

Documentar el proceso de implantación de software siguiendo estándares de calidad

Luis Fernando Escobar Llanten

Servicio Nacional de Aprendizaje

GA10- 220501097-AA10-EV01 Elabora documentos técnicos y de usuario del software

Doris Gonzales Martinez

31 de julio de 2024

## **PRESENTACIÓN**

El siguiente manual se ha desarrollado con la finalidad de proporcionar la información necesaria para realizar el mantenimiento, la instalación y la exploración de la plataforma de comercio electrónico, el cual permite a una ferretería vender sus productos en línea. El manual ofrece la información necesaria sobre cómo está realizado el software para que cualquier desarrollador con conocimientos en el framework Django pueda editar el software de manera apropiada, comprendiendo la estructura del desarrollo del aplicativo.

## **RESUMEN**

El manual detalla los aspectos técnicos e informáticos del software con la finalidad de explicar la estructura del aplicativo al personal que desee administrarlo, editarlo o configurarlo. La guía se encuentra dividida en las herramientas utilizadas para la creación del software, con una breve explicación paso a paso de las funcionalidades y requerimientos del sistema, así como sugerencias para el uso adecuado del sistema de información.

## **FINALIDAD DEL MANUAL**

La finalidad de este manual técnico es instruir a la persona que quiera administrar, editar o configurar el software de la Tienda Virtual utilizando las herramientas adecuadas y siguiendo las mejores prácticas de desarrollo en Django.

## INTRODUCCIÓN

Este manual se ha creado para detallar los aspectos técnicos del software de la Tienda Virtual, permitiendo a los desarrolladores administrarlo, editarlo o configurarlo de manera adecuada. El documento está dividido en las siguientes secciones:

1. **Aspectos técnicos:** explicación de los componentes del aplicativo desde un punto de vista teórico.
2. **Diagramas de modelamiento:** diagramas e ilustraciones del funcionamiento del aplicativo.
3. **Aspecto técnico del desarrollo del sistema:** Instrucciones sobre los componentes del aplicativo, configuración de la base de datos, estructura del desarrollo y recomendaciones para el uso adecuado del sistema.
4. Actividades de mantenimiento

## 1. ASPECTOS TÉCNICOS

### 1.1 Arquitectura del Software

La aplicación Construmole tiene una arquitectura monolítica en la que todos los componentes del sistema se encuentran en un solo código base. Esto incluye tanto el frontend como el backend, así como la gestión de datos y la lógica de negocio.

### 1.2 Módulos Principales

- **Aplicación Root:** Gestiona la configuración general del proyecto.
- **Aplicación store:** Maneja la lógica de productos, categorías, carritos de compra y órdenes.
- **Aplicación accounts:** Gestiona la autenticación de usuarios y el perfil de usuarios

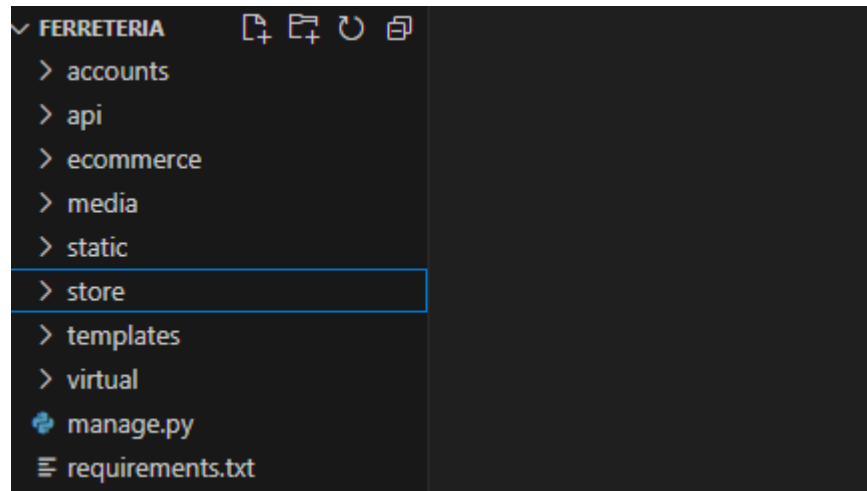
### 1.3 Lenguaje de Programación y Framework

La aplicación está desarrollada en Python utilizando el framework Django. Este framework sigue el patrón MVC (Model-View-Controller), en el que los Modelos manejan la lógica de datos, las Vistas gestionan la presentación y los Controladores dirigen las interacciones de usuario.

### 1.4 Descripción Jerárquica

La aplicación se organiza jerárquicamente en directorios que agrupan funcionalidades relacionadas.

### Estructura de Directorios:

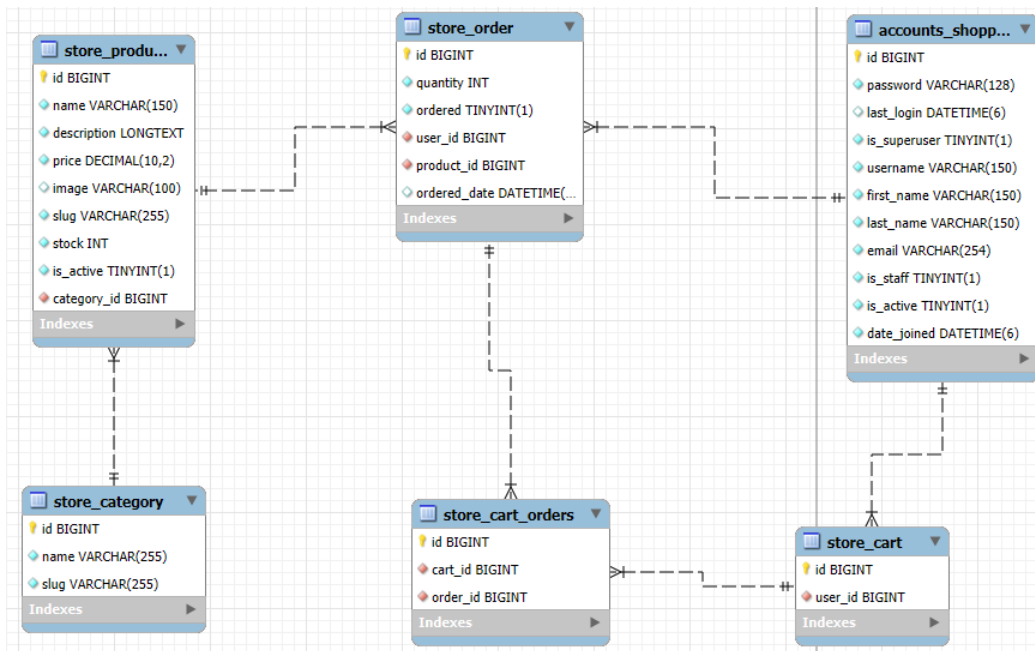


- **ecommerce:** Contiene los archivos de configuración del proyecto.
- **accounts:** Contiene los modelos y vistas relacionados con la gestión de usuarios.
- **store:** Contiene los modelos y vistas relacionados con productos, órdenes y el carrito de compras.
- **templates:** Carpeta que contiene las plantillas HTML utilizadas en las vistas.

## 2. DIAGRAMAS DE MODELAMIENTO

El diagrama relacional incluye las siguientes entidades:

- **accounts\_shopper:** Almacena la información del usuario.
- **product:** Almacena detalles sobre los productos.
- **category:** Clasifica productos.
- **order:** Guarda información sobre pedidos.
- **cart:** Mantiene los productos seleccionados antes de confirmar una compra.
- **cart\_orders:** Tabla intermedia entre las tablas cart y order.



## 2.1 Diagramas de casos de uso

En el diagrama de casos de uso se detallan los roles y actividades principales en relación con el aplicativo. Los actores principales son:

- **Administrador:** Gestiona el inventario y administra el sitio.
- **Usuario (Cliente):** Navega y realiza compras en el sitio.

**Casos de uso administrador:** CU01, CU02, CU04, CU05, CU06, CU09, CU10, CU11



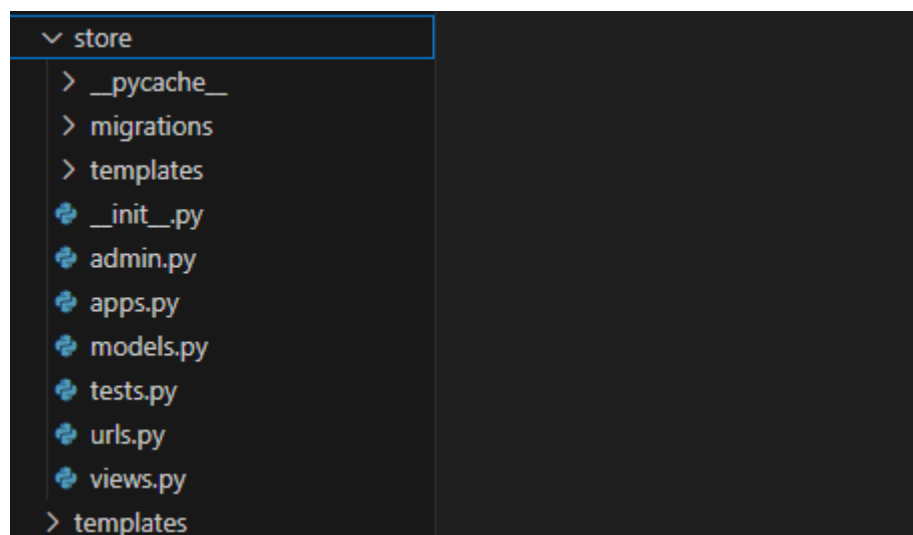
**Casos de uso cliente:** CU03, CU04, CU05, CU06, CU07, CU08, CU11, CU12.



### 3. ASPECTO TÉCNICO DEL DESARROLLO DEL SISTEMA

#### 3.1 Descripción Individual de los Módulos

##### a) Aplicación store:



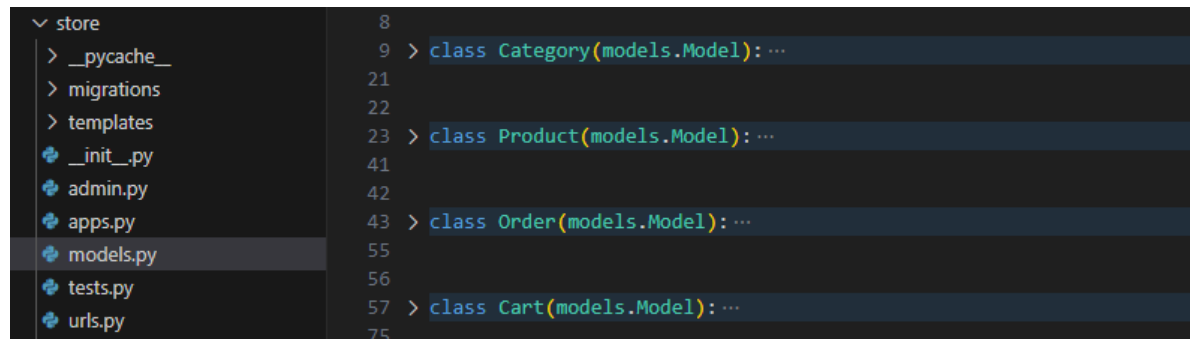
**Descripción General y Propósito:** Gestiona la lógica del negocio relacionada con los productos, el carrito de compras y las órdenes de compra.

**Responsabilidad y Restricciones:** Este módulo maneja la lógica de inventario, la actualización de órdenes, y la interacción del usuario con los productos. No gestiona la autenticación de usuarios, que es manejada por el módulo accounts.

**Dependencias:** Depende de los modelos de Django y de la base de datos para almacenar y recuperar información de productos y órdenes.

**Implementación:** Implementado en archivos dentro de la carpeta store, principalmente en los archivos models.py, views.py, urls.py.

**Modelos:** Define las estructuras de datos de la aplicación.

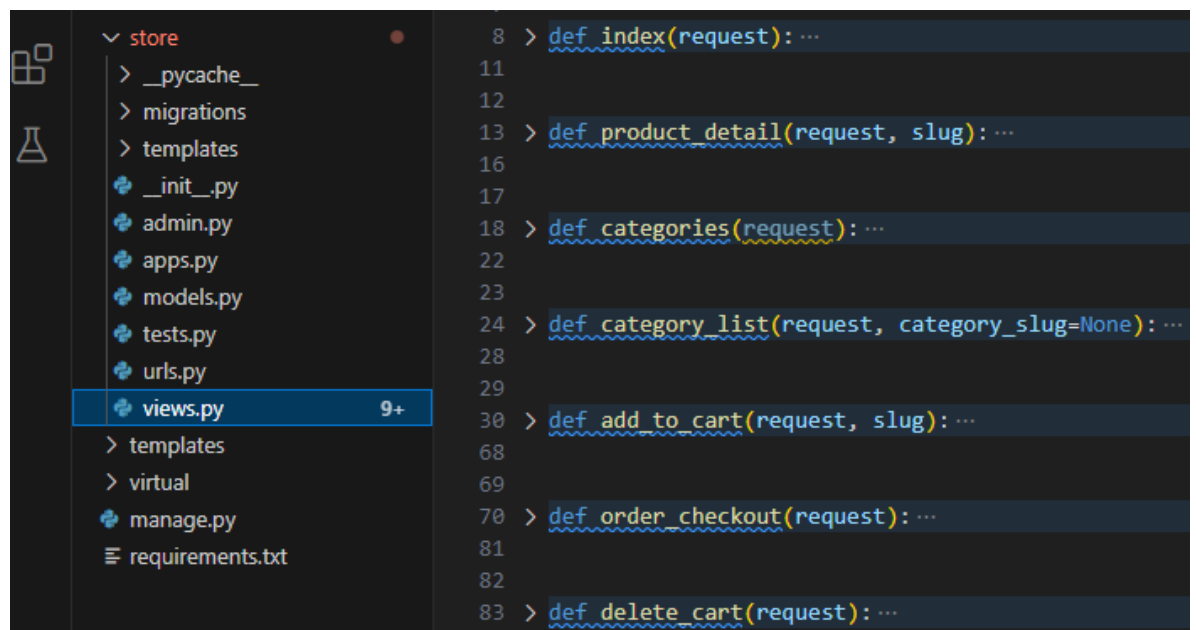


The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the 'store' directory with files: \_\_pycache\_\_, migrations, templates, \_\_init\_\_.py, admin.py, apps.py, models.py (selected), tests.py, and urls.py. The code editor shows the contents of models.py, which defines four Django models: Category, Product, Order, and Cart, all inheriting from models.Model.

```
8
9 > class Category(models.Model): ...
21
22
23 > class Product(models.Model): ...
41
42
43 > class Order(models.Model): ...
55
56
57 > class Cart(models.Model): ...
75
```

Los modelos de store representan un producto, categoría, order y carrito de compras.

**Vistas:** Contiene la lógica de la aplicación.



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows the 'store' directory with files: \_\_pycache\_\_, migrations, templates, \_\_init\_\_.py, admin.py, apps.py, models.py, tests.py, urls.py, views.py (selected), templates, virtual, manage.py, and requirements.txt. The code editor shows the contents of views.py, which defines several Django views: index, product\_detail, categories, category\_list, add\_to\_cart, order\_checkout, and delete\_cart.

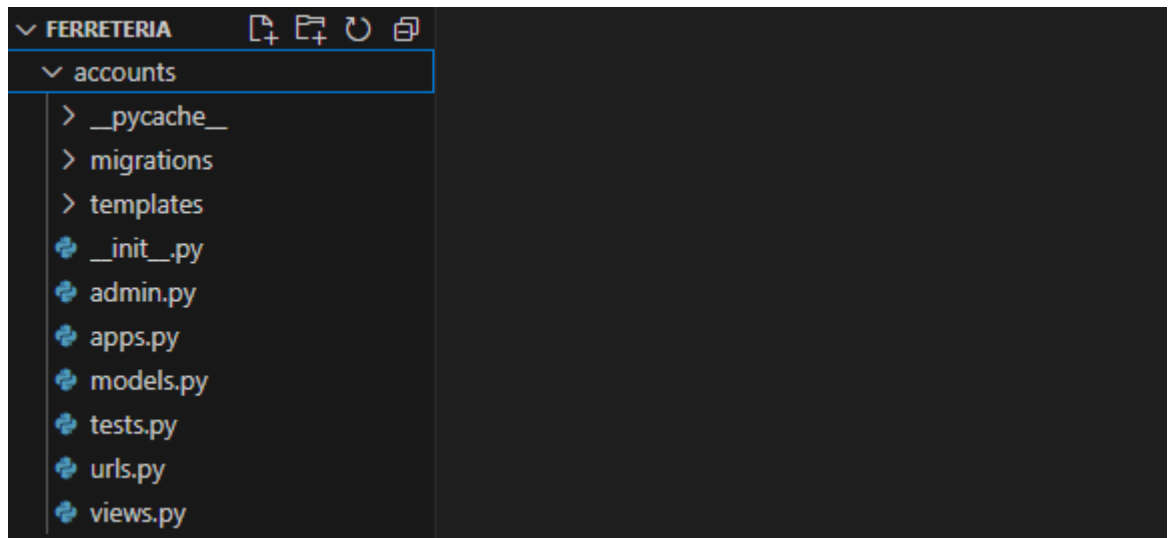
```
8 > def index(request): ...
11
12
13 > def product_detail(request, slug): ...
16
17
18 > def categories(request): ...
22
23
24 > def category_list(request, category_slug=None): ...
28
29
30 > def add_to_cart(request, slug): ...
68
69
70 > def order_checkout(request): ...
81
82
83 > def delete_cart(request): ...
```

Las funciones presentes en este archivo permiten la visualización de productos, la gestión del carrito de compras, y la compra de productos.

**Urls:** Definen las rutas que enlazan las vistas con las correspondientes acciones del usuario.

**Templates:** Archivos HTML que definen la interfaz de usuario.

**b) Aplicación accounts:**



**Descripción General y Propósito:** Gestiona la autenticación y la administración de usuarios.

**Responsabilidad y Restricciones:** Se encarga del registro, inicio de sesión, y eliminación de cuentas. No gestiona la lógica del negocio relacionada con los productos.

**Dependencias:** Utiliza el modelo User de Django y depende de la autenticación y gestión de sesiones proporcionada por Django.

**Implementación:** Implementado en archivos dentro de la carpeta accounts, como models.py, views.py, urls.py.

**Modelos:** Define las estructuras de datos de la aplicación.



```
models.py X
accounts > models.py > ...
4 class Shopper(AbstractUser):
5     pass
6
```

Este modelo extiende del AbstractUser de Django, lo que significa que hereda todos los campos y funcionalidades de un usuario estándar en Django.

**Vistas:** Contiene la lógica de la aplicación.

```
views.py X
accounts > views.py > ...
1 from django.contrib import messages
2 from django.shortcuts import redirect, render
3
4 from django.contrib.auth import get_user_model, login, logout, authenticate
5
6 # Crear usuario con la información recuperada del formulario
7 # get_user_model >> Recuperar un objeto que coincida con nuestro modelo de usuario (Shopper)
8 User = get_user_model()
9
10 > def signup(request): ...
29 |
30 # No necesitamos pasarle la info al request de que usuario esta conectado, porque ya lo tiene
31 > def logout_user(request): ...
34
35 > def login_user(request): ...
53
54 > def delete_account(request): ...
```

Las funciones presentes en este archivo permiten manejar el registro de usuarios, el inicio de sesión, el cierre de sesión, y la eliminación de las cuentas de usuario.

**Urls:** Definen las rutas que enlazan las vistas de autenticación y gestión de usuarios.

```
urls.py ×
accounts > urls.py > ...
1  from django.urls import path
2  from store.views import index
3  from accounts.views import delete_account, login_user, signup, logout_user
4
5  urlpatterns = [
6      # path('', index, name='index'),
7      path('signup/', signup, name='signup'),
8      path('logout/', logout_user, name='logout'),
9      path('login/', login_user, name='login'),
10     path('profile/', delete_account, name='profile'),
11 ]
```

**Templates:** Archivos HTML que definen la interfaz de usuario.

### c) Aplicación Raíz (root):

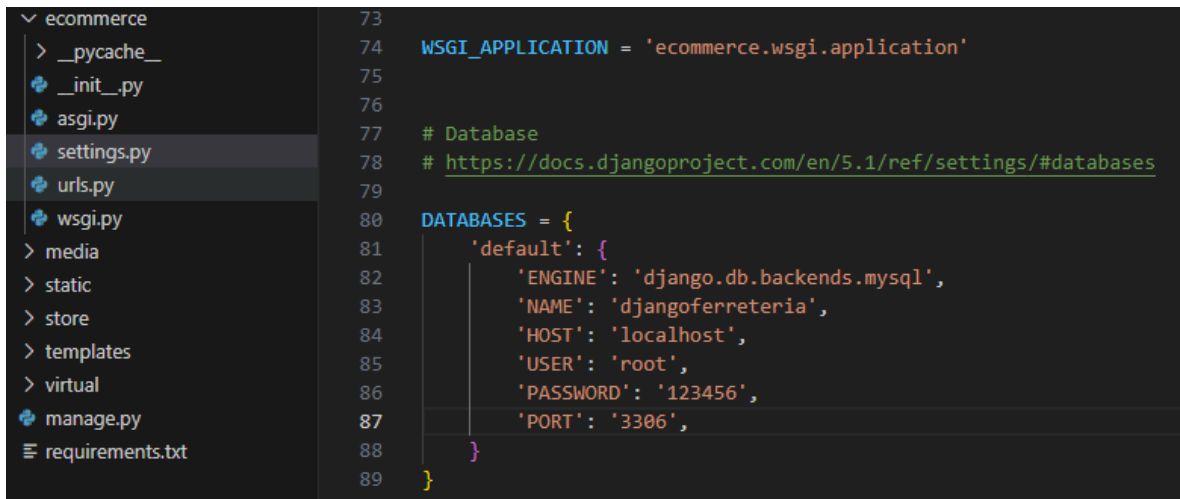
**Configuración del Proyecto:** Archivos como settings.py, urls.py, wsgi.py, y asgi.py, que manejan la configuración global del proyecto, incluyendo bases de datos, middleware, y rutas principales.

**Descripción General y Propósito:** Configura la aplicación general, incluyendo la base de datos, rutas, y el middleware.

**Responsabilidad y Restricciones:** Configura el entorno de ejecución para las aplicaciones store y accounts. No maneja directamente la lógica del negocio ni la gestión de usuarios.

**Dependencias:** Depende del framework Django para gestionar la configuración global.

**Implementación:** Implementado en archivos como settings.py, urls.py, y otros archivos de configuración.



```
73
74 WSGI_APPLICATION = 'ecommerce.wsgi.application'
75
76
77 # Database
78 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases
79
80 DATABASES = {
81     'default': {
82         'ENGINE': 'django.db.backends.mysql',
83         'NAME': 'djangoferreteria',
84         'HOST': 'localhost',
85         'USER': 'root',
86         'PASSWORD': '123456',
87         'PORT': '3306',
88     }
89 }
```

## 3.2 Preparando el ambiente para la instalación local del sistema

Para hacer una instalación local, asegúrese de cumplir con todos los requisitos técnicos que se mencionan a continuación:

### Requerimientos Técnicos de Instalación

#### Requisitos del Sistema (Hardware)

- **Sistema Operativo:** Windows 10, macOS 10.15 o superior, o distribuciones modernas de Linux.
- **Procesador:** Hexa-core 3.6 GHz o superior.
- **Memoria:** 6 GB de RAM (8 GB recomendados).
- **Disco Duro:** 800 MB de espacio libre para la instalación del software, más espacio adicional para la base de datos.
- **Conexión a Internet:** Necesaria para la descarga de dependencias y actualizaciones.

#### Requisitos de Red

- **Conexión Estable a Internet:** Necesaria para descargar dependencias y conectarse a servicios externos.

#### Requisitos del Sistema (Software)

- **Sistema Operativo:** Asegúrese de que su sistema operativo esté actualizado.
- **Python 3.12.2:** Necesario para ejecutar el código de Django.
- **PIP:** El gestor de paquetes para Python, necesario para instalar las dependencias del proyecto.

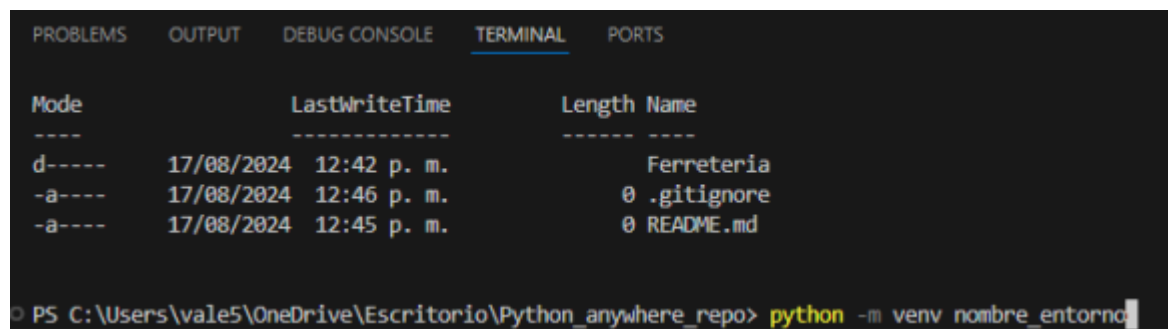
## Preparación (Antes de la Instalación)

Descargue el código fuente de la aplicación desde el repositorio oficial en GitHub.

## Creación de un Entorno Virtual ¿Por qué se crea un entorno virtual?

Un entorno virtual es una herramienta que permite aislar las dependencias del proyecto. Esto significa que las librerías instaladas para este proyecto no interferirán con las de otros proyectos en el mismo sistema, evitando conflictos de versiones y garantizando que el proyecto funcione correctamente.

**Creación del Entorno Virtual:** En la carpeta principal del proyecto escribimos el comando: `python -m venv nombre_entorno`



Mode	LastWriteTime	Length	Name
d-----	17/08/2024 12:42 p. m.		Ferreteria
-a----	17/08/2024 12:46 p. m.	0	.gitignore
-a----	17/08/2024 12:45 p. m.	0	README.md

```
PS C:\Users\vale5\OneDrive\Escritorio\Python_anywhere_repo> python -m venv nombre_entorno
```

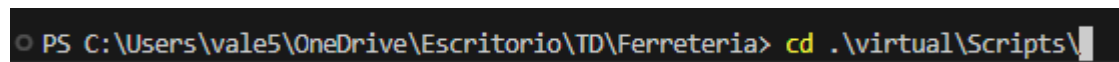
## Activation del entorno virtual

### Windows:

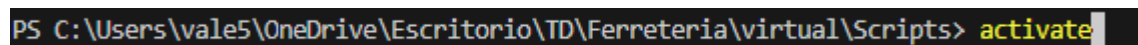
`nombre_entorno\Scripts\activate`

### macOS/Linux:

`source nombre_entorno/bin/activate`

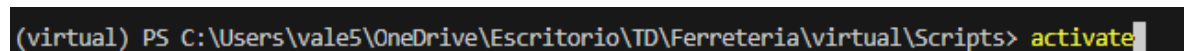


```
PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria> cd .\virtual\Scripts\
```



```
PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria\virtual\Scripts> activate
```

Si el entorno virtual se ejecuta correctamente veremos algo similar a esto:



```
(virtual) PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria\virtual\Scripts> activate
```

## Instalación del Sistema

Uso del archivo “requirements.txt”

¿Por qué descargar las librerías del archivo requirements.txt?

El archivo requirements.txt contiene una lista de todas las dependencias necesarias para ejecutar el proyecto. Descargar e instalar estas librerías asegura que todas las funcionalidades del sistema estén disponibles y funcionen como se espera.

### 1.1 Instalación de dependencias

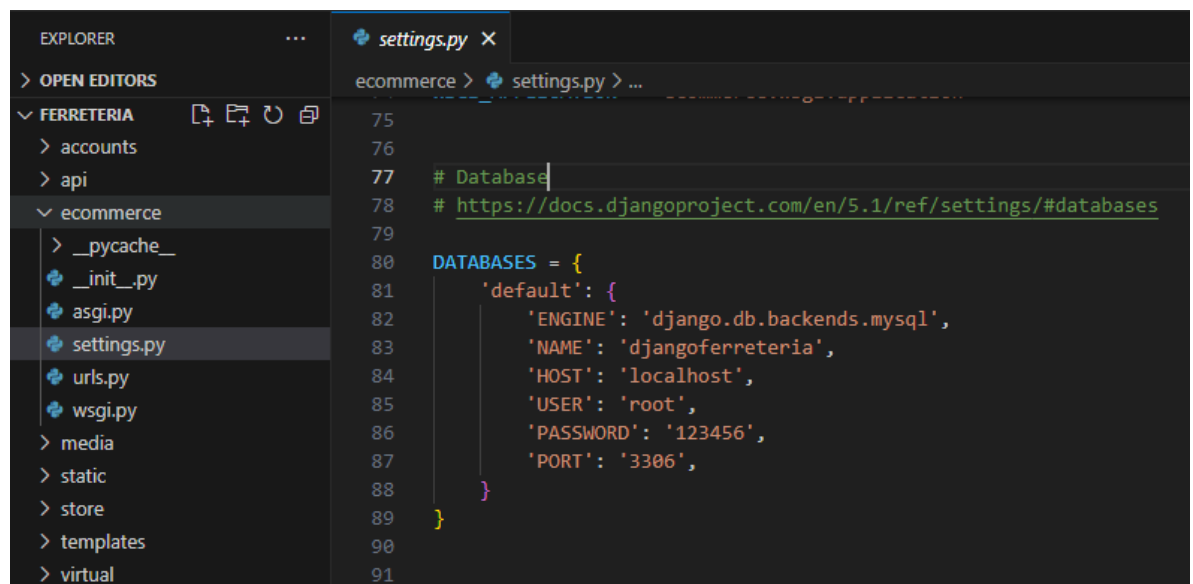
Nos situamos en la misma carpeta donde se encuentre el archivo “requirements.txt” y ejecutamos el comando:

```
pip install -r requirements.txt
```

```
PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria> pip install -r "requirements.txt"
```

### 1.2 Configuración de la Base de Datos MySQL

Asegúrese de que el archivo settings.py de Django esté configurado para utilizar MySQL como base de datos, especificando los detalles de la conexión.



```
75
76
77 # Database
78 # https://docs.djangoproject.com/en/5.1/ref/settings/#databases
79
80 DATABASES = {
81     'default': {
82         'ENGINE': 'django.db.backends.mysql',
83         'NAME': 'djangoferreteria',
84         'HOST': 'localhost',
85         'USER': 'root',
86         'PASSWORD': '123456',
87         'PORT': '3306',
88     }
89 }
90
91
```

1.3 **Crear Superusuario:** Ejecutar `python manage.py createsuperuser` para crear un usuario administrador en Django.

1.4 **Migración de Datos:** En la carpeta de nuestro proyecto donde se encuentra el archivo manage.py, ejecutamos el comando: **python manage.py makemigrations**

```
(virtual) PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria> python manage.py makemigrations
```

Django creara un archivo de migración. Terminado ese proceso ejecutamos el comando: **python manage.py migrate**

```
(virtual) PS C:\Users\vale5\OneDrive\Escritorio\TD\Ferreteria> python manage.py migrate
```

**1.5 Iniciar Servidor Local:** Ejecutar `python manage.py runserver` para iniciar el servidor local.

**1.6 Administración de Django:** Accede a la administración de Django en `http://127.0.0.1:8000/admin` utilizando el superusuario creado. Desde aquí, se pueden agregar, modificar o eliminar datos de los modelos del proyecto.

## 4. ACTIVIDADES DE MANTENIMIENTO

**Actualizaciones de Software** Las actualizaciones de software son fundamentales para mantener la seguridad, estabilidad, y funcionalidad del sistema. Estas actualizaciones pueden incluir parches de seguridad, mejoras en el rendimiento, o la incorporación de nuevas funcionalidades solicitadas por el cliente o necesarias por cambios en la tecnología.

### Actividades de Mantenimiento:

**Instalación de Parches:** Instrucciones detalladas para aplicar parches de seguridad y correcciones de errores en el código de la aplicación. **Actualización de Dependencias:** Procedimientos para actualizar bibliotecas y frameworks utilizados en la tienda virtual, asegurando la compatibilidad con nuevas versiones.

Supongamos que se descubre una vulnerabilidad en la versión de Django que estás utilizando, la cual podría permitir a un atacante inyectar código malicioso en la base de datos a través de una solicitud de usuario mal formada. El equipo de Django libera un parche de seguridad para solucionar este problema.

- **Instrucciones:** Deberías aplicar este parche lo antes posible para evitar cualquier riesgo de explotación. Esto podría implicar actualizar Django a una nueva versión, lo cual se haría mediante un comando de actualización (`pip install --upgrade Django`) y luego probar el sistema para asegurarse de que no hay problemas de compatibilidad con el código existente.

### Actualización de Dependencias

Supón que estás utilizando una biblioteca como Pillow para manejar las imágenes de los productos en tu tienda virtual. Con el tiempo, se lanza una nueva versión de Pillow que mejora la eficiencia en la compresión de imágenes, lo que podría reducir el tiempo de carga de las páginas de tu tienda.

- **Procedimientos:** Actualizar Pillow a la última versión utilizando un comando como **"pip install --upgrade Pillow"**. Luego, pruebas las funcionalidades relacionadas

con la gestión de imágenes, como subir, mostrar, y eliminar imágenes de productos, para asegurarte de que la actualización no haya roto ninguna funcionalidad.

**Verificación de la Integridad:** Post-actualización, se deben realizar pruebas para verificar que el sistema sigue funcionando correctamente y que las nuevas actualizaciones no han introducido nuevos errores.

Después de actualizar Django y Pillow, es necesario asegurarse de que todas las partes del sistema siguen funcionando correctamente. Esto podría incluir pruebas funcionales y automatizadas en áreas como el proceso de compra, la gestión de inventario, y la interacción con la base de datos.