1. **Describir las características fundamentales de la integración del framework Django con bases de datos.**
    - Explicar cómo Django se integra con diferentes sistemas de bases de datos, como SQLite, PostgreSQL o MySQL.

Django está diseñado para integrarse de forma simple y consistente con distintos sistemas de bases de datos gracias a una capa de abstracción que permite que el mismo código funcione sobre motores como SQLite, MySQL o PostgreSQL. Esta integración se controla principalmente desde el archivo settings.py, donde se define el motor a utilizar mediante la clave ENGINE, junto con el nombre de la base de datos, usuario, contraseña, host y puerto. Por ejemplo, si el proyecto se ejecuta en desarrollo se suele usar SQLite porque no requiere instalación adicional y almacena los datos en un archivo local; para entornos productivos, Django se conecta sin problemas a MySQL o PostgreSQL simplemente cambiando esa configuración.

    - Describir cómo Django maneja las conexiones y operaciones con la base de datos a través de su ORM.

Internamente, Django gestiona las conexiones mediante un sistema de conexión persistente que abre y mantiene una sesión con la base de datos mientras haya actividad, cerrándola automáticamente cuando es necesario. Todas las operaciones se realizan a través del ORM (Object Relational Mapper), una capa que permite trabajar con los datos como objetos de Python en lugar de escribir sentencias SQL manualmente. Gracias a este ORM, crear, leer o actualizar registros se hace con métodos como objects.create(), objects.filter() o objects.get(), mientras que Django traduce esas instrucciones a SQL según el motor configurado.

    - Ejemplo: Explicar cómo configurar el archivo settings.py para conectar Django con una base de datos y cómo se gestionan las conexiones.

Un ejemplo típico en settings.py para una base MySQL incluye especificar el motor django.db.backends.mysql, el nombre de la base, las credenciales y el puerto; a partir de ello, Django se encarga de abrir la conexión, ejecutar las consultas y retornar los resultados como objetos Python. Esta estructura permite que el código sea portable,

seguro y mantenible, independientemente del motor de base de datos que se utilice en el proyecto. Ejemplo:

```
DATABASES = {
   'default': {
      'ENGINE': 'django.db.backends.mysql',
      'NAME': 'mi_base',
      'USER': 'root',
      'PASSWORD': '1234',
      'HOST': 'localhost',
      'PORT': '3306',
   }
}
```

2. **Implementar la capa de modelo de acceso a datos del aplicativo utilizando entidades no relacionadas para dar solución a una problemática.**
    - Crear modelos simples en Django sin relaciones entre ellos, permitiendo la creación de tablas independientes en la base de datos.
    - Ejemplo: Crear un modelo de Producto con campos básicos como nombre, precio y cantidad, sin relaciones con otras entidades.

Productos sin relaciones, muestro el código acá:

oductos > models.py > Cliente > __str__

```python
from django.db import models

# Create your models here.


from django.db import models

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    cantidad = models.IntegerField()

    def __str__(self):
        return self.nombre


class Cliente(models.Model):
    nombre = models.CharField(max_length=100)
    email = models.EmailField(unique=True)
    telefono = models.CharField(max_length=20)

    def __str__(self):
        return self.nombre


class Pedido(models.Model):
    numero_pedido = models.CharField(max_length=20)
    fecha = models.DateField()
    total = models.DecimalField(max_digits=10, decimal_places=2)

    def __str__(self):
        return f"Pedido {self.numero_pedido}"

```

3. Implementar la capa de modelo de acceso a datos del aplicativo utilizando entidades con relaciones uno a uno, uno a muchos y muchos a muchos para dar solución a una problemática.

Asigné las siguientes relaciones:

- Uno a muchos: el modelo Pedido tiene un campo cliente definido como ForeignKey hacia Cliente, lo que permite que un cliente tenga varios pedidos.
- Muchos a muchos: el modelo Pedido incluye el campo productos definido como ManyToManyField hacia Producto, de modo que un pedido puede contener varios productos y, a la vez, un producto puede estar en distintos pedidos.
- Uno a uno: creé el modelo PerfilCliente con un campo cliente definido como OneToOneField hacia Cliente, representando información adicional única para cada cliente.

Reconocer las aplicaciones preinstaladas con el motor Django distinguiendo su utilidad como apoyo al desarrollo.

En Django vienen varias aplicaciones preinstaladas dentro de django.contrib que ayudan al desarrollo sin que tengamos que programar todo desde cero:

- django.contrib.admin: entrega el panel de administración. Gracias a esta app podemos registrar nuestros modelos (por ejemplo, Producto y Pedido) en admin.py y gestionarlos con una interfaz web: crear, editar, eliminar y buscar registros.
- django.contrib.auth: se encarga de todo lo relacionado con usuarios, autenticación y permisos. La usamos para manejar el login/logout y para restringir el acceso al CRUD de productos y pedidos solo a usuarios autenticados o con ciertos permisos.
- django.contrib.sessions: permite guardar información de la sesión del usuario en el servidor (por ejemplo, quién está logueado). Esto se usa junto con auth para recordar al usuario entre peticiones.

Otras apps como django.contrib.messages o django.contrib.staticfiles ayudan a mostrar mensajes temporales (éxito/error) y a servir archivos estáticos (CSS, imágenes, JS).

Estos son los permisos actuales, los dejé así para establecer que cualquier puede revisar los productos disponibles, el historial de clientes solo lo puede ver el administrador y los pedidos los puede ver cada usuario logueado.

Permisos por CRUD

PRODUCTOS

Operación	Permiso	Decorador
Listar	✅ Sin permisos	Pública
Detalle	✅ Sin permisos	Pública
Crear	`productos.add_producto`	`@login_required` + `@permission_required`
Editar	`productos.change_producto`	`@login_required` + `@permission_required`
Eliminar	`productos.delete_producto`	`@login_required` + `@permission_required`

CLIENTES

Operación	Permiso	Decorador
Listar	⚠️ Solo Superadmin	`@login_required` + `@superuser_required`
Detalle	⚠️ Solo Superadmin	`@login_required` + `@superuser_required`
Crear	⚠️ Solo Superadmin	`@login_required` + `@superuser_required`
Editar	⚠️ Solo Superadmin	`@login_required` + `@superuser_required`
Eliminar	⚠️ Solo Superadmin	`@login_required` + `@superuser_required`

PEDIDOS

Operación	Permiso	Decorador
Listar	✅ Solo autenticado	`@login_required`
Detalle	✅ Solo autenticado	`@login_required`
Crear	`productos.add_pedido`	`@login_required` + `@permission_required`
Editar	`productos.change_pedido`	`@login_required` + `@permission_required`
Eliminar	`productos.delete_pedido`	`@login_required` + `@permission_required`

https://github.com/marulandia/M7_portafolio_marketplace