Proyecto tienda de piercing

Te explico todo el sistema de la tienda de piercing en detalle como una caja blanca, analizando cada componente y cómo interactúan:

**🏗️ ARQUITECTURA GENERAL DEL SISTEMA**

**Patrón MVC (Model-View-Controller) de Django**

Cliente (Browser) → URLs → Views → Models → Database

↓

Templates ← Context Data

**📁 ESTRUCTURA DE ARCHIVOS Y RESPONSABILIDADES**

tienda\_piercing/

├── manage.py # Script principal de Django

├── db.sqlite3 # Base de datos SQLite

├── media/ # Archivos subidos (imágenes)

│ └── productos/ # Imágenes de productos

├── static/ # Archivos estáticos globales

├── tienda\_piercing/ # Configuración del proyecto

│ ├── \_\_init\_\_.py

│ ├── settings.py # Configuración principal

│ ├── urls.py # URLs principales del proyecto

│ └── wsgi.py # Configuración para servidor

└── tienda/ # Aplicación principal

├── migrations/ # Migraciones de base de datos

├── static/tienda/ # Archivos estáticos de la app

├── templates/tienda/ # Templates HTML

├── \_\_init\_\_.py

├── admin.py # Configuración del panel admin

├── apps.py # Configuración de la aplicación

├── forms.py # Formularios Django

├── models.py # Modelos de datos

├── urls.py # URLs específicas de la app

└── views.py # Lógica de negocio

**🗄️ CAPA DE DATOS (Models)**

**Modelo de Base de Datos**

Diagrama de Base de Datos

Diagrama

**Análisis de Modelos (models.py)**

python

*# Cada modelo hereda de models.Model*

class Categoria(models.Model):

*# Campos de la tabla*

nombre = models.CharField(max\_length=100) *# VARCHAR(100)*

descripcion = models.TextField(blank=True) *# TEXT, opcional*

def \_\_str\_\_(self): *# Representación en string*

return self.nombre

class Meta: *# Metadatos*

verbose\_name\_plural = "Categorías" *# Nombre en plural*

**Características técnicas:**

* **ORM de Django**: Traduce objetos Python a SQL
* **Lazy Loading**: Las consultas se ejecutan cuando se accede a los datos
* **Relaciones**: ForeignKey crea relaciones 1-a-muchos
* **Validaciones**: Django valida automáticamente los tipos de datos

**🎯 CAPA DE LÓGICA (Views)**

**Flujo de Procesamiento de Requests**

Flujo de Procesamiento de Request

Diagrama

**Análisis de Views**

python

def inicio(request):

*# 1. Consulta a la base de datos*

productos\_destacados = Producto.objects.filter(activo=True)[:8] *# SELECT \* FROM producto WHERE activo=1 LIMIT 8*

categorias = Categoria.objects.all() *# SELECT \* FROM categoria*

*# 2. Preparar context (datos para el template)*

context = {

'productos': productos\_destacados,

'categorias': categorias

}

*# 3. Renderizar template con context*

return render(request, 'tienda/inicio.html', context) *# HTTP 200 + HTML*

**Tipos de Views:**

* **Function-based views**: Funciones que procesan requests
* **Class-based views**: Clases con métodos HTTP (GET, POST, etc.)
* **Decoradores**: @login\_required para control de acceso

**🌐 CAPA DE ENRUTAMIENTO (URLs)**

**Sistema de URLs Jerárquico**

python

*# tienda\_piercing/urls.py (URL raíz)*

urlpatterns = [

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

path('', include('tienda.urls')), *# Delega a tienda/urls.py*

]

*# tienda/urls.py (URLs específicas)*

urlpatterns = [

path('', views.inicio, name='inicio'), *# /*

path('productos/', views.lista\_productos, name='productos'), *# /productos/*

path('producto/<int:producto\_id>/', views.detalle\_producto, name='detalle'), *# /producto/123/*

]

**Características:**

* **Namespace**: app\_name = 'tienda' evita conflictos
* **Named URLs**: name='inicio' permite referencias en templates
* **Parámetros dinámicos**: <int:producto\_id> captura valores de URL

**🖼️ CAPA DE PRESENTACIÓN (Templates)**

**Sistema de Herencia de Templates**

Herencia de Templates

Diagrama

**Motor de Templates de Django**

html

*<!-- Sintaxis del motor de templates -->*

{% extends 'tienda/base.html' %} *<!-- Herencia -->*

{% load static %} *<!-- Cargar etiquetas -->*

{% block content %} *<!-- Definir bloque -->*

*<!-- Variables -->*

{{ producto.nombre }} *<!-- Output con escape HTML -->*

{{ producto.precio|floatformat:2 }} *<!-- Filtros -->*

*<!-- Lógica de control -->*

{% for producto in productos %}

*<!-- Contenido del loop -->*

{% empty %}

*<!-- Si no hay productos -->*

{% endfor %}

{% if user.is\_authenticated %}

*<!-- Contenido condicional -->*

{% endif %}

*<!-- URLs dinámicas -->*

<a href="{% url 'tienda:detalle' producto.id %}">

{% endblock %}

**📝 CAPA DE FORMULARIOS (Forms)**

**Procesamiento de Formularios**

Flujo de Procesamiento de Formularios

Diagrama

**Análisis de Forms**

python

class ProductoForm(forms.ModelForm):

class Meta:

model = Producto *# Modelo asociado*

fields = ['nombre', 'descripcion'] *# Campos a incluir*

widgets = { *# Widgets HTML personalizados*

'nombre': forms.TextInput(attrs={

'class': 'form-control', *# Clases CSS*

'placeholder': 'Nombre' *# Atributos HTML*

})

}

def clean\_precio(self): *# Validación personalizada*

precio = self.cleaned\_data['precio']

if precio <= 0:

raise forms.ValidationError("El precio debe ser positivo")

return precio

**🔐 SISTEMA DE AUTENTICACIÓN**

**Decoradores y Middleware**

python

@login\_required *# Decorador que verifica autenticación*

def crear\_producto(request):

*# Solo usuarios autenticados pueden acceder*

if not request.user.is\_staff: *# Verificación adicional de permisos*

return HttpResponseForbidden()

**Middleware de Django (orden de ejecución):**

1. SecurityMiddleware - Headers de seguridad
2. SessionMiddleware - Manejo de sesiones
3. CsrfViewMiddleware - Protección CSRF
4. AuthenticationMiddleware - Autenticación de usuarios
5. MessageMiddleware - Sistema de mensajes

**📊 FLUJO DE DATOS COMPLETO**

**Crear un Producto (Ejemplo Completo)**

Flujo Completo: Crear Producto

Diagrama

**⚙️ CONFIGURACIÓN DEL SISTEMA (settings.py)**

**Análisis de Configuraciones Críticas**

python

*# Base de datos*

DATABASES = {

'default': {

'ENGINE': 'django.db.backends.sqlite3', *# Motor de BD*

'NAME': BASE\_DIR / 'db.sqlite3', *# Archivo de BD*

}

}

*# Archivos estáticos y media*

STATIC\_URL = '/static/' *# URL para archivos CSS/JS*

STATICFILES\_DIRS = [BASE\_DIR / 'static'] *# Directorio de desarrollo*

STATIC\_ROOT = BASE\_DIR / 'staticfiles' *# Directorio de producción*

MEDIA\_URL = '/media/' *# URL para archivos subidos*

MEDIA\_ROOT = BASE\_DIR / 'media' *# Directorio de archivos subidos*

*# Templates*

TEMPLATES = [{

'BACKEND': 'django.template.backends.django.DjangoTemplates',

'DIRS': [], *# Directorios globales*

'APP\_DIRS': True, *# Buscar en apps*

'OPTIONS': {

'context\_processors': [ *# Variables globales en templates*

'django.contrib.auth.context\_processors.auth', *# user, perms*

'django.contrib.messages.context\_processors.messages', *# messages*

],

},

}]

**🔄 CICLO DE VIDA DE UN REQUEST**

**Procesamiento Interno Detallado**

python

*# 1. RECEPCIÓN DEL REQUEST*

*# Django recibe: GET /productos/?categoria=1*

*# 2. URL RESOLUTION*

*# tienda\_piercing/urls.py → tienda/urls.py → views.lista\_productos*

*# 3. MIDDLEWARE PROCESSING (request phase)*

*# - SessionMiddleware: Carga sesión del usuario*

*# - AuthenticationMiddleware: Identifica usuario*

*# - CsrfViewMiddleware: Prepara token CSRF*

*# 4. VIEW EXECUTION*

def lista\_productos(request):

productos = Producto.objects.filter(activo=True) *# SQL: SELECT \* FROM producto WHERE activo=1*

categoria\_id = request.GET.get('categoria') *# Obtiene parámetro GET*

if categoria\_id:

productos = productos.filter(categoria\_id=categoria\_id) *# SQL: AND categoria\_id=1*

context = {'productos': productos}

return render(request, 'tienda/productos.html', context)

*# 5. TEMPLATE RENDERING*

*# - Carga template tienda/productos.html*

*# - Resuelve herencia de base.html*

*# - Evalúa {% for producto in productos %}*

*# - Ejecuta consulta SQL lazy (si no se ejecutó antes)*

*# - Genera HTML final*

*# 6. MIDDLEWARE PROCESSING (response phase)*

*# - Procesa mensajes*

*# - Agrega headers de seguridad*

*# 7. HTTP RESPONSE*

*# Status: 200 OK*

*# Content-Type: text/html*

*# HTML renderizado*

**📈 PERFORMANCE Y OPTIMIZACIÓN**

**Query Optimization**

python

*# ❌ N+1 Problem*

productos = Producto.objects.all()

for producto in productos:

print(producto.categoria.nombre) *# Una query por producto*

*# ✅ Optimizado con select\_related*

productos = Producto.objects.select\_related('categoria').all()

for producto in productos:

print(producto.categoria.nombre) *# Una sola query con JOIN*

*# ✅ Optimizado con prefetch\_related para relaciones many-to-many*

pedidos = Pedido.objects.prefetch\_related('detallepedido\_set\_\_producto').all()

**Caching Strategy**

python

*# En views.py*

from django.views.decorators.cache import cache\_page

@cache\_page(60 \* 15) *# Cache por 15 minutos*

def lista\_productos(request):

*# Vista cacheada*

pass

*# En templates*

{% load cache %}

{% cache 500 productos\_destacados %}

<!-- Contenido cacheado -->

{% endcache %}

**🛡️ SEGURIDAD**

**Mecanismos de Protección**

python

*# CSRF Protection*

*# Cada formulario incluye: {% csrf\_token %}*

*# Django valida el token en cada POST*

*# SQL Injection Prevention*

*# El ORM escapa automáticamente las consultas:*

Producto.objects.filter(nombre=user\_input) *# Seguro*

*# Se convierte en: SELECT \* FROM producto WHERE nombre = %s*

*# XSS Prevention*

*# Las variables en templates se escapan automáticamente:*

{{ producto.descripcion }} *# HTML escapado*

{{ producto.descripcion|safe }} *# Sin escapar (peligroso)*

*# Authentication & Authorization*

@login\_required

def vista\_protegida(request):

if not request.user.is\_staff:

raise PermissionDenied()

**🔍 DEBUGGING Y LOGGING**

**Herramientas de Desarrollo**

python

*# Debug Toolbar (settings.py)*

if DEBUG:

INSTALLED\_APPS += ['debug\_toolbar']

MIDDLEWARE += ['debug\_toolbar.middleware.DebugToolbarMiddleware']

*# Logging*

LOGGING = {

'version': 1,

'handlers': {

'file': {

'level': 'DEBUG',

'class': 'logging.FileHandler',

'filename': 'django.log',

},

},

'loggers': {

'django.db.backends': {

'handlers': ['file'],

'level': 'DEBUG',

},

},

}

**📦 DEPLOYMENT CONSIDERATIONS**

**Diferencias Desarrollo vs Producción**

python

*# Desarrollo*

DEBUG = True

ALLOWED\_HOSTS = []

DATABASE = SQLite

*# Producción*

DEBUG = False

ALLOWED\_HOSTS = ['tu-dominio.com']

DATABASE = PostgreSQL/MySQL

STATIC\_ROOT = '/var/www/static/'

MEDIA\_ROOT = '/var/www/media/'

*# Variables de entorno*

import os

SECRET\_KEY = os.environ.get('SECRET\_KEY')

DATABASE\_URL = os.environ.get('DATABASE\_URL')

Este análisis te muestra cómo cada componente del sistema interactúa internamente, desde la recepción de un request HTTP hasta la generación de la respuesta final, incluyendo todos los aspectos de seguridad, performance y arquitectura que Django maneja automáticamente.

Use clause

