INVESTIGACION DATOS NO ESTRUCTURADOS

¡Excelente que quieras hacer algo al nivel de Rappi! 🔥 Es una gran idea y un buen desafío en Django.

Te explico de manera **clara y paso a paso** cómo podrías hacerlo:

**🚀 ¿Qué es lo que quieres lograr?**

Tener **3 tipos de usuarios**:

1. **Cliente** ➔ Pide productos.
2. **Tienda** ➔ Vende productos.
3. **Domiciliario (repartidor)** ➔ Entrega los pedidos.

**🛠️ ¿Cómo hacerlo en Django?**

Hay varias formas, pero la más usada (y profesional) es:

**1. Extender el modelo de usuario**

En lugar de tener un solo tipo de usuario, tu modelo de Profile (o de usuario extendido) tendría un campo llamado, por ejemplo, rol.

from django.contrib.auth.models import AbstractUser

from django.db import models

class CustomUser(AbstractUser):

ROL\_CHOICES = (

('cliente', 'Cliente'),

('tienda', 'Tienda'),

('domiciliario', 'Domiciliario'),

)

rol = models.CharField(max\_length=20, choices=ROL\_CHOICES, default='cliente')

# Puedes agregar más campos si quieres

telefono = models.CharField(max\_length=15, blank=True, null=True)

direccion = models.TextField(blank=True, null=True)

**¿Qué es esto?**

* Estamos creando nuestro propio modelo de usuario basado en AbstractUser.
* Le agregamos un campo extra llamado rol.
* Cada usuario tendrá un **rol**: cliente, tienda o domiciliario.

**2. Configurar Django para usar tu propio usuario**

En settings.py debes decirle a Django que ahora usarás tu modelo de usuario personalizado:

AUTH\_USER\_MODEL = 'tuapp.CustomUser'

(donde tuapp es el nombre de tu aplicación).

**3. Durante el registro, eliges el rol**

Cuando un usuario se registre, puedes hacer que seleccione **qué tipo de cuenta** quiere crear.

**Formulario de registro ejemplo:**

from django import forms

from .models import CustomUser

class RegistroForm(forms.ModelForm):

password = forms.CharField(widget=forms.PasswordInput)

class Meta:

model = CustomUser

fields = ['username', 'email', 'password', 'rol']

➔ Y en tu signup view, registras según el rol que elija.

**4. Restricciones por rol**

Luego, cuando un usuario quiera hacer algo, puedes verificar **qué rol tiene**:

if request.user.rol == 'tienda':

# Mostrar opciones para crear productos

elif request.user.rol == 'domiciliario':

# Mostrar pedidos asignados

elif request.user.rol == 'cliente':

# Mostrar tiendas y productos para comprar

**🎯 Resumiendo:**

| **Paso** | **Qué hacer** |
| --- | --- |
| 1 | Crear un modelo de usuario extendido con un campo rol |
| 2 | Configurar Django para usar ese modelo |
| 3 | Crear formularios y vistas de registro por rol |
| 4 | Controlar lo que cada rol puede hacer |

**📦 Funciones que podrías agregar después:**

* **Tienda** puede crear y administrar sus productos.
* **Cliente** puede buscar productos y hacer pedidos.
* **Domiciliario** puede ver los pedidos pendientes y aceptarlos.
* **Paneles diferentes** para cada tipo de usuario.
* **Asignación automática** de pedidos al domiciliario más cercano.

**🚀 ¿Quieres que te arme un pequeño ejemplo funcional de cómo serían las vistas (views.py) para crear una tienda o un domiciliario?**

Te puedo hacer una mini versión para que la uses de base si quieres. ¿Te gustaría? 🎯 (Te tomaría 5 minutos leerlo y entenderlo).

¡Genial! 🔥 Ahora sí vamos a hablar en serio: esto que describes **es totalmente posible** en Django.  
Te lo voy a explicar paso a paso de forma **sencilla y organizada**, y además te voy a dar **ideas reales de código** para que puedas construirlo.

**🚀 Tu Meta:**

**Crear una plataforma tipo Rappi, con 3 roles principales**:

| **Rol** | **Funcionalidades** |
| --- | --- |
| **Tienda** | Crear, actualizar y eliminar productos. Ver pedidos de su tienda. |
| **Cliente** | Buscar productos, agregarlos al carrito y hacer pedidos. |
| **Domiciliario** | Ver pedidos pendientes, aceptar uno, hacer la entrega. |

**📚 ¿Cómo organizamos esto en Django?**

Aquí te explico **cada módulo que necesitas**:

**1. Modelo de Usuario extendido (CustomUser)**

Ya lo vimos antes: necesitas un rol para saber si es tienda, cliente o domiciliario.

class CustomUser(AbstractUser):

ROL\_CHOICES = (

('cliente', 'Cliente'),

('tienda', 'Tienda'),

('domiciliario', 'Domiciliario'),

)

rol = models.CharField(max\_length=20, choices=ROL\_CHOICES, default='cliente')

**2. Modelo de Producto**

Solo las tiendas pueden crear productos.

class Producto(models.Model):

tienda = models.ForeignKey(CustomUser, on\_delete=models.CASCADE, limit\_choices\_to={'rol': 'tienda'})

nombre = models.CharField(max\_length=200)

descripcion = models.TextField()

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

imagen = models.ImageField(upload\_to='productos/')

disponible = models.BooleanField(default=True)

* tienda: quien subió el producto.
* Solo usuarios con rol tienda deberían poder crear productos.

**3. Modelo de Pedido**

Cuando un cliente hace un pedido.

class Pedido(models.Model):

cliente = models.ForeignKey(CustomUser, on\_delete=models.CASCADE, related\_name='pedidos', limit\_choices\_to={'rol': 'cliente'})

tienda = models.ForeignKey(CustomUser, on\_delete=models.CASCADE, related\_name='pedidos\_recibidos', limit\_choices\_to={'rol': 'tienda'})

producto = models.ForeignKey(Producto, on\_delete=models.CASCADE)

cantidad = models.PositiveIntegerField(default=1)

estado = models.CharField(max\_length=20, choices=[

('pendiente', 'Pendiente'),

('aceptado', 'Aceptado'),

('en camino', 'En camino'),

('entregado', 'Entregado')

], default='pendiente')

domiciliario = models.ForeignKey(CustomUser, on\_delete=models.SET\_NULL, null=True, blank=True, related\_name='pedidos\_asignados', limit\_choices\_to={'rol': 'domiciliario'})

fecha\_pedido = models.DateTimeField(auto\_now\_add=True)

* Un pedido pertenece a un cliente y a una tienda.
* Puede estar **asignado a un domiciliario** o no (por eso el null=True).

**4. Paneles Diferentes por Usuario**

En tu views.py, puedes hacer que según el rol lo mandes a diferentes dashboards:

@login\_required

def dashboard(request):

if request.user.rol == 'tienda':

return redirect('dashboard\_tienda')

elif request.user.rol == 'cliente':

return redirect('dashboard\_cliente')

elif request.user.rol == 'domiciliario':

return redirect('dashboard\_domiciliario')

➔ Así, cuando un usuario entre, va a su propio panel. 🎯

**5. Asignación automática al domiciliario más cercano**

Esto ya es un nivel más pro 😎. Te explico una forma sencilla de hacerlo:

* Cada **domiciliario** tiene una ubicación (por ejemplo latitud/longitud).
* Cuando un pedido es nuevo:
  + Buscas **el domiciliario más cercano** disponible usando la distancia.
  + Le asignas el pedido automáticamente.

Ejemplo sencillo de cálculo de distancia:

from math import radians, sin, cos, sqrt, atan2

def calcular\_distancia(lat1, lon1, lat2, lon2):

R = 6371 # Radio de la tierra en km

dlat = radians(lat2 - lat1)

dlon = radians(lon2 - lon1)

a = sin(dlat/2)\*\*2 + cos(radians(lat1)) \* cos(radians(lat2)) \* sin(dlon/2)\*\*2

c = 2 \* atan2(sqrt(a), sqrt(1 - a))

distancia = R \* c

return distancia

➔ Y luego haces un pequeño loop para buscar al domiciliario **más cercano** que esté **libre**.

(Ya después podrías usar librerías más potentes como geopy o APIs de mapas si quieres).

**📋 Flujo básico que debes construir:**

| **Paso** | **Acción** |
| --- | --- |
| Cliente | Se registra y navega productos. |
| Cliente | Hace un pedido. |
| Pedido | Se crea como "pendiente". |
| Sistema | Busca un domiciliario cercano y le asigna el pedido. |
| Domiciliario | Acepta o rechaza el pedido. |
| Domiciliario | Entrega el pedido. |
| Sistema | Cambia estado a "entregado". |

**🎯 Entonces, ¿qué te recomiendo hacer AHORA?**

1. Crear el CustomUser con rol.
2. Crear los modelos de Producto y Pedido.
3. Hacer dashboards separados para tienda, cliente y domiciliario.
4. Hacer que una tienda pueda crear productos.
5. Hacer que un cliente pueda hacer pedidos.
6. Hacer que un domiciliario vea y acepte pedidos.

**💬 ¿Quieres que además te arme un pequeño "plan de trabajo" en Django?**

Algo como:

* Qué modelos hacer primero.
* Qué vistas seguir.
* Qué urls configurar.
* Qué templates crear.

Así tendrías como un **mapa completo** para no perderte. 🚀

¿Te gustaría? 🔥 (Te lo puedo dejar súper claro en 5 minutos).

¡Perfecto, te entiendo! 🔥 Estás yendo hacia algo **más avanzado y realista**:  
lo que dices se llama usar **datos no estructurados** porque **cada tienda podría tener productos con distinta información** o **tener su propia base de datos**.

Ahora, **¡relax!**, no necesitas complicarte demasiado.  
Te explico bien claro:

**🎯 ¿Qué quieres lograr exactamente?**

* Cada **tienda** ya tiene **su propia base de datos** o **su propio inventario**.
* Cuando ellos se registran en tu plataforma, **no vas a crear manualmente cada producto**.
* En cambio, quieres **leer/importar automáticamente** sus productos.
* Y **NO** quieres estar atado a una estructura fija de campos (por ejemplo, una tienda puede tener "color", otra "peso", otra "marca", etc.).

**🛠️ ¿Cómo hacerlo en Django?**

**1. Guardar la información como JSONField**

Django tiene un campo llamado JSONField que es perfecto para **datos no estructurados**.  
¡Cada producto puede tener **propiedades distintas**!

**Ejemplo de modelo Producto flexible:**

from django.contrib.postgres.fields import JSONField # Si usas PostgreSQL

from django.db import models

class Producto(models.Model):

tienda = models.ForeignKey(CustomUser, on\_delete=models.CASCADE, limit\_choices\_to={'rol': 'tienda'})

nombre = models.CharField(max\_length=200)

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

disponible = models.BooleanField(default=True)

datos\_extra = models.JSONField(blank=True, null=True) # Aquí se guarda toda la info no estructurada

➔ datos\_extra puede guardar lo que quieras:

{

"color": "rojo",

"peso": "2 kg",

"material": "algodón",

"marca": "Nike"

}

O en otra tienda:

{

"volumen": "500 ml",

"ingredientes": ["agua", "azúcar", "limón"]

}

**2. ¿Cómo llenas datos\_extra?**

Cuando conectes con las tiendas (por API o por carga de archivos como CSV, JSON, etc.), puedes directamente **insertar** la data en este campo.

Ejemplo rápido:

nuevo\_producto = Producto.objects.create(

tienda=mi\_tienda,

nombre="Botella de Agua",

precio=5.99,

datos\_extra={"volumen": "500 ml", "material": "plástico reciclado"}

)

**3. ¿Cómo mostrar los datos extra en el frontend?**

En tu plantilla Django (producto\_detail.html):

{% for key, value in producto.datos\_extra.items %}

<p><strong>{{ key }}:</strong> {{ value }}</p>

{% endfor %}

➔ Así automáticamente muestras **todos los datos extra** del producto, ¡aunque cada uno tenga datos distintos!

**🚀 Ventajas de hacerlo así:**

| **Ventaja** | **Explicación** |
| --- | --- |
| Flexible | Cada tienda puede tener productos con campos diferentes. |
| Escalable | No tienes que modificar tu base de datos cada vez que una tienda nueva tiene algo distinto. |
| Rápido | Puedes integrar productos automáticamente si una tienda sube su propio catálogo. |
| Profesional | Es el enfoque real que usan marketplaces grandes como Amazon o Rappi. |

**👀 Resumiendo**

Tu modelo de **Producto** tendrá un campo datos\_extra,  
donde cada tienda puede guardar **información distinta** de cada producto.  
Tú solo te encargas de mostrar dinámicamente esa información.

**⚡¿Quieres que te prepare también un ejemplo de cómo sería integrar los productos de una tienda por archivo (JSON) o una API?**

Te puedo armar un ejemplo práctico si quieres, algo como:

* Suben un .json o .csv
* Lo lees en Django
* Automáticamente creas los productos en la tienda

¿Te interesa? 🚀  
(Te ayudaría muchísimo si planeas conectar varias tiendas automáticamente).

¡Perfecto! 🔥 Vamos a hacer un ejemplo **realista** de cómo integrar productos de una tienda a tu plataforma usando un **archivo JSON**.

Te lo voy a dividir en **tres partes** para que lo entiendas bien:

**🛠️ 1. El escenario**

Imagina que **la tienda** te envía un archivo JSON llamado productos\_tienda.json, que contiene su inventario.

Ejemplo del archivo JSON:

[

{

"nombre": "Camiseta Roja",

"precio": 19.99,

"datos\_extra": {

"color": "rojo",

"talla": "M",

"material": "algodón"

}

},

{

"nombre": "Zapatos de Deporte",

"precio": 59.99,

"datos\_extra": {

"color": "blanco",

"marca": "Nike",

"talla": "42"

}

}

]

Cada producto tiene su información básica y un campo datos\_extra.

**🛠️ 2. Crear una vista para cargar el archivo en Django**

En tu views.py, creas una vista para **subir** e **importar** productos desde el JSON:

import json

from django.shortcuts import render, redirect

from .models import Producto

from django.contrib import messages

from django.contrib.auth.decorators import login\_required

@login\_required

def importar\_productos(request):

if request.method == 'POST' and request.FILES.get('archivo'):

archivo = request.FILES['archivo']

try:

datos = json.load(archivo)

for item in datos:

Producto.objects.create(

tienda=request.user, # Asignamos la tienda actual

nombre=item['nombre'],

precio=item['precio'],

datos\_extra=item.get('datos\_extra', {})

)

messages.success(request, "Productos importados exitosamente.")

return redirect('dashboard\_tienda') # Redirige al panel de tienda

except Exception as e:

messages.error(request, f"Error importando productos: {str(e)}")

return render(request, 'importar\_productos.html')

✅ ¿Qué hace esto?

* Lee el archivo JSON.
* Crea automáticamente un producto por cada entrada.
* Asocia cada producto a la tienda que subió el archivo.

**🛠️ 3. Crear un formulario HTML simple para subir el archivo**

En importar\_productos.html:

<h2>Importar Productos</h2>

<form method="POST" enctype="multipart/form-data">

{% csrf\_token %}

<input type="file" name="archivo" accept=".json" required>

<button type="submit">Subir Productos</button>

</form>

{% if messages %}

{% for message in messages %}

<p>{{ message }}</p>

{% endfor %}

{% endif %}

**🚀 ¿Qué logras con esto?**

| **Acción** | **Resultado** |
| --- | --- |
| La tienda sube su archivo JSON |  |
| Django lee todos los productos |  |
| Django guarda los productos en tu base de datos automáticamente |  |
| Cada producto queda asociado a la tienda que hizo el upload |  |

**📦 Extras que puedes agregar fácilmente después:**

* Validar que el JSON tenga los campos correctos (nombre, precio).
* Permitir también carga por **CSV** si lo prefieres.
* Integrarlo directamente con una **API** si las tiendas tienen sistemas más avanzados.

**🎯 Resumen visual:**

Tienda ➔ Sube archivo JSON ➔ Django lee y guarda productos ➔ ¡Listo para vender!

**🧠 ¿Te gustaría que también te enseñe cómo sería si quieres que en vez de subir archivos la tienda mande sus productos directamente desde su propio sistema a tu web? (tipo API pública para tiendas).**

Podrías tener tu propio mini "API de Rappi" 🤯. ¿Quieres que te arme eso también? 🚀

¡Excelente! 🔥 Ahora sí vamos a ponernos en modo **PRO**:  
**crearás tu propia API** para que las **tiendas** puedan **enviar productos automáticamente** a tu web (sin subir archivos manualmente).

**Esto es exactamente lo que hace Rappi, Amazon, UberEats, etc. internamente.**

**🚀 ¿Qué vas a lograr?**

* Cada tienda tendrá una **clave API** (un token de acceso).
* Desde su propio sistema, las tiendas podrán **mandar productos** directamente a tu Django por **HTTP (POST JSON)**.
* Tu sistema creará esos productos automáticamente.

**📚 ¿Qué necesitas?**

* Django.
* Django REST Framework (DRF) ➔ (si no lo tienes: pip install djangorestframework).
* Un modelo que guarde los productos (ya lo tienes).
* Un endpoint seguro que **reciba** los productos.

**🛠️ PASO 1: Instalar Django REST Framework**

En tu terminal:

pip install djangorestframework

Agrega en settings.py:

INSTALLED\_APPS = [

...

'rest\_framework',

]

**🛠️ PASO 2: Crear un modelo para la API Key (opcional pero recomendado)**

Cada tienda debe tener una **llave API única**.

import uuid

from django.db import models

class ApiKey(models.Model):

tienda = models.OneToOneField(CustomUser, on\_delete=models.CASCADE, limit\_choices\_to={'rol': 'tienda'})

key = models.UUIDField(default=uuid.uuid4, editable=False, unique=True)

Cuando una tienda se registre, le generas una API Key automáticamente.

**🛠️ PASO 3: Crear la API para recibir productos**

En tu views.py:

from rest\_framework.decorators import api\_view

from rest\_framework.response import Response

from rest\_framework import status

from .models import Producto, ApiKey, CustomUser

@api\_view(['POST'])

def recibir\_producto(request):

api\_key = request.headers.get('Authorization')

if not api\_key:

return Response({'error': 'No API Key provided'}, status=status.HTTP\_401\_UNAUTHORIZED)

try:

api\_key\_obj = ApiKey.objects.get(key=api\_key)

except ApiKey.DoesNotExist:

return Response({'error': 'Invalid API Key'}, status=status.HTTP\_401\_UNAUTHORIZED)

tienda = api\_key\_obj.tienda

nombre = request.data.get('nombre')

precio = request.data.get('precio')

datos\_extra = request.data.get('datos\_extra', {})

if not nombre or not precio:

return Response({'error': 'Missing fields'}, status=status.HTTP\_400\_BAD\_REQUEST)

producto = Producto.objects.create(

tienda=tienda,

nombre=nombre,

precio=precio,

datos\_extra=datos\_extra

)

return Response({'message': 'Producto creado exitosamente', 'producto\_id': producto.id}, status=status.HTTP\_201\_CREATED)

✅ **¿Qué hace este endpoint?**

* Requiere una **API Key** en el header.
* Crea un **Producto** automáticamente con los datos recibidos.

**🛠️ PASO 4: Configurar URL**

En tu urls.py:

from . import views

urlpatterns = [

path('api/recibir-producto/', views.recibir\_producto, name='recibir\_producto'),

]

**🛠️ PASO 5: ¿Cómo una tienda usaría esta API?**

Supongamos que la tienda tiene esta API Key: 123e4567-e89b-12d3-a456-426614174000

Ellos desde su sistema (o Postman) mandan este POST:

**POST** https://tuweb.com/api/recibir-producto/  
**Headers:**

Authorization: 123e4567-e89b-12d3-a456-426614174000

Content-Type: application/json

**Body:**

{

"nombre": "Hamburguesa Triple",

"precio": 8.99,

"datos\_extra": {

"ingredientes": ["carne", "queso", "tocino"],

"peso": "300g"

}

}

Y **boom 💥**, ¡el producto se crea automáticamente en tu sistema!

**📈 Beneficios de usar API:**

| **Beneficio** | **Explicación** |
| --- | --- |
| Escalable | Puedes recibir miles de productos de muchas tiendas automáticamente. |
| Seguro | Solo aceptas peticiones de tiendas con API Key válida. |
| Profesional | Es el estándar mundial de integración. |
| Flexible | Las tiendas pueden integrar su inventario en tiempo real. |

**🎯 Resumen de cómo sería el flujo:**

Tienda → Su sistema → API Request POST → Tu servidor Django → Producto creado automáticamente

**⚡ Pregunta rápida:**

¿Quieres que además te prepare un ejemplo de **cómo una tienda podría consultar sus productos** (es decir, hacer un **GET** para listar sus productos usando su API Key también)?  
Sería el complemento ideal para tener un **API completo de Rappi-style** 🚀.

¿Te gustaría que lo arme? 🔥 (sería un bonus para tener carga + consulta desde la tienda).

¡Perfecto! Vamos a crear la **documentación de tu API** de manera profesional. Esta es la **documentación** que podrías enviarle a las tiendas para que **puedan integrar** sus productos de forma fácil y clara.

**📚 Documentación API para Tiendas**

Bienvenido a la **API de integración de productos**. Esta API permite a las tiendas **crear y consultar productos** de forma sencilla y segura.

**🚀 EndPoints Disponibles**

**1. Crear Producto (POST)**

* **Ruta:** /api/recibir-producto/
* **Método:** POST
* **Descripción:** Este endpoint permite a las tiendas **crear nuevos productos** en la plataforma.

**Headers:**

* Authorization: La **API Key** de la tienda (proporcionada al registrarse).

**Body (JSON):**

{

"nombre": "Nombre del producto",

"precio": "Precio del producto",

"datos\_extra": {

"campo\_adicional": "valor",

"otro\_campo": "valor"

}

}

**Ejemplo de solicitud:**

**POST** /api/recibir-producto/  
**Headers:**

Authorization: 123e4567-e89b-12d3-a456-426614174000

Content-Type: application/json

**Body:**

{

"nombre": "Camiseta Roja",

"precio": "19.99",

"datos\_extra": {

"color": "rojo",

"talla": "M",

"material": "algodón"

}

}

**Respuesta Exitosa (201 Created):**

{

"message": "Producto creado exitosamente",

"producto\_id": 1

}

**Errores Comunes:**

* **401 Unauthorized**: API Key inválida o no proporcionada.
* **400 Bad Request**: Faltan campos obligatorios (nombre, precio).

**2. Listar Productos (GET)**

* **Ruta:** /api/listar-productos/
* **Método:** GET
* **Descripción:** Este endpoint permite a las tiendas **consultar** todos los productos que han subido a la plataforma.

**Headers:**

* Authorization: La **API Key** de la tienda.

**Ejemplo de solicitud:**

**GET** /api/listar-productos/  
**Headers:**

Authorization: 123e4567-e89b-12d3-a456-426614174000

**Respuesta Exitosa (200 OK):**

[

{

"id": 1,

"nombre": "Camiseta Roja",

"precio": "19.99",

"datos\_extra": {

"color": "rojo",

"talla": "M",

"material": "algodón"

},

"disponible": true

},

{

"id": 2,

"nombre": "Zapatos de Deporte",

"precio": "59.99",

"datos\_extra": {

"color": "blanco",

"marca": "Nike",

"talla": "42"

},

"disponible": true

}

]

**Errores Comunes:**

* **401 Unauthorized**: API Key inválida o no proporcionada.

**🔐 Autenticación con API Key**

Todas las solicitudes deben incluir la **API Key** de la tienda en los **Headers**. La API Key se obtiene cuando una tienda se registra en tu plataforma y se asocia a su cuenta.

**Ejemplo de Header:**

Authorization: Tu-API-Key

**💡 Notas Importantes**

* Asegúrate de enviar siempre el campo **Authorization** en el header para evitar errores de autenticación.
* El campo **datos\_extra** puede ser cualquier información adicional que desees asociar con el producto (por ejemplo, color, tamaño, marca, etc.), pero no es obligatorio.
* Si en algún momento deseas **revocar** la API Key de una tienda, puedes hacerlo manualmente desde el panel de administración de tu plataforma.

**🛠️ Errores Comunes**

| **Error** | **Descripción** |
| --- | --- |
| **401 Unauthorized** | La API Key es inválida o no fue proporcionada. |
| **400 Bad Request** | Faltan campos obligatorios (por ejemplo, nombre, precio). |

**💬 Soporte**

Si tienes problemas o preguntas con el uso de la API, no dudes en contactar con nuestro equipo de soporte a través del correo [**soporte@tuweb.com**](mailto:soporte@tuweb.com).

**🎯 Resumen**

Con esta API, las tiendas podrán:

* **Crear productos** de forma automática usando la **API POST**.
* **Consultar sus productos** en cualquier momento con la **API GET**.
* Utilizar una **API Key segura** para acceder a la plataforma.

🎉 ¡Listo! Ahora tienes una API completa para gestionar productos de manera eficiente, ¡y con un nivel profesional de integración! 🚀  
Si alguna tienda tiene alguna duda, solo debes enviarles esta documentación.

**📈 Siguientes pasos:**

* Hacer tests para validar las solicitudes.
* Asegurarte de que todo funcione correctamente en el entorno de producción.
* ¡Listo para integrarlo con tiendas reales!

¡Buena suerte y que tu proyecto crezca a lo grande! 💪

¡Excelente pregunta! 🔥  
Probar tu propia API es **fundamental** para asegurarte de que todo funciona antes de que las tiendas la usen.

Te explico las 3 formas más prácticas para probarla tú misma:

**📦 Opciones para probar tu API:**

**1. Usando Postman (más recomendado)**

Una herramienta visual para hacer peticiones HTTP de manera fácil.

**Pasos:**

1. Descarga Postman gratis: <https://www.postman.com/>
2. Abre Postman y crea una nueva **Request**.
3. Configura así:

* **Método:** POST o GET (dependiendo qué quieras probar)
* **URL:**
  + Si estás en desarrollo local: http://127.0.0.1:8000/api/recibir-producto/
  + O para listar: http://127.0.0.1:8000/api/listar-productos/
* **Headers:**
  + Key: Authorization
  + Value: <tu API Key>
* **Body:** (cuando es POST)
  + Tipo: raw
  + Formato: JSON
  + Ejemplo:

{

"nombre": "Camiseta Roja",

"precio": "19.99",

"datos\_extra": {

"color": "Rojo",

"talla": "M"

}

}

1. Haz clic en **Send** y ¡listo! Verás la respuesta de tu API.

**2. Usando curl desde la terminal (rápido si sabes comandos)**

**Para crear producto:**

curl -X POST http://127.0.0.1:8000/api/recibir-producto/ \

-H "Authorization: TU\_API\_KEY" \

-H "Content-Type: application/json" \

-d '{

"nombre": "Zapatos Deportivos",

"precio": "59.99",

"datos\_extra": {

"color": "Negro",

"marca": "Nike",

"talla": "42"

}

}'

**Para listar productos:**

curl -X GET http://127.0.0.1:8000/api/listar-productos/ \

-H "Authorization: TU\_API\_KEY"

**3. Usando Django Admin + navegador**

(Esto sirve para revisar los resultados, no para enviar productos)

* Desde tu navegador: accede a /admin/ de tu proyecto Django.
* Entra a los modelos de Producto, ApiKey, etc.
* Mira si se guardaron los productos que enviaste desde Postman o curl.

**🛠️ ¿Qué necesitas antes de probar?**

* Que tu **servidor Django** esté corriendo:
* python manage.py runserver
* Que tengas una **API Key válida** (creada en tu admin Django).
* Que tus endpoints estén bien configurados en urls.py.

**⚡ Tip extra para pruebas más reales**

Cuando termines de probar en local, puedes montar tu proyecto en algo gratis como **Render.com**, **Railway.app** o **Vercel + Django** para probarlo desde internet (no solo localmente).

**🎯 Resumen**

| **Herramienta** | **Para qué sirve** | **Dificultad** |
| --- | --- | --- |
| Postman | Visualizar, probar fácil | Muy fácil |
| curl | Probar desde consola | Intermedio |
| Admin Django | Ver resultados manualmente | Muy fácil |

**🚀 ¿Quieres que te prepare un ejemplo paso a paso en Postman para que puedas copiarlo directo y hacer tu primera prueba en 2 minutos? 🎯**

(Te mando cómo configurar todo visualmente si quieres).  
¿Te lo paso? 📦✨

¡Perfecto! 🔥 Vamos a hacerlo super claro para que puedas probar tu API en **Postman** fácilmente.

**🧪 Guía paso a paso para probar tu API con Postman**

**1. Instalar Postman**

✅ Si no tienes Postman:

* Descárgalo gratis desde [postman.com](https://www.postman.com/downloads/).

**2. Crear una Nueva Colección (opcional pero organizado)**

* Abre Postman.
* Clic en **Collections** ➔ **New Collection** ➔ Ponle nombre como "Mi API Tiendas".
* Luego dentro de esta colección vas creando las requests.

**3. Crear la solicitud para crear un producto**

1. **Nueva Request** ➔ nómbrala: Crear Producto.
2. Configura así:

| **Campo** | **Valor** |
| --- | --- |
| Método | POST |
| URL | http://127.0.0.1:8000/api/recibir-producto/ |

(Recuerda que 127.0.0.1:8000 es tu servidor local de Django).

1. **Agregar Headers:**

Ve a la pestaña **Headers** ➔ agrega:

| **KEY** | **VALUE** |
| --- | --- |
| Authorization | tu-API-Key (ej: 123e4567-e89b-12d3-a456-426614174000) |
| Content-Type | application/json |

1. **Agregar Body:**

* Ve a la pestaña **Body**.
* Selecciona **raw** ➔ en el combo de la derecha pon **JSON**.
* Copia este ejemplo de Body:

{

"nombre": "Camiseta Roja",

"precio": "19.99",

"datos\_extra": {

"color": "Rojo",

"talla": "M",

"material": "Algodón"

}

}

1. **Enviar la solicitud**

* Haz clic en **Send** 🚀
* Deberías ver una respuesta como:

{

"message": "Producto creado exitosamente",

"producto\_id": 1

}

✅ Eso significa que el producto ya se creó en tu base de datos.

**4. Crear la solicitud para listar productos**

1. **Nueva Request** ➔ nómbrala: Listar Productos.
2. Configura:

| **Campo** | **Valor** |
| --- | --- |
| Método | GET |
| URL | http://127.0.0.1:8000/api/listar-productos/ |

1. **Headers:**

| **KEY** | **VALUE** |
| --- | --- |
| Authorization | tu-API-Key (el mismo) |

(No es necesario poner Body en un GET).

1. **Enviar**

* Haz clic en **Send** 🚀
* Deberías ver la lista de productos:

[

{

"id": 1,

"nombre": "Camiseta Roja",

"precio": "19.99",

"datos\_extra": {

"color": "Rojo",

"talla": "M",

"material": "Algodón"

},

"disponible": true

}

]

✅ Eso confirma que tu API está funcionando correctamente.

**🎁 Tips útiles en Postman**

* Puedes guardar las requests en una **Collection** para usarlas después.
* Puedes duplicar requests si quieres probar cambios rápidos.
* Puedes usar **Environments** para guardar la URL base o la API Key y no tener que escribirlas siempre.

**📸 ¿Quieres que además te prepare una captura visual de cómo debe verse la configuración en Postman?**

(Si quieres, te hago una imagen de ejemplo real para que compares lo que ves). 🚀  
¿Te la envío? 🎨