Tienda de Electronica
Sistema de Ventas en Linea con Django

Juanjo

November 13, 2025

Contents

Introduccion

Este reporte documenta el desarrollo de un sistema de tienda en linea de electronica basado en Django 5.2.8. El proyecto implementa un conjunto completo de funcionalidades para gestionar productos, ventas, usuarios con control de acceso y reportes analiticos.

Objetivo General

Desarrollar una aplicacion web que permita:

- Gestionar un catalogo de productos electronicos
- Implementar un carrito de compras funcional
- Registrar transacciones de ventas
- Generar reportes de ventas en formato web y PDF
- Controlar acceso mediante autenticacion y permisos
- Servir como base para despliegue en entornos de produccion

Alcance

El sistema incluye:

- **Frontend**: Interfaz web responsiva con Bootstrap 5
- **Backend**: API REST implicita mediante vistas basadas en funciones
- **Base de datos**: SQLite (desarrollo) / PostgreSQL (produccion)
- **Reportes**: Generacion de PDF con ReportLab
- **Autenticacion**: Sistema de usuarios de Django con permisos personalizados

Requisitos Tecnicos

Requisitos Funcionales

1. **Gestion de Productos**
    - Crear, leer, actualizar y eliminar (CRUD) productos
    - Organizar productos por categorias
    - Mostrar informacion de: nombre, precio, stock
2. **Carrito de Compras**
    - Agregar productos al carrito
    - Modificar cantidades
    - Eliminar productos
    - Procesar compras (crear registros de venta)
3. **Ventas y Transacciones**
    - Registrar cada compra con detalles (producto, cantidad, fecha)
    - Reducir automaticamente el stock al procesar compra
    - Asociar vendedor a cada transaccion
4. **Reportes**
    - Reporte de ventas totales por periodo
    - Desglose por categoria de producto
    - Analisis por producto individual
    - Exportar a PDF
    - Filtros por fecha, categoria, producto
5. **Autenticacion y Control de Acceso**
    - Login/logout de usuarios
    - Registro de nuevos usuarios
    - Asignacion automatica de permisos
    - Acceso restringido a reportes y modulo de ventas

Requisitos No Funcionales

- **Seguridad**: HTTPS en produccion, proteccion CSRF, cookies seguras
- **Rendimiento**: Consultas optimizadas con select_related
- **Disponibilidad**: Despliegue en Render con base de datos PostgreSQL
- **Escalabilidad**: Estructura preparada para multiples instancias
- **Mantenibilidad**: Codigo modular, documentacion clara

Arquitectura del Sistema

Arquitectura General

El sistema sigue el patron **MTV (Model-Template-View)** de Django:

Componentes MTV

```
Model    : Modelos de datos (Producto, Venta, CarritoItem)
Template : Plantillas HTML con Bootstrap 5
View     : Logica de negocio (vistas basadas en funciones)
```

Modelos de Datos

```
class Categoria(models.Model):
    nombre = models.CharField(max_length=50)
    descripcion = models.TextField(blank=True)

class Producto(models.Model):
    nombre = models.CharField(max_length=100)
    categoria = models.ForeignKey(Categoria, on_delete=models.CASCADE)
    precio = models.DecimalField(max_digits=10, decimal_places=2)
    stock = models.PositiveIntegerField(default=0)

class Venta(models.Model):
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
    cantidad = models.PositiveIntegerField()
    fecha = models.DateField(auto_now_add=True)
    vendedor = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)

    def total(self):
        return self.cantidad * self.producto.precio

class CarritoItem(models.Model):
    usuario = models.ForeignKey(User, on_delete=models.CASCADE)
    producto = models.ForeignKey(Producto, on_delete=models.CASCADE)
    cantidad = models.PositiveIntegerField(default=1)
```

```
fecha_agregado = models.DateTimeField(auto_now_add=True)
```

Flujo de Datos

1. Usuario accede a /productos/ (catalogo)
2. Elige productos y los agrega al carrito
3. Revisa carrito en /carrito/
4. Procesa compra: se crean registros de Venta, se reduce stock
5. Sistema redirige a pagina de exito
6. Usuario accede a /reportes/ventas/ para analisis

Tecnologias Utilizadas

Django 5.2.8	- Framework web
Python 3.13	- Lenguaje de programacion
Bootstrap 5	- Interfaz responsiva
SQLite/PostgreSQL	- Base de datos
ReportLab 4.4.4	- Generacion de PDF
WhiteNoise 6.5.0	- Servir archivos estaticos
Gunicorn 21.2.0	- Servidor WSGI
Render	- Plataforma de despliegue

Implementacion

Estructura del Proyecto

```
proyecto2.1/
  proyecto_dos/            (Configuracion Django)
    settings.py
    urls.py
    wsgi.py
    asgi.py
  tienda/                  (Aplicacion principal)
    models.py
    views.py
    urls.py
    admin.py
    templates/
      base.html
      home.html
      catalogo.html
      carrito.html
      login.html
      signup.html
      reportes/
        reporte_ventas.html
        reporte_categorias.html
        reporte_productos.html
    static/
      tienda/
        css/
        js/
        img/
    migrations/
  manage.py
  requirements.txt
  build.sh
  Procfile
  runtime.txt
```

```
  .gitignore
  README.md
  reportes/
    main.tex
```

Rutas de la Aplicacion

URL	Vista	Descripcion
/	home	Dashboard principal
/signup/	signup	Registro de usuarios
/login/	login	Autenticacion
/productos/	catalogo	Catalogo de productos
/carrito/	carrito	Ver carrito
/carrito/agregar/	agregar_carrito	POST: agregar item
/carrito/procesar/	procesar_compra	POST: procesar compra
/ventas/	ventas	Listado de ventas
/reportes/ventas/	reporte_ventas	Reporte de ventas
/reportes/categorias/	reporte_por_categoria	Reporte por categoria
/reportes/productos/	reporte_por_producto	Reporte por producto

Funcionalidades Clave

1. Carrito de Compras

El carrito se implementa con el modelo CarritoItem:

- Relacion ManyToOne con User
- Constraint unique_together para evitar duplicados
- Metodo subtotal() para calcular costo de cada item

2. Procesamiento de Compra

Al procesar compra:

1. Se itera sobre los items del carrito
2. Se crea un registro Venta por cada item
3. Se reduce el stock del producto
4. Se vacia el carrito
5. Se redirige a pagina de exito

3. Generacion de Reportes PDF

Usa ReportLab para crear tablas dinamicas:

- Resumen general (total ventas, ingresos, productos unicos)
- Desglose por categoria con promedios
- Manejo robusto de errores y valores ausentes

4. Autenticacion y Permisos

El sistema implementa decoradores para proteger vistas:

```
@login_required
@permission_required('tienda.view_venta', raise_exception=True)
def ventas(request):
    # Solo usuarios autenticados con permiso 'view_venta'
    pass

def signup(request):
    # Al registrarse, se asigna automaticamente 'view_venta'
    user.user_permissions.add(perm)
```

Consideraciones de Seguridad

Medidas de Seguridad Implementadas:

1. **HTTPS Obligatorio**: SECURE_SSL_REDIRECT = True en produccion
2. **Cookies Seguras**: SESSION_COOKIE_SECURE, CSRF_COOKIE_SECURE
3. **HSTS**: Cabecera Strict-Transport-Security (configurable)
4. **CSRF Protection**: Validacion de tokens en formularios
5. **XFrame Options**: Proteccion contra clickjacking
6. **Proxy Headers**: X-Forwarded-Proto, X-Forwarded-Host
7. **SECRET_KEY**: Requerida como variable de entorno en produccion
8. **DEBUG = False**: En produccion no expone informacion sensible

Despliegue en Render

Pasos de Despliegue

1. Crear repositorio en GitHub
2. Crear servicio web en Render vinculado al repo
3. Configurar variables de entorno:
    - SECRET_KEY
    - DATABASE_URL
    - ALLOWED_HOSTS
    - CSRF_TRUSTED_ORIGINS
4. Build command: bash build.sh
5. Start command: gunicorn proyecto_dos.wsgi:application

Variables de Entorno Recomendadas

```
SECRET_KEY=tu_clave_super_secreta
DATABASE_URL=postgres://user:pass@host:5432/dbname
ALLOWED_HOSTS=mi-app.onrender.com
CSRF_TRUSTED_ORIGINS=https://mi-app.onrender.com
DJANGO_DEBUG=False
SECURE_HSTS_SECONDS=31536000
SECURE_HSTS_INCLUDE_SUBDOMAINS=True
SECURE_HSTS_PRELOAD=True
```

Conclusion

Se ha desarrollado exitosamente un sistema completo de tienda en linea usando Django. El proyecto:

- Cumple con todos los requisitos funcionales
- Implementa seguridad en multiples niveles
- Esta preparado para despliegue en produccion
- Mantiene codigo modular y escalable
- Incluye documentacion y configuracion completas

El sistema es listo para ser desplegado en Render y escalar segun sea necesario. Se recomienda:

1. Realizar pruebas de carga antes de despliegue en produccion
2. Configurar backups automaticos de base de datos
3. Monitorear logs y errores en Render
4. Implementar tests unitarios en el futuro
5. Considerar cache (Redis) para mejorar rendimiento

Proyecto completado: November 13, 2025