### **Introduction**

At BeautySalon, we offer a wide range of services and features to meet the needs of our community of beauty lovers.

### **Basic Requirements**

Make sure you have the following packages installed:

* Python (latest version)
* Django (latest version)
* Django Rest Framework

### **Installation Steps**

**Configure Git**: Set up your Git user name and email.

* git config --global user.name "Your Name"
* git config --global user.email “youremail@example.com”

**Create a Local Repository**: Initialize a local Git repository.

* mkdir example-project
* cd example-project
* git init

### **Dependencies and Virtual Environment**

**Creating a Virtual Environment**: Create a virtual environment named BeautySalon.

python -m venv myprojectenv

Activate the virtual environment:

* On macOS/Linux: source myprojectenv/bin/activate
* On Windows: myprojectenv\Scripts\activate

**Installing Django and Django Rest Framework**: Install Django and DRF using pip.

* pip install django
* pip install djangorestframework

**Creating a Django Project**: Generate a new Django project.

* django-admin startproject myproject

Replace BeautySalon with your desired project name.

### **Running the Django Server**

Navigate to your project directory and start the Django server.

* cd BeautySalon
* pip install psycopg2
* pip install psycopg2-binary
* python manage.py runserver

### **\*pgadmin create database**

### We move to the folder that has the project

### **\* python3 manage.py runserver**

In Django, an application is a web application that does something – a self-contained feature or functionality of your project. Think of it as a specific district or neighborhood within our city, dedicated to a particular purpose, like residential, commercial, or industrial. Create your first app with:

python manage.py startapp myapp

Replace myapp with a suitable name for your application. This command creates a new directory with your app’s name, containing the basic files needed to start building your app.

### **Registering the App and DRF**

Update the INSTALLED\_APPS list in settings.py to include 'rest\_framework' and your app.

INSTALLED\_APPS = [...,

'rest\_framework',

'myapp',

...]

### **Configuring Django to Use PostgreSQL**

Update the DATABASES settings in settings.py with your PostgreSQL credentials.

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.postgresql',

'NAME': 'beautysalondatabase',

'USER': 'postgres',

'PASSWORD': 'yourpassword',

'HOST': 'localhost',

'PORT': '5432',

}

}

**API Specification:**

First, we need to define the endpoints that our API will have. This will depend on the specific functionality of your application, but here are some examples based on a blog system:

**Data Models Design:**

Now, let's create the data models in Django that will represent the main entities of our application. For a blog system, we could have the following models:

from django.db import models

from django.contrib.auth.models import User

class Client(models.Model):

name = models.CharField(max\_length=100)

phone = models.CharField(max\_length=15)

email = models.EmailField()

class Service(models.Model):

name = models.CharField(max\_length=100)

description = models.TextField()

price = models.DecimalField(max\_digits=10, decimal\_places=2)

details = models.JSONField()

class Appointment(models.Model):

client = models.ForeignKey(Client, on\_delete=models.CASCADE)

service = models.ForeignKey(Service, on\_delete=models.CASCADE)

date = models.DateField()

time = models.TimeField()

additional\_info = models.JSONField(blank=True, null=True)

class Booking(models.Model):

appointment = models.OneToOneField(Appointment, on\_delete=models.CASCADE)

payment\_status = models.BooleanField(default=False)

class UploadedFile(models.Model):

file = models.FileField(upload\_to='uploads/')

uploaded\_at = models.DateTimeField(auto\_now\_add=True)

**Data Serializers Design:**

from rest\_framework import serializers

from .models import Client, Service, Appointment, Booking

from django.contrib.auth.models import User

from rest\_framework import serializers

from .models import UploadedFile

class ClientSerializer(serializers.ModelSerializer):

class Meta:

model = Client

fields = '\_\_all\_\_'

class ServiceSerializer(serializers.ModelSerializer):

class Meta:

model = Service

fields = '\_\_all\_\_'

extra\_kwargs = {

'url': {'view\_name': 'service-detail', 'lookup\_field': 'pk'}

}

class AppointmentSerializer(serializers.ModelSerializer):

class Meta:

model = Appointment

fields = '\_\_all\_\_'

extra\_kwargs = {

'url': {'view\_name': 'client-detail', 'lookup\_field': 'pk'}

}

class BookingSerializer(serializers.ModelSerializer):

class Meta:

model = Booking

fields = '\_\_all\_\_'

class UserSerializer(serializers.ModelSerializer):

class Meta:

model = User

fields = ['id', 'username', 'email']

class UploadedFileSerializer(serializers.ModelSerializer):

class Meta:

model = UploadedFile

fields = '\_\_all\_\_'

**API\_VIEW:**

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import Client, Service, Appointment

from .serializers import ClientSerializer, ServiceSerializer, AppointmentSerializer

from django.contrib.auth.views import LoginView as BaseLoginView

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import Client, Service, Appointment, Booking

from .serializers import ClientSerializer, ServiceSerializer, AppointmentSerializer, BookingSerializer

from rest\_framework.views import APIView

from rest\_framework import generics

from django.contrib.auth.models import User

from django.contrib.auth.hashers import make\_password

from rest\_framework import serializers

from django.contrib.auth.models import User

from .serializers import UserSerializer

from rest\_framework.permissions import IsAuthenticated

from rest\_framework.parsers import FileUploadParser

from rest\_framework.authtoken.views import ObtainAuthToken

from rest\_framework.authtoken.models import Token

@api\_view(['GET', 'POST'])

def client\_list(request):

if request.method == 'GET':

clients = Client.objects.all()

serializer = ClientSerializer(clients, many=True)

return Response(serializer.data)

elif request.method == 'POST':

serializer = ClientSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET', 'PUT', 'DELETE'])

def client\_detail(request, pk):

try:

client = Client.objects.get(pk=pk)

except Client.DoesNotExist:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

if request.method == 'GET':

serializer = ClientSerializer(client)

return Response(serializer.data)

elif request.method == 'PUT':

serializer = ClientSerializer(client, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

elif request.method == 'DELETE':

client.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

@api\_view(['GET', 'POST'])

def service\_list(request):

if request.method == 'GET':

services = Service.objects.all()

serializer = ServiceSerializer(services, many=True)

return Response(serializer.data)

elif request.method == 'POST':

serializer = ServiceSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET', 'PUT', 'DELETE'])

def service\_detail(request, pk):

try:

service = Service.objects.get(pk=pk)

except Service.DoesNotExist:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

if request.method == 'GET':

serializer = ServiceSerializer(service)

return Response(serializer.data)

elif request.method == 'PUT':

serializer = ServiceSerializer(service, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

elif request.method == 'DELETE':

service.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

@api\_view(['GET', 'POST'])

def appointment\_list(request):

if request.method == 'GET':

appointments = Appointment.objects.all()

serializer = AppointmentSerializer(appointments, many=True)

return Response(serializer.data)

elif request.method == 'POST':

serializer = AppointmentSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET', 'PUT', 'DELETE'])

def appointment\_detail(request, pk):

try:

appointment = Appointment.objects.get(pk=pk)

except Appointment.DoesNotExist:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

if request.method == 'GET':

serializer = AppointmentSerializer(appointment)

return Response(serializer.data)

elif request.method == 'PUT':

serializer = AppointmentSerializer(appointment, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

elif request.method == 'DELETE':

appointment.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

@api\_view(['GET', 'POST'])

def booking\_list(request):

if request.method == 'GET':

bookings = Booking.objects.all()

serializer = BookingSerializer(bookings, many=True)

return Response(serializer.data)

elif request.method == 'POST':

serializer = BookingSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

@api\_view(['GET', 'PUT', 'DELETE'])

def booking\_detail(request, pk):

try:

booking = Booking.objects.get(pk=pk)

except Booking.DoesNotExist:

return Response(status=status.HTTP\_404\_NOT\_FOUND)

if request.method == 'GET':

serializer = BookingSerializer(booking)

return Response(serializer.data)

elif request.method == 'PUT':

serializer = BookingSerializer(booking, data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

elif request.method == 'DELETE':

booking.delete()

return Response(status=status.HTTP\_204\_NO\_CONTENT)

class AppointmentCreate(APIView):

def post(self, request):

serializer = AppointmentSerializer(data=request.data)

if serializer.is\_valid():

serializer.save()

return Response(serializer.data, status=status.HTTP\_201\_CREATED)

return Response(serializer.errors, status=status.HTTP\_400\_BAD\_REQUEST)

class UserCreate(generics.CreateAPIView):

queryset = User.objects.all()

serializer\_class = UserSerializer

def create(self, request, \*args, \*\*kwargs):

password = make\_password(request.data.get('password'))

request.data['password'] = password

return super().create(request, \*args, \*\*kwargs)

class ProtectedView(APIView):

permission\_classes = [IsAuthenticated]

def get(self, request):

content = {'message': 'This is a protected endpoint'}

return Response(content)

class FileUploadView(APIView):

parser\_classes = [FileUploadParser]

permission\_classes = [IsAuthenticated]

def post(self, request, \*args, \*\*kwargs):

file\_obj = request.data['file']

# Handle the uploaded file

return Response({'status': 'File uploaded successfully'})

class LoginView(ObtainAuthToken):

def post(self, request, \*args, \*\*kwargs):

serializer = self.serializer\_class(data=request.data, context={'request': request})

serializer.is\_valid(raise\_exception=True)

user = serializer.validated\_data['user']

token, created = Token.objects.get\_or\_create(user=user)

return Response({'token': token.key})

**APP URLS:**

from django.urls import path

from django.contrib import admin

from rest\_framework.authtoken.views import obtain\_auth\_token

from drf\_yasg.views import get\_schema\_view

from drf\_yasg import openapi

from rest\_framework import permissions

from . import views

from .views import AppointmentCreate, UserCreate, ProtectedView, FileUploadView, LoginView

from rest\_framework.authtoken.views import obtain\_auth\_token

# Schema view for API documentation

schema\_view = get\_schema\_view(

openapi.Info(

title="My Awesome API",

default\_version='v1',

description="Descripción de mi API",

),

public=True,

permission\_classes=[permissions.AllowAny],

)

urlpatterns = [

path('admin/', admin.site.urls),

path('clients/', views.client\_list, name='client-list'),

path('api/token/', obtain\_auth\_token, name='obtain\_auth\_token'),

path('clients/<int:pk>/', views.client\_detail, name='client-detail'),

path('services/', views.service\_list, name='service-list'),

path('services/<int:pk>/', views.service\_detail, name='service-detail'),

path('appointments/', views.appointment\_list, name='appointment-list'),

path('appointments/<int:pk>/', views.appointment\_detail, name='appointment-detail'),

path('api/appointments/create/', AppointmentCreate.as\_view(), name='create-appointment'),

path('api/users/register/', UserCreate.as\_view(), name='user-create'),

path('swagger/schema/', schema\_view.without\_ui(cache\_timeout=0), name='schema-json'),

path('swagger/', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),

path('api-docs/', schema\_view.with\_ui('swagger', cache\_timeout=0), name='schema-swagger-ui'),

path('api/login/', obtain\_auth\_token, name='login'),

path('protected/', ProtectedView.as\_view(), name='protected'),

path('upload/', FileUploadView.as\_view(), name='file-upload'),

path('accounts/login/', LoginView.as\_view(), name='custom\_login'),

]

**RAIZ URLS:**

from django.contrib import admin

from django.urls import path, include

from django.contrib.auth import views as auth\_views

urlpatterns = [

path('admin/', admin.site.urls),

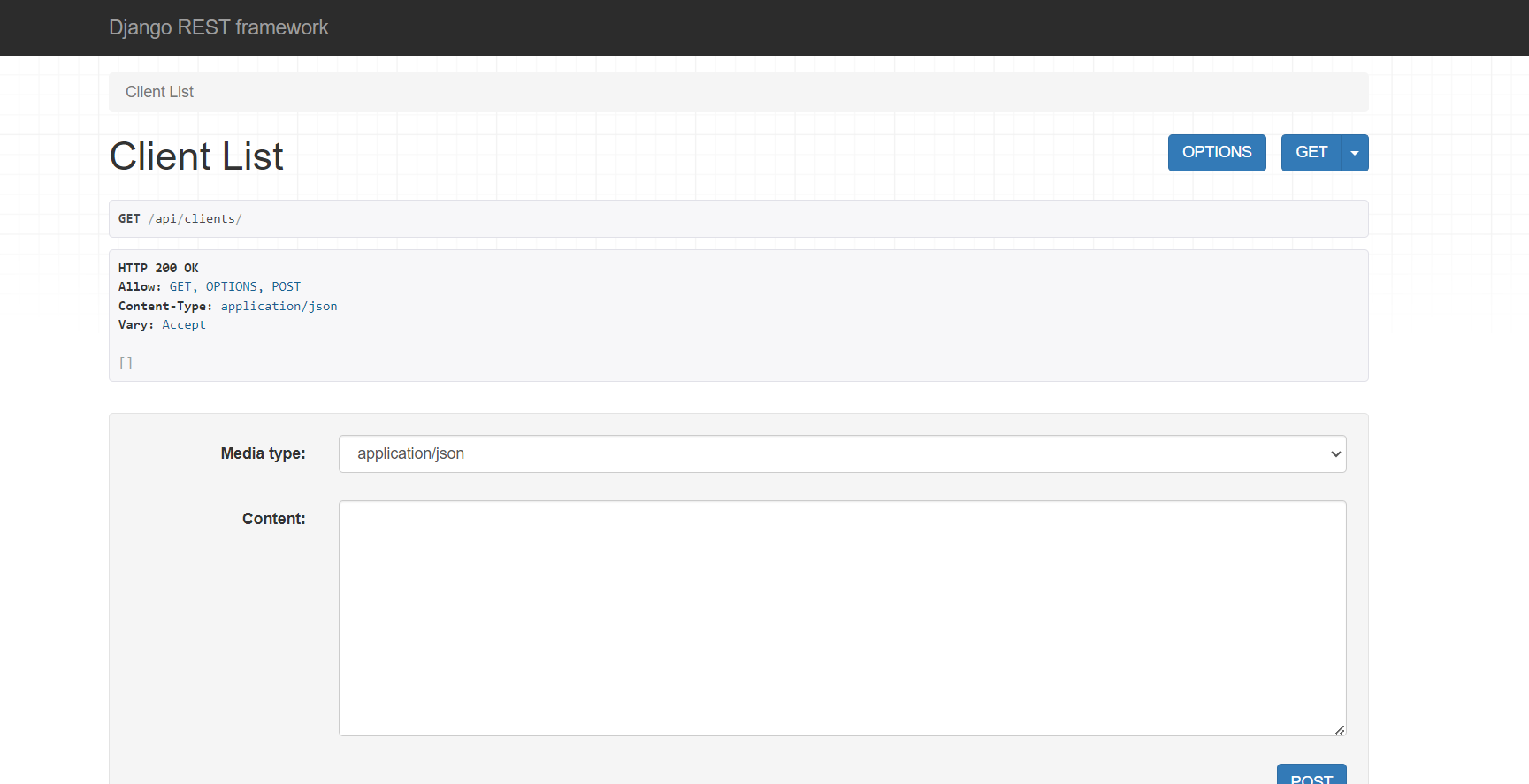
path('api/', include('BeautySalonapp.urls')),

path('accounts/login/', auth\_views.LoginView.as\_view(template\_name='registration/login.html'), name='login'),

]

**What django would look like:**

http://localhost:8000/api/posts/



use Insomnia:

In the view serializer url, the authentication and login configuration is already found.

First, you need a token. Assuming you have set up a token authentication system and a user in your Django application, you can obtain a token by making a POST request to your login or token obtain endpoint. This request will include the username and password of your user.

* **Create a New Request** in Insomnia.
* **Set the Method** to POST.
* **Enter the URL** for your token obtain endpoint (http://localhost:8000/api/login/).

**Set the Body** to JSON and include your username and password:  
{

"username": "your\_username",

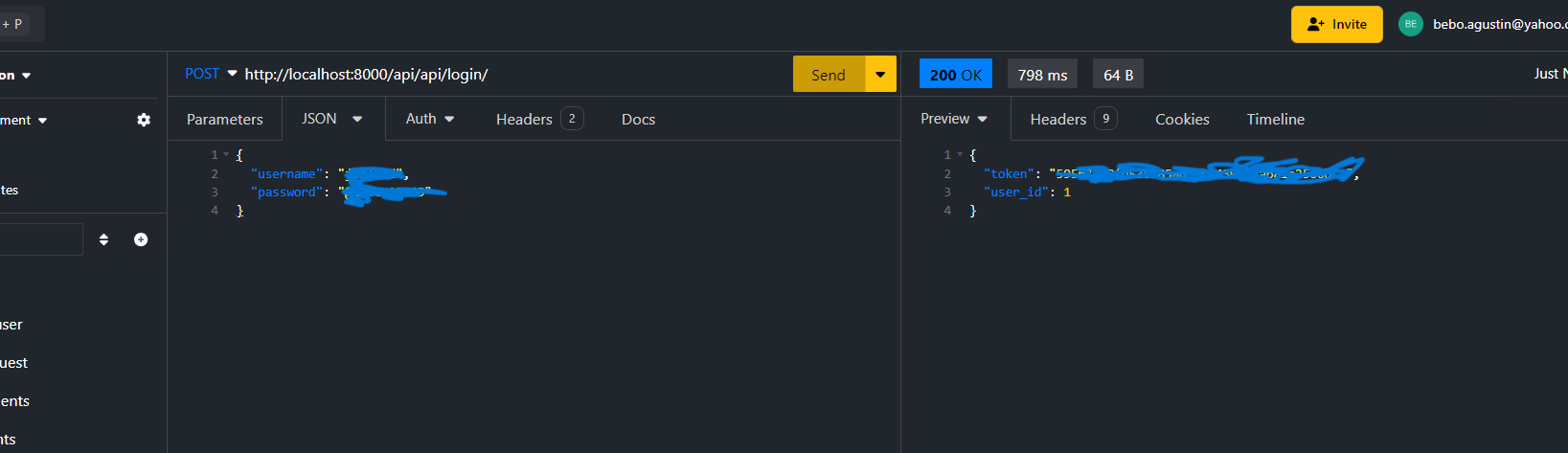
"password": "your\_password"

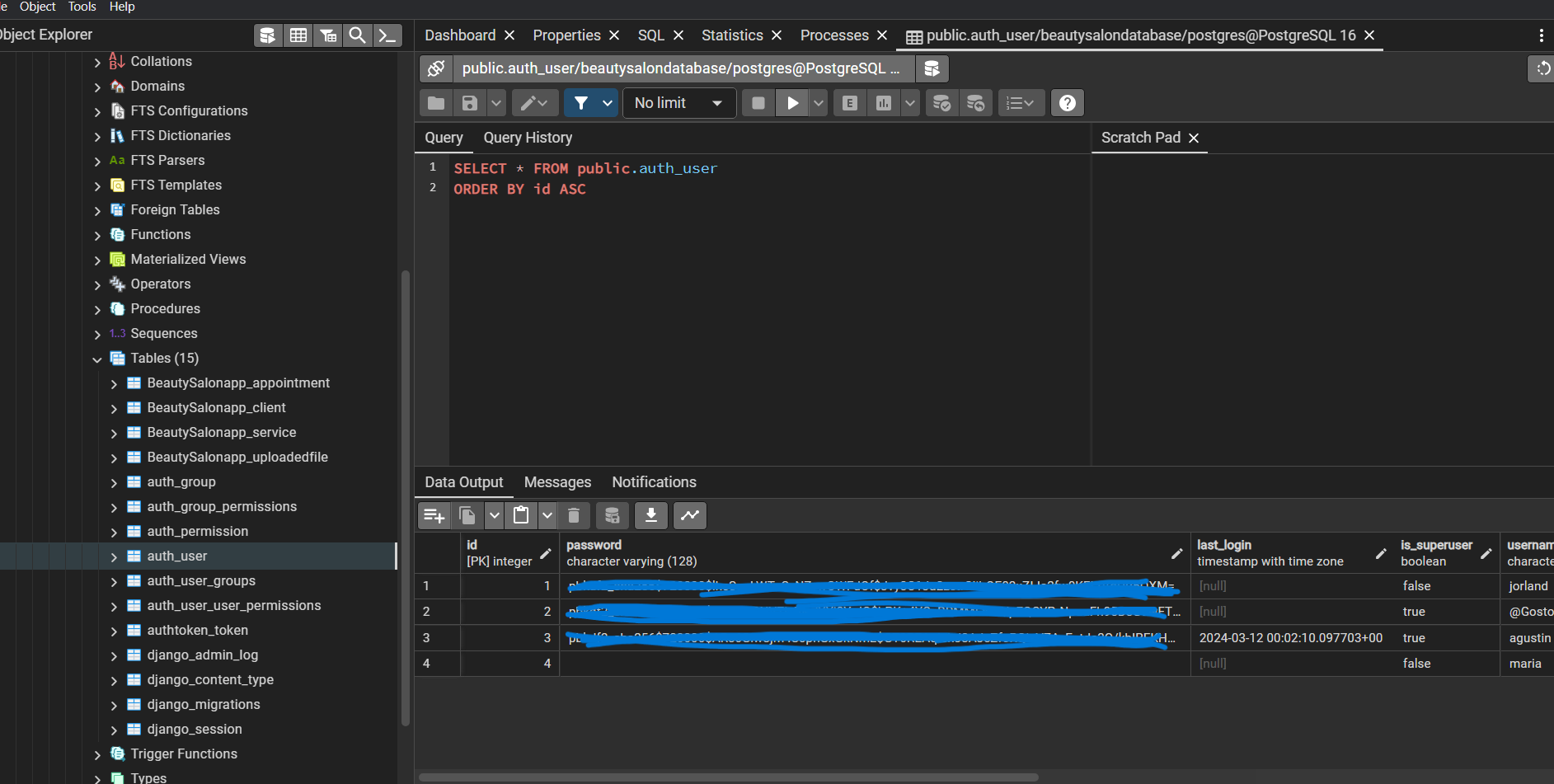
}

* **Send the Request**. You should receive a response that includes the token.

Now, with your token, you can make an authenticated request to the protected route.

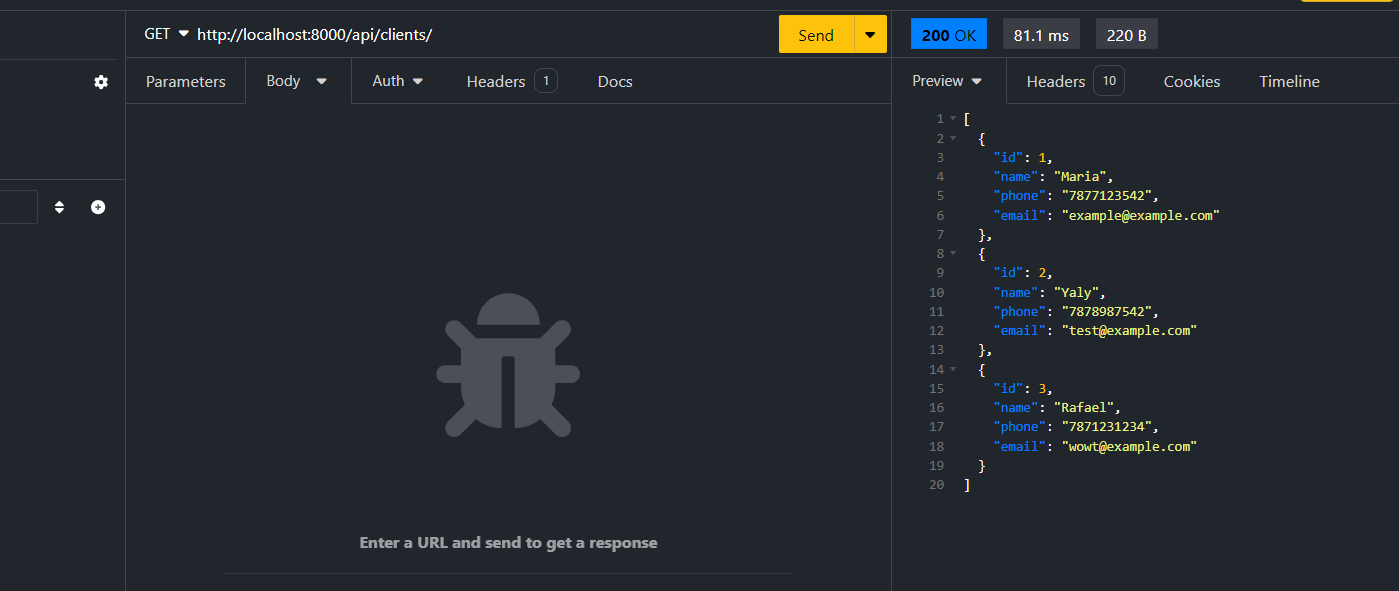
* **Create a New Request** in Insomnia for the protected route.
* **Set the Method** to GET.
* **Enter the URL** for the protected route (e.g., http://localhost:8000/protected/).
* **Add a Header** named Authorization with the value Token your\_token\_here, replacing your\_token\_here with the actual token you received from the login request.
  + It should look like this: Authorization: Token 9944b09199c62bcf9418ad846dd0e4bbdfc6ee4b
* **Send the Request**. Since you are authenticated, yo

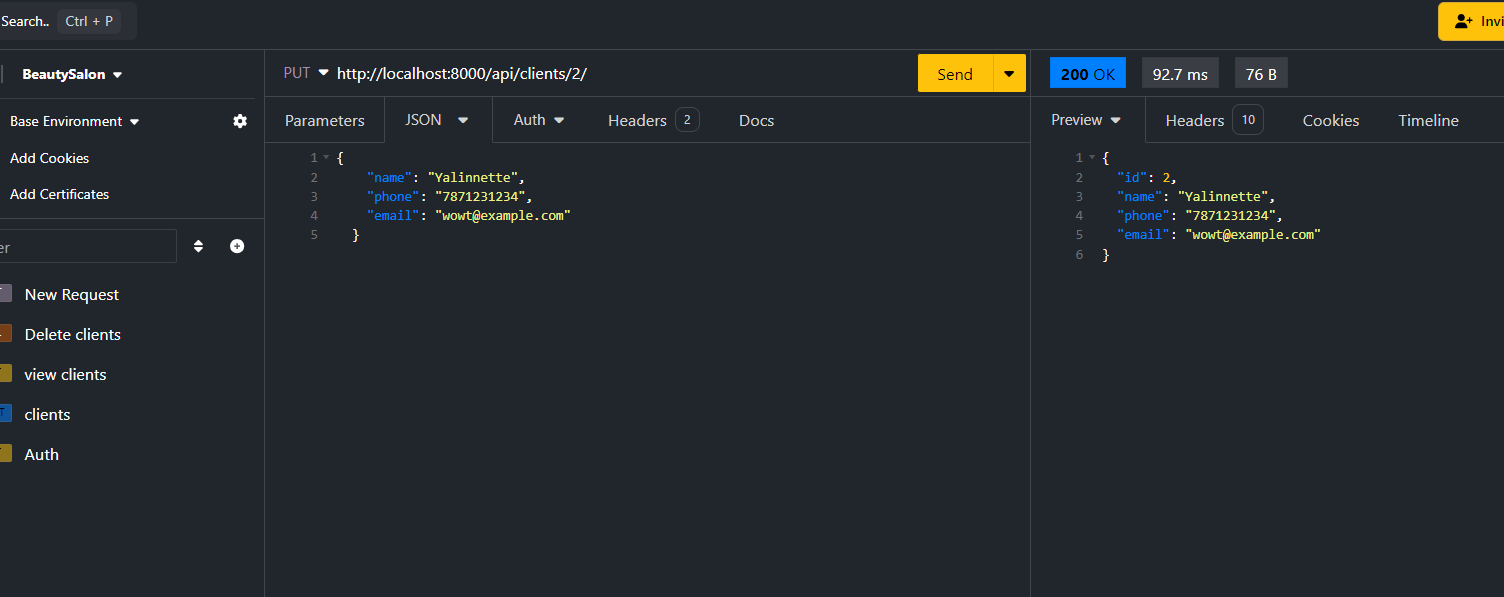




Adding data

1. GET - shows me the added information
2. POST - post the information we want
3. DELETE - deletes the information we want
4. PUT - updates the information we want





Here we are adding clients with POST we add them, and with GET we can visualize the clients. and with DELETE eliminate us.

post http://localhost:8000/api/clients/

{

"name": "Yalinnette",

"phone": "7879899898",

"email": "example@example.com"

}

get http://localhost:8000/api/clients/

{

"name": "Yalinnette",

"phone": "7879899898",

"email": "example@example.com"

}

{

"name": "Maria",

"phone": "7879890008",

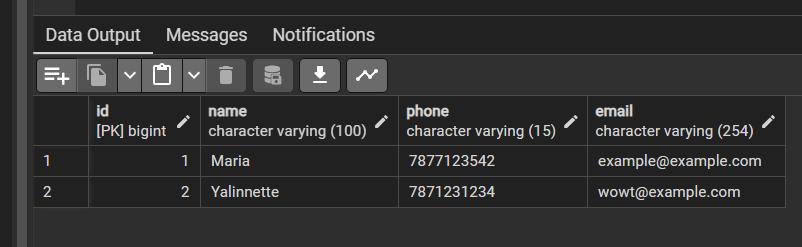
"email": "example@example.com"

}

delete http://localhost:8000/api/clients/2/

we delete that client

With the PUT we update information.



AWS

Before integrating AWS S3 with your Django project, you need to set up an S3 bucket:

* **Create an AWS Account**: Sign up or log in to your AWS account at [aws.amazon.com](https://aws.amazon.com/).
* **Create an S3 Bucket**: In the AWS Management Console, navigate to S3 and create a new bucket. Ensure the bucket name is unique across AWS, and decide on the region closest to your users for optimal performance.
* **Bucket Settings**: Disable ‘Block all public access’ settings if you want your files to be publicly accessible. Be cautious with this setting and only make public the files that need to be accessible to the end-users.
* **Get Your AWS Access Keys**: Navigate to your account’s security credentials page and create a new access key under the Access Keys (Access Key ID and Secret Access Key) section. Note these down securely.

To connect your Django project with AWS S3, you’ll use the django-storages package and configure your project’s settings:

* **Install django-storages and boto3**: Run pip install django-storages boto3 in your terminal. boto3 is the Amazon Web Services (AWS) SDK for Python, which allows Python developers to write software that uses services like Amazon S3.
* **Update settings.py**: Add storages to your INSTALLED\_APPS. Then, configure the following settings with your AWS credentials and bucket name:

AWS\_ACCESS\_KEY\_ID = 'asd1234567890wertyuiopasdfghjkl'

AWS\_SECRET\_ACCESS\_KEY = '11111111111111111111111111/222222222222222222'

AWS\_STORAGE\_BUCKET\_NAME = 'rafrodriguezbucket'

AWS\_S3\_FILE\_OVERWRITE = False

AWS\_DEFAULT\_ACL = 'public-read'

AWS\_S3\_REGION\_NAME = 'us-east-2'

AWS\_S3\_CUSTOM\_DOMAIN = f'{AWS\_STORAGE\_BUCKET\_NAME}.s3.amazonaws.com'

# Static files configuration

STATIC\_URL = f'https://{AWS\_S3\_CUSTOM\_DOMAIN}/static/'

# Media files configuration

DEFAULT\_FILE\_STORAGE = 'storages.backends.s3boto3.S3Boto3Storage'

MEDIA\_URL = f'https://{AWS\_S3\_CUSTOM\_DOMAIN}/media/'

To allow users to upload files, create a model to store file references, serialize it, and create a view:

**Model**: Define a model in models.py that includes a FileField or ImageField.  
class UploadedFile(models.Model):

file = models.FileField(upload\_to='uploads/')

uploaded\_at = models.DateTimeField(auto\_now\_add=True)

**Serializer**: Create a serializer in serializers.py for the model.  
class UploadedFileSerializer(serializers.ModelSerializer):

class Meta:

model = UploadedFile

fields = '\_\_all\_\_'

**View**: In views.py, implement a view to handle file uploads.   
class FileUploadView(APIView):

parser\_classes = [FileUploadParser]

permission\_classes = [IsAuthenticated]

def post(self, request, \*args, \*\*kwargs):

file\_obj = request.data['file']

# Handle the uploaded file

return Response({'status': 'File uploaded successfully'})

* This view uses MultiPartParser and FormParser to handle form data, including files.

**URLs**: Add a URL pattern in urls.py to route to the upload view.

use command

1. python manage.py makemigrations
2. python manage.py migrate

Install

1. pip install "uvicorn[standard]"
2. pip install wsproto
3. pip install channels

adding nested serializer and hyperlinkedmodelserializer

class ServiceSerializer(serializers.ModelSerializer):

class Meta:

model = Service

fields = '\_\_all\_\_'

extra\_kwargs = {

'url': {'view\_name': 'service-detail', 'lookup\_field': 'pk'}

}

class AppointmentSerializer(serializers.ModelSerializer):

class Meta:

model = Appointment

fields = '\_\_all\_\_'

extra\_kwargs = {

'url': {'view\_name': 'client-detail', 'lookup\_field': 'pk'}

}

class BookingSerializer(serializers.ModelSerializer):

Implementing Throttling in Django REST Framework (DRF)

REST\_FRAMEWORK = {

'DEFAULT\_AUTHENTICATION\_CLASSES': [

'rest\_framework.authentication.TokenAuthentication',

'rest\_framework.throttling.AnonRateThrottle',

'rest\_framework.throttling.UserRateThrottle'

],

'DEFAULT\_FILTER\_BACKENDS': [

'django\_filters.rest\_framework.DjangoFilterBackend'

],

'DEFAULT\_THROTTLE\_RATES': {

'anon': '100/day',

'user': '1000/day'

}

}

from rest\_framework.throttling import AnonRateThrottle, UserRateThrottle

class ExampleView(APIView):

throttle\_classes = [AnonRateThrottle, UserRateThrottle]

def get(self, request, format=None):

# Ejemplo de uso de los serializers

clients = Client.objects.all()

client\_serializer = ClientSerializer(clients, many=True)

services = Service.objects.all()

service\_serializer = ServiceSerializer(services, many=True)

appointments = Appointment.objects.all()

appointment\_serializer = AppointmentSerializer(appointments, many=True)

response\_data = {

'clients': client\_serializer.data,

'services': service\_serializer.data,

'appointments': appointment\_serializer.data,

'message': 'Hello, world!'

}

return Response(response\_data)

'DEFAULT\_FILTER\_BACKENDS': [

'django\_filters.rest\_framework.DjangoFilterBackend'

],

### Filtered out:

* from django\_filters.rest\_framework import DjangoFilterBackend from rest\_framework import filters
* class ClientViewSet(ModelViewSet): queryset = Client.objects.all() serializer\_class = ClientSerializer filter\_backends = [DjangoFilterBackend] filterset\_fields = ['name', 'email']

### Search:

* class ClientViewSet(ModelViewSet): ... filter\_backends = [filters.SearchFilter] search\_fields = ['name', 'email']

### Ordination:

* class ClientViewSet(ModelViewSet): ... filter\_backends = [filters

Create permissions.py

from rest\_framework.permissions import BasePermission

class CanViewClients(BasePermission):

def has\_permission(self, request, view):

# Verifica si el usuario puede ver todos los clientes

return request.user.has\_perm('yourapp.view\_client')

class CanAddService(BasePermission):

def has\_permission(self, request, view):

# Verifica si el usuario puede agregar un nuevo servicio

return request.user.has\_perm('yourapp.add\_service')

View.py

class ServiceCreateView(APIView):

permission\_classes = [CanAddService]

def post(self, request):

if not request.user.has\_perm('yourapp.add\_service'):

return Response({"message": "No tienes permiso para agregar servicios"}, status=status.HTTP\_403\_FORBIDDEN)

return Response({"message": "Servicio creado exitosamente"}, status=status.HTTP\_201\_CREATED)

throttling and permissions in your views, you must add the code that defines the MyThrottle class and the MyPermission class to your permissions and throttling files, respectively. Then, in your views where you want to apply these constraints, you import these classes and assign them to the corresponding views.

First, let's define the permissions and throttling classes in their respective files:

1. **Throttling (throttling.py)**:

python

from rest\_framework.throttling import UserRateThrottle

class MyThrottle(UserRateThrottle):

rate = '5/minute'

1. **Permisos (permissions.py)**:

python

from rest\_framework.permissions import IsAuthenticated

class MyPermission(IsAuthenticated):

pass

View

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from .throttling import MyThrottle

from .permissions import MyPermission

class MyProtectedView(APIView):

throttle\_classes = [MyThrottle] # Aplica throttling a esta vista

permission\_classes = [MyPermission] # Requiere autenticación para acceder

def get(self, request):

# Lógica de la vista

return Response({"message": "Bienvenido a la vista protegida"}, status=status.HTTP\_200\_OK)

To use pagination in your views, simply follow these steps:

Define the pagination class: First, define a pagination class that inherits from PageNumberPagination in your views file or in a separate file.

# pagination.py

from rest\_framework.pagination import PageNumberPagination

class MyPagination(PageNumberPagination):

page\_size = 10 # Number of objects per page

page\_size\_query\_param = 'page\_size' # Parameter to specify the page size

max\_page\_size = 1000 # Maximum page size allowed

Assign the pagination class to your views: In your views where you want to apply pagination, assign the pagination class that you have defined.

# views.py

from rest\_framework.views import APIView

from rest\_framework.response import Response

from rest\_framework import status

from .pagination import MyPagination

class MyListView(APIView):

pagination\_class = MyPagination

def get(self, request):

# Get the queryset of the objects you want to page

queryset = YourModel.objects.all()

# Paginate the results

paginator = self.pagination\_class()

paginated\_queryset = paginator.paginate\_queryset(queryset, request)

# Serialize paged objects

serializer = YourSerializer(paginated\_queryset, many=True)

# Return the paginated response

return paginator.get\_paginated\_response(serializer.data)

Configure the pagination response: The pagination class will take care of splitting the results into pages and providing metadata in the response. You just need to serialize the paginated objects and return the paginated response using paginator.get\_paginated\_response(serializer.data)