PARTE 1: Crear el proyecto en Visual Studio Code

Paso 1: Preparar el entorno

1. **Instala Python** desde python.org
2. **Instala MySQL** desde mysql.com
3. **Instala Visual Studio Code** desde code.visualstudio.com
4. Abre VS Code y abre una terminal (Ctrl + `)

Paso 2: Crear carpeta y entorno virtual

Terminal

```
mkdir burritos_to_go

cd burritos_to_go

python -m venv env

env\Scripts\activate  # En Windows

# source env/bin/activate  # En Mac/Linux
```

Paso 3: Instalar Django y dependencias

Terminal

```
pip install django djangorestframework
mysqlclient python-dotenv
```

Paso 4: Crear el proyecto y la app

bash

```
django-admin startproject burritos_project .

python manage.py startapp core
```

Agrega 'core' y 'rest_framework' a INSTALLED_APPS en burritos_project/settings.py.

Paso 5: Configurar MySQL

1. Crea una base de datos en MySQL

```
CREATE DATABASE burritos_db CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;
```

2. En settings.py, configura la base de datos:

python

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.mysql',
    'NAME': 'burritos_db',
    'USER': 'root',
    'PASSWORD': 'tu_contraseña',
    'HOST': 'localhost',
    'PORT': '3306',
  }}
```

En settings.py: (Carpeta Burritos_project)

AUTH_USER_MODEL = 'core.Usuario'

Paso 6: Crear modelos de Categoría, Producto y Pedido

En core/models.py:

```
from django.contrib.auth.models import AbstractUser

from django.db import models

from django.utils import timezone

from decimal import Decimal


class Usuario(AbstractUser):

    ROLES = (

        ('super', 'Súper Usuario'),

        ('admin', 'Administrador'),

        ('cliente', 'Cliente'),

    )

    rol = models.CharField(max_length=10, choices=ROLES, default='cliente')
```

```
    saldo = models.DecimalField(max_digits=10,
decimal_places=2, default=0.00)


class Categoria(models.Model):
    nombre = models.CharField(max_length=50)


    def __str__(self):
        return self.nombre


class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    descripcion = models.TextField()
    precio = models.DecimalField(max_digits=6,
decimal_places=2)
    categoria = models.ForeignKey(Categoria,
on_delete=models.CASCADE)
    activo = models.BooleanField(default=True)


    def __str__(self):
        return f"{self.nombre} (${self.precio})"
```

```
class Pedido(models.Model):

    cliente = models.ForeignKey('Usuario',
on_delete=models.CASCADE)

    productos =
models.ManyToManyField('Producto')

    total = models.DecimalField(max_digits=10,
decimal_places=2, default=0)

    estatus = models.CharField(max_length=20,
default='pendiente')

    fecha =
models.DateTimeField(auto_now_add=True)


    def save(self, *args, **kwargs):

        if self.pk:  # Solo al crear

            super().save(*args, **kwargs)  # Necesario
para poder usar .productos

            self.total = sum(p.precio for p in
self.productos.all())

            super().save(update_fields=['total'])

        else:

            super().save(*args, **kwargs)
```

```
    def __str__(self):
        return f"Pedido #{self.id} -
{self.cliente.username}"
```

Paso 7: Migraciones y súper usuario

Terminal

python manage.py makemigrations

python manage.py migrate

python manage.py createsuperuser

Paso 8: Crear API REST

1. En core/serializers.py:

```
from rest_framework import serializers

from .models import Usuario, Producto, Categoria, Pedido


class UsuarioSerializer(serializers.ModelSerializer):

    class Meta:

        model = Usuario

        fields = ['id', 'username', 'email', 'rol', 'saldo']
```

```
class
ProductoSerializer(serializers.ModelSeriali
zer):
    class Meta:
        model = Producto
        fields = '__all__'


class
CategoriaSerializer(serializers.ModelSerial
izer):
    class Meta:
        model = Categoria
        fields = '__all__'


class
PedidoSerializer(serializers.ModelSerialize
r):
    class Meta:
        model = Pedido
        fields = '__all__'
        extra_kwargs = {
```

```
            'cliente': {'read_only': True},
            'total': {'read_only': True}
        }


class
CrearPedidoSerializer(serializers.Serializer
):
    productos = serializers.ListField(
        child=serializers.IntegerField(),
        allow_empty=False
    )


    def validate_productos(self, value):
        productos =
Producto.objects.filter(id__in=value,
activo=True)
        if not productos.exists():
            raise serializers.ValidationError("No
se encontraron productos válidos.")
        return value
```

2. En core/views.py:

```
from rest_framework import viewsets

from .models import Usuario, Producto,
Categoria, Pedido

from .serializers import UsuarioSerializer

from .serializers import *

from rest_framework.views import APIView

from rest_framework.response import Response

from rest_framework.permissions import
IsAuthenticated

from django.utils import timezone

from .models import Producto, Pedido

from .serializers import CrearPedidoSerializer


class UsuarioViewSet(viewsets.ModelViewSet):

    queryset = Usuario.objects.all()

    serializer_class = UsuarioSerializer


class ProductoViewSet(viewsets.ModelViewSet):

    queryset = Producto.objects.filter(activo=True)
```

```
    serializer_class = ProductoSerializer


class CategoriaViewSet(viewsets.ModelViewSet):
    queryset = Categoria.objects.all()
    serializer_class = CategoriaSerializer


class PedidoViewSet(viewsets.ModelViewSet):
    queryset = Pedido.objects.all()
    serializer_class = PedidoSerializer


    def perform_create(self, serializer):
        serializer.save(cliente=self.request.user)
        productos =
serializer.validated_data.get('productos', [])
        total = sum([p.precio for p in productos])
        serializer.save(cliente=self.request.user,
total=total)


class CrearPedidoView(APIView):
    permission_classes = [IsAuthenticated]
```

```
    def post(self, request):
        productos_ids = request.data.get('productos',
[])
        productos =
Producto.objects.filter(id__in=productos_ids,
activo=True)

        if not productos.exists():
            return Response({'error': 'No se
encontraron productos válidos.'}, status=400)

        total = sum([p.precio for p in productos])
        cliente = request.user

        if cliente.saldo < total:
            return Response({
                'error': 'Saldo insuficiente.',
                'saldo_actual': cliente.saldo,
                'total_pedido': total,
```

```
            'faltante': round(total - cliente.saldo, 2)
        }, status=400)

    pedido = Pedido.objects.create(
        cliente=cliente,
        total=total,
        estatus='pendiente',
        fecha=timezone.now()
    )
    pedido.productos.set(productos)
    cliente.saldo -= total
    cliente.save()

    return Response({
        'mensaje': 'Pedido creado exitosamente.',
        'pedido_id': pedido.id,
        'total': total,
        'productos': [p.nombre for p in productos],
        'fecha': pedido.fecha,
```

```
    'saldo_restante': cliente.saldo
})
```

3. En core/urls.py:

```
from django.urls import path, include
from rest_framework.routers import
DefaultRouter
from .views import (
    ProductoViewSet,
    CategoriaViewSet,
    PedidoViewSet,
    UsuarioViewSet,
    CrearPedidoView
)

router = DefaultRouter()
router.register(r'productos', ProductoViewSet)
router.register(r'categorias', CategoriaViewSet)
router.register(r'pedidos', PedidoViewSet)
router.register(r'usuarios', UsuarioViewSet)

urlpatterns = [
```

```
    path('', include(router.urls)),

    path('crear_pedido/',
CrearPedidoView.as_view(),
name='crear_pedido'),

]
```

4. En burritos_project/urls.py:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('core.urls')),
]
```

```
burritos_to_go/
├── burritos_project/
│   ├── settings.py
│   ├── urls.py
│   └── wsgi.py
├── core/
│   ├── models.py
│   ├── views.py
│   ├── serializers.py
│   ├── urls.py
│   └── admin.py
└── manage.py
```

settings.py (Carpeta burritos_project)

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
```

```
    'django.contrib.staticfiles',
    'rest_framework',
    'core',
]

AUTH_USER_MODEL = 'core.Usuario'

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'burritos_db',
        'USER': 'root',
        'PASSWORD': 'tu_contraseña',
        'HOST': 'localhost',
        'PORT': '3306',
    }
}
```

core/admin.py

```
from django.contrib import admin
from django.core.exceptions import ValidationError
from django.contrib import messages
from django import forms
from .models import Pedido, Producto


class PedidoForm(forms.ModelForm):
    class Meta:
        model = Pedido
        fields = '__all__'

    def __init__(self, *args, **kwargs):
        self.request = kwargs.pop('request', None)
        super().__init__(*args, **kwargs)

    def clean(self):
        cleaned_data = super().clean()
```

```
        productos = cleaned_data.get('productos',
[])

        cliente = cleaned_data.get('cliente')


        if not productos or not cliente:

            return cleaned_data  # Deja que otros
validadores se encarguen


        total = sum(p.precio for p in productos)

        self._total_calculado = total  # Guardamos el
total para usarlo en save()


        if cliente.saldo < total:

            raise ValidationError(f"❌ Saldo
insuficiente. El cliente tiene ${cliente.saldo:.2f} y
el pedido cuesta ${total:.2f}.")


        return cleaned_data


    def save(self, commit=True):

        instance = super().save(commit=False)
```

```
        instance.total = getattr(self,
'_total_calculado', 0)


        if commit:
            instance.save()
            self.save_m2m()
            cliente = instance.cliente
            cliente.saldo -= instance.total
            cliente.save()


            if self.request:
                messages.success(self.request, f"✅
Pedido creado. Se descontaron
${instance.total:.2f} del saldo del cliente.")


            return instance


class PedidoAdmin(admin.ModelAdmin):
    form = PedidoForm
    exclude = ('total',)
```

```
    filter_horizontal = ('productos',)


    def get_form(self, request, obj=None,
**kwargs):
        form_class = super().get_form(request, obj,
**kwargs)
        class FormWithRequest(form_class):
            def __new__(cls, *args, **kw):
                kw['request'] = request
                return form_class(*args, **kw)
        return FormWithRequest


    def save_related(self, request, form, formsets,
change):
        # Ya no es necesario calcular ni validar aquí
        super().save_related(request, form,
formsets, change)


admin.site.register(Pedido, PedidoAdmin)
```

Abrir el sistema en el navegador

Abre tu navegador y visita:

- **Panel de administración: http://127.0.0.1:8000/admin Inicia sesión con el súper usuario que creaste.**
- **API REST (si usaste routers): http://127.0.0.1:8000/api/ Aquí puedes ver los endpoints de productos, pedidos, usuarios, etc.**

Tabla: `core_usuario`

sql

```
INSERT INTO core_usuario (id,
username, email, password, rol,
saldo, is_superuser, is_staff,
is_active)
VALUES
(1, 'admin1',
'admin1@campus.mx',
'pbkdf2_sha256$...', 'admin',
0.00, 0, 1, 1),
(2, 'cliente1',
'cliente1@correo.mx',
'pbkdf2_sha256$...', 'cliente',
150.00, 0, 0, 1),
(3, 'cliente2',
'cliente2@correo.mx',
'pbkdf2_sha256$...', 'cliente',
80.00, 0, 0, 1),
(4, 'superuser',
'super@campus.mx',
```

```
'pbkdf2_sha256$...', 'super',
0.00, 1, 1, 1);
```

Reemplaza `'pbkdf2_sha256$...'` por contraseñas encriptadas generadas por Django. Puedes obtenerlas ejecutando en el shell:

python

```
from
django.contrib.auth.hashers
import make_password
print(make_password('tu_contras
eña'))
```

Tabla: `core_categoria`

sql

```
INSERT INTO core_categoria (id,
nombre)
VALUES
(1, 'Comidas'),
(2, 'Bebidas'),
(3, 'Postres');
```

Tabla: `core_producto`

sql

```
INSERT INTO core_producto (id,
nombre, descripcion, precio,
categoria_id, activo)
VALUES
(1, 'Burrito Clásico',
'Tortilla rellena de carne',
55.00, 1, TRUE),
(2, 'Agua Natural', 'Botella de
500ml', 15.00, 2, TRUE),
```

```
(3, 'Hamburguesa BBQ', 'Con
cebolla caramelizada', 65.00,
1, TRUE),
(4, 'Galleta integral', 'Hecha
con avena y miel', 20.00, 3,
TRUE),
(5, 'Refresco 600ml', 'Bebida
azucarada sabor cola', 25.00,
2, TRUE);
```

Tabla: **core_pedido**

sql

```
INSERT INTO core_pedido (id,
cliente_id, total, estatus,
fecha)
VALUES
(1, 2, 70.00, 'pendiente',
'2025-10-16 10:30:00'),
(2, 3, 40.00, 'confirmado',
'2025-10-16 11:00:00'),
(3, 2, 80.00, 'entregado',
'2025-10-15 14:45:00');
```

Tabla intermedia:

core_pedido_productos

sql

```
INSERT INTO
core_pedido_productos (id,
pedido_id, producto_id)
VALUES
(1, 1, 1),
(2, 1, 2),
(3, 2, 4),
(4, 3, 3),
(5, 3, 5);
```

```
env\Scripts\activate

python manage.py runserver
```