Flask. Práctica 4. Panel de Gestión Mejorado para Tienda Online.

1. Objetivo General.

El objetivo de esta práctica es que desarrolles una aplicación web completa de gestión para una tienda online, utilizando Flask, Jinja2, MongoDB y Programación Orientada a Objetos (POO). Aprenderás a organizar tu código, separar la lógica de la presentación, reutilizar plantillas, trabajar con rutas dinámicas y relativas, y aplicar estilos CSS para una mejor experiencia visual.

2. Estructura y Organización del Proyecto.

Tu proyecto debe tener esta estructura:

```
tienda_gestion/
├── app.py
├── static/
│   └── css/
│       └── styles.css
└── templates/
        ├── base.html
        ├── dashboard.html
        ├── añadir_producto.html
        ├── lista_productos.html
        ├── detalle_producto.html
        ├── registro_usuario.html
        ├── lista_usuarios.html
        └── 404.html
```

¿Por qué esta estructura?

- Mantener la lógica en **app.py**.
- Separar las plantillas HTML en la carpeta **templates.**
- Guardar los archivos CSS en la carpeta **static/css/.**

3. Explicación Detallada de los Requisitos.

3.1. Herencia de Plantillas y Navegación.

¿Qué es la herencia de plantillas?

- Permite crear una plantilla base (**base.html**) con la estructura general (cabecera, barra de navegación, footer) y que el resto de páginas la reutilicen, añadiendo sólo el contenido específico de cada una.

¿Qué debe tener base.html?

- Estructura básica HTML.
- Bloques **{% block title %}** y **{% block content %}** para que cada página defina su título y contenido.
- Barra de navegación con enlaces a todas las páginas principales, usando **url_for** para construir las rutas.
- Inclusión del **CSS** con **url_for('static', filename='css/styles.css').**
- Un footer con tu nombre y el año actual.

¿Cómo deben usar las otras plantillas la herencia?

- Todas las plantillas deben comenzar con **{% extends 'base.html' %}** y definir los bloques necesarios.

3.2. Rutas y Funcionalidad Principal.

Debes crear las siguientes rutas en app.py:

a) /dashboard.

- Página principal con:
    - Saludo al administrador, nombre de la tienda y fecha actual.
    - Tabla/lista de productos (nombre, precio, stock, categoría). Si el stock es 0, mostrar “Agotado” en rojo.
    - Total de unidades en stock.
    - Lista de clientes (nombre, email, estado, número de pedidos), cuántos están activos y el cliente con más pedidos.
    - Lista de pedidos recientes (cliente, total, fecha) y suma total de ingresos.

b) /añadir-producto.

- Formulario para añadir un producto (nombre, precio, stock, categoría).
- El formulario debe enviar datos por POST a la misma ruta.
- Al añadir el producto, mostrar un mensaje de éxito en la misma página.

c) /productos.

- Tabla/lista con todos los productos.
- Cada producto debe tener un enlace a su página de detalle (ruta dinámica).

d) /productos/<int:id_producto>.

- Muestra la información detallada del producto seleccionado.
- Si el producto no existe, muestra la página **404.html.**

e) /registro-usuario.

- Formulario para registrar un nuevo usuario (nombre, email, contraseña).
- Al enviar, crea una instancia de la **clase Usuario** y guárdala en una lista o base de datos.
- Muestra un mensaje de éxito tras el registro.

f) /usuarios.

- Muestra una tabla/lista con todos los usuarios registrados (nombre, email, fecha de registro).

g) Página de error 404.

- Si se accede a una ruta no existente o a un producto inexistente, debe mostrarse una página de error personalizada (**404.html**).

3.3. Programación Orientada a Objetos (POO) – Gestión de Usuarios.

¿Qué tienes que hacer?

- Define una clase **Usuario** en **app.py** con los atributos: nombre, email, contraseña, fecha_registro.
- Cuando se registre un usuario desde el formulario, crea una instancia de **Usuario** y guárdala en una lista o en la base de datos.
- En la página **/usuarios**, muestra la información de los usuarios accediendo a los atributos de la clase.

3.4. Rutas Relativas y Dinámicas con url_for.

¿Qué significa esto?

- **Rutas relativas:** No escribas nunca rutas directamente como **/dashboard** o **/static/css/styles.css** en los enlaces o formularios. Usa siempre **url_for('nombre_funcion')** o **url_for('static', filename='...').**

- **Rutas dinámicas:** Implementa rutas que reciben variables, por ejemplo **/productos/<int:id_producto>**, para mostrar el detalle de cada producto.
  En la plantilla, genera los enlaces así:

```
<a href="{{ url_for('detalle_producto', id_producto=producto.id) }}">{{
producto.nombre }}</a>
```

3.5. Estilos CSS.

¿Qué debe tener el CSS?

- Estilos para la barra de navegación, tablas, formularios, mensajes de éxito/error y el footer.
- El diseño debe ser limpio, legible y agradable.
- El archivo debe estar en **static/css/styles.css** y enlazarse desde **base.html** usando **url_for.**

4. Ejemplo de barra de navegación en base.html

```
<nav>
  <a href="{{ url_for('dashboard') }}">Dashboard</a>
  <a href="{{ url_for('añadir_producto') }}">Añadir Producto</a>
  <a href="{{ url_for('ver_productos') }}">Lista de Productos</a>
  <a href="{{ url_for('registro_usuario') }}">Registro de Usuario</a>
  <a href="{{ url_for('lista_usuarios') }}">Lista de Usuarios</a>
</nav>
```

5. Ampliaciones Opcionales.

- Página de error 404 personalizada con diseño especial.
- Uso de Bootstrap o Bulma para el diseño (solo si terminas lo obligatorio).
- Footer con información adicional o enlaces a redes sociales.
- Implementar edición y borrado de productos/usuarios usando rutas dinámicas.

6. Buenas Prácticas y Consejos.

- **Organiza tu código**: Mantén la lógica en Python y la presentación en las plantillas.
- **Calcula en Python**: Sumas, totales y datos complejos deben calcularse en **app.py** antes de enviarlos a la plantilla.
- **Usa bucles y condicionales en las plantillas solo para mostrar datos**.
- **Comenta tu código** y cuida la indentación.
- **No repitas código**: Usa la herencia de plantillas para evitar duplicidad.

7. Entrega.

- Sube todo en un ZIP o un repositorio de GitHub con la estructura indicada.
- Incluye un archivo **README.md** con:
    - Instrucciones para instalar dependencias y ejecutar la app.
    - Breve descripción de la funcionalidad.
    - Capturas de pantalla de las páginas principales.
- La aplicación debe funcionar correctamente al ejecutar **python app.py**.