PONTIFICIA UNIVERSIDAD CATÓLICA DEL ECUADOR

Unidad Académica de Formación Técnica y Tecnológica – PUCE TEC

DESARROLLO DE UN CATALOGO WEB PARA EMPRENDIMIENTO VARIEDADES MEDINA

Proyecto de titulación previo a la obtención del título de: Tecnólogo Superior en Desarrollo de Software

Autor: Josselyn Simbaña

Tutor: Patricio Alvear

Quito, Ecuador

2025

Dedicatoria

Deseo dedicar este trabajo a todos quienes creyeron, confiaron y me apoyaron durante este trayecto de mi vida, comenzando por Dios quien me brindó la bondad y sabiduría para seguir en cada momento de dificultad.

A mi madre, la persona más importante de mi vida, quien es la motivación de seguir a adelante, a quien no solo debo la vida, sino también los aprendizajes que he tenido.

Y por último a mis hermanos que siempre estuvieron ahí para apoyarme en cada momento que lo necesitaba, quienes me dieron fuerza y por quienes quisiera en algún momento ser un ejemplo de que a pesar de todos los obstáculos siempre podemos superarnos a nosotros mismos.

Tabla de Contenido

Lista de tablas

Lista de figuras

DECLARACIÓN y AUTORIZACIÓN

Yo, Josselyn Simbaña con C.I. 1728608298 autora del trabajo de Titulación intitulado: "Catalogo web para emprendimiento Variedades Medina", previa a la obtención del título de Desarrollo de Software en la Unidad Académica de Formación Técnica y Tecnológica PUCE TEC:

1.- Declaro tener pleno conocimiento de la obligación que tiene la Pontificia Universidad Católica del Ecuador, de conformidad con el artículo 144 de la Ley Orgánica de Educación Superior, de entregar a la SENESCYT en formato digital una copia del referido trabajo de graduación para que sea integrado al Sistema Nacional de Información de la Educación Superior del Ecuador para su difusión pública respetando los derechos de autor.

2.- Autorizo a la Pontificia Universidad Católica del Ecuador a difundir a través de sitio web de la Biblioteca de la PUCE el referido trabajo de titulación, respetando las políticas de propiedad intelectual de Universidad.

Quito, 2025

Josselyn Pamela Simbaña Calapiña

C.I. 1728608298

Agradecimientos

Agradezco primeramente a Dios, quien proporcionó la fuerza necesaria para concluir con éxito este proyecto, bridándome esperanza y sabiduría en aquellos momentos que más necesitaba.

Agradezco a mi madre por todo el esfuerzo y sacrificio, quien me enseño que solamente luchando y siendo constantes podemos lograr nuestros objetivos, quien con su amor y cuidado ha llevado al éxito a mis hermanos y a mí. Siendo el primer ejemplo de que el trabajo duro permite lograr grandes cosas.

Agradezco también a mis hermanos, quienes han sido parte importante de mi desarrollo como persona, me han acompañado en cada etapa de la vida y guiándome los aprendizajes a lo largo del camino, quienes ofrecieron su tiempo y dedicación para cuidarme y protegerme, para enseñarme que la vida se debe enfrentar siempre con valentía y perseverancia.

Por último, agradezco a una Belén, quien me abrió una nueva perspectiva del mundo, mostrándome que la resiliencia permite afrontar cada obstáculo de la vida y llevarse un gran aprendizaje consigo, quien ha sido una gran maestra, amiga, compañera, hermana y segunda madre, con quien aprendí que para los momentos difíciles es mejor ir un paso a la vez, que el limite no existe siempre que tenga el valor para construir hermosas alas y volar tan alto como podamos.

Introducción

Hoy en día, la tecnología ha tenido grandes avances y con eso todas las industrias han tenido la obligación de adaptarse a este estilo de vida.

Por esta razón, la competencia en la industria del comercio ha desarrollado una brecha significativa, en la que los grandes, medianos y pequeños negocios no tienen un acceso equitativo a recursos, en este caso a la tecnología, visibilidad y canales de venta. Esta desigualdad limita la posibilidad de crecimiento de negocios pequeños en el mercado.

Frente a esta problemática en cuanto al acceso a recursos digitales, se ha vuelto necesario crear y desarrollar herramientas que permitan a los pequeños eprendedores entrear en el mercado en mejores condiciones.

La creación de un catálogo web proporciona una reducción en esa desigualdad, para brindar una plataforma mas accesible, funcional y de costos bajos para aquellos negocios con recursos económicos limitados, de manera que puedan tener una mayor exposición de sus productos y así poder llegar a nuevos clientes, al igual que podrá tener una mejor organización y gestión de su negocio.

El proyecto en desarrollo tiene como objetivo crear una página web en la que la emprendedora podrá disponer de un servicio de visualización de sus productos, además de proporcionar información detallada sobre la talla y valor de cada artículo mejorando así la experiencia de comprar en línea.

Al desarrollar este catálogo se enfoca en lo importante e imprescindible que es la tecnología para el comercio en la actualidad, debido a la creciente competencia que usa la tecnología para exponer y promocionar sus productos, con una orientación más atractiva, intuitiva y funcional que facilite la navegación la toma de decisiones de los clientes.

Al desarrollar una interfaz se debe tomar en cuenta la simplicidad de ejecución en acciones de compra ya que algunos usuarios suelen experimentar dificultades para encontrar

los productos deseados o realizar el pago, lo que puede producir una baja conversión de ventas y pérdida de clientes potenciales.

Desde una perspectiva más técnica, el problema se encuentra que muchos emprendedores no suelen usar herramientas tecnológicas, ya sea por la falta de accesibilidad o falta de conocimiento en el manejo de estas, lo que no permite a las tiendas gestionar sus ventas y exhibir sus productos de forma atractiva. Por tal razón la solución propuesta es el desarrollo de un catálogo digital basado en tecnologías web modernas, para así optimizar la presentación de los productos. La implementación de lenguajes de programación como JavaScript en el Front End para asegurar una interfaz más interactiva y atractiva, además de ser responsiva para los distintos dispositivos que pueda tener el cliente.

En cuanto al Back End, se desarrollará con Node.js y MongoDB, lo que permitirá gestionar de manera eficiente de la información de los productos e interacciones de los usuarios.

Capítulo I

1.1Levantamiento de Requisitos y Diseño del Sistema

1.1.1 Requisitos Funcionales

Funcionalidad N°1: Ingreso al sistema

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F1	Ingreso al sistema	19/5/2025	19/5/2025	Crear un sistema que tendrá una pantalla de ingresa en la cual el usuario digitará su nombre de usuario y su contraseña, además de que habrá un usuario que tendrá los permisos de agregar, modificar o eliminar productos del catalogo	Alta
	Entrada	**Proceso**		**Salida**	
	Ingreso de nombre de usuario y contraseña	El sistema verifica las credenciales en la base de datos		Acceso permitido o denegado según la validación	
	Solicitud de acciones sobre productos	El sistema verifica el rol del usuario autenticado		Se autoriza la gestión de productos solo si es administrador	

Tabla 1: Ingreso al sistema

Funcionalidad N°2: Creación de productos

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F2	Creación de productos	19/5/2025	19/5/2025	El administrador podrá actualizar los detalles del producto	Alta
	Entrada	**Proceso**		**Salida**	

	Selección de un producto existente	El sistema muestra un formulario con los datos actuales del producto	El formulario se activa para modificaciones
	Ingreso de nuevos datos	El sistema valida y guarda los cambios en la base de datos	El producto se actualiza en el catálogo con los nuevos detalles

Tabla 2: Creación de productos

Funcionalidad N°3: Categorización de productos

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F3	Categorización de productos	19/5/2025	19/5/2025	El catálogo mostrara los productos organizados por categorías	Media
	Entrada	**Proceso**		**Salida**	
	Ingreso a la sección de catálogo	El sistema recupera los productos desde la base de datos		Los productos se agrupan y muestran según su categoría correspondiente	
	Selección de una categoría específica	El sistema filtra los productos de esa categoría		Se muestran solo los productos que pertenecen a esa categoría	

Tabla 3: Categorización de productos

Funcionalidad N°4: Detalle del producto

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F4	Detalle del producto	19/5/2025	19/5/2025	Cada producto mostrara detalles disponibles para los clientes	Alta
	Entrada	**Proceso**		**Salida**	
	Selección o visualización de un producto	El sistema consulta la información del producto en la base de datos		Se muestra una vista con los detalles completos del producto	

Tabla 4: Detalle del producto

Funcionalidad N°5: Muestra de productos en stock

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F5	Muestra de productos en stock	19/5/2025	19/5/2025	El sistema permitirá mostrar únicamente los productos en stock, los productos que ya fueron vendidos se ocultaran o eliminaran de la base de datos.	Alta
	Entrada	**Proceso**		**Salida**	
	Consulta al catálogo	El sistema filtra los productos con stock disponible		Se muestran solo productos con cantidad mayor a cero	
	Actualización de stock tras una compra	El sistema descuenta la cantidad y verifica si el stock llega a cero		Si el producto no tiene stock, se oculta o se elimina del catálogo	

Tabla 5:Muestra de productos en stock

Funcionalidad N°6: Visualización de productos

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F6	Visualización de productos	19/5/2025	19/5/2025	El cliente podrá visualizar los productos sin necesidad de registrarse al sitio web	Alta
	Entrada	**Proceso**		**Salida**	
	Ingreso a la página web	El sistema carga el catálogo de productos disponibles		El cliente visualiza los productos sin requisitos de autenticación	

Tabla 6: Visualización de productos

Funcionalidad N°7: Selección de productos

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F7	Selección de productos	19/5/2025	19/5/2025	El cliente podrá seleccionar los productos y agregarlos a un carrito de compras	Alta
	Entrada	**Proceso**		**Salida**	
	Clic en el botón "agregar al carrito"	El sistema agrega el producto seleccionado a una lista temporal		El producto aparece en el carrito de compras con sus detalles	

Tabla 7:Selección de productos

Funcionalidad N°8: Generación de factura

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F 8	Generación de factura	19/5/2025	19/5/2025	El sistema generara una lista de os productos agregados al carrito y generara el total de a compra	Alta
	Entrada	**Proceso**		**Salida**	
	Productos agregados al carrito	Listado y suma del precio de los productos unitarios		Valor total de a suma de la compra	

Tabla 8: Generación de factura

Funcionalidad N°9: Revisión de actividad

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F9	Revisión de stock	19/5/2025	19/5/2025	Si un producto que el cliente selecciono se agota antes de completar la compra, el sistema	Media

				notificará al cliente y eliminará el producto del carrito	
	Entrada	**Proceso**		**Salida**	
	Verificación de stock al confirmar la compra	El sistema revisa la disponibilidad de cada producto en el carrito		Se notifica al cliente y se elimina el producto agotado del carrito	

Tabla 9: Revisión de actividad

Funcionalidad N°10: Proceso de pago

No.	**Nombre actividad**	**Fecha inicio**	**Fecha fin**	**Descripción**	**Prioridad**
F10	Proceso de pago	19/5/2025	19/5/2025	Al presionar el botón de pagar el sistema enviara automáticamente los detalles de la compra y redireccionara al chat de WhatsApp del vendedor	Alto

Entrada	**Proceso**	**Salida**
El cliente hace clic en una categoría disponible desde el menú o interfaz principal del catálogo.	El sistema busca en la base de datos todos los productos relacionados con la categoría seleccionada, los ordena y los prepara para ser mostrados.	Se presenta en pantalla una lista de productos correspondientes a la categoría elegida, incluyendo datos como nombre, precio y disponibilidad.

Tabla 10: Proceso de pago

Funcionalidad N°11: Coordinación de pago

No.	Nombre actividad	Fecha inicio	Fecha fin	Descripción	Prioridad
F11	Coordinación de pago	19/5/2025	19/5/2025	El vendedor podrá coordinar el pago y entrega del pedido directamente con el cliente	Alta
	Entrada	**Proceso**		**Salida**	
	El vendedor recibe una notificación al chat de WhatsApp del vendedor relacionada con un pedido generado por el cliente.	El vendedor revisa la información del cliente y del pedido, posteriormente se contactará con el cliente para definir la forma de pago y generar la venta.		Venta concluida	

Tabla 11: Coordinación de pago

1.1.2 Requisitos No Funcionales

a. Rendimiento

- El sistema deberá cargar el catálogo de los productos en el menor tiempo posible para garantizar una buena experiencia en el usuario.
- El sistema debe soportar alrededor de 50 clientes simultáneos en las primeras pruebas de los primeros prototipos.

b. Seguridad

- El sistema permitirá registrarse únicamente al administrador, quien podrá realizar la gestión de los productos dentro de la plataforma.

c. Usabilidad

- El catálogo será claro para los clientes, con un diseño ordenado para que sea fácil encontrar lo que buscan.

- Un panel del administrador sencillo de usar, con opciones claras para agregar, editar o borrar productos.
- Diseño responsivo para que sea adaptable a los distintos dispositivos de los clientes y del administrador.

d. Escalabilidad

- El sistema debe ser usar una arquitectura adaptable que permita una mejor comprensión para futuras mejoras.
- El sistema debe dejar que se agreguen más productos sin que se vuelva un caos.
- La base de datos (MongoDB Atlas) debe estar lista para manejar más pedidos si el negocio crece.

e. Compatibilidad

- La redirección a WhatsApp debe funcionar correctamente en los distintos dispositivos móviles o de escritorio

f. Almacenamiento

- El almacenamiento de los productos de forma eficiente usando servicios en la nube como MongoDB que sea adaptable a las tecnologías de producción.
- La base de datos debe ser capaz de manejar el almacenamiento que permitan realizar consultas rápidas.

1.2. Tecnologías y su Impacto en el Sistema

Para el desarrollo del sistema se ha seleccionado tecnologías que cumplen con los requerimientos planteados de rendimiento, facilidad en el uso y escalabilidad del proyecto:

1.2.1. Tecnologías para el FrontEnd

Debido a que el frontend corresponde a la parte visual del sistema, como es la interfaz en la cual interactúa directamente el usuario se implementaran las siguientes tecnologías:

- **HTML**

  Es un leguaje estándar para poder estructurar sitios web, lo que permite una mejor organización de componentes y elementos que componen la página.

- **CSS**

  Brindara estilos visuales al contenido HTML, de manera que las interfaces sean mas atractivas y adaptables a los distintos dispositivos que puedan ser usados.

- **JavaScript**

  Es el lenguaje de programación que se encargara de la interactividad dentro de la interfaz, además de ser los encargados de las funciones, validaciones y manejo del contenido del sistema.

1.2.2. Tecnologías para el BackEnd

- **Node.js**

  Es el entorno que se encarga de la ejecución del sistema, maneja la lógica dentro del BackEnd, permitiendo el procesamiento de las distintas funciones que requiere el proyecto en desarrollo.

- **Express.js**

  Es el Framework establecido que facilitara la creación de las rutas y controladores del sistema. Permite organizar las funcionalidades de CRUD requerido en el proyecto.

- **Mongoose**

  Es la librería que permite definir los esquemas para la estructura de datos en MongoDB.

- **Dotenv**

  Permite una gestión en las variables de entorno como la URL para la conexión de la base de datos y claves secretas para los tokens de autorización.

- **Json Web Token**

  Es una herramienta que se utilizo para manejar la generación y verificación de tokens, permitiendo la protección de datos.

- **Multer y Cloudinary**

  Multer se usa para procesar la subida de archivos desde formularios HTML, por otro lado, Cloudinary se utiliza para almacenar las imágenes de los productos en la nube.

- **Cors**

  Es el middleware que permite la comunicación entre el FrontEnd y BackEnd.

- **MongoDB Atlas**

  Se ha establecido el uso de esta base de datos por su capacidad almacenamiento, permitiendo guardar los productos e información del sistema.

1.3. Diseños de Mock Ups

En esta sección se muestra los prototipos de los diseños de las pantallas o interfaces del sistema para una mejor comprensión de cómo se lucirá y utilizará en el proyecto

1.3.1. Diseño de inicio de Sesión de Administrador

Tendrá espacios, uno para poner el correo y otro para la contraseña. Más abajo hay un botón Azul que dice "Ingresar. Todo está hecho para que se vea bien en celular o computadora, y sea súper fácil de usar, como pide el proyecto para que el vendedor entre rápido y empiece a manejar sus productos.

Ilustración 1: Inicio de sesión

1.3.2. Diseño de visualización de página principal del catalogo

Este diseño muestra la página principal del catálogo, donde los clientes ven los productos al entrar con el enlace del vendedor. Arriba tiene el nombre del emprendimiento y algunos datos importantes.

Se puede observar cómo hay categorías como: " Mujer", " Hombre" o "Cobijas", cada con los detalles. Al elegir una, se muestran solo los productos de esa categoría. En el centro, hay varios productos en cuadraditos, cada uno con su foto, nombre (por ejemplo, "Pantalón Negro"), precio y un botón de "Agregar al Carrito". Al final, un botón de "Ver Más" deja cargar más productos. Todo está ordenado y fácil de entender, para que los clientes encuentren lo que buscan rápido, como pide el proyecto.

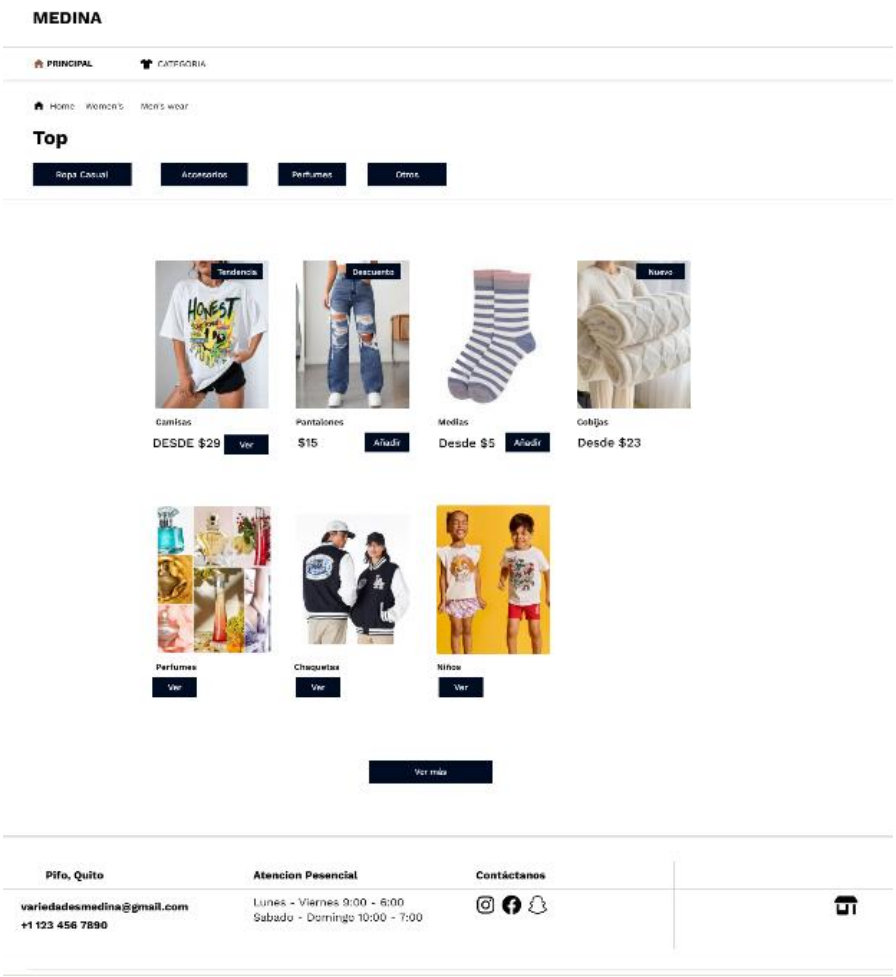

Ilustración 2: Página principal del catalogo

1.3.3. Diseño de Visualización de Categorías

En este diseño se muestra la página en donde se puede notar ms explícitamente las categorías, precios y detalles de los productos, además del carrito de compras de manera que los clientes lo encuentren fácil de manejar y fácil de hallar lo que buscan.

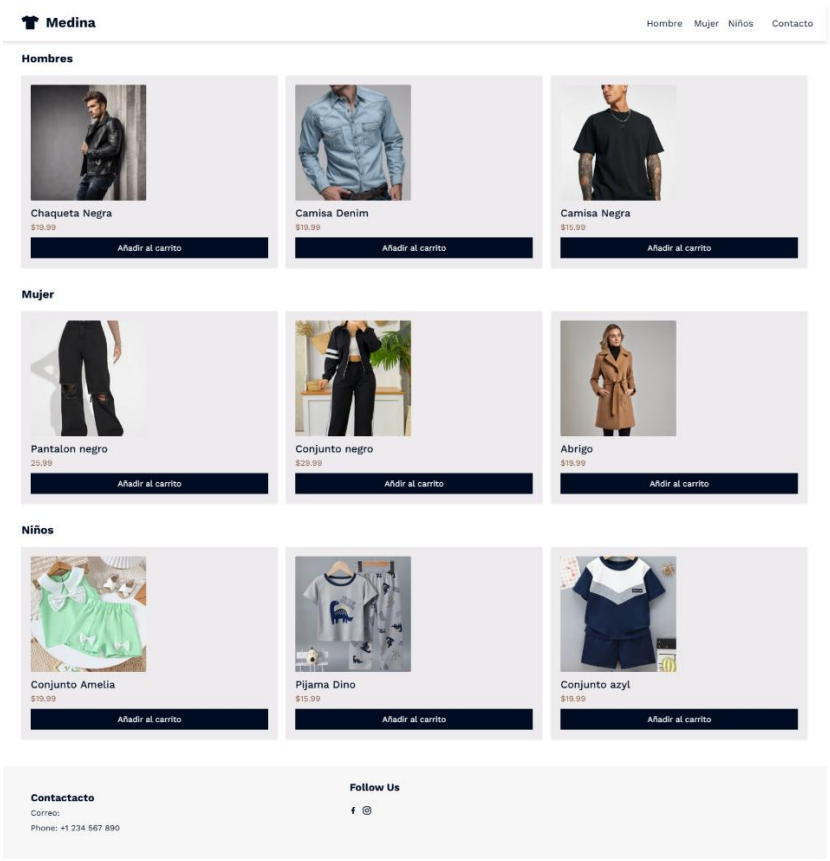

Ilustración 3: Visualización de Categorías

1.3.4. Diseño de Listado de productos y Total de Venta

Esta Interfaz muestra el carrito del cliente, en la parte izquierda se puede observar la lista de productos con todos los detalles como el nombre, precio y talla del producto.

Además de la opción de poder eliminar de la lista algún producto que ya no quiere llevárselo. El lado derecho podemos encontrar el valor total de la compra y un botón de "Pagar" que llevara al cliente al chat de la vendedora con el resumen.

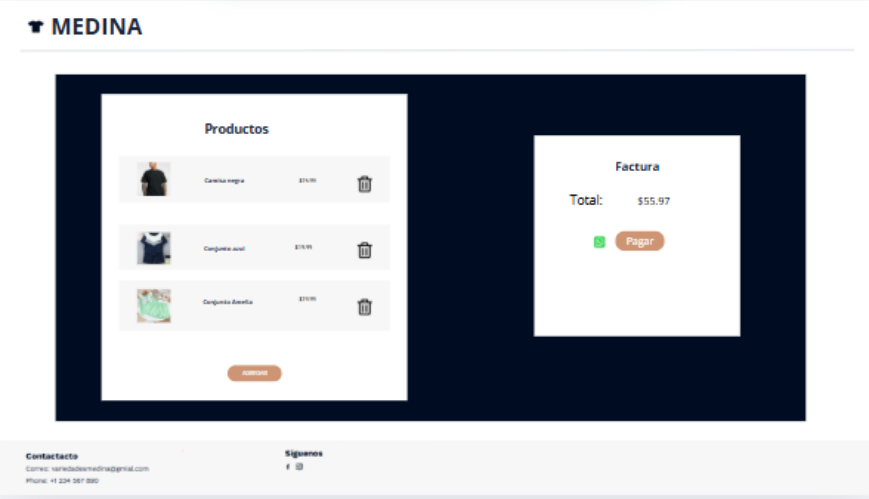

Ilustración 4: Listado de productos y Total de Venta

1.4. Diagramas

1.4.1. Diseño de Diagrama Entidad-Relación (ER)

En esta sección se presenta el diseño de la base de datos del catálogo web mediante un diagrama Entidad-Relación, en el que se define la estructura y relación de los datos para cumplir con las funcionalidades del sistema. Este modelo se implementará en MongoDB Atlas el cual se ha seleccionado para cumplir con las necesidades tanto del administrador comí de los clientes, alienándose con los requerimientos identificados previamente.

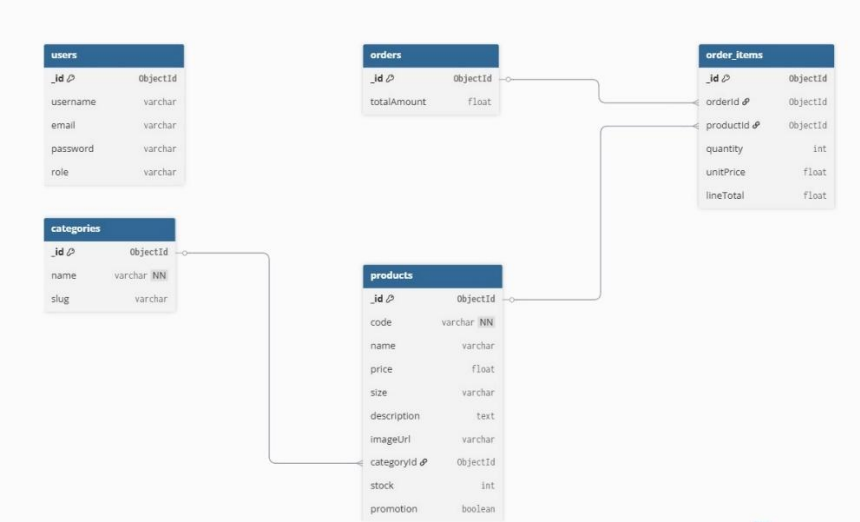

Ilustración 5. Diseño de Diagrama Entidad-Relación (ER)

El diagrama ER engloba siete entidades principales, cada una diseñada para soportar una funcionalidad específica del catálogo. A continuación, se detalla cada entidad, sus atributos y su propósito en el sistema.

1.4.1.1. Entidad: Usuario

Atributos:

- **id_usuario** (PK, ObjectId/String): Identificador único del usuario.
- **nombre** (String): Nombre del administrador.
- **correo** (String): Correo del administrador
- **contraseña** (String): Contraseña del administrador que estará encriptada.
- **rol** (String): Rol del usuario ("administrador").

Esta entidad representa al usuario administrador, el que se encargara de gestionar el catálogo, el **rol** permite asegurarse de que solo los administradores puedan acceder al panel de gestión.

1.4.1.2. Entidad Producto

Atributos:

- **id_producto** (PK, ObjectId/String): Identificador único del producto.

- **nombre** (String): Nombre del producto
- **descripcion** (String): Detalles del producto como talla, color, etc.
- **precio** (Double): Precio del producto.
- **código** (String): Codigo del producto.
- **talla** (String): Talla del producto.
- **stock** (Number): Cantidad disponible en inventario.
- **imagen_url** (String): Enlace a la foto del producto.
- **categoria_id** (FK, ObjectId/String): Identificador de la categoría a la que pertenece el producto.

Esta entidad se encarga de el almacenamiento de los productos que el administrador sube al catálogo. Los clientes podrán ver estos detalles en el catálogo.

1.4.1.3. Entidad Categoría

Atributos:

- **id_categoria** (PK, ObjectId/String): Identificador único de la categoría.
- **nombre_categoria** (String): Nombre de la categoría

Esta entidad organiza los productos en categorías permitiendo un catálogo más ordenado y fácil de navegar para los clientes.

1.4.1.4. Entidad OrdenItem

Atributos:

- **_id** (PK, ObjectId/String): Identificador único del ítem en laorden
- **Orden_id** (FK, ObjectId/String): Identificador de la orden al que pertenece.

- **producto_id** (FK, ObjectId/String): Identificador del producto que se agregó.
- **cantidad** (Number): Cantidad de unidades de ese producto en el carrito.
- **precio_unitario** (Double): Precio del producto

Esta entidad representa a cada producto que el cliente seleccione en el catálogo, la cantidad permite que el cliente pueda conocer cuantas unidades quedan de dicho producto y el catálogo asegura que el precio este visible para el cliente y permita a la plataforma calcular el valor de la compra/venta.

1.4.1.5. Entidad Orden

Atributos:

- **id_orden** (PK, ObjectId/String): Identificador único del pedido.
- **total** (Number/Decimal): Precio total del pedido.
- **estado_envio** (String): Estado del envío
- **cliente_whatsapp** (String): Número de WhatsApp del cliente

Esta entidad representa el pedido que se genera cuando el cliente presiona el botón pagar, mostrando el total de la compra calculando la suma de los ítems escogidos, además de la redirección del pedido al numero de la vendedora, así logrando un seguimiento del pedido.

1.4.2. Diagrama de casos de uso

Representación las funcionalidades principales del sistema web de catálogo desde la perspectiva de los usuarios. En este proyecto, se identifican dos actores principales: el

cliente, que navega y realiza pedidos**;** y el administrador, que gestiona productos y atiende pedidos.

Este diagrama permite visualizar qué acciones puede realizar cada usuario, sin entrar en detalles técnicos, facilitando la validación de los requerimientos funcionales y guiando el diseño del sistema

1.4.2.1. Actor Usuario Administrador

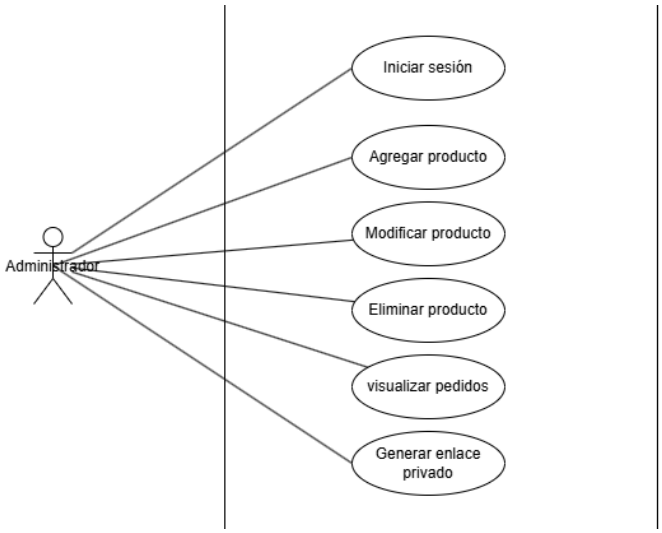

Ilustración 6 Diagrama de casos de uso (Administrador)

Acciones en el Sistema:

- **Entrar al panel**: El administrador inicia sesión con sus datos, asegurando acceso autorizado.
- **Crear producto**: Añade nuevos artículos al catálogo, como ropa o cobijas.
- **Editar producto:** Permite actualizar detalles de productos, manteniendo el catálogo al día.
- **Borrar producto**: Elimina productos que ya no se ofrecen, completando la gestión del catálogo.

- **Revisar pedidos**: Muestra los pedidos de los clientes para coordinar su entrega.
- **Crear enlace de acceso:** Genera un enlace para que los clientes vean el catálogo sin registrarse.

1.4.2.2. Actor Usuario Cliente

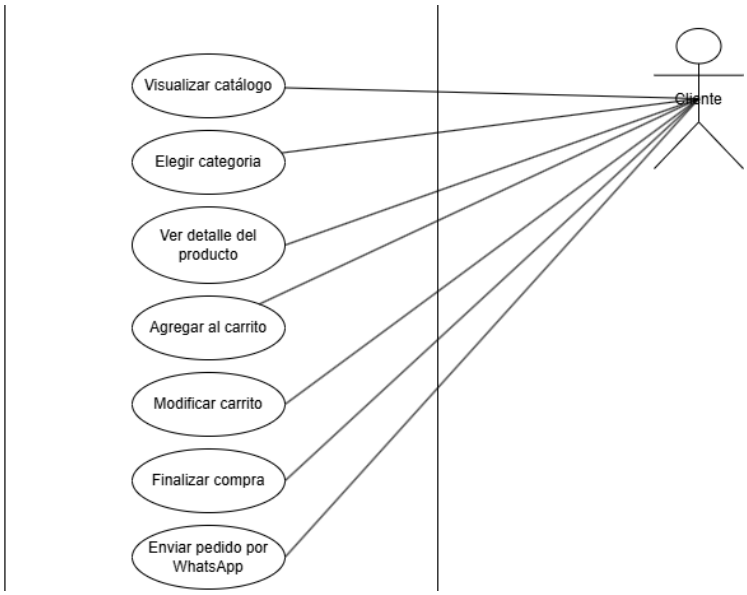

Ilustración 7. Diagrama de casos de uso (Cliente)

Acciones en el sistema

- **Visualizar catálogo:** El cliente puede ver la lista de productos disponibles.
- **Elegir categoría:** El cliente puede filtrar los productos por categorías.
- **Ver detalle del producto:** Puede consultar información más específica de un producto como el precio, talla, etc.
- **Agregar al carrito:** Puede seleccionar un producto y agregarlo a su carrito de compras.

- **Modificar carrito:** Puede editar las cantidades de productos o eliminar artículos del carrito.
- **Finalizar compra:** Acción para completar el proceso de compra
- **Enviar pedido por WhatsApp:** En lugar de una compra tradicional, el pedido puede enviarse directamente por WhatsApp para gestionar el pago o entrega.

1.4.3. Diagrama de clases UML

Se muestra la estructura general del sistema web de catálogo para una tienda de ropa, incluyendo las entidades del modelo de datos (MongoDB Atlas), los controladores del Backend (Node.js), las páginas del Frontend (JavaScript) y los servicios integrados (como WhatsApp).

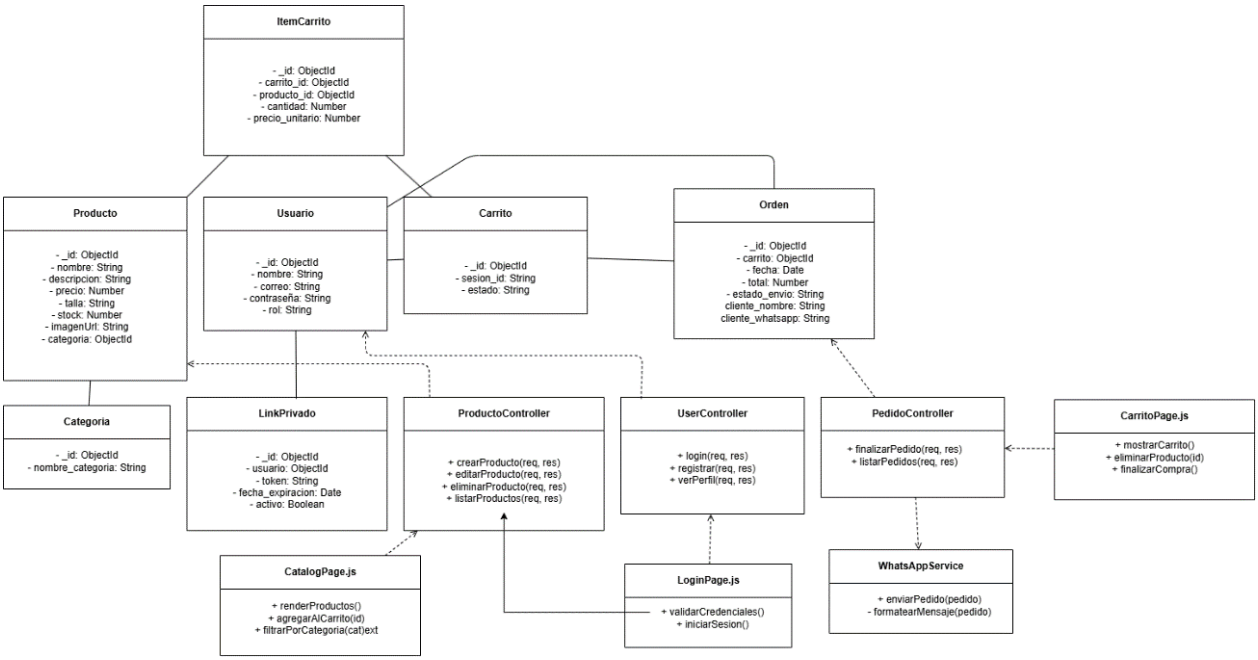

Ilustración 8: Diagrama de clases UML

Capítulo II

2.1 Construcción del Sistema FrontEnd para Administrador

2.1.1 Estándares de Construcción

En el desarrollo del FrontEnd del sistema para el Catálogo Web MEDINA, se eligió usar un enfoque modular mas organizado, separando responsabilidades con archivos HTML, CSS y JavaScript independientes. Esto permite que el código sea mucho más legible y fácil de mantener de manera escalar para ampliación en el futuro.

Las convenciones que se adoptaron para la codificación son:

- **Lenguaje:** Todo el código como funciones, variables, clases, nombres de archivos están escritos en inglés, para así poder asegurar una compatibilidad con documentación técnica y practicas globales de desarrollo.
- **Estructura de capas:** El proyecto está estructurado en carpetas principales ara así mantener un orden:

  - html/: Se encuentran los archivos de interfaces como login.html, admin.html.

  - css/: Se encuentran los estilos separados por pantalla, como login.css o admin.css.

  - js/: Se encuentran las funciones como el control de acceso, validaciones y manejo de DOM.

- **Nomenclatura de funciones:**
    - Para las funciones en JavaScript, se uso UpperCamelCase. Ejemplo: ValidateLogin, RenderProductList, UpdateSidebarCart.
    - Para variables de CSS, se opto por lowerCamelCase, como .productCard, adminContainer o cartTotalPrice.
- **Separación de responsabilidades**
    - **login.js :** maneja la lógica de validación y acceso al sistema.
    - **admin.js:** gestiona las operaciones del panel de control de administración, como la carga de productos al sistema.
    - **Main.js:** contiene funciones compartidas del proyecto.
- **Metodología Visual:**

    No se implementó o uso Frameworks como Bootstrap, sin embargo, se aplicó principios de diseño responsivo y reutilización de estilos en archivos css para mantener la coherencia y que la interfaz se vea clara, que sea de mantenimiento fácil y sea escalable a largo plazo.

2.1.2. Definición de Páginas

2.1.2.1. LoginPage

La página de inicio de sesión, implementada en el archivo login.html, constituye el punto de entrada principal para los usuarios administradores del sistema de catálogo web “Medina”. Su función primordial es garantizar un acceso seguro al panel de administración mediante la validación de credenciales.

Características principales:

- **Estructura HTML:** La interfaz se basa en un formulario simplificado que incluye dos campos obligatorios: un campo para el correo electrónico (email) y otro para la contraseña (password). Ambos campos están configurados como requeridos para prevenir el envío de formularios incompletos, asegurando la integridad de los datos ingresados.
- **Diseño visual**: Los estilos de la página se definen en el archivo css/login.css, adoptando un diseño centrado, minimalista y alineado con la identidad visual del sistema, lo que proporciona una experiencia estética y funcionalmente coherente.
- **Funcionalidades adicionales:**
  - Un campo de selección (checkbox) para la opción “Recordar sesión”, que permite mantener activa la sesión del usuario.
  - Un enlace para la recuperación de contraseñas, facilitando el acceso en caso de olvido de credenciales.

Lógica de funcionamiento:

El archivo js/login.js gestiona la validación de las credenciales y la interacción con el BackEnd. Este script implementa verificaciones previas en el lado del cliente y administra el token de sesión, que habilita la transición segura hacia la interfaz administrativa tras una autenticación exitosa.

Diseño visual:

La interfaz presenta un encabezado con el título “Medina”, seguido por los campos de ingreso y un botón principal etiquetado como “Ingresar”. Este diseño prioriza la claridad y la usabilidad, eliminando elementos que puedan distraer al usuario.

Usabilidad y accesibilidad:

La página fue diseñada con un enfoque en la simplicidad y la experiencia del usuario, garantizando un acceso rápido y eficiente al sistema. Se optimizó para ser accesible, con una disposición intuitiva que facilita la interacción sin comprometer la seguridad ni la funcionalidad.

Características estructurales y de diseño:

- **Estructura del formulario**:

  La página presenta un formulario optimizado que incluye dos campos obligatorios:

  - Un campo para el correo electrónico (**type="email"**), configurado para validar el formato del correo ingresado.
  - Un campo para la contraseña (**type="password"**), que protege la visibilidad de los datos introducidos.

  Ambos campos están marcados como obligatorios, utilizando validaciones nativas de HTML5 para garantizar que no se envíen formularios incompletos.

- **Funcionalidades complementarias:**

  - Una casilla de verificación (checkbox) para la opción de "Recordar sesión", que permite mantener la sesión activa para accesos posteriores.
  - Un enlace hipertextual para la recuperación de contraseñas, que dirige al usuario a un proceso de restablecimiento de credenciales en caso de olvido.

- **Diseño visual:** Los estilos de la interfaz se gestionan mediante el archivo css/login.css, que implementa un diseño centrado, minimalista y visualmente coherente con la identidad gráfica del sistema. La presentación es moderna y limpia, priorizando la usabilidad y eliminando elementos distractores para ofrecer una experiencia de usuario fluida.

Lógica operacional:

El archivo js/login.js contiene la lógica funcional que gestiona la interacción del formulario y la comunicación con el backend. El proceso de autenticación sigue los siguientes pasos:

- Al enviar el formulario, se utiliza el método preventDefault() para evitar la recarga automática de la página, optimizando la experiencia del usuario.
- Los valores de los campos de correo electrónico y contraseña se extraen mediante JavaScript.
- Se realiza una solicitud HTTP de tipo POST a la ruta de la API (api/auth/login), enviando las credenciales en formato JSON a través de la función fetch.
- Si la respuesta del servidor incluye un token válido basado en JSON Web Token (JWT):
    - El token se almacena en localStorage para gestionar la sesión del usuario.
    - El usuario es redirigido automáticamente a la interfaz administrativa (admin.html).

- En caso de error, como credenciales incorrectas, se muestra un mensaje de notificación al usuario mediante una alerta nativa, proporcionando retroalimentación inmediata sobre el problema.

Ventajas de la implementación:

- **Seguridad:** La utilización de JWT como mecanismo de autenticación proporciona un enfoque robusto para la gestión de sesiones, garantizando un control de acceso seguro.
- **Experiencia del usuario:** La prevención de recargas de página y la redirección automática optimizan la interacción, ofreciendo un flujo de navegación intuitivo y eficiente.
- **Modularidad y escalabilidad:** La estricta separación entre la estructura (HTML), el diseño visual (CSS) y la lógica funcional (JavaScript) facilita el mantenimiento del código y permite la incorporación de mejoras futuras, como validaciones avanzadas o integración con sistemas de autenticación más complejos, como autenticación multifactor.
- **Accesibilidad:** El diseño simplificado asegura que la interfaz sea accesible y fácil de usar, minimizando barreras para los usuarios administradores.

2.1.2.2. DashboardPage

La interfaz de administración, implementada en el archivo admin.html, representa el núcleo operativo del sistema de catálogo web "Medina", diseñada específicamente para usuarios con permisos administrativos. Esta página centraliza la gestión de los productos del

catálogo, ofreciendo un entorno funcional y eficiente para la administración de contenidos.

Características estructurales y de diseño:

- **Estructura visual:**
    - Barra lateral (sidebar): Ubicada en el lado izquierdo de la interfaz, permanece fija y contiene el logotipo del sistema junto con un menú de navegación que facilita el acceso a las principales funcionalidades.
    - Área principal (main): Diseñada para albergar secciones dinámicas que permiten visualizar, editar, crear y eliminar productos del catálogo.
- **Estilo personalizado:** Los estilos se gestionan mediante el archivo css/admin.css, que implementa un diseño responsivo adaptado a diferentes dispositivos. Se utilizan colores institucionales, sombras suaves y tipografías personalizadas importadas desde Google Fonts, asegurando una presentación visual moderna, coherente y profesional.

Lógica funcional:

El DashBoardPage brinda acceso a las operaciones interactivas de la página, incluyendo las siguientes funcionalidades:

- **Carga dinámica de datos:** Los productos existentes en el catálogo se obtienen automáticamente desde el backend y se renderizan en la interfaz mediante JavaScript, permitiendo una visualización actualizada.

- **Interacciones dinámicas:** La página incluye botones interactivos para acciones como “Agregar producto”, “Guardar cambios”, “Eliminar producto” y “Marcar como promoción”, que facilitan la gestión del catálogo.
- **Gestión de enlaces:** El sistema permite crear enlaces de catálogo personalizados, que pueden compartirse con clientes para promover los productos.
- **Gestión de categorías:** Se da un manejo y control de las categorías que existen dentro del sistema.
- **Manejo De WhatsApp:** en el que el administrado tiene acceso para enviar mensajes a los clientes a su elección, el cual contendrá el link del catalogo o enlaces que maneja personalmente.

Gestión de productos:

El panel administrativo ofrece un conjunto de herramientas para la gestión integral de productos, permitiendo al usuario registrar y modificar información detallada, incluyendo:

- Nombre del producto.
- Descripción.
- Precio.
- Talla.
- Categoría.
- Imagen asociada.

Estas operaciones se realizan a través de formularios dinámicos que se actualizan en tiempo real mediante JavaScript, optimizando la interacción y reduciendo la necesidad de recargas de página.

2.1.2.3.ProductFormPage

El formulario de productos forma parte integral de la página admin.html y está diseñado para permitir al administrador registrar y modificar productos del catálogo de ropa. Este formulario es dinámico y responde a las acciones del usuario según se trate de una creación nueva o una edición existente.

Componentes del formulario:

- **Campos principales:**
  - Nombre del producto
  - Descripción
  - Precio
  - Talla
  - Categoría
  - Imagen (archivo)
- **Botones de acción:**
  - **Guardar producto:** crea un nuevo producto o actualiza uno existente.
  - **Cancelar:** limpia el formulario y vuelve a la vista principal.
  - **Eliminar:** disponible cuando se edita un producto, permite eliminarlo permanentemente.
  - **Marcar como promoción / stock:** según el estado del producto.

Lógica funcional (admin.js):

- Al cargar la página, se ejecuta la función initAdmin() si el usuario está autenticado.
- El formulario se activa mediante un botón flotante (“+”) que abre el contenedor del formulario.
- Cuando el formulario se envía:
    - Se recopilan los datos desde el DOM.
    - Se empaquetan en un Form Data si hay imagen incluida.
    - Se envían al BackEnd mediante fetch a la ruta /api/products usando el método POST o PUT según corresponda.

Interacción con la base de datos:

- El formulario se comunica con una API RESTful protegida por tokens JWT.
- Los productos se guardan en una base de datos remota MongoDB.
- Al editar un producto, el sistema identifica el productId y rellena el formulario con los datos actuales.

2.1.2.4. ProductListPage

La **ProductListPage** forma parte de la interfaz principal del panel de administración (admin.html) y corresponde a la sección donde se visualizan todos los productos registrados en el sistema. Esta lista ofrece al administrador un acceso rápido a cada artículo para su gestión inmediata.

Presentación visual:

- Cada producto se representa como una tarjeta o contenedor individual con información clave:
    - Imagen del producto

    - Nombre
    - Precio
    - Talla
    - Categoría
  - Indicadores de stock o promoción
- Las tarjetas incluyen botones para:
    - **Editar**: carga el producto en el formulario para modificarlo.
    - **Eliminar**: borra permanentemente el producto.
- **Lógica funcional (admin.js):**

  •Al inicializarse la vista, se llama a una función que obtiene todos los productos desde la API mediante GET /api/products.

  •Los productos son renderizados dinámicamente usando JavaScript, con elementos creados y organizados en el DOM.

  •Cuando el usuario realiza una acción (como eliminar o editar), se captura el productId y se ejecutan llamadas a la API correspondientes (DELETE, PUT, etc.).

  Actualización en tiempo real:

  Después de cada operación (crear, editar, eliminar o cambiar estado), la lista se vuelve a cargar automáticamente para reflejar los cambios sin necesidad de recargar toda la página.

2.1.2.5. Tecnologías Utilizadas para FrontEnd de Administrador

FrontEnd:

- **HTML5:** utilizado para estructurar las páginas del sistema, permitiendo una base semántica clara y organizada.
- **CSS3:** define el diseño visual del sistema. Se usaron selectores personalizados, flexbox y media queries para lograr un diseño limpio y responsivo.
- **JavaScript (vanilla):** lenguaje principal para implementar la lógica del lado del cliente. Se manejan eventos, validaciones, manipulación del DOM y consumo de servicios REST mediante fetch.

Librerías y APIs externas:

- **Google Fonts:** se integró la familia tipográfica "Roboto" desde Google Fonts para un estilo moderno y profesional.
- **API REST (backend):** el frontend se comunica con una API construida en Node.js (según el contexto del proyecto), utilizando peticiones GET, POST, PUT y DELETE.
- **JSON Web Token (JWT):** se utiliza para validar sesiones. El token se guarda en localStorage y se envía en los headers de cada petición para proteger rutas sensibles.

Buenas prácticas adoptadas:

- Separación de responsabilidades (HTML para estructura, CSS para estilo, JS para lógica).
- Modularización del código JavaScript por archivo (login.js, admin.js).
- Convenciones de nombres en inglés, uso de UpperCamelCase para funciones y lowerCamelCase para variables y clases CSS.

2.2. Construcción del Sistema FrontEnd para Cliente

2.2.1. Estándares de Construcción

El desarrollo del sistema Frontend desarrollado para los clientes se realizo bajo criterios de simplicidad, eficaz y alto grado de compatibilidad para los múltiples dispositivos posibles.

A continuación, se detallará los estándares utilizados:

Lenguaje del código

Todo el código fuente del sistema cliente (HTML, CSS y JavaScript) está escrito en inglés, incluyendo nombres de funciones, variables, clases y archivos. Esto favorece la legibilidad técnica y permite escalar o integrar el proyecto con herramientas internacionales.

Estructura del proyecto

El sistema se organiza en tres carpetas principales:

- html/ → Contiene la interfaz principal catalog.html
- css/ → Contiene los estilos personalizados catalog.css
- js/ → Contiene la lógica del catálogo catalog.js

Tecnologías utilizadas

- **HTML5**: estructura semántica de la página del catálogo.
- **CSS3**: diseño responsive, colores personalizados, flexbox y estilo limpio.
- **JavaScript (vanilla)**: usado para:

- consumir la API de productos,
- manejar el estado del carrito,
- renderizar productos dinámicamente,
- preparar el enlace de WhatsApp.

Convenciones de codificación

- Funciones en **UpperCamelCase**, ej: addToCart, renderProducts.
- Variables y clases CSS en **lowerCamelCase**, ej: cartSidebar, productsGrid.
- Separación de lógica, estructura y estilos en archivos independientes.

Diseño responsive

Se aplicaron principios de diseño adaptable para garantizar la experiencia en dispositivos móviles. Esto incluye:

- Contenedores flexibles
- Ajustes automáticos de márgenes
- Vista del carrito lateral oculta en pantallas pequeñas

2.2.2. PublicCatalogPage

La vista catalog.html representa la interfaz principal para los clientes en el sistema web de la tienda Medina. Su objetivo es mostrar los productos disponibles de forma clara, permitir su exploración mediante filtros o búsqueda, y facilitar el proceso de compra utilizando WhatsApp como canal de contacto.

2.2.2.1 Estructura general

- **Cabecera fija** con el nombre de la tienda, un campo de búsqueda de productos y un botón de acceso al carrito.
- **Sección de categorías** que permite filtrar productos como “Hombre”, “Mujer”, “Niños”, “Ropa interior”, entre otros.
- **Grid de productos**: muestra tarjetas con imagen, nombre, precio y botón para añadir al carrito.
- **Carrito lateral (sidebar)**: se despliega cuando el cliente lo solicita, mostrando los productos añadidos, cantidades, subtotal y un botón de “Enviar pedido”.

2.2.2.2 Comportamiento dinámico (catalog.js)

- **Carga de productos** desde el backend mediante una solicitud GET /api/products, usando fetch.
- **Filtrado interactivo**:
  - Por categoría, mediante botones.
  - Por texto, desde la barra de búsqueda.
- **Carrito de compras local**:
  - Se almacena en localStorage mientras el usuario navega.
  - Permite añadir y eliminar productos, así como ajustar cantidades.
- **Generación de pedido**:

- El botón “Enviar pedido” genera un enlace a WhatsApp con un mensaje preformateado.
- El mensaje contiene el detalle del pedido, subtotal, y se envía al número del administrador.

2.2.2.3 Estilos (catalog.css)

- Uso de colores institucionales, bordes redondeados, sombreados suaves y diseño amigable para el usuario.
- Compatibilidad total con pantallas móviles.
- Tipografía moderna importada desde Google Fonts para mantener coherencia con la identidad visual de la tienda.

2.2.3 Tecnologías utilizadas

El FrontEnd del catálogo público de la tienda Medina fue desarrollado utilizando tecnologías web estándar, priorizando la compatibilidad con dispositivos móviles, la rapidez de carga y una experiencia de usuario intuitiva.

2.2.3.1 Tecnologías principales

- **HTML5**

Utilizado para construir la estructura semántica de la página principal (catalog.html). Incluye etiquetas accesibles, contenedores bien organizados y soporte para atributos responsivos.

- **CSS3**

Definido en **catalog.css,** se empleó para el diseño visual del catálogo. Se aplicaron:

- **Flexbox** para la distribución del grid de productos.
- **Media queries** para adaptar la interfaz a diferentes tamaños de pantalla.
- Paleta de colores personalizada y tipografía importada desde Google Fonts.

- **JavaScript (vanilla)**

Toda la lógica de interacción fue implementada sin frameworks. En el archivo catalog.js se encuentran:

- La carga de productos desde el backend (fetch a /api/products)
- El filtrado dinámico por categoría y texto
- La gestión del carrito de compras en memoria y en localStorage
- La generación automática del mensaje de pedido por WhatsApp

2.2.3.2 Integraciones externas

- **WhatsApp Web**

Se utilizó el esquema de URL oficial https://wa.me/ para abrir el chat de WhatsApp con el administrador, incluyendo el mensaje del pedido generado automáticamente desde el carrito.

- **Google Fonts**

Se cargó la fuente “Roboto” para mantener consistencia visual con el sistema de administración y mejorar la legibilidad.

2.2.3.3 Buenas prácticas implementadas

- Separación de responsabilidades (HTML / CSS / JS)
- Nombres de variables y funciones en inglés y formato camelCase
- Código comentado en puntos clave
- Uso de localStorage para mantener persistencia sin necesidad de backend adicional
- Reutilización de funciones como renderProducts() y updateCartUI() para mantener DRY (Don't Repeat Yourself)

2.3. Construcción del Sistema BackEnd

2.3.1 Estructura general del sistema

El sistema BackEnd fue desarrollado utilizando Node.js y el FramEwork Express.js, junto con MongoDB Atlas como base de datos NoSQL. Su estructura se organiza de manera modular para mantener una arquitectura limpia, escalable y fácil de mantener.

2.3.1.1 Estructura de carpetas principal:

- **controllers/:** contiene la lógica de negocio de cada entidad, como userController.js.
- **models**/: define los esquemas de Mongoose para las colecciones de la base de datos (User.js, Product.js).
- **middleware/:** funciones intermedias que protegen rutas (verifyToken.js, verifyAdmin.js).

- **config/:** configuración general del proyecto, como la conexión a la base de datos (db.js).
- **routes/:** define las rutas de la API REST (user, product, auth, etc.).
- app.js: archivo principal donde se configura el servidor, middlewares y rutas.
- .**env:** archivo de variables de entorno (como claves JWT o credenciales de la base de datos).

2.3.1.2 Estructura de ejecución:

Al iniciar el servidor con node app.js:

1. Se conecta a la base de datos remota MongoDB Atlas.
2. Se cargan las rutas de autenticación y gestión de productos.
3. Se habilita la recepción de datos en formato JSON y archivos (mediante multer).
4. El sistema queda listo para recibir solicitudes desde el frontend.

2.3.2 Estándares de construcción

Durante el desarrollo del sistema backend de Medina, se aplicaron estándares de codificación que permiten mantener un código limpio, legible, seguro y fácil de escalar. A continuación, se detallan los principales criterios adoptados:

2.3.2.1 Lenguaje y entorno

- El backend fue desarrollado con Node.js, utilizando JavaScript ES6+.

- Se utilizó Express.js como framework principal para la construcción de la API REST.
- El proyecto se ejecuta con nodemon en desarrollo y está preparado para ser desplegado en servicios como Render o Railway.

2.3.2.2 Estructura modular

Cada responsabilidad se encuentra separada en carpetas específicas:

- **controllers/** para la lógica de negocio
- **routes/** para definir endpoints
- **middleware/** para proteger rutas
- **models/** con esquemas de Mongoose
- **config/** para conexión a MongoDB

2.3.2.3 Seguridad y autenticación

- Se implementó JWT (JSON Web Token) para autenticar usuarios y proteger rutas.
- El token se valida mediante verifyToken.js, y los privilegios de administrador se controlan con verifyAdmin.js.
- Las claves secretas y URI de conexión están almacenadas en un archivo .env.

2.3.2.4 Convenciones de codificación

- Código escrito en inglés, incluyendo nombres de funciones, variables y archivos.
- Uso de camelCase para variables y funciones (getUserById, createProduct).
- Archivos y módulos nombrados con estilo **kebab-case** o **PascalCase** según el tipo (ej: userController.js).
- Funciones reutilizables y modularizadas para evitar duplicidad de lógica (DRY principle).

2.3.2.5 Comunicación con frontend

- Todas las rutas están estructuradas como **API RESTful**, devolviendo respuestas en formato JSON.
- Se emplea multer para la carga de imágenes de productos.
- Las rutas protegidas requieren token en el header (Authorization: Bearer <token>).

2.3.2.6 Buenas prácticas adoptadas

- Validación básica de datos antes de guardar en la base.
- Manejo de errores con respuestas estándar (códigos 200, 400, 401, 403, 500).
- Código comentado en secciones críticas para facilitar su comprensión.

2.3.3 Definición de rutas

El backend del sistema expone una API RESTful organizada en rutas públicas y rutas protegidas. Cada conjunto de rutas está definido en un archivo específico dentro de la carpeta routes.

A continuación, se describen los principales grupos de rutas implementados:

2.3.3.1 Rutas públicas

Estas rutas están disponibles sin autenticación. Se utilizan principalmente para iniciar sesión y consultar productos.

- **POST /api/auth/login**

Permite a un usuario autenticarse. Devuelve un token JWT si las credenciales

son válidas.

Definido en auth.js.

- **GET /api/products**

Retorna todos los productos activos del catálogo, visibles para cualquier usuario (vista cliente).

Definido en product.js.

2.3.3.2 Rutas protegidas

Estas rutas requieren un token JWT válido para acceder. Algunas también requieren permisos de administrador.

Autenticación:

- **GET /api/auth/verify-token**

Verifica si el token enviado es válido y devuelve información del usuario autenticado.

Definido en auth.js.

Usuario:

- **POST /api/users**

Crea un nuevo usuario administrador (registrado solo por otros admins o en desarrollo).

Definido en user.js.

Productos:

- **POST /api/products**

Registra un nuevo producto. Requiere autenticación y rol de administrador.

- **PUT /api/products/:id**

Edita los datos de un producto existente.

- **DELETE /api/products/:id**

Elimina un producto del sistema.

- **PATCH /api/products/:id/status**

Permite marcar un producto como stock o promoción.

Todas estas rutas están definidas en product.js y protegidas por los middlewares verifyToken y verifyAdmin.

2.3.4 Controladores

Los controladores son responsables de manejar la lógica de negocio de las rutas del backend. Se encargan de recibir los datos de las peticiones, validarlos, procesarlos y devolver una respuesta adecuada al cliente.

userController.js

Este archivo contiene las funciones asociadas al manejo de usuarios. Su funcionalidad principal es el **registro de nuevos administradores**.

- **createUser(req, res)**

Esta función:

- Recibe los datos del nuevo usuario desde **req.body** (nombre, correo, contraseña).
- Verifica si el usuario ya existe en la base de datos.
- Encripta la contraseña usando **bcrypt.**
- Guarda el nuevo usuario en MongoDB utilizando el modelo

User.

- Devuelve una respuesta con los datos del usuario creado (sin contraseña).

Se aplica una lógica segura y clara que incluye validación básica y hashing de contraseñas antes de almacenar los datos en la base de datos.

Los controladores para productos o autenticación están integrados directamente en los archivos de rutas como **product.js y auth.js**, en lugar de estar desacoplados. Si se decide escalar el proyecto, se recomienda trasladar esa lógica a controladores independientes para mantener la estructura modular.

2.3.5 Modelo de Datos

El sistema backend utiliza **Mongoose** para definir la estructura de los documentos almacenados en la base de datos MongoDB Atlas. Los dos modelos principales que se manejan en el sistema son Product y User.

2.3.5.1 Modelo Product

Este modelo representa los productos que se gestionan desde el panel de administración y se visualizan en el catálogo público. Se encuentra en el archivo Product.js.

Los campos definidos en el modelo son:

- **name**: nombre del producto (tipo String, requerido).
- **price**: precio del producto (tipo Number, requerido, mínimo 0).
- **code** : tiene el código único del producto (tipo String, requerido)

- **size**: talla del producto (tipo String, requerido).
- **description**: descripción opcional (tipo String, no requerido).
- **imageUrl**: arreglo de URLs de imágenes (tipo [String], requerido).
- **category**: categoría del producto (tipo String, requerido).
- **stock**: cantidad disponible (tipo Number, requerido, mínimo 0).

También se incluye la opción timestamps, lo que permite registrar automáticamente las fechas de creación y actualización del producto.

2.3.5.2 Modelo User

Este modelo representa a los usuarios del sistema, específicamente administradores, y se encuentra en el archivo User.js.

Los campos definidos son:

- **username**: nombre de usuario (tipo String, requerido).
- **email**: correo electrónico (tipo String, requerido).
- **password**: contraseña encriptada (tipo String, requerido).
- **role**: rol del usuario, puede ser 'admin' o 'user' (tipo String, con valor por defecto 'user').

Este modelo implementa un middleware de pre-guardado (pre('save')) que se encarga de encriptar la contraseña con bcrypt antes de almacenarla, siempre que el usuario sea nuevo o la contraseña haya sido modificada.

2.3.6 Middlewares

En el sistema backend de Medina se implementan middlewares personalizados para proteger las rutas sensibles y garantizar que solo los usuarios autenticados y autorizados puedan realizar ciertas acciones, especialmente aquellas relacionadas con la administración del catálogo.

2.3.6.1 verifyToken.js

Este middleware se encarga de validar que la solicitud incluya un **token JWT** válido en el encabezado **Authorization.** Su funcionamiento es el siguiente:

- Verifica que el encabezado tenga el formato correcto: **Bearer <token>.**
- Si el token es válido, lo decodifica utilizando la clave secreta definida en **env. JWT_SECRET.**
- Extrae del token la información del usuario (ID, correo y rol) y la almacena en **user** para que esté disponible en los siguientes controladores.
- Si el token está ausente, mal formado, vencido o inválido, retorna un código de estado 401 y un mensaje de error descriptivo.

Este middleware se aplica en rutas que requieren autenticación, como la creación o edición de productos.

2.3.6.2 verifyAdmin.js

Este middleware se encarga de verificar que el usuario autenticado tenga el rol de **administrador**. Funciona de la siguiente manera:

- Evalúa el objeto **user** (que debe haber sido establecido previamente por **verifyToken**).
- Si el rol no es **admin**, devuelve un error **403 Forbidden** indicando que el acceso está denegado.
- Si el usuario sí es administrador, permite que la solicitud continúe.

Este middleware se usa en conjunto con **verifyToken** para restringir operaciones sensibles como eliminar productos, crear nuevos o marcar productos en stock o promoción.

2.3.7 Conexión con base de datos

El sistema se conecta a una base de datos remota utilizando MongoDB Atlas, una solución en la nube que permite gestionar bases de datos NoSQL con alta disponibilidad y escalabilidad.

2.3.7.1 Proceso de conexión

La conexión a la base de datos se realiza mediante la librería **Mongoose**, que proporciona una interfaz amigable para interactuar con MongoDB. Aunque el archivo **db.js** no contiene configuración explícita en el momento actual, lo usual es que allí se defina y exporte la función de conexión.

En el archivo principal **app.js,** se importa esta configuración y se invoca la conexión utilizando una URI que está almacenada como variable de entorno en el archivo .**env.**

2.3.7.2 Seguridad

Para evitar exponer información sensible como contraseñas o URIs de conexión, se utiliza un archivo **.env** que contiene, por ejemplo:

```
MONGODB_URI=mongodb+srv://<usuario>:<contraseña>@<cluster>.mongodb.net/medinaDB
JWT_SECRET=claveSecretaParaToken
```

Esta URI es leída dentro del código mediante **.env (MONGODB_URI)** y pasada a **mongoose.connect(),** permitiendo una conexión segura y desacoplada del código fuente.

2.3.7.3 Estabilidad

Una vez conectada, la aplicación puede realizar operaciones de lectura, escritura, actualización y eliminación sobre las colecciones de products y users. Además, Mongoose maneja internamente la reconexión automática en caso de pérdida de conexión temporal.

Anexos

Repositorio de Github del BackEnd:

https://github.com/JossPS/Medina_Backend

Repositorio de GitHub del FrontEnd(Cliente y Administrador)

https://github.com/JossPS/Medina_Frontend

Capítulo III

3.1. Pruebas y Estabilización FrontEnd Administrador

3.1.1. Inicio De Sesión (Administrador)

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida el correcto funcionamiento del formulario de inicio de sesión del administrador en la interfaz web. Se busca comprobar que los campos sean completados, que al hacer clic en "Iniciar sesión" se procese la información, y que la interfaz reaccione adecuadamente según el rol del usuario.	
Precondiciones	• El navegador debe estar abierto y cargado con la interfaz de inicio de sesión (/login). • El usuario debe estar registrado previamente con credenciales válidas. • El frontend debe tener conexión activa con el backend (API) para procesar el inicio de sesión.
Pasos para seguir	1. Acceder a la URL del sistema: http://localhost:9000/login 2. Ingresar el correo del administrador en el campo correspondiente. 3. Ingresar la contraseña en el campo correspondiente. 4. Presionar el botón "Iniciar sesión". 5. Observar si se muestra mensaje de validación, redirección o errores.
Datos de Entrada	• Detalle los datos de entrada Correo: admin@medina.com • Contraseña: admin123
Resultados Esperados	**Resultados Esperados** • El formulario valida que ambos campos estén llenos. • Se envía una solicitud POST al backend. • Si las credenciales son válidas, se almacena el token en localStorage. • Se redirige automáticamente al **panel de administrador** (/admin).
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** Todos los campos fueron llenados correctamente y se redirige al panel sin errores. • **RECHAZO:** Campos vacíos, credenciales incorrectas o no se produce redirección.
Desarrollador Asignado	Josselyn Simbaña

Tabla 12 Inicio De Sesión (Administrador)

3.1.2 Registro de un nuevo producto desde el panel de administrador

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa el funcionamiento del formulario de registro de productos en el panel del administrador. Se verifica que los campos requeridos sean completados correctamente, que se seleccione una imagen válida y que al presionar el botón de guardar, se genere la solicitud al backend y se actualice la interfaz.	
Precondiciones	• El administrador debe haber iniciado sesión correctamente. • El sistema debe estar en la ruta del panel (/admin). • El formulario debe estar visible y operativo. • Debe existir conexión activa con el backend.
Pasos para seguir	1. Acceder al panel del administrador. 2. Dirigirse a la sección "Agregar producto". 3. Llenar los campos: nombre, precio, categoría, stock. 4. Seleccionar una imagen desde el equipo local. 5. Presionar el botón “Guardar”. 6. Verificar el mensaje de éxito y visualización del producto en la lista.
Datos de Entrada	• Nombre: Camiseta Oversize • Precio: 12.99 • Categoría: Mujer • Stock: Disponible • Imagen: camiseta.jpg
Resultados Esperados	• Todos los campos son validados en el frontend antes de enviar. • El archivo de imagen es procesado y adjuntado correctamente. • Se muestra una alerta o notificación de éxito. • El nuevo producto aparece en la lista de productos del panel.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • El formulario fue completado, se envió correctamente la solicitud y el producto aparece en la lista. **RECHAZO:** • Campos vacíos, imagen no seleccionada, mensaje de error o producto no agregado a la lista.
Desarrollador Asignado	Josselyn Simbaña

Tabla 13 Registro de un nuevo producto desde el panel de administrador

3.1.3 Edición de un producto existente desde el panel de administrador

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el administrador puede modificar la información de un producto ya registrado. Se valida que los campos se carguen correctamente en el formulario, que puedan editarse, y que al guardar los cambios se actualice la información en pantalla.ste caso evalúa el funcionamiento del formulario de registro de productos en el panel del administrador. Se verifica que los campos requeridos sean completados correctamente, que se seleccione una imagen válida y que al presionar el botón de guardar, se genere la solicitud al backend y se actualice la interfaz.	
Precondiciones	• El administrador debe estar autenticado. • El producto debe existir en el sistema previamente. • El botón de "Editar" debe estar habilitado en la lista de productos. • El backend debe estar activo y respondiendo solicitudes.
Pasos para seguir	1. Iniciar sesión como administrador y acceder al panel. 2. Localizar un producto en la lista y hacer clic en el botón "Editar". 3. Verificar que se cargue el formulario con los datos actuales del producto. 4. Modificar los campos deseados (ej. nombre, precio, categoría). 5. Presionar el botón "Guardar cambios". 6. Observar si se actualiza correctamente en la lista.
Datos de Entrada	• **Producto original:** Nombre: Camiseta Oversize Precio: 12.99 • **Cambios aplicados:** Nombre: Camiseta Oversize Blanca Precio: 13.50
Resultados Esperados	• El formulario de edición se carga con los datos actuales del producto. • Los nuevos valores se validan antes del envío. • Se muestra mensaje de éxito tras la edición. • La lista de productos se actualiza con los nuevos datos.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** Los cambios se reflejan inmediatamente en la interfaz sin errores. **RECHAZO:** • Error al guardar, cambios no aplicados, mensaje de validación fallido.
Desarrollador Asignado	Josselyn Simbaña

Tabla 14 Edición de un producto existente desde el panel de administrador

3.1.4 Eliminación de un producto desde el panel de administrador

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el administrador puede eliminar un producto existente desde la interfaz del panel. Se verifica que el botón de eliminación funcione correctamente, que se confirme la acción y que el producto desaparezca de la lista después de ser eliminado.	
Precondiciones	• El administrador debe haber iniciado sesión correctamente. • Debe existir al menos un producto previamente registrado en el sistema. • La lista de productos debe estar visible en el panel. • El backend debe estar activo y permitir eliminar productos.
Pasos para seguir	1. Ingresar al panel de administrador. 2. Buscar un producto en la lista. 3. Presionar el botón "Eliminar" correspondiente al producto. 4. Confirmar la acción en el mensaje de confirmación (si aplica). 5. Verificar que el producto desaparece de la lista tras la eliminación.
Datos de Entrada	**Producto a eliminar:** • Nombre: Camiseta Oversize Blanca • ID interno: (obtenido por el sistema, no visible al usuario)
Resultados Esperados	• Se dispara una alerta o confirmación antes de eliminar. • El producto es eliminado visualmente del DOM. • Se muestra un mensaje de éxito. • La API devuelve respuesta exitosa (status 200 OK).
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • Producto eliminado de la interfaz sin errores y confirmado por mensaje. **RECHAZO:** • El producto permanece en la lista, no se confirma la eliminación, o se presenta error visual.
Desarrollador Asignado	Josselyn Simbaña

Tabla 15 Eliminación de un producto desde el panel de administrador

3.1.5 Visualización de la lista de productos registrados en el panel de administrador

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que la interfaz del administrador cargue correctamente todos los productos almacenados en el sistema. Se evalúa si la información es presentada en forma de lista o tarjeta y que cada producto muestre sus datos principales.	
Precondiciones	• El administrador debe haber iniciado sesión correctamente. • Deben existir productos previamente registrados en el sistema. • El backend debe estar activo y responder la solicitud GET /api/products. • El frontend debe estar conectado correctamente con la API.

Pasos para seguir	1. Iniciar sesión como administrador. 2. Acceder al panel principal (/admin). 3. Esperar la carga automática de la lista de productos. 4. Verificar que se rendericen visualmente los productos disponibles. 5. Revisar que se muestren: nombre, precio, categoría, imagen y estado.
Datos de Entrada	• No aplica datos de entrada manuales. • El sistema consulta automáticamente los productos desde la base de datos.
Resultados Esperados	• Se envía una solicitud GET a la API de productos. • La respuesta es procesada correctamente por el frontend. • Todos los productos se renderizan en el panel. • Cada producto muestra sus atributos clave.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • Los productos se muestran correctamente en pantalla al cargar el panel. **RECHAZO:** • No se muestra ningún producto, error de conexión o información incompleta.
Desarrollador Asignado	Josselyn Simbaña

Tabla 16 Visualización de la lista de productos registrados en el panel de administrador

3.2. Pruebas y Estabilización FrontEnd Cliente

3.2.1 Visualización de productos al acceder al catálogo del cliente

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que, al ingresar al enlace público del catálogo, el sistema cargue correctamente los productos disponibles. Se espera que el catálogo sea funcional y que los productos aparezcan con su imagen, nombre, precio, y estado (ej. en promoción o sin stock).	
Precondiciones	• El administrador debe haber registrado previamente productos en el sistema. • El backend debe estar corriendo y disponible para la ruta pública (/api/products). • El cliente debe tener conexión a internet y acceder al enlace correcto del catálogo (/catalog.html)
Pasos para seguir	1. Abrir un navegador web e ingresar a la URL pública del catálogo. 2. Esperar la carga automática de los productos. 3. Observar cómo se renderiza la interfaz de productos. 4. Verificar que se muestren las tarjetas o bloques con cada producto.
Datos de Entrada	No se ingresan datos manuales; el sistema carga automáticamente los productos desde la base de datos.
Resultados Esperados	• Se realiza una solicitud GET al backend (/api/products). • Los productos se muestran en pantalla, incluyendo: Nombre Imagen Precio Estado (promoción, agotado, disponible)

	• El diseño es responsive y limpio.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • La lista de productos aparece correctamente al cargar la página. **RECHAZO:** • No se muestra ningún producto, la página queda vacía o aparecen errores de carga**.**
Desarrollador Asignado	Josselyn Simbaña

Tabla 17 Visualización de productos al acceder al catálogo del cliente

3.2.2 Filtrado de productos por categoría en el catálogo público

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que al hacer clic sobre una categoría del catálogo, se filtren y muestren únicamente los productos que pertenecen a dicha categoría. El objetivo es facilitar la navegación del cliente a través del catálogo, mejorando la experiencia de usuario.	
Precondiciones	• La página del catálogo debe estar cargada correctamente (/catalog.html). • Deben existir productos asignados a diferentes categorías. • El backend debe haber entregado la lista completa de productos al frontend.
Pasos para seguir	1. Ingresar a la página pública del catálogo. 2. Esperar que se cargue la lista de productos. 3. Localizar la barra o menú de categorías. 4. Hacer clic en una categoría (ej. “Ropa de mujer”). 5. Verificar que solo los productos de esa categoría se muestren en pantalla.
Datos de Entrada	Categoría seleccionada: Ropa de mujer.
Resultados Esperados	• La interfaz actualiza los productos en pantalla sin recargar la página. • Solo se muestran los productos cuya categoría coincide con la seleccionada. • El filtro visual se resalta o indica la categoría activa. • Si no hay productos en esa categoría, se muestra un mensaje tipo: “No hay productos disponibles”.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** El sistema filtra correctamente los productos al seleccionar una categoría. **RECHAZO:** • La lista no cambia, se muestran productos incorrectos o no responde el clic.
Desarrollador Asignado	Josselyn Simbaña

Tabla 18 Filtrado de productos por categoría en el catálogo público

3.2.3 Búsqueda de productos por palabra clave en el catálogo público

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el buscador del catálogo permite filtrar productos en tiempo real según las palabras escritas por el cliente. La funcionalidad mejora la experiencia al permitir encontrar productos específicos sin navegar manualmente por todo el catálogo.	
Precondiciones	• La página del catálogo (/catalog.html) debe estar cargada completamente. • Deben existir productos cuyos nombres coincidan con palabras clave comunes. • El buscador debe estar visible en la parte superior o en una sección accesible.
Pasos para seguir	1. Ingresar a la página del catálogo. 2. Localizar el campo de búsqueda. 3. Escribir una palabra clave relacionada con un producto (ej. "camiseta"). 4. Observar cómo se actualiza la lista de productos en pantalla. 5. Borrar o cambiar la palabra para validar que los resultados se ajustan dinámicamente.
Datos de Entrada	Palabra clave ingresada: camiseta
Resultados Esperados	• Se filtran automáticamente los productos en pantalla mientras se escribe. • Solo se muestran los productos que contienen la palabra "camiseta" en el nombre (o descripción, si aplica). • Si no hay coincidencias, se muestra un mensaje como: "No se encontraron resultados".
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • El filtrado se realiza en tiempo real y muestra productos relevantes. **RECHAZO:** • La lista no cambia, muestra productos incorrectos o no responde a la búsqueda.
Desarrollador Asignado	Josselyn Simbaña

Tabla 19. Búsqueda de productos por palabra clave en el catálogo público

3.2.4 Agregar un producto al carrito desde el catálogo público

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que los clientes puedan añadir productos al carrito correctamente desde la vista del catálogo. Se comprueba que al hacer clic en el botón correspondiente, el producto se agregue al carrito visual, se actualice el contador y pueda consultarse posteriormente.	
Precondiciones	• El catálogo debe estar cargado y mostrar productos disponibles. • Cada tarjeta de producto debe tener un botón o icono de "Agregar al carrito". • El carrito debe estar implementado y visible (ícono o sidebar). • El producto debe tener stock disponible.
Pasos para seguir	1. Acceder a la página del catálogo. 2. Localizar un producto en la lista. 3. Hacer clic en el botón "Agregar al carrito" de ese producto. 4. Observar que el contador del carrito se incrementa.

	5. Abrir el carrito para verificar que el producto se encuentra en la lista interna.
Datos de Entrada	Producto seleccionado: Camiseta Oversize Cantidad esperada en el carrito: 1 unidad
Resultados Esperados	• El carrito incrementa su contador visual (ej. de 0 a 1). • El producto aparece listado dentro del carrito (nombre, cantidad, subtotal). • Si el mismo producto se agrega varias veces, la cantidad se incrementa correctamente. • Se muestra el total acumulado.
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • El producto se agrega al carrito, se actualiza el contador y se muestra correctamente en el resumen. **RECHAZO:** • No cambia el contador, el producto no se refleja en el carrito o da errores de visualización.
Desarrollador Asignado	Josselyn Simbaña

Tabla 20. Agregar un producto al carrito desde el catálogo público

3.2.5 Visualización del carrito y modificación de productos seleccionado

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso comprueba que los usuarios puedan acceder al contenido del carrito, ver los productos añadidos, modificar cantidades o eliminar artículos antes de confirmar su pedido. Se valida que la interfaz responda correctamente a los cambios y actualice el total de la compra.	
Precondiciones	• El cliente debe haber agregado uno o más productos al carrito. • El botón o ícono del carrito debe estar visible y funcional. • El panel del carrito debe poder abrirse sin recargar la página. • El carrito debe tener funciones para modificar cantidades o eliminar productos.
Pasos para seguir	1. Ingresar a la página del catálogo. 2. Agregar dos o más productos al carrito. 3. Hacer clic en el ícono del carrito para desplegar el contenido. 4. Revisar que aparezcan los productos añadidos con sus cantidades y subtotales. 5. Cambiar la cantidad de un producto y verificar la actualización del total. 6. Eliminar un producto del carrito. 7. Confirmar que el total general se actualiza en tiempo real.
Datos de Entrada	• Productos añadidos: Camiseta Oversize x2 Jeans Azul x1 • Acción: Cambiar Camiseta a x3 Eliminar Jeans
Resultados Esperados	• Los productos aparecen con su nombre, cantidad, precio unitario y subtotal. • Al cambiar la cantidad, se actualiza el subtotal del producto y el total del carrito.

	• Al eliminar un producto, desaparece del carrito y se recalcula el total. • Si el carrito queda vacío, se muestra un mensaje del tipo "Tu carrito está vacío".
Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • El contenido del carrito se muestra correctamente y responde a los cambios en tiempo real. **RECHAZO:** • El carrito no se despliega, no actualiza cantidades o da errores al eliminar productos.
Desarrollador Asignado	Josselyn Simbaña

Tabla 21. Visualización del carrito y modificación de productos seleccionados

3.2.6 Envío del contenido del carrito a través de WhatsApp

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que al presionar el botón de "Enviar pedido por WhatsApp", se genere correctamente un mensaje con el resumen de los productos seleccionados, incluyendo cantidades, nombres y total. El sistema debe abrir una pestaña con WhatsApp Web (o app móvil) prellenada con ese contenido y dirigida al número de la tienda.	
Precondiciones	• El cliente debe haber agregado productos al carrito. • El número de WhatsApp de la tienda debe estar configurado correctamente en el archivo JS (WHATSAPP_NUMBER). • El dispositivo del cliente debe tener acceso a WhatsApp Web o la aplicación móvil.
Pasos para seguir	1. Acceder al catálogo (/catalog.html) y agregar varios productos al carrito. 2. Ir al carrito y verificar que el contenido esté correcto. 3. Hacer clic en el botón "Enviar pedido por WhatsApp". 4. Observar si se abre WhatsApp Web o la app con el mensaje preformateado. 5. Validar que el mensaje incluya: 6. Nombre de los productos 7. Cantidades 8. Precio de cada producto 9. Precio total de la compra
Datos de Entrada	• Productos en el carrito: + Camiseta Blanca x2 – $13.00 + Jean Azul x1 – $18.00 • Total: $44.00 • WhatsApp de la tienda: +593 0987654321
Resultados Esperados	• Se abre una nueva pestaña con una URL de Whatsapp • El mensaje contiene el resumen claro del carrito. • El número es el correcto y permite continuar la conversación.

Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** El mensaje se genera correctamente, se abre WhatsApp y el contenido está bien estructurado. **RECHAZO:** • El botón no hace nada, abre WhatsApp pero con mensaje incompleto o al número incorrecto.
Desarrollador Asignado	Josselyn Simbaña

Tabla 22. Envío del contenido del carrito a través de WhatsApp

3.3. Pruebas y Estabilización BackEnd

3.3.1. LOGIN EXITOSO

3.3.1.1. Inicio de sesión con credenciales válidas

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que el sistema permita el inicio de sesión de un administrador cuando se proporcionan credenciales válidas. Se espera una respuesta con un token JWT y los datos del usuario autenticado.	
Precondiciones	• El usuario administrador debe estar previamente registrado en la base de datos con correo y contraseña válidos. • El servidor debe estar corriendo y conectado a la base de datos MongoDB. • Ruta activa: POST /api/auth/login
Pasos para seguir	1. Enviar una solicitud POST al endpoint /api/auth/login. 2. En el body, enviar las credenciales válidas en formato JSON 3. Esperar por la respuesta desde el servidor.

Datos de Entrada	{
	"email":"admin@medina.com",
	"password":"admin123"
	}

Resultados Esperados	• Código HTTP: 200 OK
	• Respuesta JSON contiene:
	{
	"token": "<JWT generado>",
	"user": {
	"_id": "...",
	"email": "admin@medina.com",
	"role": "admin"
	}
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** Se recibe un token válido, junto con los datos del usuario autenticado. **RECHAZO:** • El token no se genera, la respuesta está incompleta, o los datos no son los correctos.
Desarrollador Asignado	Josselyn Simbaña

Tabla 23. Inicio de sesión con credenciales válidas

3.3.2. LOGIN CON CREDENCIALES INVALIDAS

3.3.2.1 Intento de inicio de sesión con credenciales inválidas

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema rechace correctamente los intentos de inicio de sesión con datos incorrectos, ya sea un correo no registrado o una contraseña inválida. Se espera que no se genere token y que se informe adecuadamente al usuario.	
Precondiciones	• Debe existir un usuario administrador registrado en la base de datos. • El backend debe estar activo y escuchando en la ruta: POST /api/auth/login. • La solicitud debe enviarse con credenciales incorrectas.
Pasos para seguir	1. Enviar una solicitud POST al endpoint /api/auth/login. 2. Enviar en el body datos inválidos 3. Observar la respuesta del servidor.

Datos de Entrada	1. Prueba con usuario inexistente
	{
	"email": "prubamedina@medina.com",
	"password": "admin123"
	}
	2. Prueba con contraseña ncorrecta
	{
	"email": "variedades12@medina.com",
	"password": "contrasenaincorrecta"
	}

Resultados Esperados	• Código HTTP: 401 Unauthorized
	• Respuesta en formato JSON:
	
	{
	"error": "Credenciales inválidas"
	}
	
	• Se genera token aunque la contraseña es incorrecta.
	• Se devuelve un 200 OK con información de usuario.
	• No se muestra ningún mensaje de error claro.

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** No se emite token, y se retorna un mensaje claro de error con código 401. • **RECHAZO:** El sistema responde como si fuera válido, genera token incorrectamente o retorna estado 200.
Desarrollador Asignado	Josselyn Simbaña

Tabla 24. Intento de inicio de sesión con credenciales inválidas

3.3.3. LOGIN CON CAMPOS VACÍOS

3.3.3.1. Intento de inicio de sesión sin completar los campos requeridos

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que el sistema maneje correctamente las solicitudes de inicio de sesión incompletas, es decir, cuando el cliente no envía el correo, la contraseña o ambos. El backend debe validar los campos requeridos y rechazar la solicitud.	
Precondiciones	• El servidor debe estar activo y conectado. • Ruta POST /api/auth/login habilitada. • Middleware o validaciones deben estar configuradas para revisar campos vacíos.
Pasos para seguir	1. Enviar una solicitud POST a /api/auth/login.

Datos de Entrada	{
	"email": "",
	"password": ""
	}

Resultados Esperados	• Código HTTP: 400 Bad Request
	• Respuesta en formato JSON:
	{
	"error": "Todos los campos son obligatorios"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El BackEnd detecta la omisión de campos y retorna un mensaje de error con código 400. • **RECHAZO:** Se acepta la solicitud sin validación, se genera token, o la respuesta es ambigua.
Desarrollador Asignado	Josselyn Simbaña

Tabla 25. Intento de inicio de sesión sin completar los campos requeridos

3.3.4. VERIFICACION CON UN TOKEN VÁLIDO

3.3.4.1 Acceso exitoso a una ruta que está protegida con un token valido

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que un usuario autenticado con un token JWT válido puede acceder a una ruta protegida del sistema. Se prueba que el middleware verifyToken funcione correctamente y permita el acceso cuando el token es correcto.	
Precondiciones	• El administrador debe haber iniciado sesión previamente y obtenido un token válido. • El backend debe tener rutas protegidas con middleware verifyToken y verifyAdmin (por ejemplo: POST /api/products). • El token debe enviarse en la cabecera de autorización.
Pasos para seguir	1. Iniciar sesión mediante POST /api/auth/login para obtener un token válido. 2. Enviar una solicitud a una ruta protegida (por ejemplo: POST /api/products) con un Authorization header. 3. Incluir un body válido para crear un producto (opcional, si se quiere testear también el contenido).

Datos de Entrada	• Token válido obtenido previamente
	• Ruta: POST /api/products
	• Headers:
	Authorization: Bearer (eyJhbGciOiJIUzI1NiIsInR...)
	• Body :
	{
	"name": "Vestido Floral",
	"price": 25.99,
	"category": "Ropa de mujer"
	}

Resultados Esperados	• Código HTTP: 200 OK (o 201 Created si se trata de una creación). • Se procesa la solicitud y se devuelve la respuesta esperada de la API (ej. producto creado). • El token es reconocido y decodificado correctamente.
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El servidor acepta el token y permite acceder a la ruta protegida sin errores. • **RECHAZO:** El token es rechazado incorrectamente o se retorna un error pese a estar autenticado.
Desarrollador Asignado	Josselyn Simbaña

Tabla 26.Acceso exitoso a una ruta que está protegida con un token valido

3.3.5. VERIFICAIÓN CON UN TOKEN INVÁLIDO

3.3.5.1. Acceso a una ruta usando un token inválido

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema rechace correctamente las solicitudes hacia rutas protegidas cuando se incluye un token JWT que es inválido, mal formado o ha expirado. El middleware debe detectar el problema y denegar el acceso, retornando un mensaje claro.	
Precondiciones	• El BackEnd debe tener rutas protegidas con verifyToken. • Se debe utilizar un token:

	• Incorrectamente formado, • Manipulado, • Con firma inválida, • O uno que ya haya expirado.
Pasos para seguir	1. Enviar una solicitud GET, POST, o DELETE a una ruta protegida (por ejemplo: GET /api/products). 2. Enviar el encabezado Authorization con uno de los casos. 3. Observar la respuesta que manda el servidor.
Datos de Entrada	• Token inválido (manipulado o caducado) • Ruta protegida: GET /api/products • Header: Authorization: Bearer (eyJhbGciOiJIUzI1NiIsInR…)

Resultados Esperados	• Código HTTP: 403 Forbidden o 401 Unauthorized
	• Mensaje en formato JSON como:
	{
	"error": "Token inválido o expirado"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El sistema bloquea el acceso correctamente y da un mensaje claro. • **RECHAZO:** Permite el acceso con token inválido o genera error no controlado.
Desarrollador Asignado	Josselyn Simbaña

Tabla 27. Acceso a una ruta usando un token inválido

3.3.6. ACCESO SIN TOKEN

3.3.6.1. Prueba de acceso a una ruta protegida sin enviar token de autenticación

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema impida el acceso a rutas protegidas cuando no se incluye el token JWT en el encabezado Authorization. Se espera que el middleware rechace la solicitud y devuelva un mensaje claro indicando que el token es requerido.	
Precondiciones	• El backend debe tener implementado el middleware verifyToken en la ruta a probar. • La ruta protegida debe requerir autenticación (ej. POST /api/products) • No debe enviarse ningún encabezado Authorization.
Pasos para seguir	1. Enviar una solicitud POST, GET, o DELETE a una ruta protegida como: **POST /api/products** 2. No incluir el encabezado Authorization. 3. Observar la respuesta del servidor.
Datos de Entrada	• Token: Ninguno enviado • Ruta protegida: POST /api/products

Resultados Esperados	• Código HTTP: 403 Forbidden o 401 Unauthorized
	• Respuesta JSON:
	{
	"error": "Token no proporcionado"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El BackEnd bloquea el acceso y devuelve un mensaje específico indicando que el token es obligatorio. • **RECHAZO:** La solicitud se procesa sin token o se genera un error ambiguo**.**
Desarrollador Asignado	Josselyn Simbaña

Tabla 28. Prueba de acceso a una ruta protegida sin enviar token de autenticación

3.3.7 OBTENCION DE LISTA DE PRODUCTOS

3.3.7.1 Obtener la lista de productos disponibles para el publico

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso verifica que cualquier usuario (autenticado o no) pueda obtener correctamente la lista de productos mediante una solicitud GET. Se comprueba que la respuesta contenga los productos en orden reciente y con los campos esperados..	
Precondiciones	• El backend debe estar ejecutándose. • Deben existir productos previamente registrados en la base de datos. • La ruta pública debe estar habilitada: GET /api/products.
Pasos para seguir	1. Enviar una solicitud GET a la ruta: /api/products 2. No enviar token ni encabezado especial. 3. Observar la respuesta del servidor.
Datos de Entrada	• Ninguno (solo la solicitud GET sin parámetros).
Resultados Esperados	• Código HTTP: 200 OK • Respuesta en formato JSON con un array de productos: [{ "_id": "123abc", "name": "Camiseta Oversize", "price": 14.99, "category": "Ropa de mujer", "imageUrl": "http://...", "inStock": true, "isPromo": false, },] • La lista debe estar ordenada por fecha de creación (más recientes primero).

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** La respuesta es correcta, completa y llega sin token. • RECHAZO: Se retorna error, lista vacía sin razón, o campos incompletos.
Desarrollador Asignado	Josselyn Simbaña

Tabla 29. Obtener la lista de productos disponibles para el publico

3.3.8. CREACION DE UN PRODUCTO

3.3.8.1 Crear un nuevo producto con los datos validos por parte del administrador

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el sistema permite que un administrador autenticado registre un nuevo producto mediante la API, incluyendo campos como nombre, precio, categoría y una imagen opcional. Se espera una respuesta confirmando la creación y el objeto guardado.	
Precondiciones	• El backend debe estar activo y conectado a MongoDB. • El administrador debe estar autenticado y tener un token JWT válido. • La ruta protegida debe estar habilitada: POST /api/products. • Middleware verifyToken y verifyAdmin deben estar configurados correctamente. • El sistema debe permitir carga de imágenes (opcional, usando multer).
Pasos para seguir	1. Obtener un token válido mediante POST /api/auth/login. 2. Enviar una solicitud POST a /api/products con el token en la cabecera: Authorization: Bearer <TOKEN> 3. Enviar un FormData con los siguientes requeridos.
Datos de Entrada	name: Camiseta Oversize price: 14.99 category: Ropa de mujer imageUrl: archivo PNG
Resultados Esperados	{ "message": "Producto creado correctamente", "product": { "_id": "...", "name": "Camiseta Oversize", "price": 14.99, "category": "Ropa de mujer", "imageUrl": "url del archivo" } }

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El producto es creado y almacenado correctamente con respuesta completa. • **RECHAZO:** Error en validación, respuesta incompleta o fallo al guardar imagen.
Desarrollador Asignado	Josselyn Simbaña

Tabla 30. Crear un nuevo producto con los datos validos por parte del administrador

3.3.9. FALLO EN LA CREACION POR CAMPOS FALTANTES

3.3.9.1. Intento de creación de un producto sin completar los campos obligatorios

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el backend rechace correctamente las solicitudes para crear productos cuando faltan campos requeridos como name, price o category. El middleware o la lógica del controlador debe detectar esta omisión y retornar un mensaje de error sin almacenar el producto.	
Precondiciones	• Ruta activa: POST /api/products. • Middleware verifyToken y verifyAdmin habilitado. • El administrador debe enviar un token JWT válido. • El backend debe tener validación de campos
Pasos para seguir	1. Obtener un token válido como administrador. 2. Enviar una solicitud POST a /api/products con un campo faltante 3. Observar la respuesta del servidor.

Datos de Entrada	{
	"price": 12.99,
	"category": "Ropa de mujer"
	}

Resultados Esperados	{
	"error": "Todos los campos son obligatorios"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El servidor valida los campos y bloquea la creación cuando hay omisiones. • **RECHAZO:** Se permite la creación parcial o no se indica el error claramente.
Desarrollador Asignado	Josselyn Simbaña

Tabla 31. Intento de creación de un producto sin completar los campos obligatorios

3.3.10 EDICION DE PRODUCTO EXISTENTE

3.3.10.1 Alteración de un producto ya registrado con nuevos datos valiosos

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que un administrador pueda actualizar los datos de un producto existente. Se espera que el sistema reciba los nuevos valores, los aplique correctamente en la base de datos, y devuelva una respuesta confirmando la actualización.	
Precondiciones	• Debe existir al menos un producto previamente registrado. • El administrador debe estar autenticado con un token válido. • La ruta protegida debe estar habilitada: PUT /api/products/:id. • Middleware verifyToken y verifyAdmin deben estar activos.
Pasos para seguir	1. Obtener el id de un producto previamente creado. 2. Enviar una solicitud PUT a: 3. Incluir el token de autenticación en el encabezado. 4. Enviar un body JSON con los nuevos datos del producto: 5. Observar la respuesta del servidor.

Datos de Entrada	{
	"name": "Vestido Largo Negro",
	"price": 29.99,
	"category": "Ropa de mujer"
	}

Resultados Esperados	{
	"message": "Producto actualizado correctamente",
	"product": {
	"_id": "64e9e184bc3f2730b29dfdb7",
	"name": "Vestido Largo Negro",
	"price": 29.99,
	}
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** Los datos se actualizan correctamente y la respuesta es clara. • **RECHAZO:** La respuesta es incompleta, se devuelve error, o el cambio no se refleja.
Desarrollador Asignado	Josselyn Simbaña

Tabla 32. Alteración de un producto ya registrado con nuevos datos valiosos

3.3.11. ELIMINACION DE UN PRODUCTO EXISTENTE

3.3.11.1. Eliminar un producto registrado usando un ID válido

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema permita a un administrador eliminar correctamente un producto existente mediante la API, usando un ID válido. El sistema debe confirmar la eliminación con un mensaje y código adecuados.	
Precondiciones	• El backend debe estar ejecutándose y conectado a MongoDB. • Debe existir al menos un producto previamente creado.

	• El administrador debe estar autenticado con un token válido. • Ruta activa: DELETE /api/products/:id. • Middleware verifyToken y verifyAdmin deben estar activos.
Pasos para seguir	1. Obtener el ID de un producto válido en la base de datos. 2. Enviar una solicitud DELETE a: /api/products/64f27f9dcb654f2021a1bc9e 3. Incluir en los headers: Authorization: Bearer <token_válido> 4. Observar la respuesta del servidor.
Datos de Entrada	• ID del producto: valido y existente • Token de administrador

Resultados Esperados	{
	"message": "Producto eliminado correctamente"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El producto es eliminado exitosamente y el backend lo confirma con mensaje. • **RECHAZO:** No se elimina, código incorrecto o falla inesperada.
Desarrollador Asignado	Josselyn Simbaña

Tabla 33. Eliminar un producto registrado usando un ID válido

3.3.13 ELIMINACION DE PRODUCTO

3.3.13.1 Prueba para intentar eliminar un producto con un ID inexistente

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema maneje correctamente una solicitud para eliminar un producto cuyo ID no existe en la base de datos. El backend debe devolver una respuesta clara indicando que no se encontró el producto, sin causar errores internos.	
Precondiciones	• El backend debe estar activo y conectado. • El administrador debe estar autenticado con token válido. • La ruta protegida DELETE /api/products/:id debe estar habilitada. • Se debe utilizar un ID con formato válido de MongoDB pero que no esté en uso.
Pasos para seguir	1. Enviar una solicitud de DELETE a la api con un ID invalido. 2. Incluir encabezado: Authorization: Bearer <token_válido> 3. Observar la respuesta del servidor.
Datos de Entrada	□ ID: 828458000049fjk □ Token valido del administrador
Resultados Esperados	{ "error": "Producto no encontrado"

	}
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El sistema informa correctamente que el producto no existe. • **RECHAZO:** El sistema permite la operación, da error interno o no informa adecuadamente.
Desarrollador Asignado	Josselyn Simbaña

Tabla 34. Prueba para intentar eliminar un producto con un ID inexistente

3.3.14. SUBIR UNA IMAGEN VALIDA

3.3.14.1. Subida correcta de una imagen de producto con un formato valido

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el sistema acepta y procesa correctamente una imagen de producto enviada al crear o editar un producto, usando el middleware multer. Se valida que el archivo sea recibido, guardado y referenciado en la respuesta.	
Precondiciones	• Ruta activa: POST /api/products • Middleware multer configurado correctamente para manejar imágenes JPG. • El administrador debe estar autenticado con token válido.
Pasos para seguir	1. Obtener un token del administrador. 2. Enviar una solicitud POST a /api/products con encabezado: Authorization: Bearer <token_válido> Content-Type: multipartaform-data 3. Enviar FormData con: * Campos del producto * Archivo de imagen JPG en campo imagenUrl
Datos de Entrada	name: “Cartera beige” price: 25.00 category: “Accesorios” imageUrl: cartera.jpg

Resultados Esperados	{
	"message": "Producto creado correctamente",
	"product": {
	"imageUrl": "uploads/cartera-1691241846763-8234230.jpg"
	}
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** La imagen se procesa, almacena y asocia correctamente al producto. • **RECHAZO:** La imagen no es procesada o no se refleja en la respuesta.
Desarrollador Asignado	Josselyn Simbaña

Tabla 35. Subida correcta de una imagen de producto con un formato valido

3.3.15. SUBIDA DE ARCHIVONO VALIDO

3.3.15.1. Intento de subida de in archivo que no es imagen en al crear un producto

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema rechace correctamente los archivos que no sean imágenes válidas (por ejemplo, PDF, DOCX, EXE, etc.) al intentar subir un producto. El middleware multer debe estar configurado para filtrar extensiones o tipos no permitidos.	
Precondiciones	• El backend debe tener multer configurado con filtro de tipo (fileFilter). • Ruta POST /api/products activa y protegida. • El administrador debe tener token JWT válido. • Formulario de subida debe permitir multipartaform-data.
Pasos para seguir	1. Obtener un token del administrador. 2. Enviar una solicitud POST a /api/products con encabezado: Authorization: Bearer <token_válido> Content-Type: multipartaform-data 3. Enviar FormData con: * Campos del producto * Archivo incompatible con el campo
Datos de Entrada	Archivo: documento.pdf Campos: name, price, category Token válido

Resultados Esperados	{
	"error": "Tipo de archivo no permitido"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El archivo es rechazado correctamente con un mensaje claro. • **RECHAZO:** El archivo se guarda o el servidor falla sin controlar el error.
Desarrollador Asignado	Josselyn Simbaña

Tabla 36. Intento de subida de in archivo que no es imagen en al crear un producto

3.3.16. CREACION DE PRODUCTO SIN IMAGEN

3.3.3.16.1. Prueba de creación de un producto sin adjuntar imagen

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que el sistema maneje correctamente la creación de un producto cuando el administrador no adjunta ninguna imagen. Dependiendo de la lógica implementada, el backend puede: • Permitir la creación con una imagen por defecto, o • Rechazar la solicitud si la imagen es obligatoria.	
Precondiciones	• El backend debe estar corriendo. • Middleware multer habilitado.

	• Ruta POST /api/products protegida. • El administrador debe tener token JWT válido.
Pasos para seguir	1. Obtener un token del administrador. 2. Enviar una solicitud POST a /api/products con encabezado: Authorization: Bearer <token_válido> Content-Type: multipartaform-data 3. Enviar FormData sin el campo de imagen: * Campos del producto
Datos de Entrada	□ name: “Pantalón Negro” □ price: 22.50 □ category: “Ropa de hombre” □ imageUrl: (sin enviar)

Resultados Esperados	{
	"error": "La imagen es obligatoria"
	}
	Código HTTP: 400 Bad Request

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El sistema responde correctamente según la política definida: acepta o rechaza con claridad. • **RECHAZO:** Se crea el producto sin control, o hay errores no gestionados.
Desarrollador Asignado	Josselyn Simbaña

Tabla 37. Prueba de creación de un producto sin adjuntar imagen

3.3.17. VERIFICACION DE REFLEJO DE CAMBIOS EN EL FRONTEND

3.3.17.1. Comprobar que la edición de un producto se refleja correctamente en el catálogo para el cliente.

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si, tras editar un producto mediante la API del backend, los cambios aplicados se visualizan correctamente en el catálogo público del cliente (frontend). Se comprueba tanto la API como la integración del frontend.	
Precondiciones	• El backend debe estar corriendo. • El frontend público (catalog.html) debe estar conectado correctamente a /api/products. • El producto editado debe estar visible previamente en el catálogo. • Cache del navegador desactivada o forzada a recargar.
Pasos para seguir	1. Ingresar a la URL pública del catálogo y anotar el nombre/precio del producto original. 2. Hacer login como administrador. 3. Ir al endpoint PUT /api/products/:id y editar el producto (nombre, precio o categoría). 4. Volver al catálogo público. 5. Refrescar la página y verificar que los nuevos datos se muestran correctamente.

Datos de Entrada	{
	"name": "Camiseta Oversize Blanca",
	"price": 15.99,
	"category": "Ropa de mujer"
	}

Resultados Esperados	• El producto aparece actualizado en el catálogo público: * Nombre modificado * Precio actualizado * Categoría reflejada correctamente • La imagen no se pierde (si no fue modificada)
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El cambio hecho desde el backend es visible en el catálogo público al recargar. • **RECHAZO:** El cambio no se refleja o requiere acciones forzadas (borrar caché, recargar muchas veces, etc.).
Desarrollador Asignado	Josselyn Simbaña

Tabla 38. Comprobar que la edición de un producto se refleja correctamente en el catálogo para el cliente.

3.3.18. VERIFICACION DE ELIMINACION DE PRODUCTOS EN EL FRONTEND

3.3.18.1. Confirmar que el producto eliminado se refleje en el frontend del catálogo para clientes

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso comprueba si al eliminar un producto usando la API del backend, este desaparece correctamente del catálogo visible por los clientes. Se evalúa que la integración frontend-consumo de API funcione dinámicamente tras una eliminación.	
Precondiciones	• El producto debe existir y ser visible actualmente en el catálogo público. • El backend debe estar corriendo y conectado a MongoDB. • El frontend (catalog.html) debe estar funcional y correctamente enlazado a GET /api/products. • El administrador debe estar autenticado con token válido.
Pasos para seguir	1. Ingresar al catálogo público y verificar la presencia del producto (anotar nombre, ID o imagen). 2. Iniciar sesión como administrador. 3. Enviar una solicitud DELETE a: /api/products/:id con token en el encabezado. 4. Volver al catálogo y recargar la página. 5. Observar si el producto eliminado ya no aparece.
Datos de Entrada	ID del producto a eliminar: valido y visible Token del administrador en el header Ruta usada: DELETE /api/products/64feb…

Resultados Esperados	• Código HTTP: 200 OK • Producto eliminado de la base de datos • El catálogo público se actualiza al recargar y ya no muestra el producto eliminado
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El producto eliminado desaparece del catálogo del cliente tras recargar la página. • **RECHAZO:** El producto sigue visible, aunque ya fue eliminado desde el backend.
Desarrollador Asignado	Josselyn Simbaña

Tabla 39. Confirmar que el producto eliminado se refleje en el frontend del catálogo para clientes

3.3.19. FILTRADO DE LOS PRODUCTOS POR CATEGORIA

3.3.19.1. Obtener productos filtrados por categorías en promoción

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso evalúa si el backend permite realizar filtros al consultar los productos públicos, por ejemplo: • Mostrar solo productos de una categoría específica • Mostrar solo los que están en promoción (isPromo: true) • Mostrar los que están fuera de stock (inStock: false)	
Precondiciones	• El backend debe estar corriendo y tener productos con distintos estados y categorías. • Ruta GET /api/products debe aceptar parámetros opcionales (query strings). • El frontend o Postman puede ser usado para probar el filtro.
Pasos para seguir	1. Enviar una solicitud a la siguiente URL con parámetros de filtro: * Filtrar por categoría: GET /api/products/category=Ropa%20de%20mujer • Filtrar por promoción: GET /api/products/isPromo=true • Filtrar por stock: GET /api/products/inStock=false 2. Observar la respuesta desde el BackEnd
Datos de Entrada	• category=Ropa de mujer • isPromo=true • inStock=false
Resultados Esperados	• Código HTTP: 200 OK [{ "_id": "...", "name": "Camiseta Rosa", "category": "Ropa de mujer", "isPromo": true, "inStock": false

	}, ...]
Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** El backend filtra correctamente según los parámetros enviados y devuelve la lista esperada. • **RECHAZO:** El filtro no tiene efecto, devuelve error o datos erróneos.
Desarrollador Asignado	Josselyn Simbaña

Tabla 40. Obtener productos filtrados por categorías en promoción

3.3.20. ACCESO A UNA RUTA INEXISTENTE(404)

3.3.20.1. Solicitud a una ruta no definida en la API

DESCRIPCIÓN DEL CASO DE PRUEBA Valida que el backend devuelva un error 404 claro y controlado cuando se accede a una ruta no definida.	
Precondiciones	• El backend debe estar corriendo. • No debe existir ninguna ruta definida en /api/productoos (mal escrita). • El sistema debe tener un manejador global para errores 404.
Pasos para seguir	1. Enviar una solicitud GET a una ruta inexistente, como: /api/productoos 2. Observar el estado y el contenido de la respuesta.
Datos de Entrada	• Método: GET • Ruta: /api/productoos

Resultados Esperados	{
	"error": "Ruta no encontrada"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** • El sistema devuelve un 404 con un mensaje claro y estructurado. • **RECHAZO:** • El sistema responde con error 500, sin mensaje, o sin formato consistente.
Desarrollador Asignado	Josselyn Simbaña

Tabla 41. Solicitud a una ruta no definida en la API

3.3.20. FORMATO CONSISTENTE DE ERRORES

3.3.20.1. Validar que todos los errores compartan la misma estructura de respuesta

DESCRIPCIÓN DEL CASO DE PRUEBA Este caso valida que todos los errores devueltos por la API tengan una estructura consistente en formato JSON, útil para manejar errores desde el frontend.	
Precondiciones	• El backend debe tener controladores con validación de errores. • • El middleware global de manejo de errores debe estar configurado. • • Se deben usar errores conocidos (login fallido, token inválido, etc.).
Pasos para seguir	1. Provocar un error común, por ejemplo: • Login con datos incorrectos • Token inválido • POST sin campos obligatorios Observar la estructura de la respuesta.
Datos de Entrada	• { • "email": "admin@medina.com", • "password": "contrasenaequivocada" • }

Resultados Esperados	{
	"error": "Credenciales inválidas"
	}

Criterios de Aceptación / Rechazo	• **ACEPTACIÓN:** • Todos los errores siguen el mismo formato JSON con clave error.. • **RECHAZO:** Algunos errores retornan mensajes sin estructura o en texto plano.
Desarrollador Asignado	Josselyn Simbaña

Tabla 42. Validar que todos los errores compartan la misma estructura de respuesta

3.3.20. ALLA DE CONEXIÓN A LA BASE DE DATOS

3.3.20.1. Validar comportamiento del backend si falla la conexión con MongoDB

DESCRIPCIÓN DEL CASO DE PRUEBA Evalúa la reacción del backend cuando no puede conectarse a la base de datos, por ejemplo, al tener mal configurada la URI de MongoDB Atlas o si el servicio no está disponible.	
Precondiciones	• El archivo .env debe tener una URI incorrecta de MongoDB o el servidor de MongoDB debe estar detenido. • • El backend debe estar configurado para mostrar mensajes claros en la consola y manejar fallos de conexión. •
Pasos para seguir	1.Cambiar temporalmente MONGODB_URI por un valor inválido. • Iniciar el backend. • Observar: • Los mensajes en consola • La respuesta al enviar un GET /api/products

	.
Datos de Entrada	• URI invalida en .env • Peticion: GET /api/products

Resultados Esperados	{
	"error": "No se pudo conectar a la base de datos"
	}

Criterios de Aceptación / Rechazo	**ACEPTACIÓN:** • El backend no se cae, lanza error controlado y mantiene consistencia. **RECHAZO:** • El backend lanza excepción no capturada o se detiene inesperadamente.
Desarrollador Asignado	Josselyn Simbaña

Tabla 43. Validar comportamiento del backend si falla la conexión con MongoDB

3.4. Despliegue del FrontEnd (Administrador y Cliente)

El proceso llevado a cabo para publicar el sistema desarrollado, tanto para los clientes como para el administrador, el cual permite su acceso en línea desde cualquier ubicación o dispositivo. Para lograrlo, se empleó la plataforma AWS S3 (Amazon Simple Storage Service), el cual es una herramienta que facilita el alojamiento de sitios web estáticos con alta disponibilidad y accesibilidad desde diversos dispositivos. El despliegue no se limitó únicamente a la carga de archivos, sino que incluyó optimizaciones para mejorar el rendimiento en navegadores y la implementación de medidas básicas de seguridad.

3.4.1. Configuracion de AWS S3

Para este despliegue, se crearon dos buckets independientes en AWS S3, cada uno cumple con un propósito específico:

medina-catalog-frontend: Es destinado al catálogo web accesible para los clientes.

medina-admin-frontend: Espacio utilizado por el administrador para la gestión de productos, enlaces y pedidos.

En cada bucket se realizaron las siguientes configuraciones:

Habilitación de hosting estático: Se activó la funcionalidad de Static Website Hosting para que los archivos funcionen como un sitio web completo.

Carga de archivos optimizados: Los archivos generados tras el proceso de construcción del frontend fueron subidos a los buckets.

Acceso público configurado: Se implementó una política JSON que permite el acceso general a los archivos HTML, CSS y JavaScript, asegurando que el contenido sea visible públicamente.

3.4.2 Optimización y seguridad

Antes de publicar el sistema, se optimizó su rendimiento mediante la compresión de archivos HTML, CSS y JavaScript a través de la comprensión, la eliminación de recursos no utilizados para ahorrar espacio y la organización de los archivos en una estructura modular lo que facilita el mantenimiento; además, se implementó medidas básicas de seguridad, evitando exponer datos sensibles en el frontend y estableciendo la comunicación con el backend mediante HTTPS, utilizando dominios proporcionados por Render.

3.4.3. Accesibilidad

La infraestructura de AWS S3 permitió que los sitios web fueran accesibles a través de URLs públicas generadas por la plataforma, sin necesidad de adquirir un dominio propio. Esto garantizó la compatibilidad con dispositivos móviles (Android e iOS) y computadoras y navegadores modernos.

En el proceso de construcción se llevaron a cabo pruebas para verificar la correcta visualización y funcionalidad desde diferentes dispositivos, con un enfoque especial en teléfonos móviles, considerando que estos son el principal medio de acceso para los clientes en este tipo de emprendimientos. Este enfoque permitió que el sistema estuviera disponible de manera eficiente y accesible, cumpliendo con los objetivos establecidos para el proyecto.

3.5. Despliegue del Backend

3.5.1 Despliegue y Configuración en Render

El backend, desarrollado en Node.js y Express, fue desplegado en Render, una plataforma práctica que actualiza el sistema automáticamente desde el repositorio principal en GitHub, el cual configura HTTPS sin complicaciones. Se uso variables de entorno para guardar datos importantes, como la conexión a MongoDB Atlas o el secreto de los tokens JWT, además de la llave secreta para Cloudinry manteniéndolos seguros y fáciles de ajustar.

Los comandos: "**npm instal**"l y "**npm run start"** permiten al servidor iniciar en el puerto que Render asigna, y los logs en su panel ayudan a detectar errores, como problemas de conexión, haciendo que el despliegue sea sencillo y eficiente el catálogo web.

3.5.2. Optimización y Seguridad

Para que el backend tenga mejor velocidad y seguridad, se optimizó usando variables de entorno para proteger datos sensibles, como credenciales de la base de datos, y se configuró CORS para permitir solo conexiones desde el frontend

autorizado. Render ofrece HTTPS automáticamente, asegurando que la información viaje protegida, y se usaron herramientas como Helmet y límites de solicitudes para evitar ataques o sobrecargas, logrando así un sistema confiable.

3.5.3. Accesibilidad

El sistema está diseñado para ser accesible desde cualquier dispositivo, como celulares o computadoras, en navegadores modernos, usando una URL pública que Render proporciona sin necesidad de un dominio propio. Se probaron las funciones principales, como ver productos o gestionar pedidos, para probar su funcionalidad. La autenticación con JWT permite a los usuarios acceder a sus funciones según su rol, ofreciendo una experiencia práctica y segura.

Conclusiones

- El desarrollo del sistema web "Medina Variety" representa la consolidación de un proyecto integral que abarca desde la recolección de requerimientos funcionales hasta la validación completa mediante pruebas de frontend y backend. Este proyecto tuvo como objetivo central crear una solución digital accesible y eficiente para la gestión y visualización de un catálogo de productos de una tienda de ropa, facilitando tanto la administración interna como la experiencia de los clientes.
- Desde sus inicios, el análisis de requerimientos permitió definir claramente las funcionalidades necesarias: un panel de administrador con autenticación, registro y gestión de productos (crear, editar, eliminar, marcar como promoción o sin stock) y una vista pública del catálogo donde los clientes pudieran visualizar productos por categoría, agregarlos a un carrito y generar un pedido a través de WhatsApp. Esta definición guió todas las fases posteriores del desarrollo.
- Técnicamente, el sistema se desarrolló con una arquitectura dividida en frontend y backend, el frontend fue construido en HTML, CSS y JavaScript puro, segmentado en dos vistas independientes: una para administradores y otra para clientes Mientras que el backend se implementó con Node.js y Express, y utiliza MongoDB Atlas como base de datos remota, garantizando escalabilidad y persistencia centralizada.
- Para la gestión segura de usuarios administradores, se utilizó JWT (JSON Web Tokens), junto con middlewares personalizados para la verificación de roles y autenticación. La carga de imágenes fue manejada por multer, lo que permitió la subida de archivos desde formularios tipo multipart/form-data.
- Durante la etapa de validación del sistema se desarrollaron y ejecutaron un total de 22 casos de prueba del backend, abarcando todas las funcionalidades del sistema lo que permitió asegurar el correcto funcionamiento de la capa de backend

- Asimismo, se implementaron casos de prueba para el frontend del administrador y del cliente, que verificaron desde el inicio de sesión hasta la experiencia completa de navegación por el catálogo, interacción con el carrito, y generación de pedidos vía WhatsApp.
- Los resultados obtenidos demostraron la correcta integración entre el backend y el frontend, permitiendo que todas las acciones del administrador se reflejen en tiempo real en la vista del cliente. Esto, sumado al diseño modular y estructurado del sistema, garantiza no solo el cumplimiento de los requerimientos iniciales, sino también una base sólida para futuras extensiones como gestión de pedidos, panel de estadísticas, o integración con pasarelas de pago.
- En conclusión, el catálogo web "Medina Variety" es un sistema funcional, seguro, escalable y adaptado a las necesidades reales de una tienda de ropa, demostrando la aplicación de buenas prácticas de desarrollo, validación rigurosa, y orientación a la experiencia del usuario.

Recomendaciones

Tras la implementación y validación del sistema "Medina Variety", se presentan las siguientes recomendaciones técnicas y funcionales que permitirán fortalecer, mantener y escalar la solución desarrollada:

- Al integrar un sistema de pedidos, se recomienda que el stock de los productos se actualice automáticamente según las compras realizadas para evitar inconsistencias y sobreventa.
- Incluir un panel gráfico que permita gestionar y revisar los productos más vistos, más vendidos, visitas al catálogo, entre otros. Esto con el fin de poder tomar decisiones informadas sobre el inventario y estrategias de promoción.
- Al publicar el sistema, se deben aplicar medidas de seguridad adicionales como cifrado HTTPS, protección contra inyecciones, manejo seguro de variables de entorno y control de acceso más granular.
- A medida que el numero de visitas y de productos agregados a la plataforma, sugiero dar una mejor experiencia de usuario, convirtiéndola más llamativa y atractiva, en el cual pueda adaptarse a los distintos gustos de los clientes.

Referencias bibliográficas

Wikipedia contributors. (2025). Competencia (economía). En Wikipedia, La enciclopedia libre. Recuperado de https://es.wikipedia.org/wiki/Competencia_(econom%C3%ADa)

Arribas, E. H. (s.f.). Condiciones que favorecen el emprendimiento: análisis económico y propuestas. Fundación Funcas. https://www.funcas.es/articulos/condiciones-que-favorecenel-emprendimiento-analisis-economico-y-propuestas/

FasterCapital. (s.f.). Desafíos y competencia en el mercado. Recuperado de https://fastercapital.com/es/tema/desaf%C3%ADos-y-competencia-en-el-mercado.html

Cinco Días. (2025, 11 de marzo). Las startups que innovan, se internacionalizan y dejan autonomía a sus equipos crecen más rápido. https://cincodias.elpais.com/companias/2025-0 3-11/las-startups-que-innovan-se-internacionalizan-y-dejan-autonomia-a-sus-equipos-crecenmas-rapido.html

HubSpot. (2025). 15 frameworks de desarrollo web que debes conocer. https://blog.hubspot.es/website/framework-desarrollo-web

Togrow Agencia. (s.f.). Tecnologías clave para proyectos de desarrollo web. https://togrowagencia.com/tecnologias-para-proyectos-de-desarrollo-web/

Carmatec. (2024). Guía completa de arquitectura de aplicaciones web. https://www.carmatec.com/es_mx/blog/guia-completa-de-arquitectura-de-aplicaciones-web/

Wikipedia contributors. (2024). Arquitectura orientada a servicios. Wikipedia. https://es.wikipedia.org/wiki/Arquitectura_orientada_a_servicios

Apiumhub. (2023). Desarrollo de bajo código: definición, ventajas y futuro. https://apiumhub.com/es/tech-blog-barcelona/desarrollo-de-bajo-codigo/

ThoughtWorks. (s.f.). La plataforma de ingeniería: clave para maximizar la eficacia del desarrollo de software. https://www.thoughtworks.com/es-es/insights/blog/platforms/engineering-platform-key-to-maximizing-software-development-effectiveness

SEGIB. (2022). Caja de herramientas para la transformación digital. https://www.segib.org/wp-content/uploads/3.-Caja-de-Herramientas.pdf

Informática PRO. (s.f.). Servicios informáticos para páginas web y catálogos online. https://informaticapro.es/desarrollo-informatico-paginas-web