



EP2. Elaboración de API

Ingeniería Tecnologías de la Información

8°A

INTEGRANTES:

- Mendieta Chimal Sony Luis	MCS0220598
- Ascencio Onofre Carlos Gerardo	AOC0220155
- Nava Sanchez Axel	NSA0220388

Docente: LORENZO ANTONIO CARDOSO CONTRERAS

Materia: Tecnologías y aplicaciones en Internet

12/03/2025

Índice

Índice	1
Planteamiento del caso práctico.	2
Introducción.	3
Diseño de la base de datos.	4
Figura 1.1 Diagrama entidad-relación.	4
Figura 1.2 Código de la creación de los modelos.	5
Figura 1.3 Comando makemigrations.	5
Figura 1.4 Comando migrate.	6
Figura 1.5 Tablas de la base de datos.	6
Figura 1.6 Tabla producto.	7
Figura 1.7 Tabla proveedor.	7
Figura 1.8 Tabla usuario.	8
Prueba de la creación de la API.	9
Figura 1.9 Instalación de djangorestframework.	9
Figura 2.0 Instalación de djangorestframework-simplejwt.	9
Figura 2.1 Configuración de REST_FRAMEWORK.	10
Figura 2.2 Configuración de aplicaciones.	11
Figura 2.3 Serializador	12
Figura 2.4 Definir Serializador	13
Figura 2.5 Rutas de la API	14
Figura 2.6 Creacion Superusuario.	14
Figura 2.7 Iniciar servidor.	15
Figura 2.8 Petición de token.	15
Figura 2.9 Petición sin Token.	16
Figura 3.0 Petición GET con token	16
Figura 3.1 Petición POST con Token	17
Figura 3.2 Petición GET con Token	17
Figura 3.3 Petición GET con Token	18
Figura 3.4 Petición DELETE con Token	18
Figura 3.5 Petición PUT con Token	19
Conclusiones.	20
Bibliografía.	21

Planteamiento del caso práctico.

Descripción del Caso Práctico

El presente proyecto tiene como objetivo el desarrollo de un sistema de gestión para una perfumería, permitiendo la administración eficiente de productos, proveedores y clientes. A través de una API desarrollada con Django, se busca facilitar el manejo de inventario, la actualización de precios y la gestión de pedidos de manera estructurada y automatizada.

Situación Problemática

Actualmente, la perfumería enfrenta diversas dificultades en la administración de su inventario y relaciones con proveedores. Algunas de las problemáticas identificadas incluyen:

- Registro manual y desorganizado de productos y stock, lo que dificulta el control y actualización de inventario.
- Falta de un sistema unificado para gestionar la información de proveedores, dificultando la comunicación y abastecimiento de productos.
- Limitaciones en la seguridad de los datos de clientes y proveedores, lo que puede generar riesgos en la protección de información.
- Falta de automatización en procesos clave como la actualización de precios y el seguimiento de pedidos, lo que impacta en la eficiencia operativa.

Situación Ideal

Con el desarrollo de este sistema, se busca alcanzar las siguientes mejoras:

- Implementación de una API estructurada que permita la gestión eficiente de usuarios, productos y proveedores.
- Automatización del control de stock, facilitando el monitoreo en tiempo real de la disponibilidad de productos.
- Integración de mecanismos de autenticación y seguridad para garantizar la protección de los datos.
- Optimización de la relación con proveedores mediante un sistema centralizado que agilice el registro y seguimiento de pedidos.

Introducción.

El presente documento detalla el desarrollo de un sistema de gestión para una perfumería, el cual busca optimizar la administración de productos, proveedores y clientes a través de una API desarrollada con Django. En este reporte se documenta el proceso de diseño de la base de datos, la implementación de la API con operaciones CRUD (Create, Read, Update, Delete), así como la autenticación mediante tokens para garantizar la seguridad de los datos.

A lo largo del informe, se presentan las distintas secciones que conforman el desarrollo del proyecto. Primero, se expone el planteamiento del caso práctico, donde se describe la problemática actual y la situación ideal a la que se desea llegar con la implementación del sistema. Posteriormente, se detalla el diseño de la base de datos, incluyendo los modelos creados y su relación. Luego, se documenta la creación de la API con evidencia de pruebas realizadas en Insomnia, cubriendo las operaciones esenciales.

Finalmente, se presentan las conclusiones obtenidas tras la realización del proyecto, destacando los aprendizajes adquiridos, los retos enfrentados y las soluciones implementadas. Asimismo, se incluye la bibliografía utilizada, siguiendo el formato APA, para respaldar los conceptos y tecnologías empleadas en el desarrollo del sistema.

Diseño de la base de datos.

Se muestra nuestro diagrama entidad-relación (figura 1.1) de nuestra base de datos que utilizaremos para nuestro sistema.

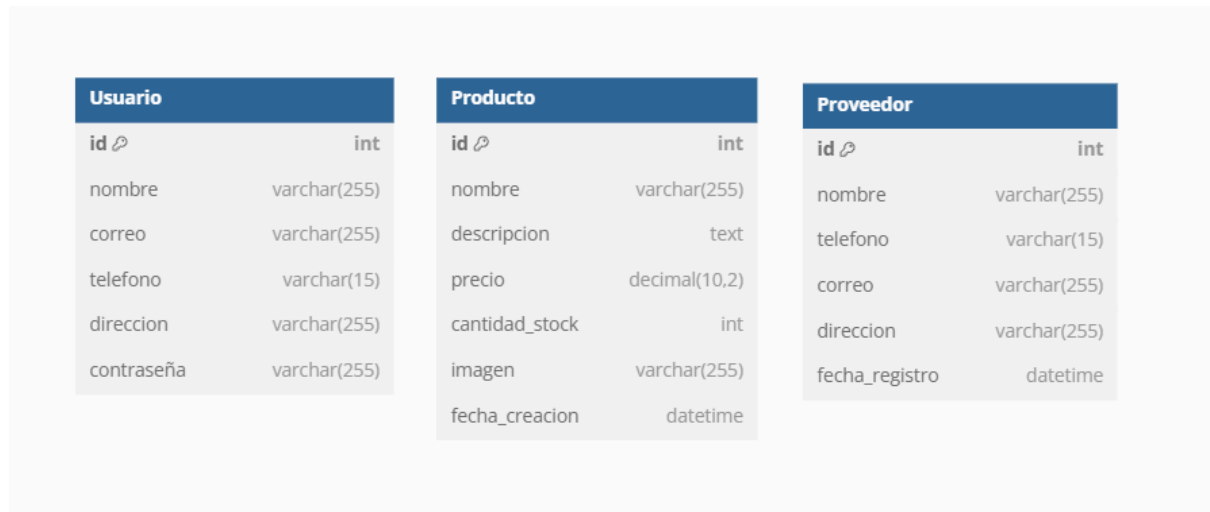


Figura 1.1 Diagrama entidad-relación.

Para el desarrollo del sistema de gestión de perfumería, se diseñó una base de datos relacional con tres entidades principales: **Usuario**, **Producto** y **Proveedor**. Cada una de estas entidades está representada mediante modelos en Django, los cuales definen la estructura y comportamiento de los datos dentro del sistema como se muestra en la figura 1.2.

```
from django.db import models

# Create your models here.

class Usuario(models.Model):
    nombre = models.CharField(max_length=255)
    correo = models.EmailField(unique=True)
    telefono = models.CharField(max_length=15, null=True, blank=True)
    direccion = models.CharField(max_length=255, null=True, blank=True)
    contraseña = models.CharField(max_length=255)

class Producto(models.Model):
    nombre = models.CharField(max_length=255)
    descripcion = models.TextField()
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    cantidad_stock = models.PositiveIntegerField()
    imagen = models.CharField(max_length=255, null=True, blank=True)
    fecha_creacion = models.DateTimeField(auto_now_add=True)

class Proveedor(models.Model):
    nombre = models.CharField(max_length=255)
    telefono = models.CharField(max_length=15)
    correo = models.EmailField()
    direccion = models.CharField(max_length=255, null=True, blank=True)
    fecha_registro = models.DateTimeField(auto_now_add=True)
```

Figura 1.2 Código de la creación de los modelos.

Una vez definidos los modelos en Django, se debe crear una migración para reflejar estos cambios en la base de datos. Para hacerlo, se ejecuta el siguiente comando (figura 1.3).

```
PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria> python manage.py makemigrations
Migrations for 'appPerfumeria':
  appPerfumeria\migrations\0003_initial.py
    + Create model Producto
    + Create model Proveedor
    + Create model Usuario
```

Figura 1.3 Comando makemigrations.

Luego de crear la migración, es necesario aplicarla a la base de datos para que los cambios sean reflejados. Para ello, se utiliza el siguiente comando (figura 1.4).

```
PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria> python manage.py migrate
Operations to perform:
  Apply all migrations: admin, appPerfumeria, auth, contenttypes, sessions
Running migrations:
  Applying appPerfumeria.0002_delete_producto_delete_proveedor_delete_usuario... OK
  Applying appPerfumeria.0003_initial... OK
```

Figura 1.4 Comando migrate.

Se demuestra la correcta creación de la base de datos así como la creación de las tablas dentro de la base de datos (figura 1.5).

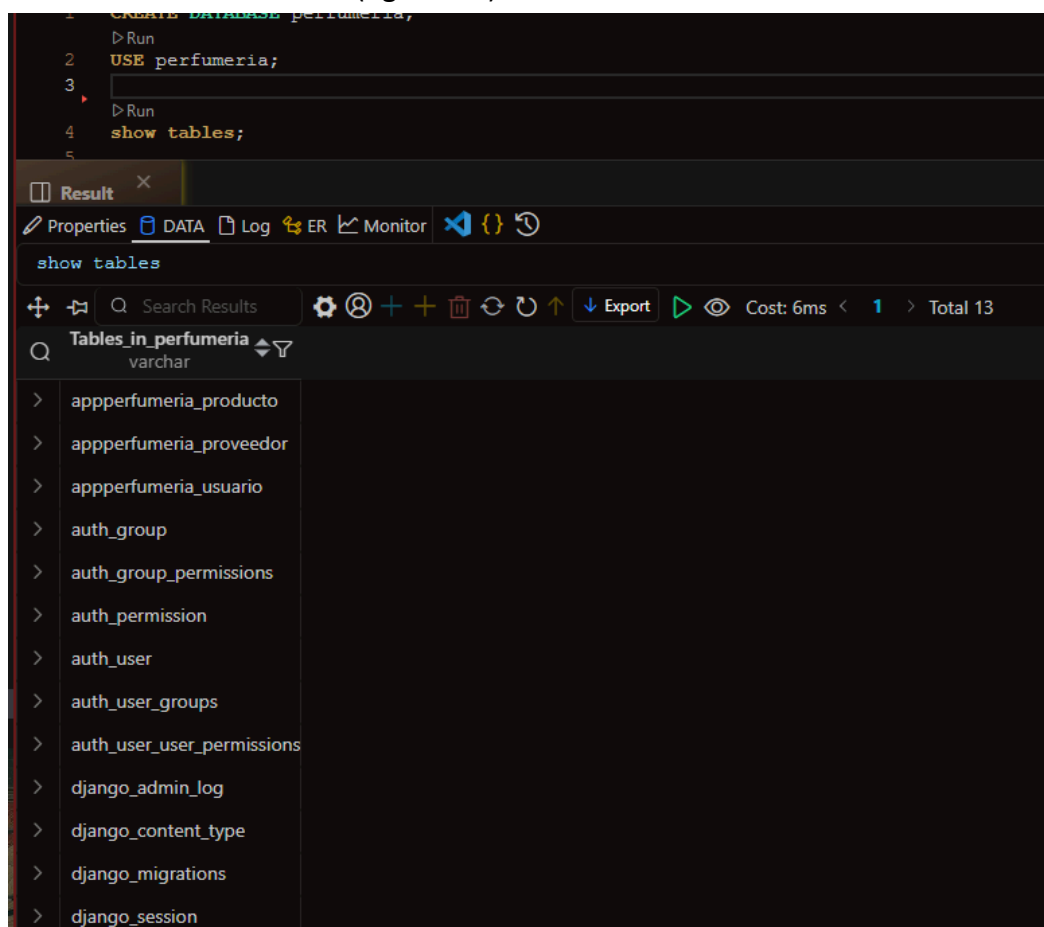


Figura 1.5 Tablas de la base de datos.

Creación del modelo de producto dentro de la base de datos (figura 1.6).

Result

Properties DATA Log ER Monitor

describe appperfumeria_producto

Search Results

Cost: 33ms < 1 > Total 7

Field	Type	Null	Key	Default	Extra
varchar	blob	varchar	varchar	blob	varchar
id	bigint(20)	NO	PRI	(NULL)	auto_increment
nombre	varchar(255)	NO		(NULL)	
descripcion	longtext	NO		(NULL)	
precio	decimal(10,2)	NO		(NULL)	
cantidad_stock	int(10) unsigned	NO		(NULL)	
imagen	varchar(255)	YES		(NULL)	
fecha_creacion	datetime(6)	NO		(NULL)	

Figura 1.6 Tabla producto.

Creación del modelo de proveedor dentro de la base de datos (figura 1.7).

Result

Properties DATA Log ER Monitor

describe appperfumeria_proveedor

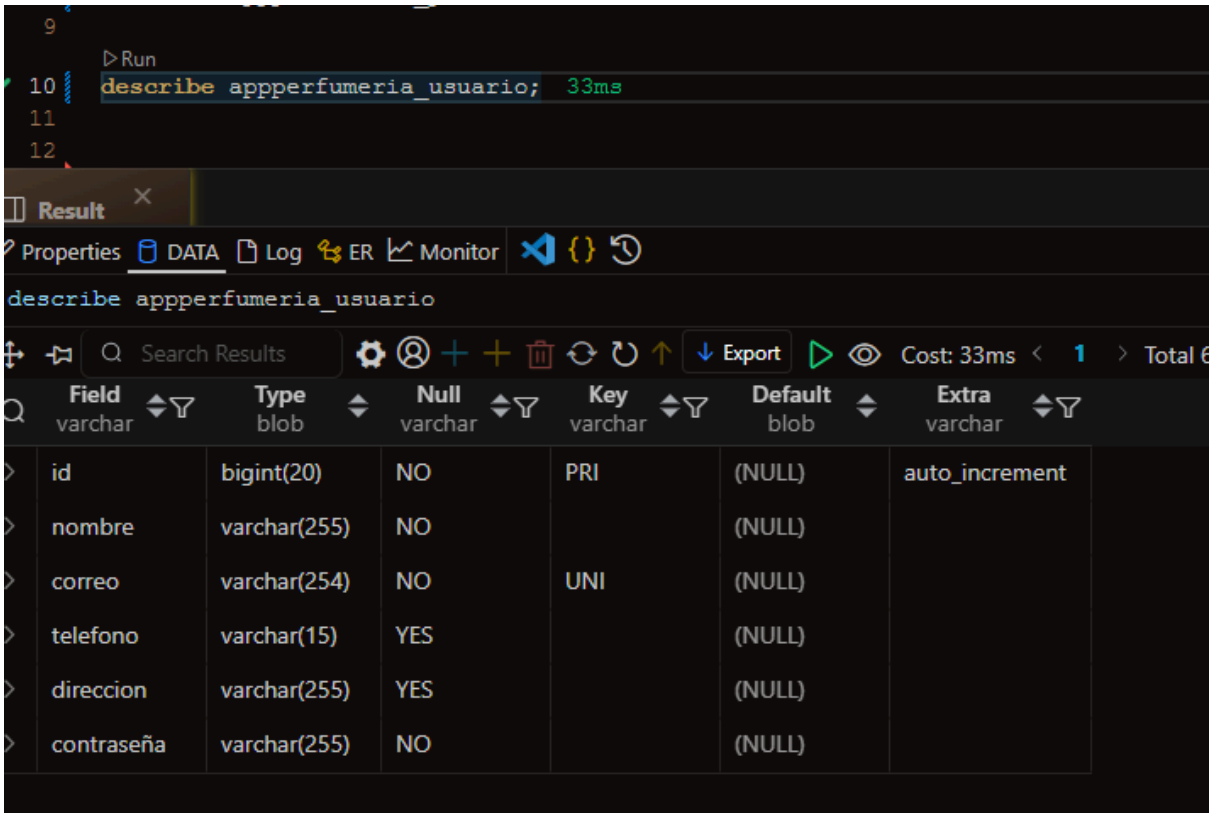
Search Results

Cost: 36ms < 1 > Total 6

Field	Type	Null	Key	Default	Extra
varchar	blob	varchar	varchar	blob	varchar
id	bigint(20)	NO	PRI	(NULL)	auto_increment
nombre	varchar(255)	NO		(NULL)	
telefono	varchar(15)	NO		(NULL)	
correo	varchar(254)	NO		(NULL)	
direccion	varchar(255)	YES		(NULL)	
fecha_registro	datetime(6)	NO		(NULL)	

Figura 1.7 Tabla proveedor.

Creación del modelo de usuario dentro de la base de datos (figura 1.8).



The screenshot shows a database IDE interface. At the top, a SQL editor displays the command `describe appperfumeria_usuario;` with a green execution time of 33ms. Below the editor, a 'Result' tab is active, showing the table's structure. The table has 6 columns: `id`, `nombre`, `correo`, `telefono`, `direccion`, and `contraseña`. The `id` column is the primary key (PRI) and is auto-incrementing. The `correo` column is the unique key (UNI). All columns are of type `varchar` except for `id` which is `bigint(20)`. All columns allow null values.

Field	Type	Null	Key	Default	Extra
id	bigint(20)	NO	PRI	(NULL)	auto_increment
nombre	varchar(255)	NO		(NULL)	
correo	varchar(254)	NO	UNI	(NULL)	
telefono	varchar(15)	YES		(NULL)	
direccion	varchar(255)	YES		(NULL)	
contraseña	varchar(255)	NO		(NULL)	

Figura 1.8 Tabla usuario.

Prueba de la creación de la API.

Para poder crear una api dentro de framework se necesita instalar djangoestframework con el siguiente comando figura 1.9.

```
(venv) PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria>
(venv) PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria> pip install djangoestframework
Collecting djangoestframework
  Downloading djangoestframework-3.15.2-py3-none-any.whl (1.1 MB)
    1.1/1.1 MB 98.4 kB/s eta 0:00:00
Requirement already satisfied: django>=4.2 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from djangoestframework) (5.1.7)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from django>=4.2->djangoestframework) (0.5.3)
Requirement already satisfied: asgiref<4,>=3.8.1 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from django>=4.2->djangoestframework) (3.8.1)
Requirement already satisfied: tzdata in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from django>=4.2->djangoestframework) (2025.1)
Requirement already satisfied: typing-extensions>=4 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from asgiref<4,>=3.8.1->django>=4.2->djangoestframework) (4.12.2)
Installing collected packages: djangoestframework
Successfully installed djangoestframework-3.15.2
```

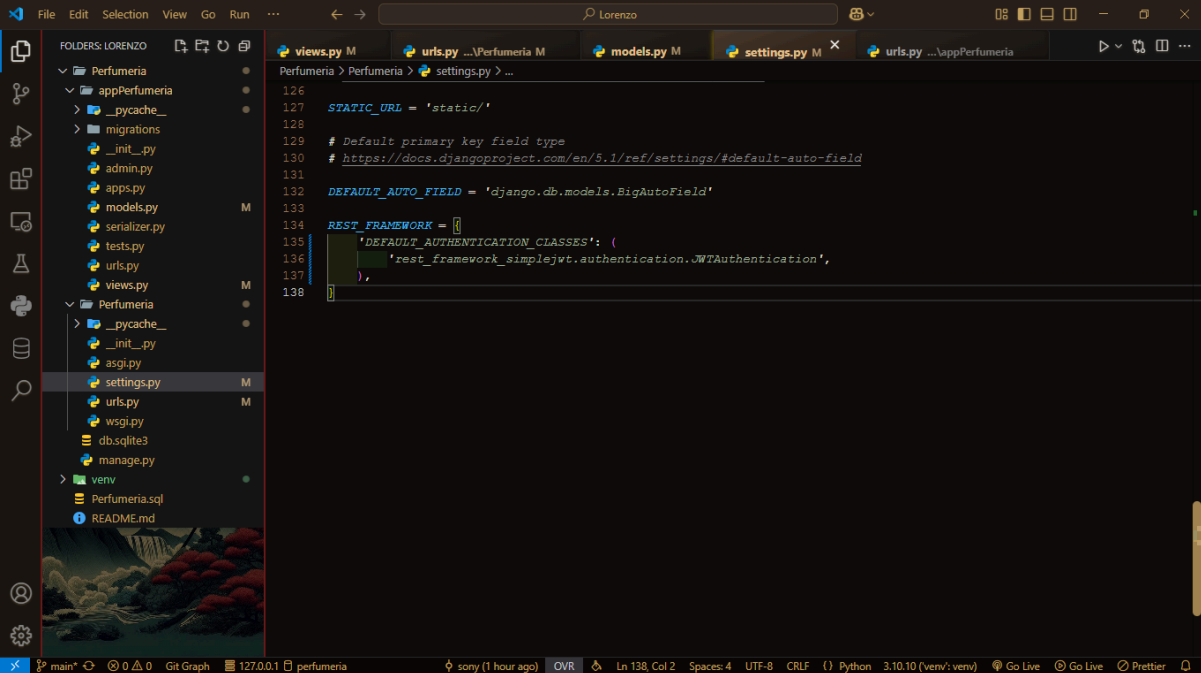
Figura 1.9 Instalación de djangoestframework.

Este comando (figura 2.0), pip install djangoestframework-simplejwt, se usa para instalar un paquete adicional en el entorno de Python que permite integrar el sistema de autenticación basado en JSON Web Tokens (JWT) en tu aplicación Django con Django REST Framework (DRF).

```
PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria> pip install djangoestframework-simplejwt
Collecting djangoestframework-simplejwt
  Downloading djangoestframework-simplejwt-5.5.0-py3-none-any.whl.metadata (4.6 kB)
Requirement already satisfied: django>=4.2 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from djangoestframework-simplejwt) (5.1.7)
Requirement already satisfied: djangoestframework>=3.14 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from djangoestframework-simplejwt) (3.8.1)
Collecting pyjwt<2.10.0,>=1.7.1 (from djangoestframework-simplejwt)
  Downloading PyJWT-2.9.0-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: asgiref<4,>=3.8.1 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from django>=4.2->djangoestframework-simplejwt) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from djangoestframework-simplejwt) (0.5.3)
Requirement already satisfied: tzdata in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from django>=4.2->djangoestframework-simplejwt) (2025.1)
Requirement already satisfied: typing-extensions>=4 in c:\users\sonyc\desktop\lorenzo\lorenzo\venv\lib\site-packages (from
```

Figura 2.0 Instalación de djangoestframework-simplejwt.

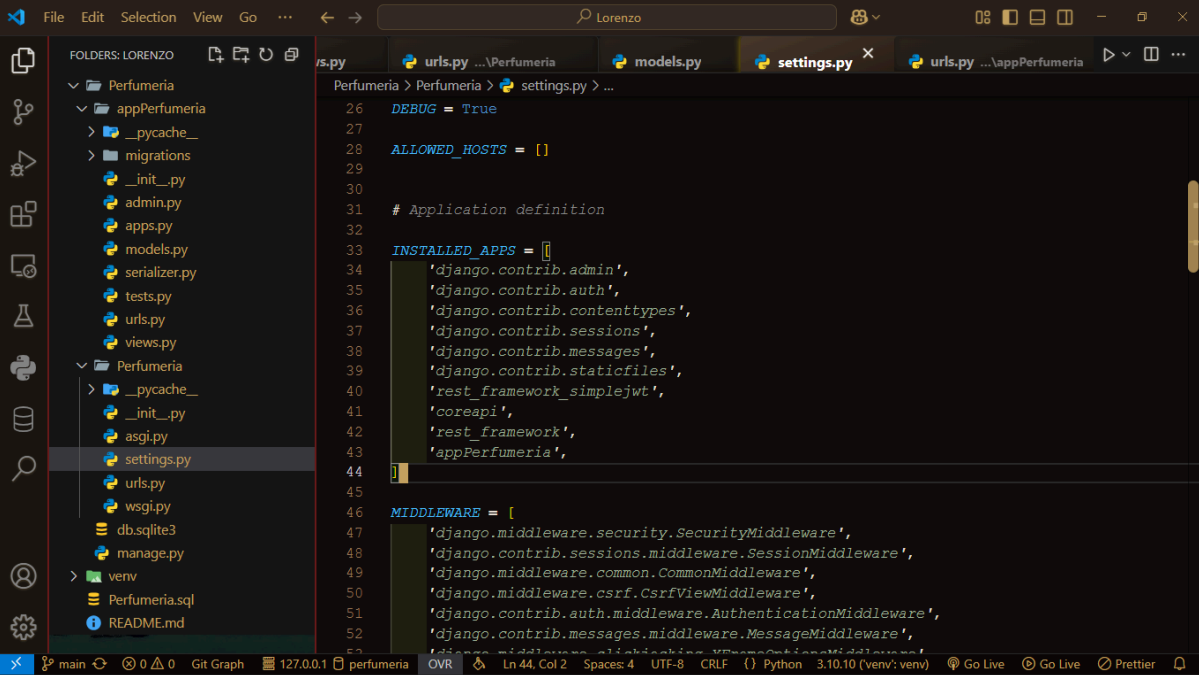
En el archivo de configuración settings.py de Django, se debe establecer el sistema de autenticación para la API. Para habilitar la autenticación basada en JSON Web Tokens (JWT), se debe configurar el parámetro DEFAULT_AUTHENTICATION_CLASSES en la sección REST_FRAMEWORK. Esto asegura que todas las solicitudes a la API estén protegidas por el esquema de autenticación JWT, que permite validar las solicitudes mediante tokens seguros en lugar de credenciales tradicionales (usuario y contraseña) así como se muestra en la figura 2.1



```
126 STATIC_URL = 'static/'
127
128 # Default primary key field type
129 # https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field
130
131 DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
132
133 REST_FRAMEWORK = {
134     'DEFAULT_AUTHENTICATION_CLASSES': (
135         'rest_framework_simplejwt.authentication.JWTAuthentication',
136     ),
137 },
138 }
```

Figura 2.1 Configuración de REST_FRAMEWORK.

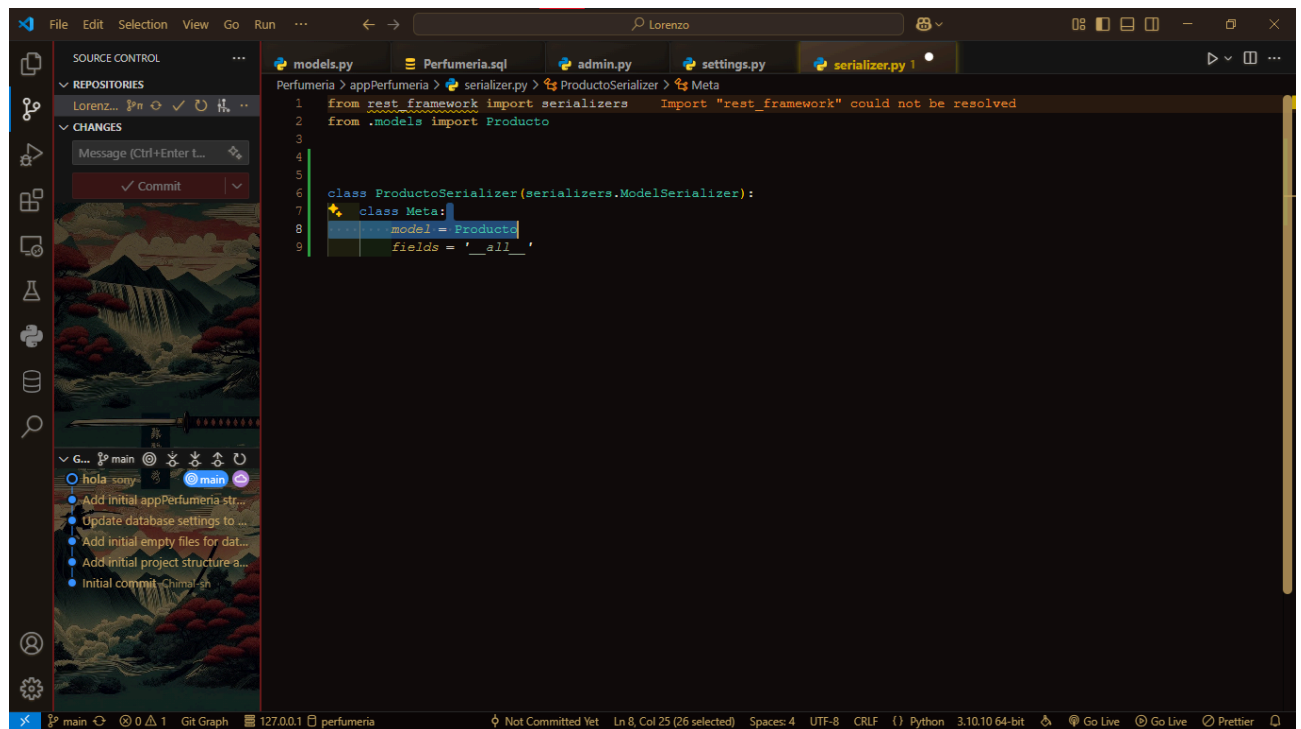
El siguiente fragmento de código (figura 2.2) muestra cómo se configuran las aplicaciones necesarias para que el proyecto funcione correctamente, incluyendo las aplicaciones esenciales de Django y las bibliotecas necesarias para la implementación de la API RESTful y la autenticación JWT.



```
26 DEBUG = True
27
28 ALLOWED_HOSTS = []
29
30
31 # Application definition
32
33 INSTALLED_APPS = [
34     'django.contrib.admin',
35     'django.contrib.auth',
36     'django.contrib.contenttypes',
37     'django.contrib.sessions',
38     'django.contrib.messages',
39     'django.contrib.staticfiles',
40     'rest_framework_simplejwt',
41     'coreapi',
42     'rest_framework',
43     'appPerfumeria',
44 ]
45
46 MIDDLEWARE = [
47     'django.middleware.security.SecurityMiddleware',
48     'django.contrib.sessions.middleware.SessionMiddleware',
49     'django.middleware.common.CommonMiddleware',
50     'django.middleware.csrf.CsrfViewMiddleware',
51     'django.contrib.auth.middleware.AuthenticationMiddleware',
52     'django.contrib.messages.middleware.MessageMiddleware',
53 ]
```

Figura 2.2 Configuración de aplicaciones.

El siguiente fragmento de código (figura 2.0) define un serializador para el modelo Producto, el cual se encargará de estructurar y validar los datos que la API enviará y recibirá:

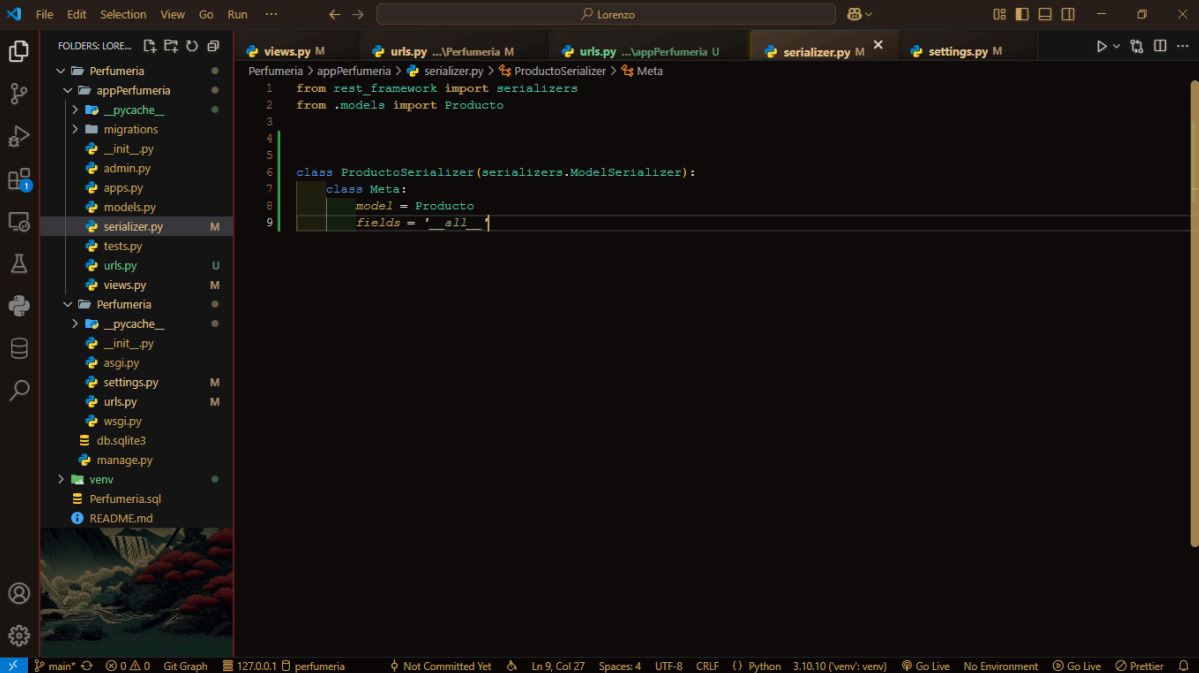


```
1 from rest_framework import serializers
2 from .models import Producto
3
4
5
6 class ProductoSerializer(serializers.ModelSerializer):
7     class Meta:
8         model = Producto
9         fields = '_all'
```

Figura 2.3 Serializador

En el proyecto de perfumería, se define una vista para gestionar los productos de la base de datos mediante el uso de viewsets de Django REST Framework. Esta vista permite realizar operaciones CRUD sobre el modelo Producto, y está configurada para requerir que el usuario esté autenticado antes de realizar cualquier operación.

El siguiente fragmento de código figura 2.4 muestra la implementación de la vista para el modelo Producto



```
1 from rest_framework import serializers
2 from .models import Producto
3
4
5
6 class ProductoSerializer(serializers.ModelSerializer):
7     class Meta:
8         model = Producto
9         fields = '__all__'
```

The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure for 'Perfumeria' with various files and folders. The code editor shows the implementation of a Django REST Framework serializer for the 'Producto' model. The code is as follows:

Figura 2.4 Definir Serializador

En el archivo `urls.py` del proyecto, se definen las rutas URL que manejarán las peticiones a diferentes vistas del sistema, como el panel de administración de Django, la API de productos de la perfumería y las rutas relacionadas con la autenticación utilizando JWT. En la figura 2.5 se muestran las configuraciones de las rutas.

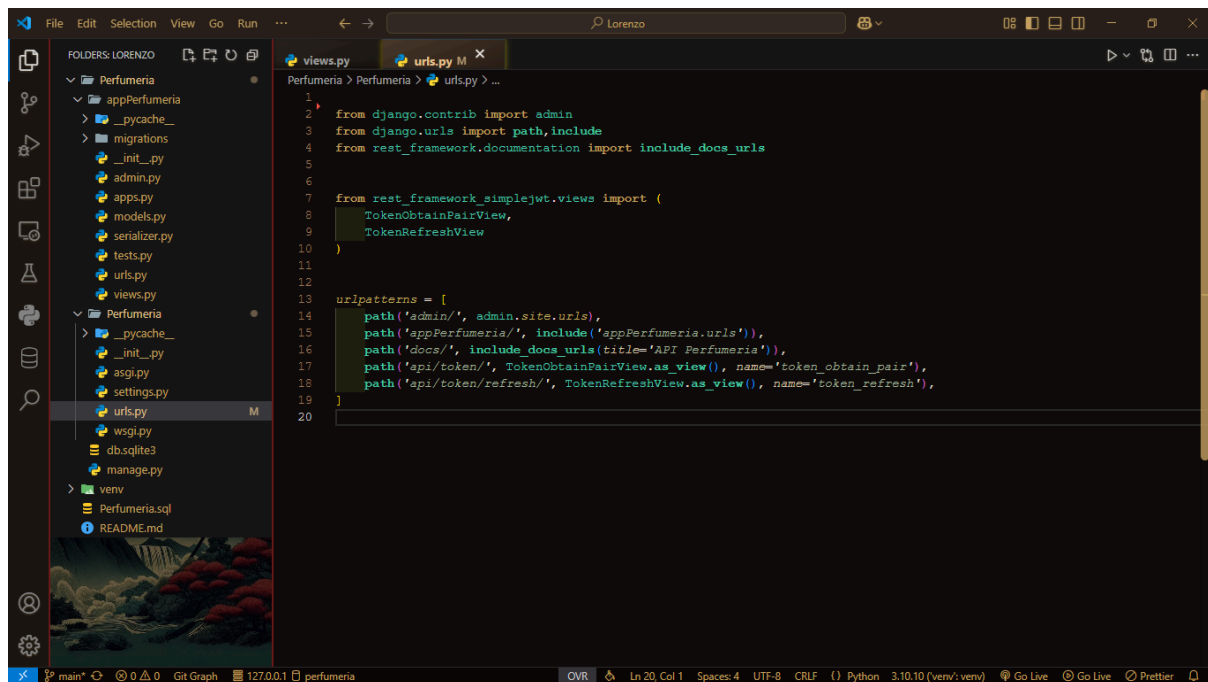


Figura 2.5 Rutas de la API

Para crear un superusuario en Django, se utiliza el comando `python manage.py createsuperuser`, que permite establecer un usuario con privilegios de administración para acceder al panel de administración de Django y para poder recibir el token como se ve en la figura 2.6.

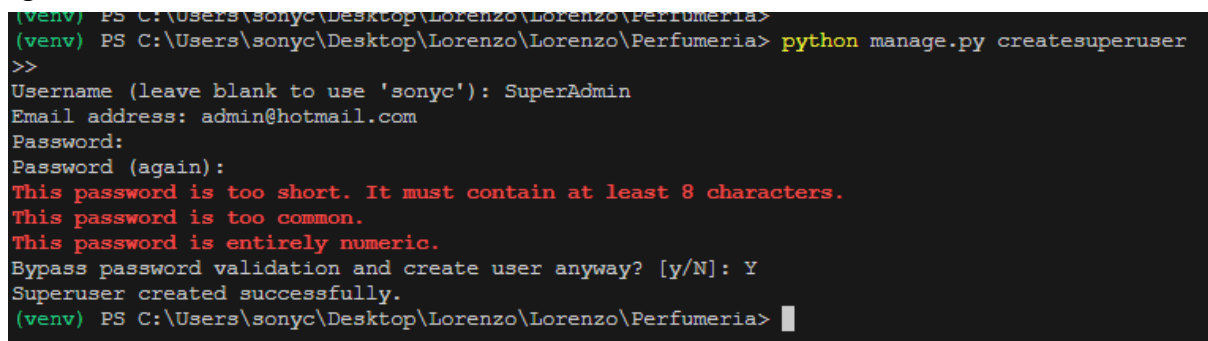


Figura 2.6 Creacion Superusuario.

Iniciamos el servidor para realizar las pruebas con la api (figura 2.7).

```
PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo> cd Perfumeria
PS C:\Users\sonyc\Desktop\Lorenzo\Lorenzo\Perfumeria> python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
March 12, 2025 - 09:40:19
Django version 5.1.7, using settings 'Perfumeria.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Figura 2.7 Iniciar servidor.

Realizamos la petición del token al servidor en insomnia como se muestra en la figura 2.8

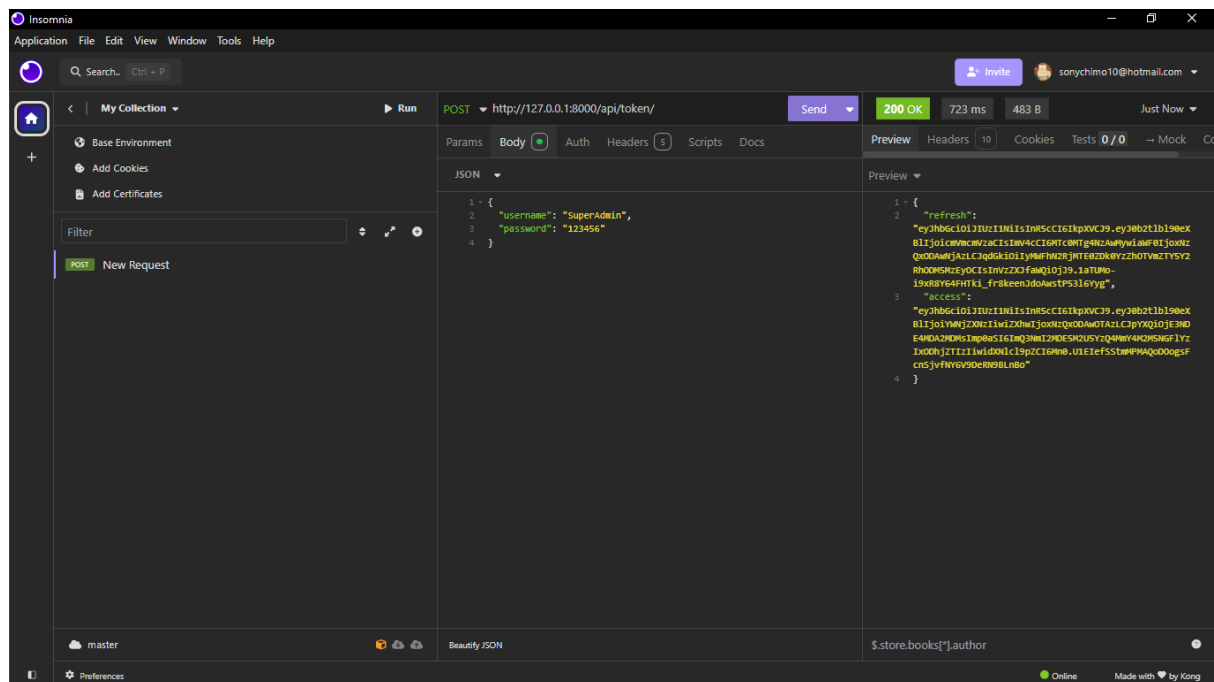
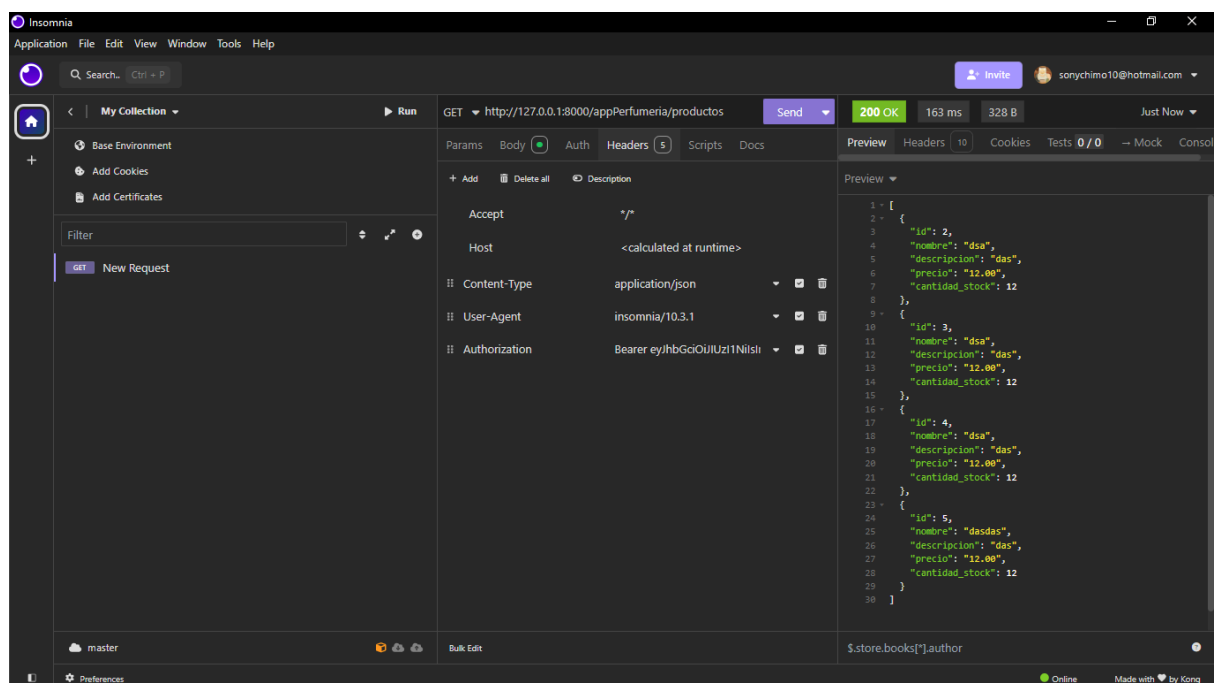


Figura 2.8 Petición de token.

The screenshot displays the Insomnia application window. The top menu bar includes Application, File, Edit, View, Window, Tools, and Help. Below the menu is a search bar and user information (Invite, sonychimo10@hotmail.com). The main workspace is divided into three vertical panes. The left pane shows a sidebar with 'My Collection' selected, containing options like Base Environment, Add Cookies, and Add Certificates. A filter input and a '+ POST New Request' button are also visible. The middle pane shows a REST client request configuration for a POST method to the URL 'http://127.0.0.1:8000/appPerfumeria/productos/'. It features tabs for Params, Body, Auth, Headers (selected), Scripts, and Docs. Under the Headers tab, there are entries for Accept (*/*), Host (<calculated at runtime>), Content-Type (application/json), User-Agent (insomnia/10.3.1), and Authorization (Bearer eyJhbGogOiJIUzI1NiIs... Bulk Edit is shown at the bottom. The right pane displays the response status '401 Unauthorized' with a duration of 125 ms and size of 58 B. It includes tabs for Preview, Headers, Cookies, Tests, Mock, and Console. The Preview tab shows a JSON response: { "detail": "Authentication credentials were not provided." }. The bottom status bar indicates the master branch, bulk edit mode, and the API endpoint \$store.books[*].author.

Realizamos una petición get ahora con el token en Insomnia puesto en el header para ver los productos registrados como se muestra en la figura 3.0.



16

Realizamos una petición POST para ingresar un nuevo producto como se muestra en la figura 3.1.

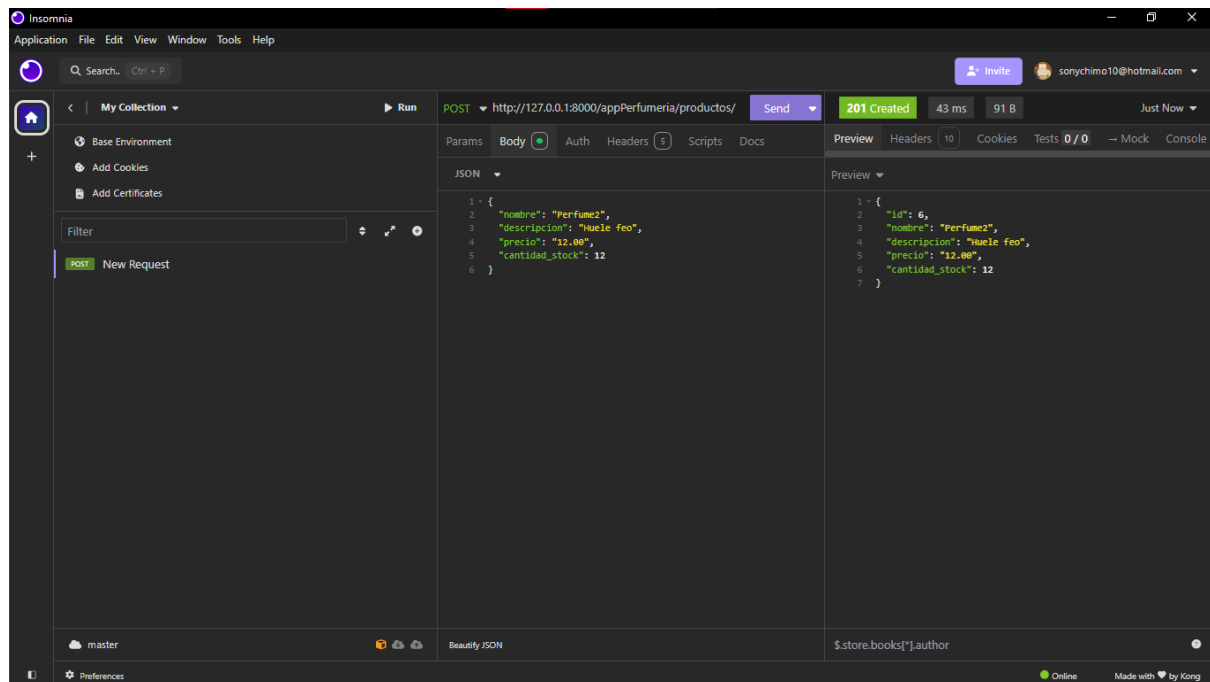


Figura 3.1 Petición POST con Token

Realizamos una petición GET para ver los productos ingresados como se muestra en la figura 3.2.

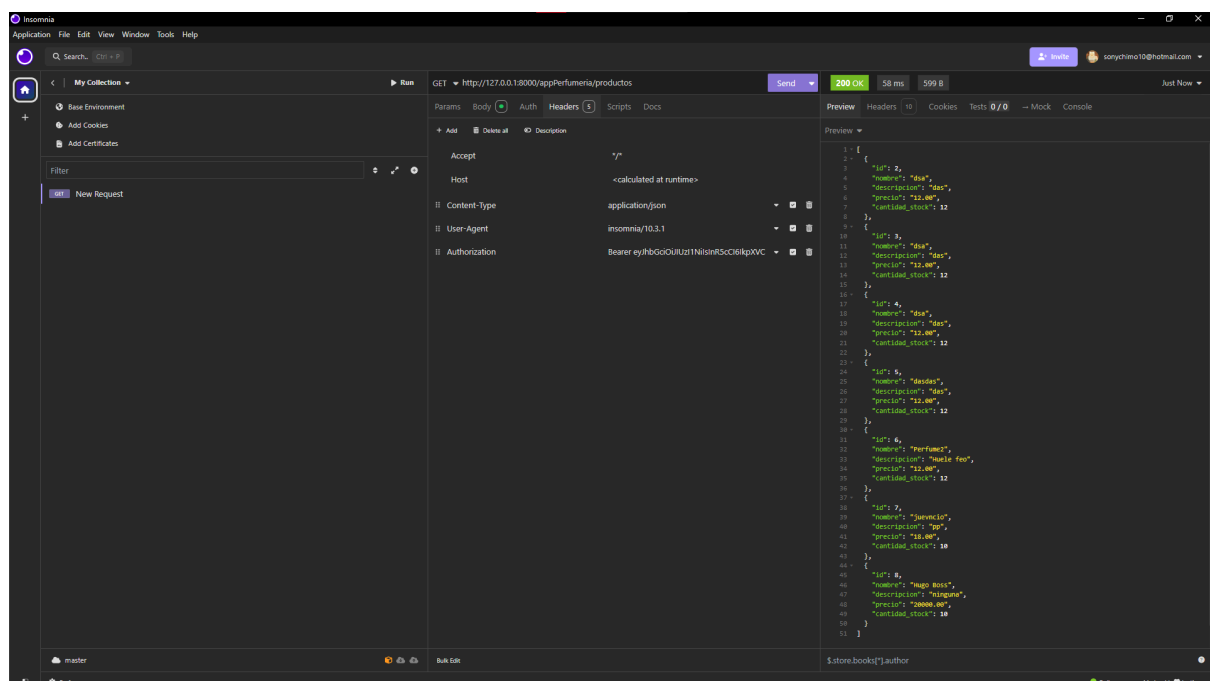


Figura 3.2 Petición GET con Token

Realizamos una petición DELETE para borrar el producto 3 como se muestra en las figuras 3.3 y 3.4.

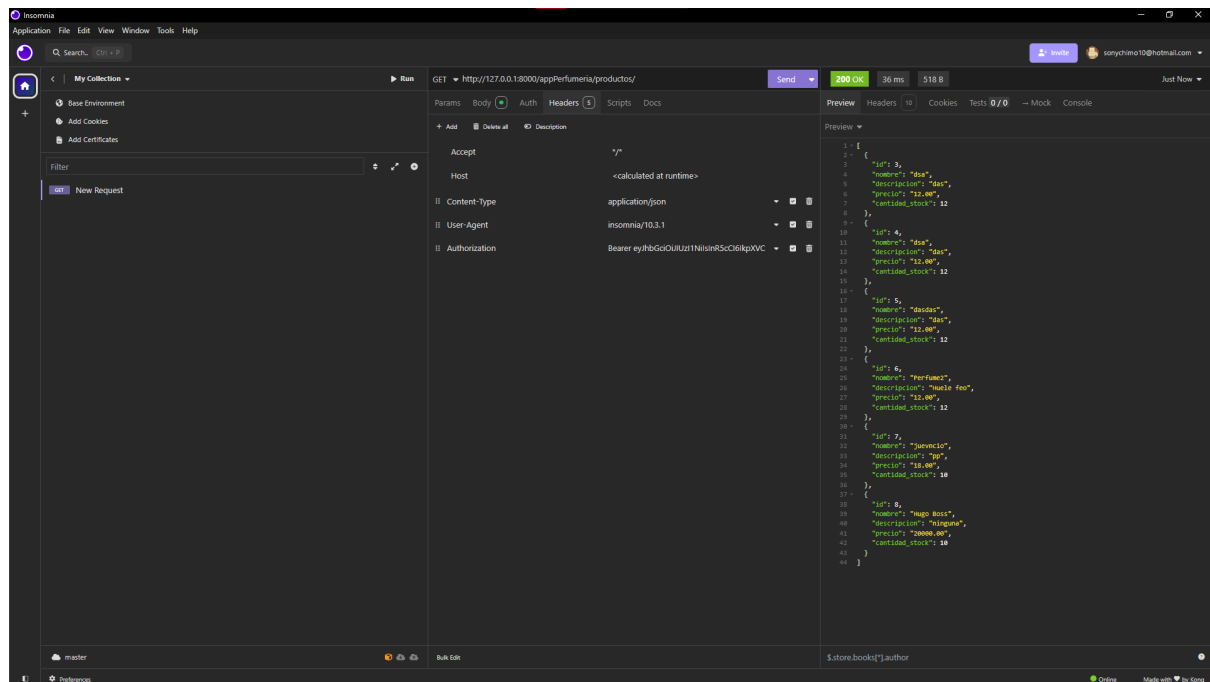


Figura 3.3 Petición GET con Token

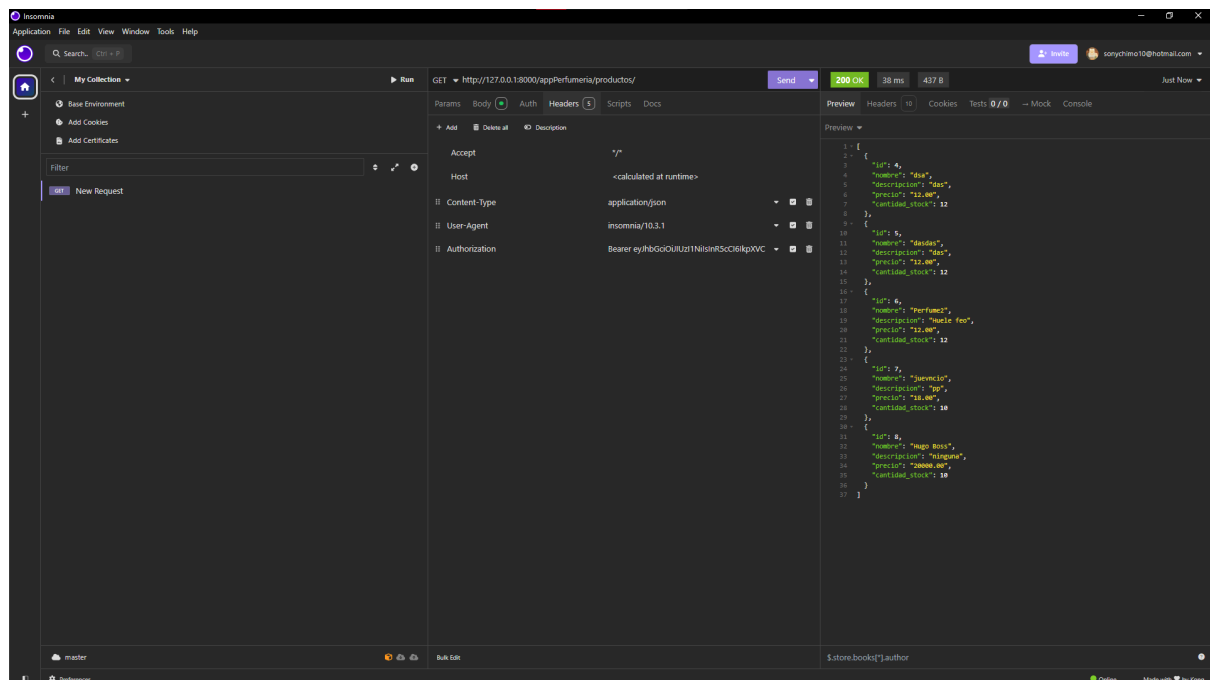


Figura 3.4 Petición **DELETE** con Token

Realizamos una petición PUT para actualizar el producto 4 como se muestra en las figuras 3.5.

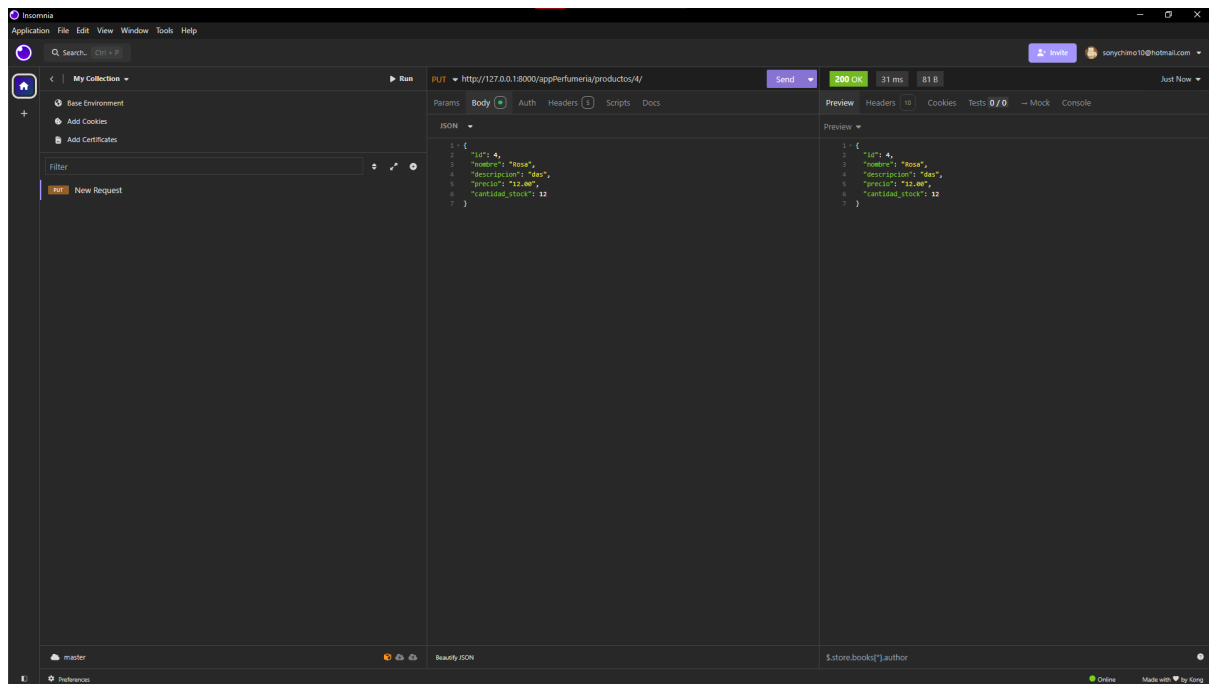


Figura 3.5 Petición PUT con Token

Conclusiones.

Al finalizar la práctica, se logró desarrollar una API funcional para la gestión de productos en una perfumería utilizando Django Rest Framework (DRF) y JWT para la autenticación de usuarios. A lo largo del proceso, se adquirieron conocimientos clave sobre la creación de endpoints, la seguridad en las API y la validación de datos.

Uno de los principales retos fue la configuración de JWT, ya que al inicio hubo dificultades para gestionar los tokens de autenticación. También fue necesario familiarizarse con Insomnia para realizar pruebas de las solicitudes con autenticación, asegurando que los endpoints protegidos funcionaran correctamente. Además, se presentaron desafíos en la validación y manipulación de datos en la base de datos, lo que requirió ajustes en los modelos y serializadores.

A pesar de estos desafíos, se lograron realizar exitosamente las operaciones CRUD con seguridad, garantizando que solo los usuarios autenticados pudieran acceder a la información. También se documentaron los pasos del proceso, facilitando la comprensión y replicación del proyecto en otros entornos.

En general, esta práctica permitió reforzar habilidades en el desarrollo de APIs REST, la implementación de seguridad con JWT y el uso de herramientas como Insomnia para la prueba y validación de servicios web, lo que representa una base sólida para proyectos futuros en este ámbito.

Bibliografía.

Casr, D. (2018). Construir un API REST con Django REST Framework y APIView. Recuperado de

<https://davidcasr.medium.com/construir-un-api-rest-con-django-rest-framework-y-apiview-5ea4b2823307>

Jaime. (2020). Desarrollo de API Restful con Python, Django y Django Rest Framework. Recuperado de

<https://devjaime.medium.com/desarrollo-de-api-restful-con-python-django-y-django-rest-framework-e6f019dbe177>

Viera, U. (2025.). Creación de una API REST con Django Rest Framework. Recuperado de

<https://www.urianviera.com/django/django-rest-api-guide>

Hektor Profe. (2025). Proyecto API con Django Rest Framework. Recuperado de

<https://docs.hektorprofe.net/academia/django/api-rest-framework/>

Urian121. (2025). Consumir una API REST con Django y Python en 10 líneas. Recuperado de

<https://github.com/urian121/Consumir-una-API-REST-con-Django-y-Python-en-10-lineas>