

Listing 15: dniAPI/dni/views.py

```
1 import requests
2 from django.shortcuts import render, redirect
3 from django.contrib.auth import login
4 from .forms import LoginForm, RegisterForm
5 from .models import userData
6
7 def homeView(request):
8     return render(request, 'home.html', {'user': request.user})
9
10 def loginView(request):
11     if request.method == 'POST':
12         form = LoginForm(request.POST)
13         if form.is_valid():
14             dni = form.cleaned_data.get('dni')
15             password = form.cleaned_data.get('contrasea')
16             user = userData.objects.authenticate_user(dni=dni, password=password)
17             if user is not None:
18                 login(request, user)
19                 return redirect('home')
20         else:
21             form = LoginForm()
```

```

22     return render(request, 'login.html', {'form': form})
23
24 def registerView(request):
25     if request.method == 'POST':
26         form = RegisterForm(request.POST)
27         if form.is_valid():
28             dni = form.cleaned_data['dni']
29             nombres = form.cleaned_data['nombres']
30             apellido_paterno = form.cleaned_data['apellido_paterno']
31             apellido_materno = form.cleaned_data['apellido_materno']
32             password = form.cleaned_data['contrasea']
33
34             if validate_dni(dni, nombres, apellido_paterno, apellido_materno):
35                 userData.objects.create(dni=dni, nombres=nombres,
36                                         apellido_paterno=apellido_paterno, apellido_materno=apellido_materno,
37                                         password=password)
38                 return redirect('home')
39             else:
40                 alert_message = "Los datos del DNI no coinciden con la API. Verifcalos e
41                               intenta nuevamente."
42                 return render(request, 'register.html', {'form': form, 'alert_message':
43                                                         alert_message})
44         else:
45             form = RegisterForm()
46     return render(request, 'register.html', {'form': form})
47
48 def validate_dni(dni, nombres, apellido_paterno, apellido_materno):
49     url =
50         f'https://dniruc.apisperu.com/api/v1/dni/{dni}?token=eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJlbWFPbCI6IHNiJ9.eyJlbWFPbCI6IHNiJ9.eyJlbWFPbCI6IHNiJ9'
51     response = requests.get(url)
52     data = response.json()
53     if data.get('success', False):
54         api_nombres = data.get("nombres", "").upper().replace(" ", "")
55         api_apellidos = (data.get("apellidoPaterno", "") +
56                          data.get("apellidoMaterno", "")).upper().replace(" ", "")
57         return api_nombres == nombres.upper().replace(" ", "") and api_apellidos ==
58             (apellido_paterno + apellido_materno).upper().replace(" ", "")
59     return False

```

Por ultimo, mostraremos nuestro Home.html dependiendo de si está autenticado o no:

Listing 16: dniAPI/dni/templates/home.html

```

1 <DOCTYPE html>
2 <html lang="es">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Casino FastMoney</title>
7 </head>
8 <body>
9   {% if user.is_authenticated %}
10   <h1>Bienvenido, {{ user.nombres }}</h1>
11   <form action="{% url 'logout' %}" method="post">
12     {% csrf_token %}
13     <button type="submit">Cerrar sesin</button>
14   </form>

```

```
15     {% else %}  
16         <h1>Bienvenido, por favor inicia sesin o regstrate</h1>  
17         <a href="{% url 'login' %}">Iniciar sesin</a>  
18         <a href="{% url 'register' %}">Registrarse</a>  
19     {% endif %}  
20 </body>  
21 </html>
```

### 2.2.3. Ejecución

Ahora que vimos como funciona nuestro proyecto con el código fuente, podemos proceder a mostrar como es la ejecución

#### Home sin autenticación

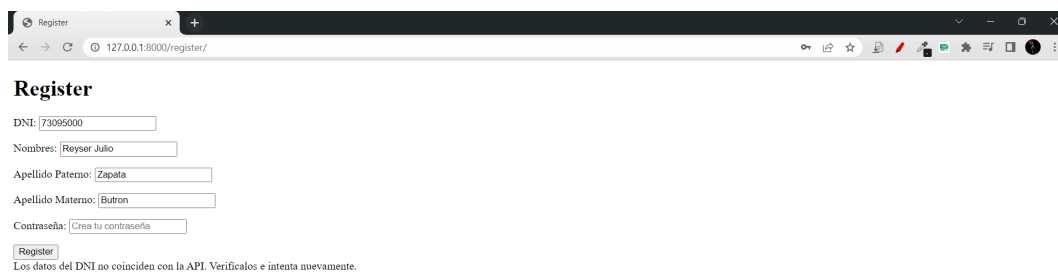
Esto es como se mostrará nuestro Home cuando lo abrimos primera vez, es decir, sin registrarnos:



Figura 10: Home sin autenticarse

#### Register inválido

Cuando hay un register invalido mandará un mensaje de alerta, es decir, los datos ingresados, DNI, nombres, apellidos, no coinciden con el dni ingresado, es decir, datos falsos:



**Register**

DNI:

Nombres:

Apellido Paterno:

Apellido Materno:

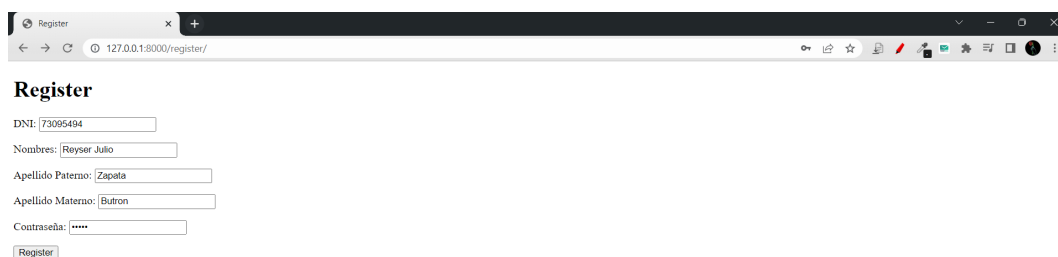
Contraseña:

Los datos del DNI no coinciden con la API. Verificalos e intenta nuevamente.

Figura 11: Registro inválido

### Register válido

Cuando ingresamos el dni válido, junto con el nombre y apellidos correctos, se nos mandará al Home, para hacer el login correspondiente:



**Register**

DNI:

Nombres:

Apellido Paterno:

Apellido Materno:

Contraseña:

Figura 12: Register con datos válidos

Respuesta del Api:

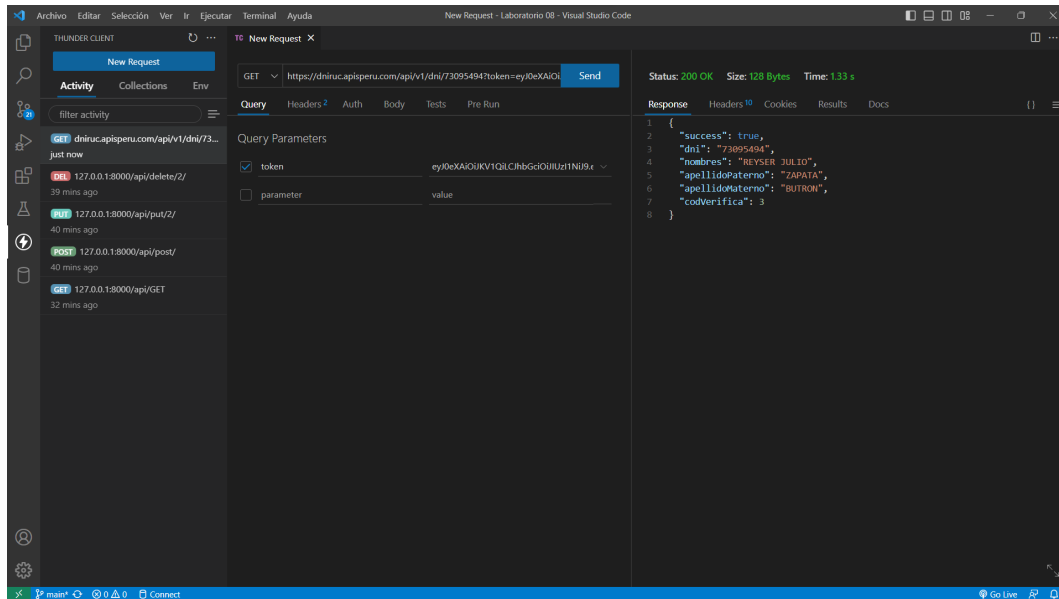


Figura 13: Respuesta de la API don el dni

## Login

Este es el apartado del Login, es decir, ingresaremos nuestro dni y contraseña:

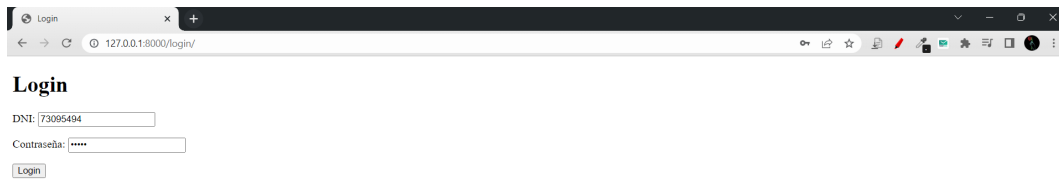


Figura 14: Login.html

## Home autenticado

Cuando hicimos un Login exitoso, el home se nos mostrará de esta forma:



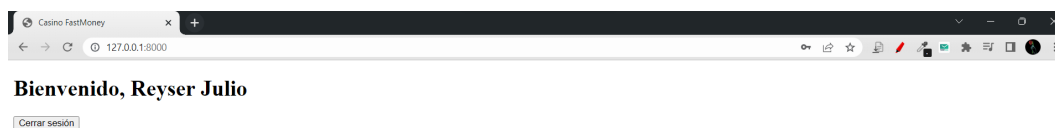


Figura 15: Home con login exitoso

### 3. Equipos, materiales y temas utilizados

- Visual Studio 1.80
- Python 3.11
- Git 2.39.2.
- Entorno virtual
- Cuenta en GitHub con el correo institucional.
- Django 4
- Django Rest Framework

### 4. URL de Repositorio Github

- URL del Repositorio GitHub para clonar o recuperar.
- <https://github.com/ReyserLyn/pweb-lab-08.git>

### 5. Actividades con el repositorio GitHub

#### 5.1. Clonamos el repositorio GitHub

- Al ser un proyecto en repositorio independiente, para su ejecución debemos clonar el repositorio.

Listing 17: Clonando Repositorio del Laboratorio 08

```
$ git clone https://github.com/ReyserLyn/pweb-lab-08.git
```

Listing 18: Dirigiéndonos al directorio de trabajo

```
$ cd pweb-lab-08
```

Una vez estemos dentro de nuestro directorio de trabajo, tenemos que crear el entorno virtual, activarlo e instalar las dependencias necesarias para la correcta ejecución de los proyectos.

Listing 19: Creando entorno virtual

```
$ py -m venv venv
```

Listing 20: Activamos nuestro entorno virtual

```
$ venv/Scripts/activate
```

Listing 21: Descargando dependencias

```
$ pip install -r requirements.txt
```

Una vez tengamos esto hecho, ya podremos abrir los servidor de los diferentes proyectos y probar su funcionalidad.

## 5.2. Estructura de laboratorio 08

- El contenido que se entrega en este laboratorio es el siguiente:

```
pweb-lab-08/
|--- djaangoApiRest
|   |--- api
|       |--- migrations
|       |--- __init__.py
|       |--- admin.py
|       |--- apps.py
|       |--- models.py
|       |--- serializers.py
|       |--- tests.py
|       |--- urls.py
|       |--- views.py
|   |--- asgi.py
|   |--- middleware.py
|   |--- settings.py
|   |--- urls.py
|   |--- wsgi.py
|--- dniAPI
|   |--- dni
|       |--- migrations
|       |--- static
|       |--- templates
|       |--- __init__.py
|       |--- admin.py
|       |--- apps.py
|       |--- forms.py
|       |--- models.py
|       |--- tests.py
|       |--- views.py
```

```
|--- __init__.py
|--- asgi.py
|--- db.sqlite3
|--- manage.py
|--- settings.py
|--- urls.py
|--- wsgi.py
|--- Latex (Informe Latex)
|--- Tutorial (Ejercicio Django - QuickStart)
|--- .gitignore
|--- db
|--- manage.py
|--- requirements.txt
```

Siendo el directorio más importante "DjangoApiRest", es donde se evidencia nuestra CRUD API.

### 5.3. Commits

En la captura de pantalla mostrada a continuación, se pueden ver los commits más importantes tratados para este Laboratorio 08:

```
commit 35c329b8a75e7e96d08fd99dd1c647647ba5f824 (HEAD -> main, origin/main, origin/HEAD)
Author: Reyser Julio Zapata Butron <rzapata@unsa.edu.pe>
Date: Wed Jul 26 18:52:40 2023 -0500

    Modificado y arreglado

commit 3afbcad9ff5e6f74ad96e5579bf8ad9734df8d10
Author: Reyser Julio Zapata Butron <rzapata@unsa.edu.pe>
Date: Wed Jul 26 18:52:21 2023 -0500

    requirements hecho

commit d35e0bbfd0456df03241f4b3560d2bdaf67bdcde
Author: Reyser Julio Zapata Butron <rzapata@unsa.edu.pe>
Date: Wed Jul 26 18:51:22 2023 -0500

    API REST DJANGO Creada y funcional

commit 9d6bbec2d308c212a3ef4743c7b0d5152f34c540
Author: Reyser Julio Zapata Butron <rzapata@unsa.edu.pe>
Date: Wed Jul 26 17:32:47 2023 -0500

    Proyecto consumiendo API de DNI

commit 97e9cdfacd0fdd8b0a59e2920c52476bdb0b5b69
Author: ReyserLyn <rzapata@unsa.edu.pe>
Date: Tue Jul 25 13:41:57 2023 -0500

    Avance Laboratorio 08

commit 5743f3a81eb3f6dd8eda55f9c9b5915c8b91befd
Author: ReyserLyn <rzapata@unsa.edu.pe>
Date: Tue Jul 25 13:41:40 2023 -0500

    Tutorial corregido

commit 03082b78c2dd72f9c0e29e8b600b3d484685af06
Author: ReyserLyn <rzapata@unsa.edu.pe>
Date: Tue Jul 25 13:17:49 2023 -0500

    Tutorial completo y correcto
```

Figura 16: Captura de pantalla de los commits más importantes

## 6. Pregunta: ¿Cuál fué la mayor dificultad del uso de este framework?

La mayor dificultad que enfrentamos al usar este framework fue familiarizarnos con su estructura y conceptos clave. Al principio, nos resultó un poco abrumador entender cómo se organiza el proyecto, cómo definir modelos, serializadores y vistas, y cómo configurar las rutas para las API. Sin embargo, a medida que avanzamos en el desarrollo y consultamos la documentación oficial y otros recursos, pudimos superar esta dificultad y comprender mejor cómo aprovechar todas las capacidades que ofrece Django Rest Framework. Una vez superado este obstáculo inicial, pudimos aprovechar al máximo las funcionalidades del framework y desarrollar una API robusta y eficiente para nuestro proyecto.

## 7. Rúbricas

### 7.1. Entregable Informe

Tabla 1: Tipo de Informe

<b>Informe</b>	
<b>Latex</b>	El informe está en formato PDF desde Latex, con un formato limpio (buena presentación) y facil de leer.

## 7.2. Rúbrica para el contenido del Informe y demostración

- El alumno debe marcar o dejar en blanco en celdas de la columna **Checklist** si cumple con el ítem correspondiente.
- Si un alumno supera la fecha de entrega, su calificación será sobre la nota mínima aprobada, siempre y cuando cumpla con todos los ítems.
- El alumno debe autocalificarse en la columna **Estudiante** de acuerdo a la siguiente tabla:

Tabla 2: Niveles de desempeño

	Nivel			
<b>Puntos</b>	Insatisfactorio 25 %	En Proceso 50 %	Satisfactorio 75 %	Sobresaliente 100 %
<b>2.0</b>	0.5	1.0	1.5	2.0
<b>4.0</b>	1.0	2.0	3.0	4.0

Tabla 3: Rúbrica para contenido del Informe y demostración

	Contenido y demostración	Puntos	Checklist	Estudiante	Profesor
<b>1. GitHub</b>	Hay enlace URL activo del directorio para el laboratorio hacia su repositorio GitHub con código fuente terminado y fácil de revisar.	2	X	2	
<b>2. Commits</b>	Hay capturas de pantalla de los commits más importantes con sus explicaciones detalladas. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>3. Código fuente</b>	Hay porciones de código fuente importantes con numeración y explicaciones detalladas de sus funciones.	2	X	2	
<b>4. Ejecución</b>	Se incluyen ejecuciones/pruebas del código fuente explicadas gradualmente.	2	X	2	
<b>5. Pregunta</b>	Se responde con completitud a la pregunta formulada en la tarea. (El profesor puede preguntar para refrendar calificación).	2	X	2	
<b>6. Fechas</b>	Las fechas de modificación del código fuente están dentro de los plazos de fecha de entrega establecidos.	2		0	
<b>7. Ortografía</b>	El documento no muestra errores ortográficos.	2	X	2	
<b>8. Madurez</b>	El Informe muestra de manera general una evolución de la madurez del código fuente, explicaciones puntuales pero precisas y un acabado impecable. (El profesor puede preguntar para refrendar calificación).	4	X	4	
<b>Total</b>		20		18	

## 8. Referencias

- <https://www.django-rest-framework.org/tutorial/quickstart/>
- <https://www.django-rest-framework.org/tutorial/4-authentication-and-permissions/>
- <https://coffeebytes.dev/como-personalizar-el-modelo-user-en-django/>
- <https://www.django-rest-framework.org/>