



El.

Ingeniería Tecnologías de la Información

8°A

INTEGRANTES:

- Mendieta Chimal Sony Luis	MCS0220598
- Ascencio Onofre Carlos Gerardo	AOC0220155
- Nava Sanchez Axel	NSA0220388

Docente: LORENZO ANTONIO CARDOSO CONTRERAS

Materia: Tecnologías y aplicaciones en Internet

6/04/2025

Índice

Índice	1
Planteamiento del caso práctico.	2
Introducción.	3
Arquitectura WEB.	4
Diagramas de presentación.	5
Aspectos de usabilidad.	6
Desarrollo Front-end.	7
API.	8
Middleware.	9
Manejo de sesiones.	10
Desarrollo Back-end.	11
Diseño de la base de datos:	12
Conclusiones.	13
Bibliografía.	14

Planteamiento del caso práctico.

El sistema que se desarrollará está orientado a resolver las necesidades de gestión de una perfumería. Actualmente, la perfumería enfrenta una situación en la que la gestión de productos, pedidos y clientes se realiza de manera manual y desorganizada. Esto provoca errores en el manejo de inventarios, dificultades para realizar un seguimiento adecuado de los pedidos y una atención al cliente ineficiente.

La situación ideal que se busca con el desarrollo de esta aplicación es contar con un sistema automatizado que permita gestionar de manera eficiente:

1. **Inventarios de productos:** Controlar la cantidad de productos disponibles, actualizar el stock en tiempo real y generar alertas cuando los productos estén por agotarse.
2. **Pedidos de clientes:** Facilitar el proceso de compra en línea, registrar los pedidos, gestionar su estado y ofrecer un seguimiento detallado de los mismos.
3. **Datos de clientes:** Permitir el registro de clientes, almacenar su información personal y mantener un historial de compras para ofrecer una experiencia personalizada.
4. **Proveedor:** Gestionar las relaciones con los proveedores, controlar la llegada de nuevos productos y optimizar el proceso de compra.

El sistema está basado en una arquitectura web, utilizando tecnologías como Django y MongoDB, lo que permitirá una mayor escalabilidad y eficiencia en el manejo de grandes cantidades de datos. Se espera que la aplicación mejore la productividad de la perfumería, reduzca los errores humanos y proporcione un entorno más organizado tanto para los empleados como para los clientes.

Introducción.

El presente reporte tiene como objetivo detallar el desarrollo de un sistema web para la gestión integral de una perfumería. A lo largo de este documento, se describen los aspectos fundamentales del proyecto, incluyendo la arquitectura web, el desarrollo tanto del front-end como del back-end, la conexión con la base de datos MongoDB y la creación de APIs, entre otros.

El reporte está estructurado en varias secciones que cubren todos los aspectos técnicos y conceptuales del desarrollo. Se inicia con una descripción general del caso práctico que el sistema pretende resolver, seguido de la arquitectura lógica y física del sitio web. Se presentan también diagramas de presentación, prototipos y aspectos de usabilidad que son clave en el diseño del sitio.

Además, se detallan los procedimientos relacionados con el desarrollo front-end y back-end, el manejo de sesiones, la creación de APIs, el uso de middleware y la interacción con la base de datos. También se incluye el diseño del modelo de datos y la forma en que la base de datos NoSQL, MongoDB, se integra con el sistema.

Finalmente, se presentan las conclusiones obtenidas al finalizar el proyecto, destacando los aprendizajes clave, los retos enfrentados durante el proceso de desarrollo y las soluciones implementadas.

Este documento tiene como fin ofrecer una visión completa sobre el desarrollo y funcionamiento del sistema web, proporcionando los detalles técnicos necesarios para comprender la solución implementada.

Arquitectura WEB.

Arquitectura Lógica

La arquitectura lógica representa la organización funcional del sistema, mostrando los principales componentes y cómo interactúan entre sí.

Componentes:

- Cliente (Front-end)
 - Interfaz desarrollada en HTML, CSS y JavaScript.
 - Formularios de inicio de sesión, gestión de productos, pedidos, etc.
 - Comunicación con el servidor a través de solicitudes HTTP.
- Servidor Web (Back-end con Django)
 - Lógica de negocio, rutas y controladores.
 - Middleware para validación de sesiones y tipo de usuario.
 - Conexión con la base de datos MongoDB.
- Base de Datos (MongoDB)
 - Almacena colecciones de usuarios, productos, pedidos, etc.
 - Modelo NoSQL usando MongoEngine.

Flujo lógico:

1. El usuario accede a la interfaz web desde su navegador.
2. Las solicitudes se envían al servidor Django.
3. El middleware verifica permisos según el tipo de usuario.
4. Django accede a MongoDB si es necesario (ej. cargar productos).

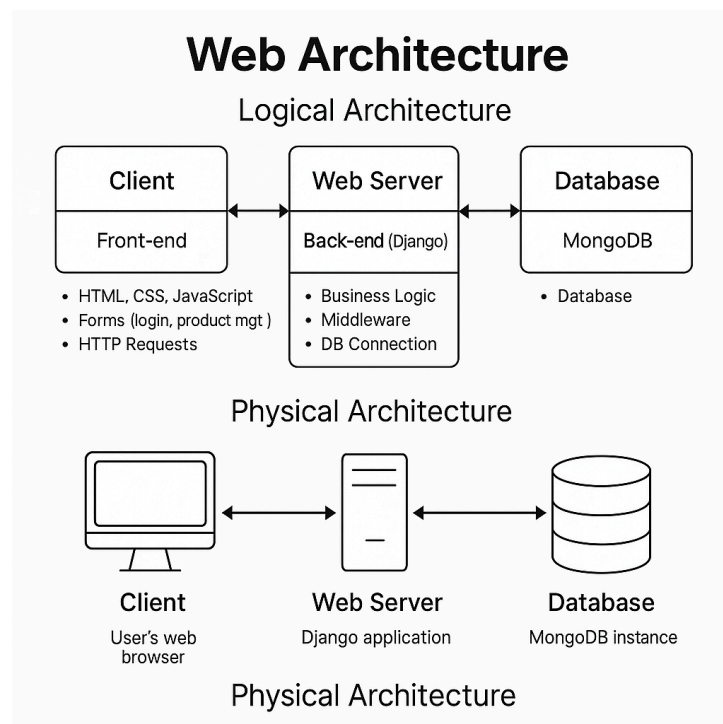
5. Se responde al cliente con la página renderizada o redirigida.

Arquitectura Física

La arquitectura física describe cómo se distribuyen los componentes del sistema en los dispositivos reales o virtuales.

Componentes Físicos:

- Cliente (Navegador Web del usuario)
 - Cualquier dispositivo con acceso a internet (PC, tablet, móvil).
- Servidor Web (Local o en la nube)
 - Alojamiento del proyecto Django.
 - Servidor corriendo Gunicorn/WSGI + Nginx o entorno de desarrollo local.
- Base de Datos (MongoDB)
 - Puede estar en el mismo servidor o en un servicio externo (como Atlas).



Diagramas de presentación.

La Figura D.1 muestra el diagrama del diseño de la página principal del sistema web.

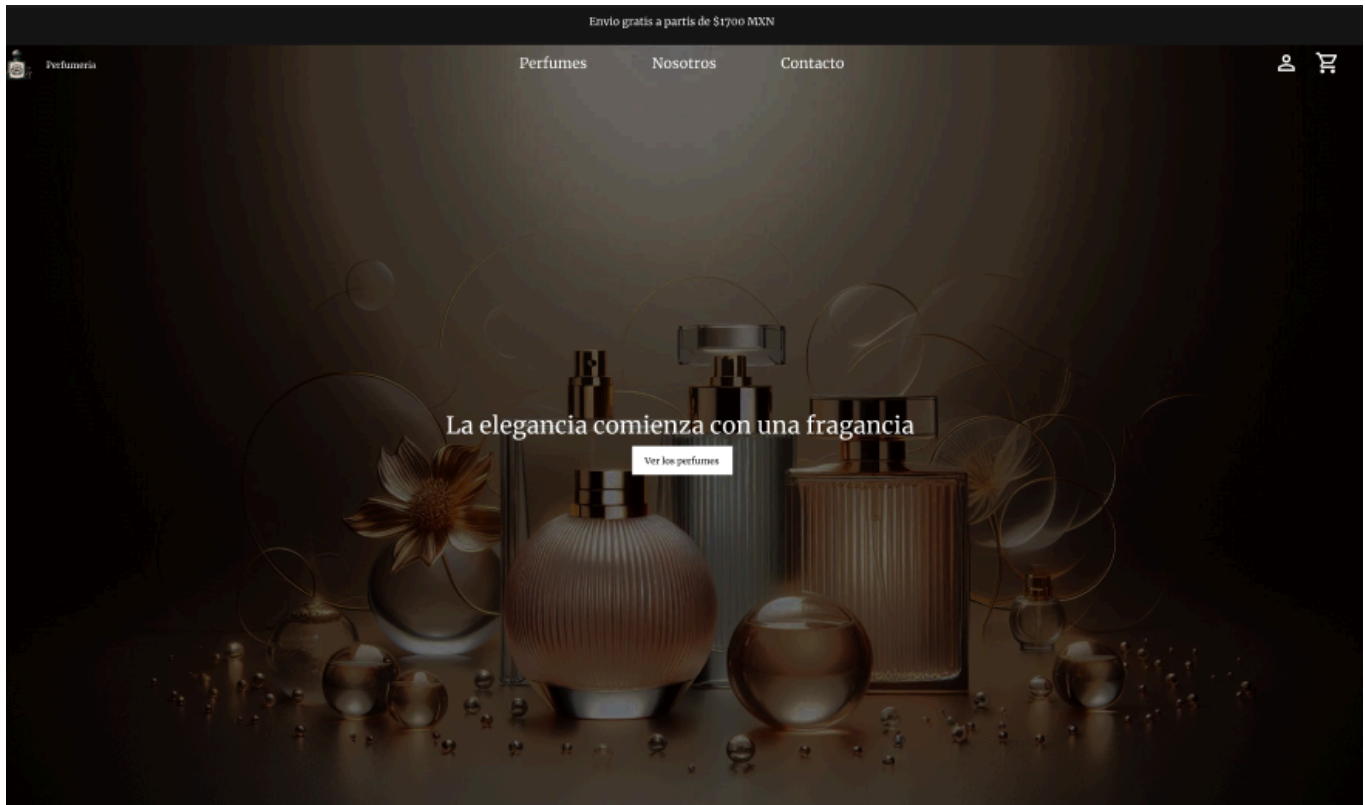


Figura D.1 Diseño Avanzado de la página principal

La Figura D.2 muestra el diagrama del diseño de la página principal donde se ven los productos

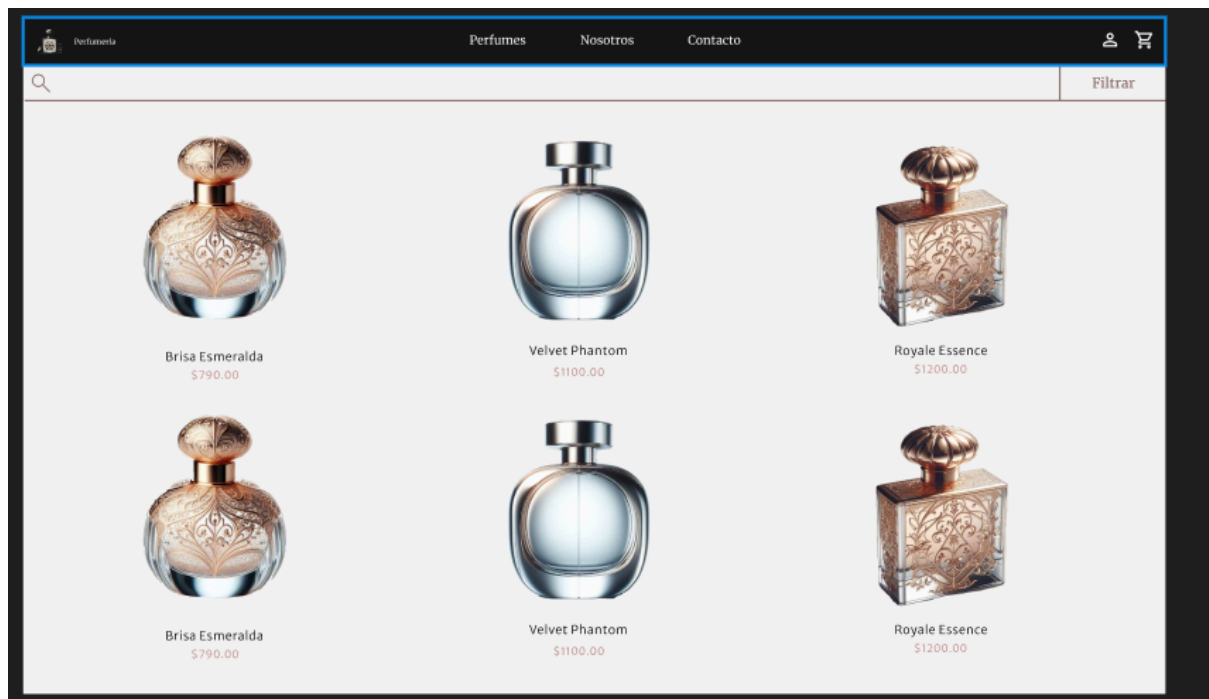


Figura D.2 Diseño Avanzado

La Figura D.3 muestra el diagrama del diseño de la página principal donde se inicia sesión

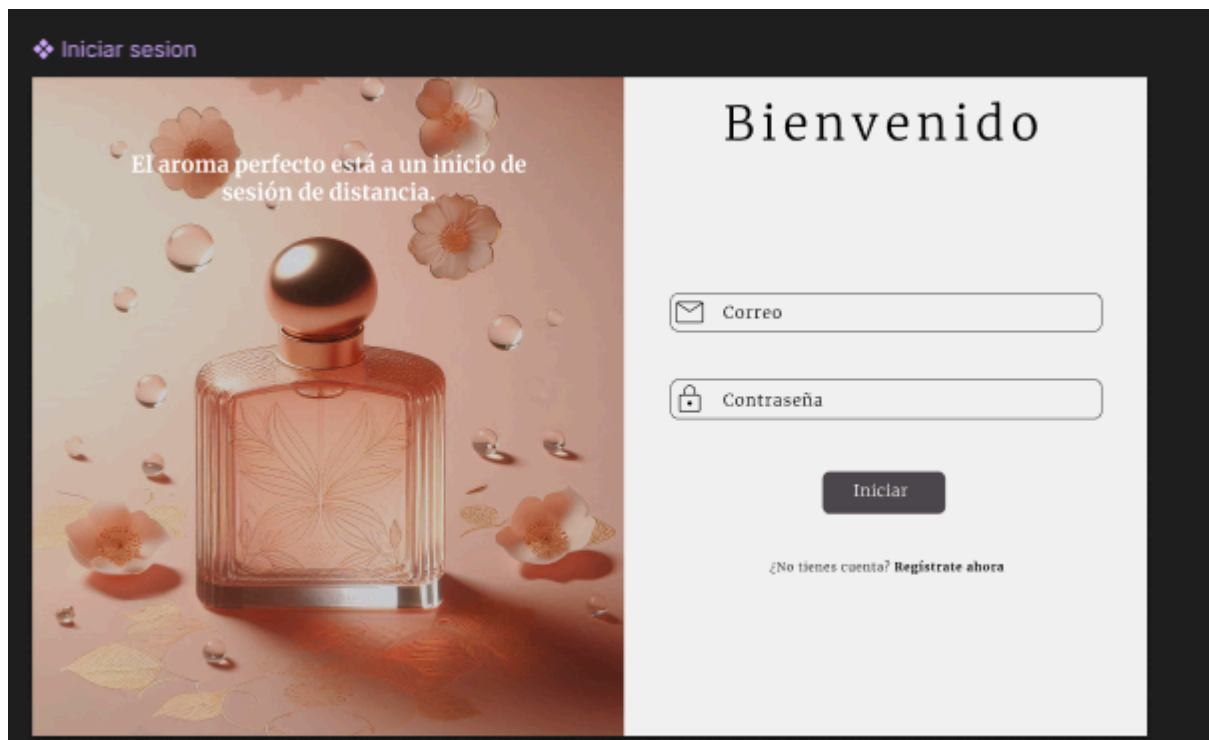


Figura D.3 Diseño Avanzado

La Figura D.4 muestra el diagrama del diseño de la página principal donde se registra el usuario,

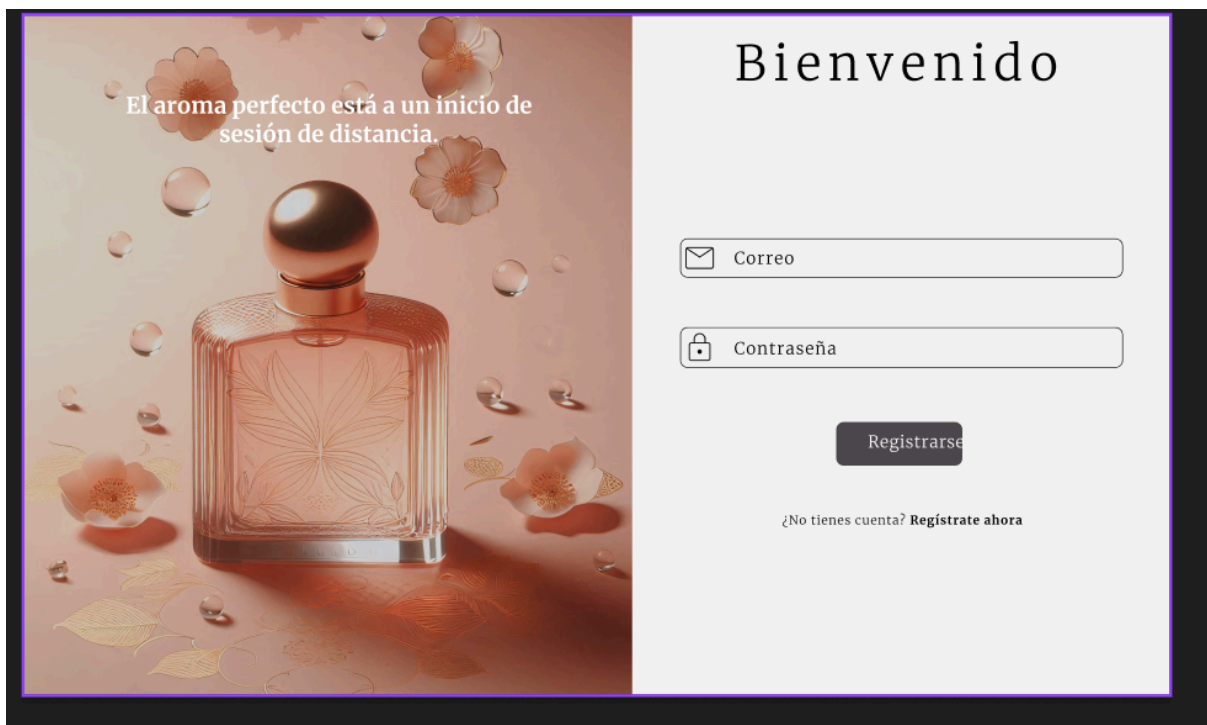


Figura D.4 Diseño Avanzado

La Figura D.5 muestra el diagrama del diseño de la página principal donde se administran los productos.

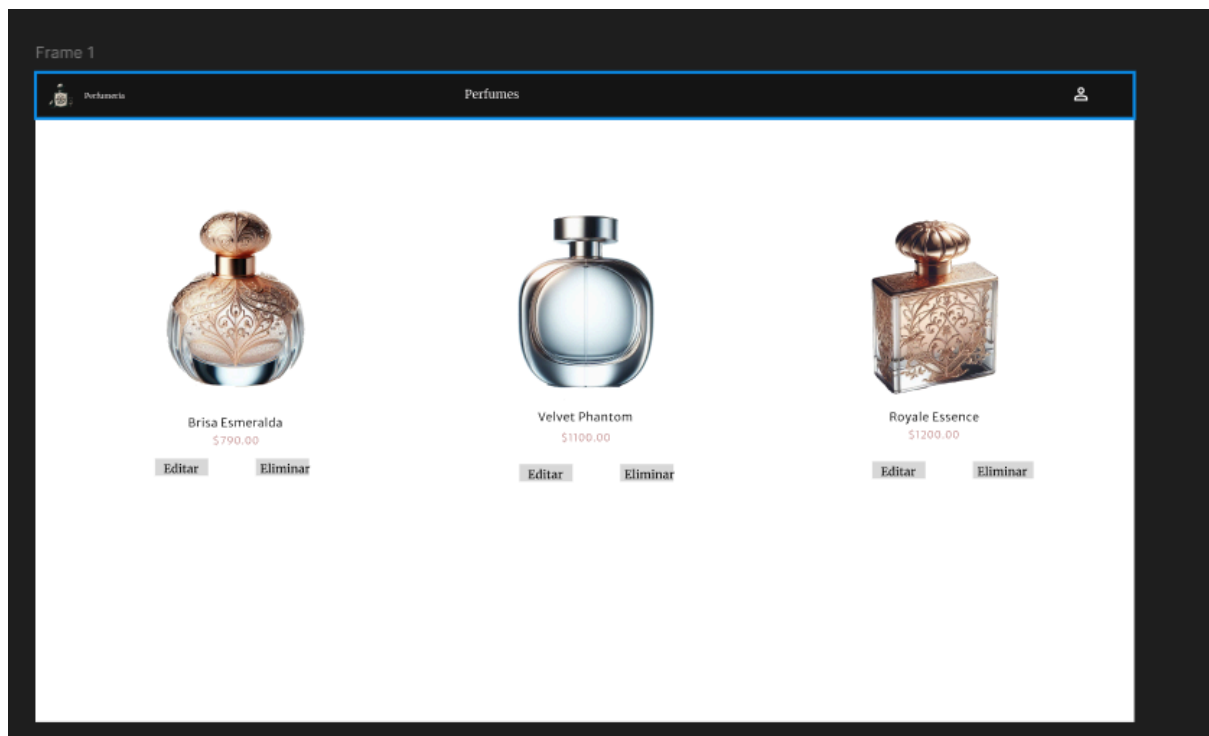


Figura D.5 Diseño Avanzado.

Aspectos de usabilidad.

Navegación clara y estructurada

- Se organizaron las rutas del sistema de forma intuitiva: Inicio, Productos, Proveedores, Usuarios, etc.
- Se utilizaron menús desplegables (hamburguesa en mobile) para facilitar el acceso en distintos dispositivos.

Compatibilidad con dispositivos móviles (diseño responsivo)

- Se utilizó CSS y media queries o frameworks como Bootstrap/Tailwind para que la página se vea bien tanto en computadoras como en celulares o tablets.

Retroalimentación visual

- El sistema muestra mensajes de error cuando el usuario introduce mal una contraseña o deja un campo vacío en los formularios.
- Se usan colores o alertas para confirmar acciones exitosas (por ejemplo: "Producto agregado correctamente").

Formularios simples y validados

- Los formularios tienen validaciones que previenen errores y ayudan al usuario a completarlos correctamente.
- Campos como "correo", "contraseña" o "nombre del producto" tienen validaciones específicas.

Accesibilidad

- Se usaron etiquetas semánticas (<label>, <input>, <nav>, etc.) para ayudar a los lectores de pantalla y mejorar la experiencia de personas con discapacidad.
- Los botones y enlaces tienen nombres claros como "Agregar producto" o "Editar proveedor".

Minimizar clics y facilitar tareas

- Las acciones importantes están al alcance de pocos clics (como editar, eliminar o agregar registros).
- Al iniciar sesión, se redirige automáticamente según el tipo de usuario (admin, empleado) para una experiencia personalizada.

Consistencia en el diseño

- Todos los botones, formularios y tablas tienen un estilo visual coherente en todas las vistas.
- Se reutilizaron componentes para no confundir al usuario cambiando la forma en que interactúa en distintas secciones.

Desarrollo Front-end.

1. Colores

- Se utilizó una **paleta de colores armónica y coherente** con el rubro de perfumería (por ejemplo, tonos suaves, pasteles o elegantes como el rosado, blanco, dorado, lavanda, etc.).
- Colores distintos para **acciones específicas**, como:
 - Botones primarios: azul o verde (para acciones como “Agregar” o “Guardar”).
 - Botones de advertencia o eliminar: rojo.
 - Fondo claro para facilitar la lectura.

2. Fondos

- Fondos neutros o con degradados suaves para no saturar visualmente.
- En algunas páginas (como el inicio o login), se utilizó una **imagen o fondo personalizado** relacionado con la temática de perfumes.

3. Animaciones y Efectos

- Se integraron **animaciones suaves** con CSS para:
 - Aparición de menús desplegables (navbar tipo hamburguesa en vista móvil).
 - Transiciones en botones al pasar el mouse.
 - Efecto de "hover" sobre tarjetas de productos o proveedores.
- **Animaciones al cargar elementos**, como formularios o tablas, para mejorar la estética y suavidad del sitio.

4. Diseño Responsive

- Se aplicaron reglas de diseño adaptable (responsive) usando media queries o frameworks como Tailwind CSS/Bootstrap.
- El sitio se adapta automáticamente a **dispositivos móviles, tablets y escritorios**, garantizando accesibilidad desde cualquier dispositivo.

5. Coherencia visual y estructura

- Se mantuvo un **estilo visual consistente** en todas las páginas:
 - Tipografía uniforme.
 - Tamaño y forma de botones coherentes.
 - Uso de íconos (por ejemplo, para eliminar, editar o regresar).
- Las secciones están bien diferenciadas con bordes, sombras o separadores visuales.

API.

El ProveedorAPI y el ProveedorDetailAPI (figuras 2.1, 2.2) son vistas de API para gestionar los proveedores en la aplicación. Estas vistas están protegidas por autenticación JWT (JSON Web Token) a través del middleware MongoEngineJWTAuthentication, lo que garantiza que solo los usuarios autenticados puedan acceder a ellas.

Las funcionalidades incluyen:

1. Obtener todos los proveedores (GET /proveedores).
2. Crear un nuevo proveedor (POST /proveedores).
3. Obtener un proveedor específico por ID (GET /proveedores/{id}).
4. Actualizar los datos de un proveedor (PUT /proveedores/{id}).
5. Eliminar un proveedor (DELETE /proveedores/{id}).

```
class ProveedorAPI(APIView):
    authentication_classes = [MongoEngineJWTAuthentication] # Usa tu backend
    permission_classes = [IsAuthenticated]
    # Obtener todos los proveedores o crear uno nuevo
    def get(self, request, *args, **kwargs):
        proveedores = Proveedor.objects.all() # Queryset de MongoEngine
        serializer = ProveedorSerializer(proveedores, many=True)
        return Response(serializer.data)

    def post(self, request, *args, **kwargs):
        serializer = ProveedorSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figura 2.1 API Proveedor

```
class ProveedorDetailAPI(APIView):
    authentication_classes = [MongoEngineJWTAuthentication] # Usa tu backend
    permission_classes = [IsAuthenticated]
    # Helper para obtener un proveedor por ID
    def get_object(self, id):
        try:
            return Proveedor.objects.get(id=id)
        except Proveedor.DoesNotExist:
            raise status.HTTP_404_NOT_FOUND

    # Obtener un proveedor específico
    def get(self, request, id, *args, **kwargs):
        proveedor = self.get_object(id)
        serializer = ProveedorSerializer(proveedor)
        return Response(serializer.data)

    # Actualizar un proveedor
    def put(self, request, id, *args, **kwargs):
        proveedor = self.get_object(id)
        serializer = ProveedorSerializer(proveedor, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    # Eliminar un proveedor
    def delete(self, request, id, *args, **kwargs):
        proveedor = self.get_object(id)
        proveedor.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Figura 2.2 API Proveedor

Las clases ProductoAPI y ProductoDetailAPI (figura 2.3, 2.4) son vistas basadas en clases (APIView) que exponen una API RESTful para el manejo de productos. Estas vistas están protegidas por autenticación con JSON Web Tokens (JWT), usando el middleware personalizado MongoEngineJWTAuthentication, garantizando que solo usuarios autenticados (como el administrador) puedan realizar operaciones sobre los productos.

Funcionalidades de la API:

- GET /productos/ → Obtener todos los productos.
- POST /productos/ → Crear un nuevo producto.
- GET /productos/{id} → Obtener un producto por ID.
- PUT /productos/{id} → Actualizar un producto.
- DELETE /productos/{id} → Eliminar un producto.

```
class ProductoAPI(APIView):
    authentication_classes = [MongoEngineJWTAuthentication] # Usa tu backend
    permission_classes = [IsAuthenticated]
    # Obtener todos los productos o crear uno nuevo
    def get(self, request, *args, **kwargs):
        productos = Producto.objects.all() # Queryset de MongoEngine
        serializer = ProductoSerializer(productos, many=True)
        return Response(serializer.data)

    def post(self, request, *args, **kwargs):
        serializer = ProductoSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data, status=status.HTTP_201_CREATED)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)
```

Figura 2.3 API Producto.

```
class ProductoDetailAPI(APIView):
    authentication_classes = [MongoEngineJWTAuthentication] # Usa tu backend
    permission_classes = [IsAuthenticated]
    # Helper para obtener un producto por ID
    def get_object(self, id):
        try:
            return Producto.objects.get(id=id)
        except Producto.DoesNotExist:
            raise status.HTTP_404_NOT_FOUND

    # Obtener un producto específico
    def get(self, request, id, *args, **kwargs):
        producto = self.get_object(id)
        serializer = ProductoSerializer(producto)
        return Response(serializer.data)

    # Actualizar un producto
    def put(self, request, id, *args, **kwargs):
        producto = self.get_object(id)
        serializer = ProductoSerializer(producto, data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data)
        return Response(serializer.errors, status=status.HTTP_400_BAD_REQUEST)

    # Eliminar un producto
    def delete(self, request, id, *args, **kwargs):
        producto = self.get_object(id)
        producto.delete()
        return Response(status=status.HTTP_204_NO_CONTENT)
```

Figura 2.4 API Producto.

Middleware.

Se desarrolló un middleware personalizado en Django para garantizar que las rutas del sistema web sean accesibles únicamente por usuarios con el rol correspondiente. Este middleware no depende del sistema de autenticación de Django (auth) ya que se utiliza una estructura de sesiones personalizadas almacenadas en cookies (no se utiliza una base de datos SQL).

¿Qué hace este middleware?

- Valida la existencia de una sesión activa (usuario_id).
- Verifica el tipo de usuario (usuario_tipo), el cual puede ser "admin" o "empleado".
- Controla el acceso a ciertas rutas según el rol del usuario.
- Redirige a la página de inicio de sesión si el usuario no tiene permisos.

¿Cómo se conecta con la base de datos?

El middleware no consulta directamente la base de datos, ya que los datos relevantes (usuario_id y usuario_tipo) se guardan previamente en la sesión al iniciar sesión correctamente, en una vista como esta:

```

class AdminMiddleware:
    def __init__(self, get_response):
        self.get_response = get_response

    def __call__(self, request):
        rutas_solo_admin = [
            '/proveedores/', '/proveedor/crear', '/proveedor/editar', '/proveedor/eliminar',
            '/producto/crear', '/producto/editar', '/producto/eliminar',
            '/Administrador/productos/', '/Administrador/usuarios/', '/Administrador/proveed
            '/Administrador/usuario/eliminar/',
        ]

        rutas_admin_y_empleado = [
            '/Administrador/usuario/', '/Administrador/usuario/editar/'
        ]

        path_actual = request.path
        usuario_id = request.session.get('usuario_id')
        usuario_tipo = request.session.get('usuario_tipo')

        # Rutas exclusivas para admin
        if any([path_actual.startswith(ruta) for ruta in rutas_solo_admin]):
            if not usuario_id or usuario_tipo != 'admin':
                return redirect('iniciar_sesion')

        # Rutas accesibles para admin y empleado
        elif any([path_actual.startswith(ruta) for ruta in rutas_admin_y_empleado]):
            if not usuario_id or usuario_tipo not in ['admin', 'empleado']:
                return redirect('iniciar_sesion')

        return self.get_response(request)

```

Manejo de sesiones.

El sistema web de la perfumería implementa un sistema de autenticación basado en sesiones para gestionar el acceso de los usuarios y sus permisos dentro de la plataforma. A continuación, se describen los diferentes tipos de usuarios y sus permisos, así como el uso de tokens para la API.

1. Administrador

- **Permisos:** El Administrador tiene acceso completo a todas las funcionalidades del sistema. Puede modificar los productos, proveedores, y usuarios, así como realizar otras operaciones administrativas.
- **Acceso:** Solo el Administrador puede acceder a rutas que permiten la modificación de datos, configuración del sistema y gestión de otros usuarios.
- **Acción:** El Administrador tiene la capacidad de modificar los productos, proveedores, y usuarios, además de gestionar configuraciones del sistema.
- **API y Tokens:** El Administrador es el único usuario que podrá obtener y utilizar un token de autenticación para interactuar con la API. El token se utiliza para validar las solicitudes a las rutas protegidas de la API y garantizar que solo el Administrador pueda realizar acciones de gestión (como modificar productos o proveedores).

2. Empleado

- **Permisos:** Los Empleados tienen permisos limitados y solo pueden editar su propio perfil, sin acceso a la gestión de datos del sistema ni funciones administrativas.
- **Acceso:** Los Empleados solo tienen acceso a las rutas que les permiten actualizar su información personal (por ejemplo, nombre, correo, etc.).
- **Acción:** Los Empleados pueden editar su perfil, pero no tienen acceso a ninguna función de administración o modificación de datos en el sistema.
- **API y Tokens:** Los Empleados no tendrán acceso a la API para realizar modificaciones de datos. Solo pueden interactuar con las funcionalidades básicas que no requieren autenticación vía token.

3. Usuarios no autenticados

- Permisos: Los Usuarios que no han iniciado sesión no podrán acceder a las funcionalidades protegidas del sistema.
- Acceso: Este grupo de usuarios solo podrá visualizar el contenido público del sitio (como el catálogo de productos), pero no tendrá acceso a ninguna funcionalidad restringida (como la edición de perfil o la gestión de datos).
- Acción: No podrán realizar ninguna acción de edición o gestión hasta que inicien sesión y obtengan los permisos correspondientes.

API y Tokens: Los Usuarios no autenticados no tienen acceso a la API ni a las rutas protegidas por token. Solo podrán acceder a las funcionalidades públicas del sistema.

Desarrollo Back-end.

El desarrollo del Back-end del sitio web fue implementado utilizando el framework Django de Python, adaptado para trabajar con MongoDB como base de datos no relacional a través de la librería mongoengine. A continuación se detallan los componentes clave del back-end:

Conexión con la Base de Datos (MongoDB)

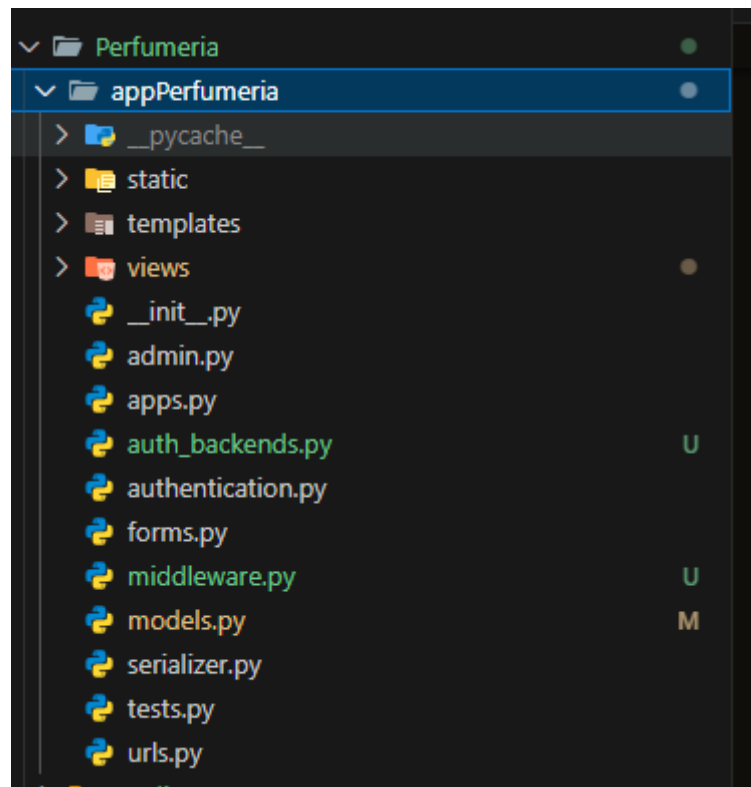
El proyecto establece la conexión con MongoDB de la siguiente forma:

```
connect(
    'Perfumeria', # Nombre de la base de datos
    host='mongodb://localhost:27017/', # Si usas MongoDB local
    # Si usas MongoDB Atlas (en la nube), usa la cadena de conexión de Atlas
    # host='mongodb+srv://<usuario>:<contraseña>@cluster.mongodb.net/<base_de_datos>'
)
```

El back-end se encarga de:

- Autenticación de usuarios mediante sesiones personalizadas (guardando usuario_id y usuario_tipo en request.session).
- Protección de rutas con un middleware personalizado, que controla el acceso según el tipo de usuario (admin, empleado, cliente).
- CRUD de entidades como productos, proveedores, pedidos, etc., implementados mediante vistas Django y MongoEngine para operaciones en la base de datos.
- Controladores de vistas que redirigen a diferentes interfaces según el tipo de usuario.
- Validaciones de formularios (login, creación/edición de entidades) usando formularios personalizados.

El back-end está estructurado de la siguiente forma:



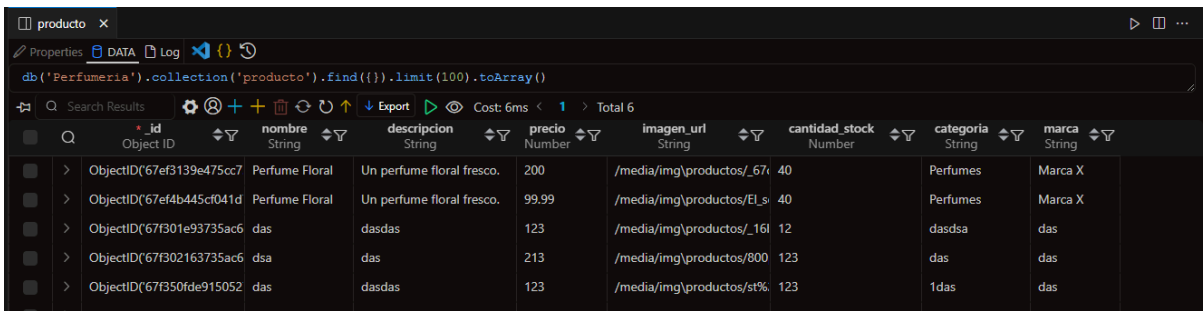
Diseño de la base de datos:

Para el desarrollo del sistema web de la perfumería se ha utilizado MongoDB, una base de datos NoSQL de tipo documento. Este tipo de base de datos permite almacenar los datos en formato JSON (o BSON internamente), facilitando la representación flexible de la información y adaptándose perfectamente a las necesidades del sistema, donde los datos pueden tener estructuras dinámicas.

Se ha diseñado una base de datos llamada perfumeria, la cual contiene varias colecciones que representan las entidades principales del sistema. Estas colecciones no están relacionadas de forma rígida como en una base de datos relacional, lo que permite escalar y modificar su estructura con mayor facilidad.

A continuación, se muestra el esquema físico con ejemplos de documentos de dos de las colecciones principales:

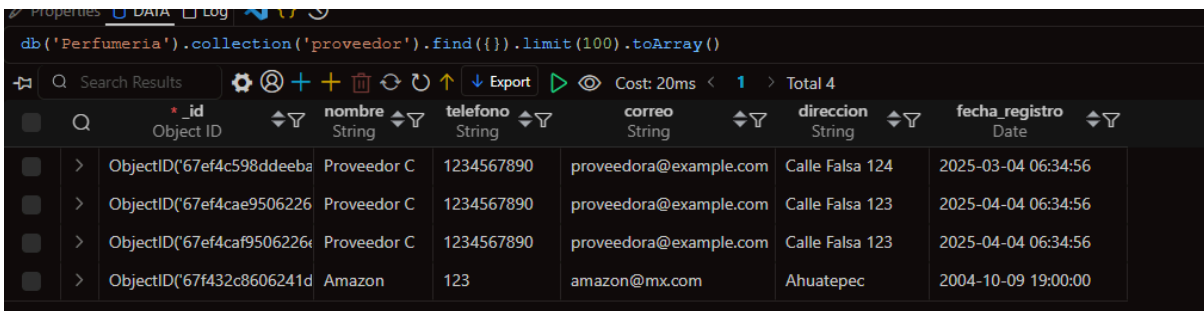
En la siguiente figura se muestra la colección de producto.



	*_id Object ID	nombre String	descripcion String	precio Number	imagen_url String	cantidad_stock Number	categoria String	marca String
>	ObjectID('67ef3139e475cc7')	Perfume Floral	Un perfume floral fresco.	200	/media/img/productos/_67c	40	Perfumes	Marca X
>	ObjectID('67ef4b445cf041d')	Perfume Floral	Un perfume floral fresco.	99.99	/media/img/productos/El s	40	Perfumes	Marca X
>	ObjectID('67f301e93735ac6')	das	dasdas	123	/media/img/productos/_16l	12	dasdsa	das
>	ObjectID('67f302163735ac6')	dsa	das	213	/media/img/productos/800	123	das	das
>	ObjectID('67f350fde915052')	das	dasdas	123	/media/img/productos/st%	123	1das	das

Figura B.1 Colección Producto.

En la siguiente figura se muestra la colección de proveedor.



	*_id Object ID	nombre String	telefono String	correo String	direccion String	fecha_registro Date
>	ObjectID('67ef4c598ddeeba')	Proveedor C	1234567890	proveedora@example.com	Calle Falsa 124	2025-03-04 06:34:56
>	ObjectID('67ef4cae9506226')	Proveedor C	1234567890	proveedora@example.com	Calle Falsa 123	2025-04-04 06:34:56
>	ObjectID('67ef4caf9506226')	Proveedor C	1234567890	proveedora@example.com	Calle Falsa 123	2025-04-04 06:34:56
>	ObjectID('67f432c8606241d')	Amazon	123	amazon@mx.com	Ahuatepec	2004-10-09 19:00:00

Figura B.2 Colección Producto

Conclusiones.

Durante el desarrollo de esta práctica, se adquirieron conocimientos fundamentales para la creación de aplicaciones web completas, abarcando tanto el front-end como el back-end. Los principales aprendizajes incluyen:

- Implementación de un sistema de autenticación personalizado con sesiones.
- Uso de middlewares en Django para proteger rutas según el tipo de usuario.
- Conexión de Django con MongoDB mediante MongoEngine.
- Estructuración de una base de datos NoSQL para una aplicación web.
- Desarrollo de una interfaz responsiva utilizando HTML, CSS y JavaScript.

Principales retos enfrentados:

- Adaptar Django para que funcione sin base de datos relacional.
- Manejar la autenticación sin el sistema auth de Django.
- Configurar el middleware correctamente para que funcionara con sesiones personalizadas.

Sucesos relevantes:

- Se logró implementar un middleware funcional que protege las rutas para distintos tipos de usuario.
- Se almacenan imágenes de productos directamente en el servidor, lo cual optimiza el rendimiento y el control de los recursos.

Bibliografía.

Casr, D. (2018). Construir un API REST con Django REST Framework y APIView. Recuperado de

<https://davidcasr.medium.com/construir-un-api-rest-con-django-rest-framework-y-apiview-5ea4b2823307>

Jaime. (2020). Desarrollo de API Restful con Python, Django y Django Rest Framework. Recuperado de

<https://devjaime.medium.com/desarrollo-de-api-restful-con-python-django-y-django-rest-framework-e6f019dbe177>

Viera, U. (2025.). Creación de una API REST con Django Rest Framework. Recuperado de

<https://www.urianviera.com/django/django-rest-api-guide>

Hektor Profe. (2025). Proyecto API con Django Rest Framework. Recuperado de

<https://docs.hektorprofe.net/academia/django/api-rest-framework/>

Urian121. (2025). Consumir una API REST con Django y Python en 10 líneas. Recuperado de

<https://github.com/urian121/Consumir-una-API-REST-con-Django-y-Python-en-10-lineas>