TIENDA ONLINE FARKO

DESARROLLO DE APLICACIONES WEB 2ª DAW 2025

Fernando Sánchez Herráez

ENLACE A GITHUB

https://github.com/Nanocot/ProyectoFinal

ENLACE A PRODUCCION

https://tienda-online-farko.infinityfreeapp.com/index.php?action=home

ENLACE A LA MEMORIA

https://github.com/Nanocot/ProyectoFinal/blob/main/S%C3%A1nchezHerr%C3%A1ez Fernando.pdf

ENLACE A GUIA DE USUARIO

https://github.com/Nanocot/ProyectoFinal/blob/main/Manual-Usuario.pdf

USUARIO:

Nombre de usuario: usuario1@farko.es

Contraseña: usuario1

ADMINISTRADOR:

Nombre de usuario: admin@farko.es

Contraseña: admin123

El script SQL para crear la base de datos se encuentra en el siguiente enlace

https://github.com/Nanocot/ProyectoFinal/blob/main/CreacionBaseDatos/script-2.0.sql

Índice

1 DESCRIPCIÓN DEL PROYECTO

En este documento se desarrolla el proyecto final del curso de Desarrollo de Aplicaciones Web. Su meta principal es hacer una tienda virtual llamada "Tienda virtual FARKO". Esta app quiere expandir las habilidades técnicas al crear apps web desde cero, usando lo aprendido en el curso.

1.1 CARACTERÍSTICAS GENERALES

La creación de "Tienda virtual FARKO" usará un sistema cliente-servidor, con estas tecnologías clave:

- Frontend: HTML5, CSS para la forma y el diseño visual, y JavaScript para que el cliente interactúe. Se buscará un diseño responsive para su correcta visualización en diferentes dispositivos y con una interfaz fácil de usar, tanto para usuarios como para administradores.
- Backend: PHP para la lógica del servidor, MySQL para guardar datos (productos, usuarios, pedidos, etc.), y Apache para manejar las peticiones y mostrar el contenido.
- Otros: Se usará Git para controlar las versiones del código fuente al crear la app.

1.2 OBJETIVOS Y ALCANCE

Objetivo General:

Crear una aplicación web útil de tienda online que deje ver productos, manejar un carrito y generar pedidos. Esto mostrará las habilidades técnicas adquiridas en este proceso de aprendizaje.

Objetivos Específicos:

Manejo de Usuarios:

- Poner en marcha un sistema de acceso (iniciar y cerrar sesión) para clientes dados de alta y administradores.
- Dar vía libre al alta de cuentas nuevas para clientes sin registro, pasándolos a ser clientes registrados.
- Crear la herramienta para que los clientes dados de alta cambien sus datos personales (nombre, apellidos, clave, dirección).
- Sumar un sistema de alta y baja a un sistema de noticias mediante correo electrónico (newsletter).

Listado y Muestra de Artículos:

- Crear una galería de los artículos ofertados, sumando fotos, opiniones de otros usuarios, precios, medidas y colores existentes.
- Poner en marcha un sistema para buscar artículos de forma avanzada por:
    - Tipo (ropa, pegatinas, láminas, gorros, etc.)
    - Colecciones

- Ropas
- Medidas
- Color de ropa
- Color de diseño
- Precios

Cesta de Compra:

- Crear la herramienta para añadir artículos a la cesta desde la página donde se ven.
- Crear la herramienta para permitir la visualización completa de la cesta de la compra en cualquier momento.
- Crear la función de poder quitar artículos que estén en la cesta.

Trámite de Compra:

- Poner en marcha un sistema de compra para clientes con o sin registro.

Incidencias y Opiniones:

- Crear un sistema para que los clientes dados de alta dejen opiniones y valoren artículos.
- Poner en marcha un sistema para que los clientes dados de alta puedan realizar reclamaciones por las distintas incidencias.

Datos y Movimiento:

- Hacer un apartado con datos sobre la empresa.
- Crear un registro de compras para clientes que estén dados de alta.
- Poner en marcha un sistema para aceptar o no las cookies.
- Asegurar un movimiento fácil usando un menú que sea principal.

Panel de Administrador:

- Poner en marcha un sistema de acceso para los administradores del sitio web.

Gestión de Artículos:

- Dar vía libre a ver, sumar, cambiar el coste, poner ofertas, cambiar medidas/tonos, controlar el stock y quitar artículos.

Gestión de Colecciones:

- Permitir visualizar, añadir, modificar, aplicar descuentos y borrar las distintas colecciones disponibles.

Manejo de Compras:

- Permitir visualizar, consultar estado de las compras, además de, buscar compras por día, número de cosas, persona, cosas en concreto y precio total.

Manejo de Usuarios:

- Permitir visualizar los usuarios registrados y cambiar el estado (activo/no activo).

Manejo de Descuentos:

- Permitir visualizar, poner, cambiar qué tipo y cuánto es, borrar descuentos existentes, además de manejar cuándo empiezan y terminan los descuentos.

Manejo de Problemas:

- Permitir visualizar, contestar y cerrar problemas que tienen los clientes con los productos.

Alcance:

Este proyecto hará lo siguiente para los diferentes tipos de usuarios:

Usuarios sin Cuenta: Crear una cuenta, ver y buscar artículos, añadir y quitar artículos del carrito, ver el carrito, comprar (sin ver el historial ni cambiar datos de la persona), ver datos de la empresa, manejar cookies y navegar por el sitio web mediante un menú.

Usuarios Registrados: Todas las funcionalidades de los usuarios no registrados, además de inicio y cierre de sesión, modificación de ajustes de usuario (nombre, apellidos, contraseña, dirección, suscripción/baja a Newsletter), escritura de reseñas y valoración de productos, realización de reclamaciones y visualización del historial de compras.

Administradores: Inicio y cierre de sesión, gestión completa de productos, colecciones, compras, usuarios, descuentos e incidencias.

2 ANÁLISIS DEL SECTOR/MERCADO

2.1 PROSPECTIVA DEL TÍTULO EN EL SECTOR

Para hablar de la proyección que tiene este sector vamos a usar los datos que nos proporciona el **Gobierno de España**, en el documento "**2025 Tendencias del mercado de trabajo en España NIPO**" (1) pero sobre todo nos centraremos en la parte que nos concierne, la Tecnología de la Información y las Comunicaciones a la cual a partir de este momento nos referiremos como TIC.

En la sección de correspondiente a nuestro sector del documento citado anteriormente podemos ver un poco más en profundidad el futuro que les depara a los profesionales del sector teniendo en cuenta los siguientes datos:

"*El número de empresas de esta sección (71.979) ha crecido por encima de la media nacional (4,36 %); y aún ha sido mayor el aumento en la actividad de Programación, consultoría y otras actividades relacionadas con la informática (43,79 %) que son 11.756 empresas más que hace diez años.*"

"*Las organizaciones empresariales han cuantificado el número de vacantes en cada una de las especialidades que, en orden de mayor a menor, son las siguientes:*

- *Desarrollo de Software*
- *Inteligencia Artificial*

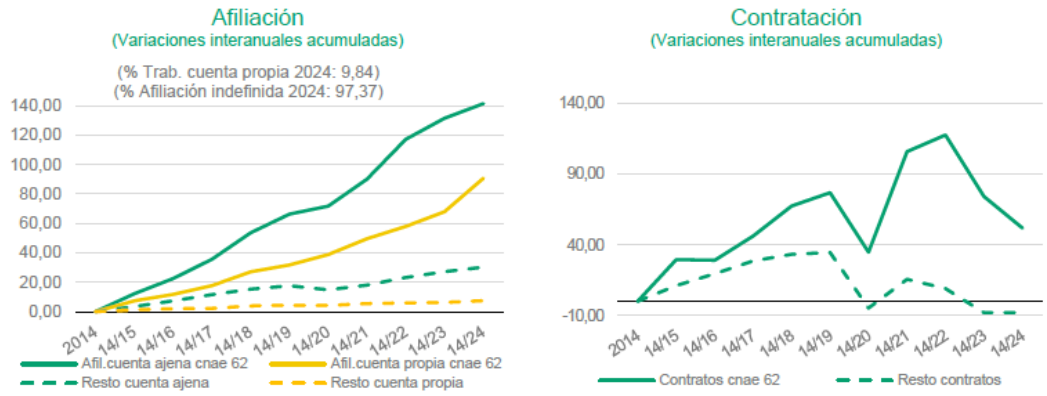
- *Hardware*
- *ERP*
- *CPD's "*

Prestando atención a estos dos datos podemos concluir que este sector tiene una proyección bastante brillante, haciendo así que la formación dentro del mismo sea una opción bastante demandada y endulzada para los jóvenes.

2.2 EVOLUCIÓN Y TENDENCIA DEL SECTOR

Siguiendo con el documento citado en el punto anterior vamos a desarrollar la evolución y tendencia que está siguiendo el sector, en este caso el punto de partida que vamos a utilizar, para la comparación será el 2014.

Gráfico 24 Tendencia de la actividad CNAE 62. Programación, consultoría y otras actividades relacionadas con la informática



Fuentes: Elaborado por el Observatorio de las Ocupaciones del SEPE a partir de los datos de la Tesorería General de la Seguridad Social, último día hábil de diciembre de cada año y del SISPE, total contratos anuales. Año base 2014.

Vemos que hay una tendencia al alza llegando hasta un crecimiento de un 141% desde el 2014.

Según el SEPE este sector cuenta con una de las mejores estabilidades del mercado laboral contando con 84 puntos sobre los 42 que tiene la media.

2.3 NORMATIVA Y DOCUMENTACIÓN TÉCNICA ESPECÍFICA

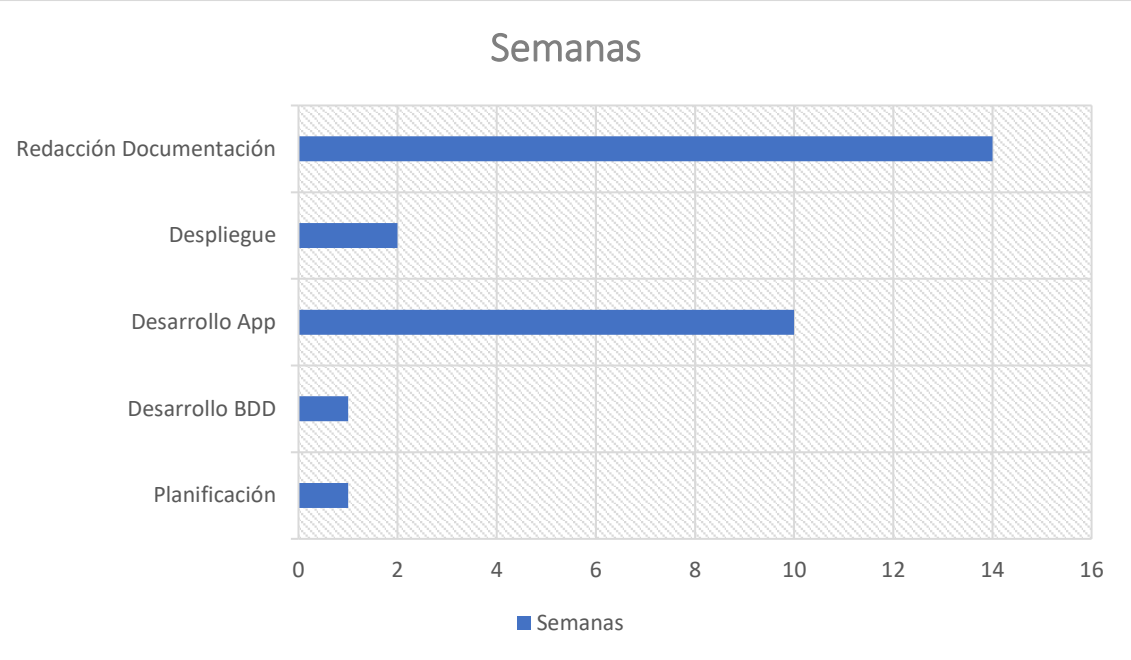
Para poder desarrollar de forma correcta y segura esta profesión hay que tener en cuenta diferentes normativas y reglas dictadas tanto a nivel estatal por el **Gobierno de España** como a nivel continental dictadas por Europa. Las más importantes y necesarias son las siguientes:

- Reglamento General de Protección de datos (2)
- Ley de Servicios de la Sociedad de la Información (3)
- Estándares WCAG :
    - Perceptibilidad
    - Operabilidad
    - Comprensibilidad
    - Robustez

3 PLAN DE EJECUCIÓN

3.1 DIAGRAMA DE FLUJO DE PROCESOS

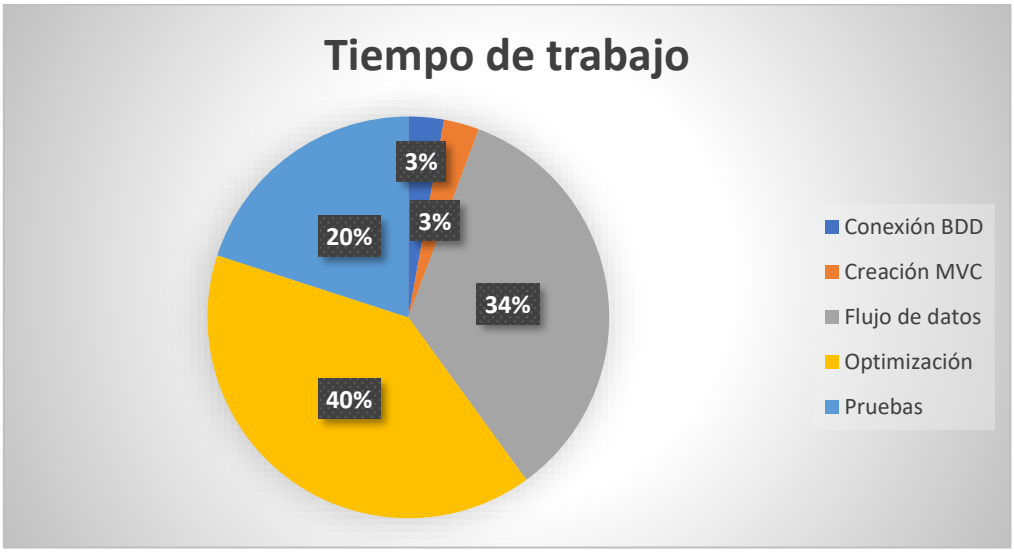
Fases del proyecto	Tiempo estimado
Planificación	1 semana
Desarrollo Base de datos	1 semana
Desarrollo de Aplicación	10 semanas
Despliegue	2 semanas
Redacción de la documentación	14 semanas



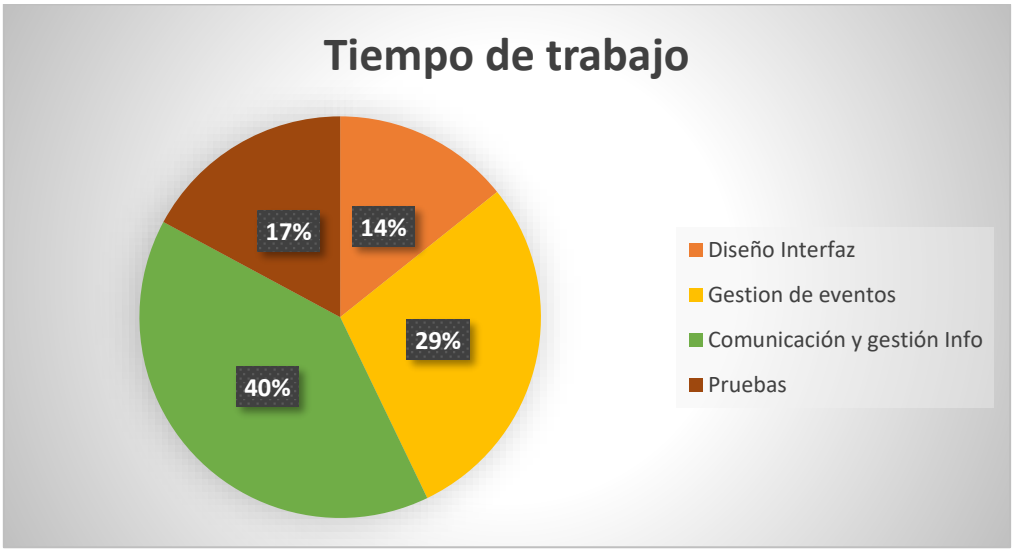
Fase del desarrollo	Tiempo estimado
Desarrollo parte Servidor	5 semanas
Desarrollo parte Cliente	5 semanas

Esta división no significa que se hagan de forma independiente, ya que a la hora del desarrollo de la aplicación con el MVC, tenemos que combinar el desarrollo de la parte del cliente y el desarrollo de la parte servidor, para ello hemos estipulado un tiempo aproximado de 10 semanas, siendo así repartidas equitativamente entre las dos partes, en la siguiente tabla desglosaremos más a fondo los procesos seguidos en cada una de las fases del desarrollo.

Desarrollo Servidor (5 semanas)	
Fase del desarrollo	Tiempo estimado
Conexión a la base de datos y pruebas	1 día
Creación y estructuración del MVC	1 día
Gestión del flujo de datos	12 días
Optimización de recursos del servidor	14 días
Pruebas de rendimiento y depuración	7 días



Desarrollo Cliente (5 semanas)	
Fase del desarrollo	Tiempo estimado
Diseño de la interfaz	5 días
Gestión de eventos de usuario	10 días
Comunicación con servidor y gestión de información cliente	14 días
Pruebas de flujo de usuario y rendimiento	6 días



3.2 PROCESO DE DESARROLLO SOFTWARE

3.2.1 FASE DE ANÁLISIS

3.2.1.1 TIPOS DE USUARIO

- Usuarios Registrados
- Usuarios No Registrados
- Administrador

3.2.1.2 DESCRIPCIÓN DE REQUISITOS

Usuarios Registrados:

- Iniciar Sesión
- Visualizar productos
- Añadir productos al carrito
- Eliminar productos del carrito
- Ver carrito
- Filtrar los productos
    - Por categoría (ya que no se vende solo ropa, también podremos comprar pegatinas, laminas dibujadas a mano, gorras, etc.)
    - Por colecciones
    - Por prendas
    - Por tallas
    - Por colores de prenda
    - Por colores de dibujo
    - Por precio
- Comprar productos
- Escribir reseñas y valorar productos
- Realizar reclamaciones
- Modificar ajustes de usuario
    - Modificar nombre
    - Modificar apellidos
    - Modificar contraseña
    - Modificar dirección
    - Modificar métodos de pago
        - Añadir método de pago
        - Eliminar método de pago
    - Dar de alta la suscripción al NewsLetter
    - Dar de baja la suscripción al NewsLetter
        - Estás opciones serán un botón que puede estar activo o no
- Ver información sobre la empresa
- Ver historial de las compras
- Aceptar las cookies
- Rechazar las cookies
- Navegación mediante un menú
- Cerrar Sesión

Usuarios No Registrados:

- Registro de una cuenta nueva (esto hará que pasen a ser Usuarios Registrados)
- Visualizar productos
- Comprar productos
- Añadir productos al carrito
- Eliminar productos del carrito
- Ver carrito
- Filtrar los productos
    - Por categoría
    - Por colecciones
    - Por prendas
    - Por tallas
    - Por colores de prenda
    - Por colores de dibujo
    - Por precio
- Ver información de la empresa
- Aceptar las cookies
- Rechazar las cookies
- Navegación mediante un menú

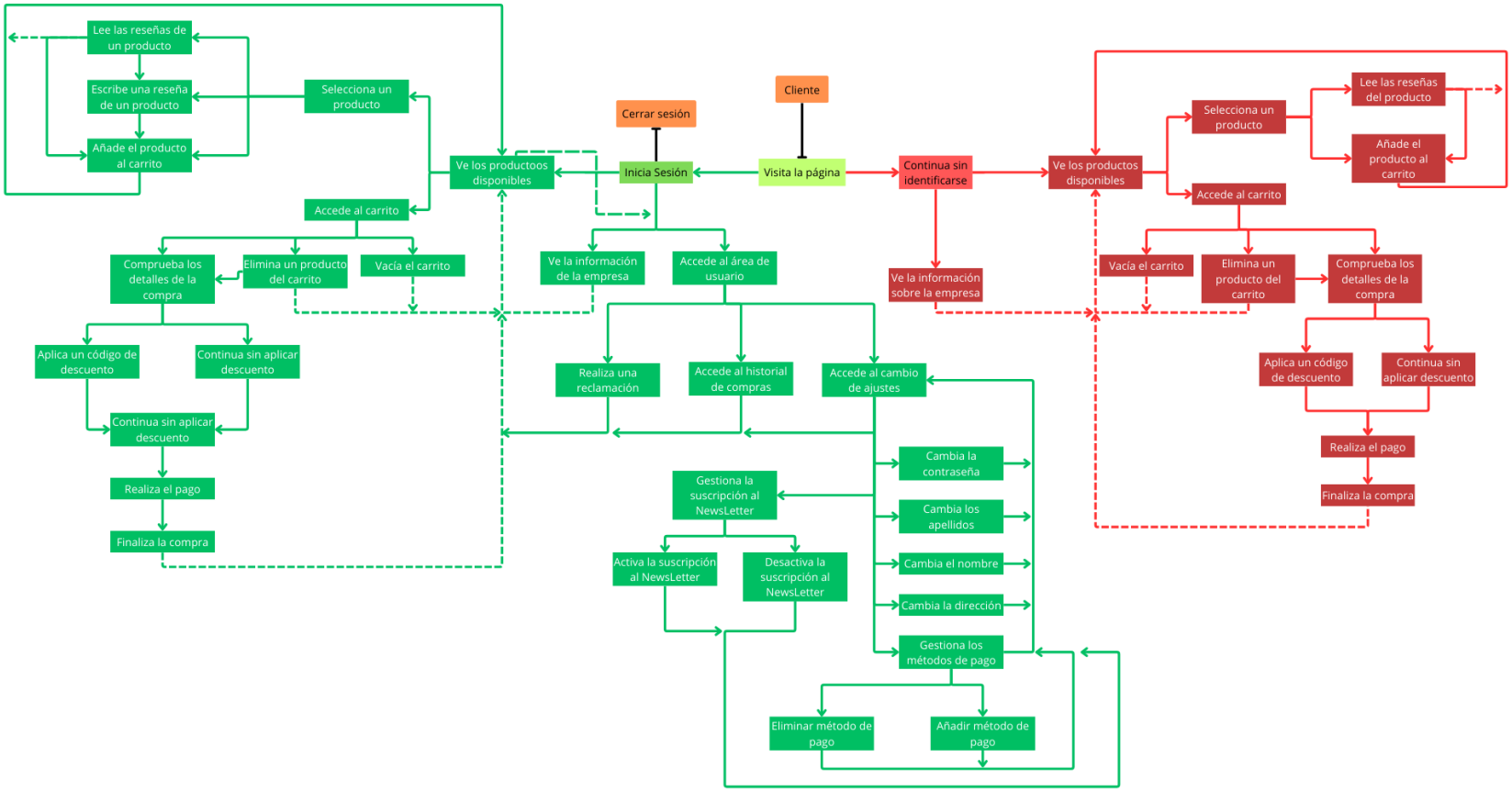
Administrador

- Iniciar Sesión
- Gestionar productos
    - Visualizar los productos existentes
    - Añadir nuevos productos
    - Modificar el precio de productos existentes
    - Aplicar descuentos a productos
    - Modificar tallas disponibles
    - Modificar colores disponibles
    - Comprobar stock
    - Eliminar productos
- Gestionar colecciones
    - Visualizar las colecciones existentes
    - Añadir nuevas colecciones
    - Modificar el nombre
    - Aplicar descuentos a colecciones
    - Eliminar colecciones
- Gestionar las compras
    - Visualizar las compras
    - Cambiar el estado de las compras
    - Filtrar las compras
        - Por fecha
        - Por cantidad de productos
        - Por usuario
        - Por productos específicos
        - Por dinero
- Gestionar los usuarios

    - Visualizar los usuarios
    - Cambiar el estado de un usuario
- Gestionar descuentos
    - Visualizar descuentos existentes
    - Añadir nuevos descuentos
    - Modificar el tipo del descuento (cantidad o porcentaje)
    - Cambiar la fecha de inicio de aplicación del descuento
    - Cambiar la fecha de finalización del descuento
    - Modificar la cuantía del descuento, por ejemplo, si es de porcentaje, cambiarlo de un 5% a un 10%.
    - Borrar un descuento
- Gestionar incidencias
    - Visualizar todas las incidencias
    - Responder incidencias
    - Cerrar incidencias
- Cerrar Sesión

3.2.1.3 CASOS DE USO

Casos de uso cliente



Casos de uso administrador

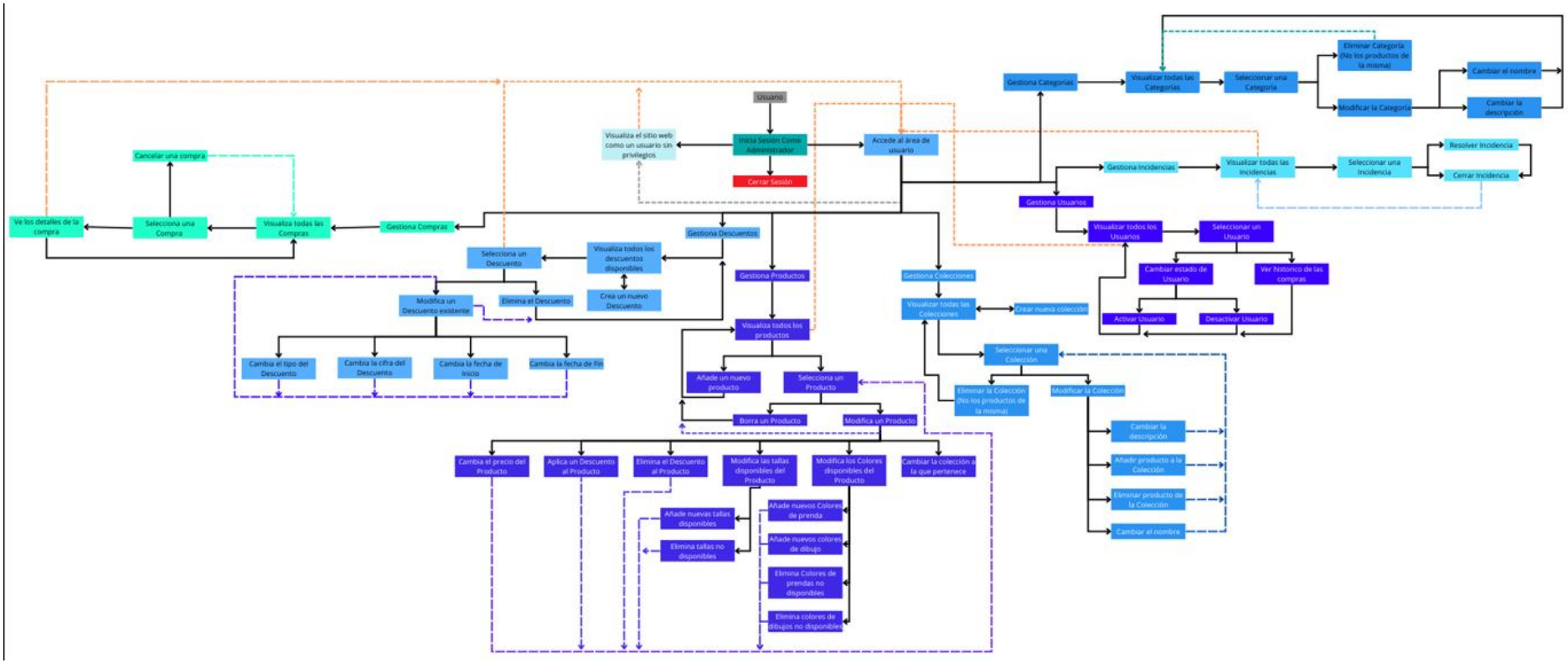


Imagen Casos de Uso cliente:

https://www.canva.com/design/DAGkh_u5LgI/2cTW8iltqk3tnVfNtJzESA/view?utm_content=DAGkh_u5LgI&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=hb710a92154

Imagen Casos de Uso administrador:

https://www.canva.com/design/DAGkiKK_zRw/BhmlFfqu8cM342vWmLGWXQ/view?utm_content=DAGkiKK_zRw&utm_campaign=designshare&utm_medium=link2&utm_source=uniquelinks&utlId=h5ddaa3dc3e

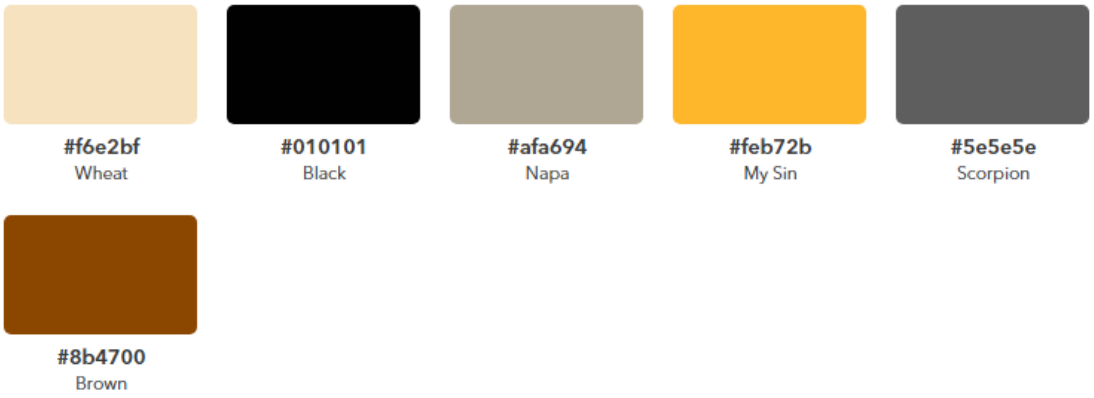
3.2.1.4 Guía de estilos

Estilos

Como podemos observar la página del proyecto, tiene una estética, simple, reflejando que se trata de una página enfocada a la compra de ropa.

Paleta

Hablando de la paleta de colores, encontramos unos colores fríos.



Siendo simple la paleta, consigue una gran calidez y seriedad en la página.

Tipografía

Vemos que la tipografía usada en esta página es “Arial”.

Gracias a esta tipografía tenemos una sensación de fluidez y ligereza dentro de la página. Los tamaños del texto, desde títulos a párrafos gozan de tamaños fácilmente legibles que se calculan de forma dinámica en función del tamaño del dispositivo.

Ilustraciones, iconos y elementos.

En esta página no vemos que tengan muchas ilustraciones, por no llegar a decir, que no se aprecia ningún elemento de este tipo

3.2.1.5 Prototipo del sitio web

Vistas Comunes

Página Inicio

Página de inicio Sesión (Usuario)

Página de Información (Sobre nosotros)

Página de producto

Página de carrito

Vistas Usuario

Página de Ajustes de Usuario

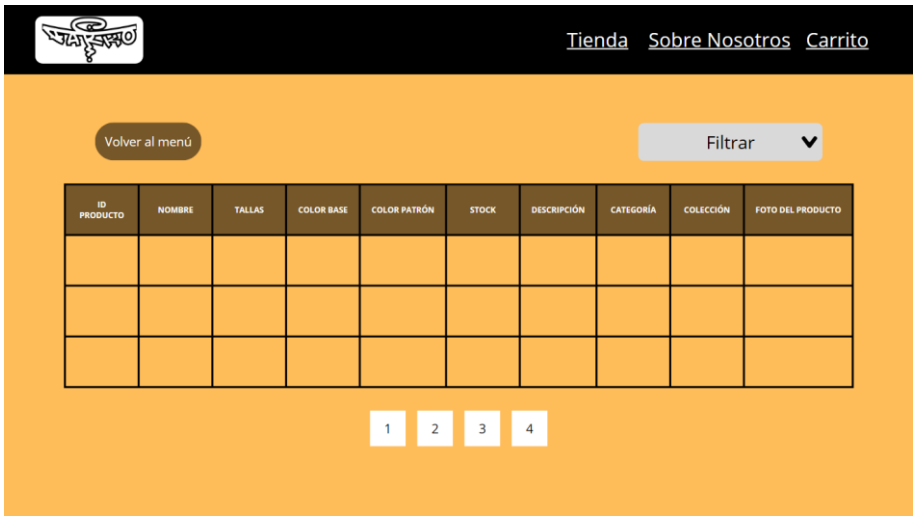
Página de Incidencias

Página de Historial de compras

Vista de Administrador

Panel de Administrador

Página de Productos



Página de usuarios

Página de descuentos

Página de colecciones

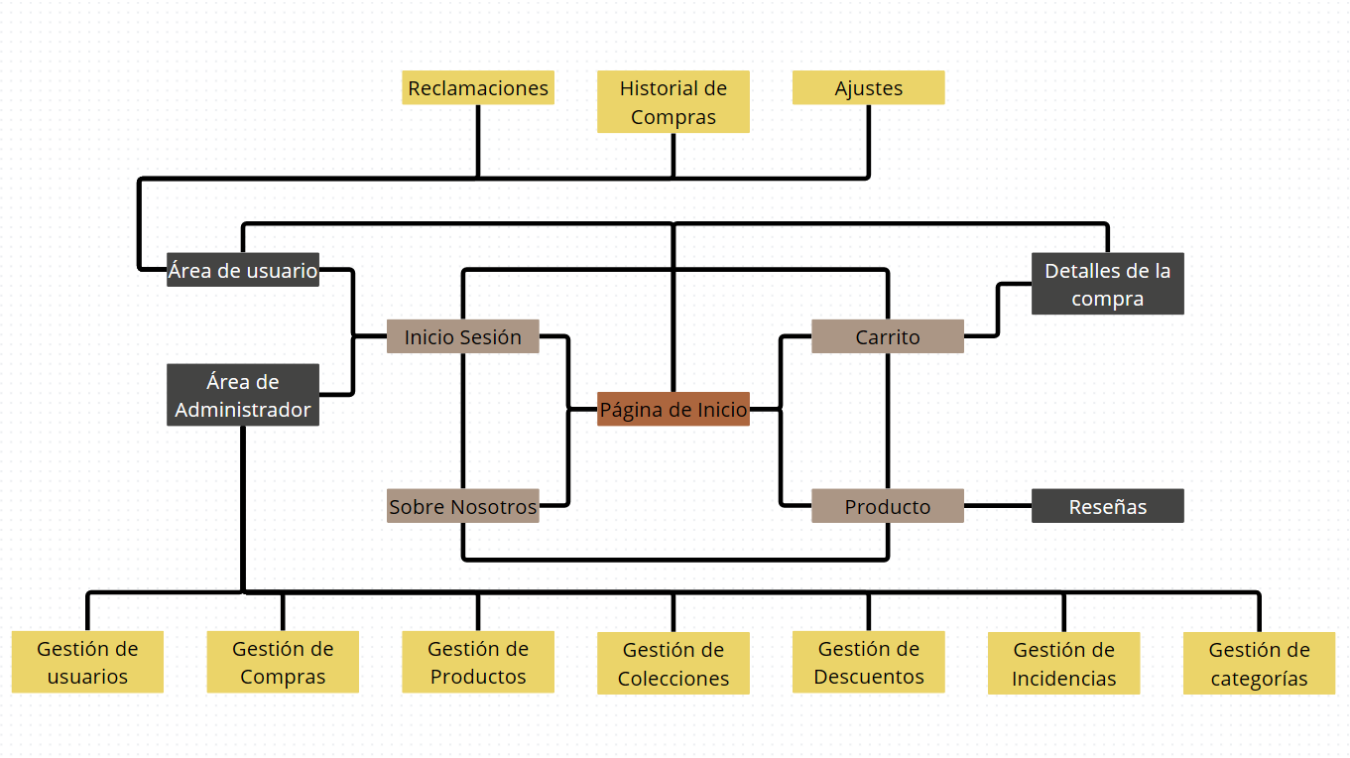
Página de incidencias

Página de Compras

Página de Categorías

3.2.1.6 Mapa de navegación

Este es el mapa de navegación que se aplica a todos los usuarios, teniendo luego la distinción entre los usuarios registrados y los administradores, los cuales tendrán funciones específicas.



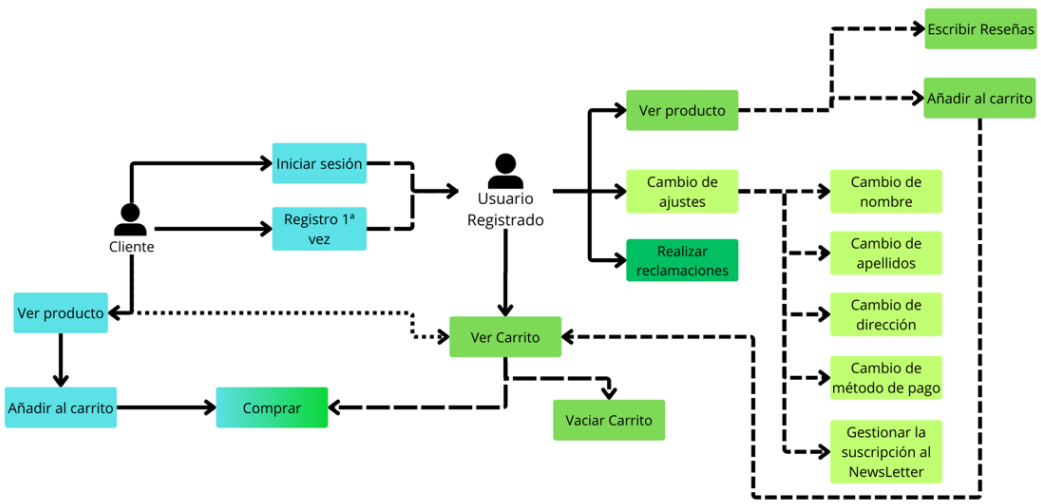
En marrón tenemos la página de inicio, que es común a todos los usuarios.

En gris tenemos las páginas comunes que accedemos desde la página de inicio.

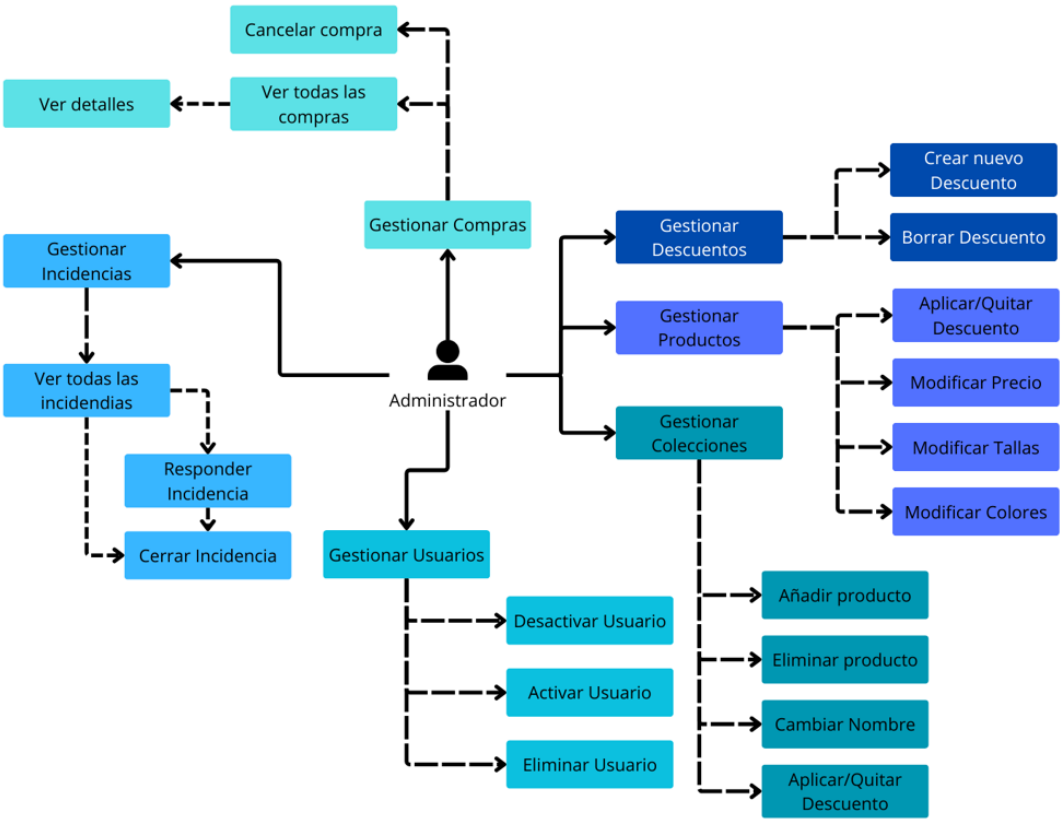
En gris oscuro tenemos las páginas especificas a las que solo podemos acceder desde una página distinta a la de inicio.

En amarillo tenemos las páginas específicas de cada tipo de usuario.

Mapa de navegación Cliente:



Mapa de navegación Administrador:



3.2.2 FASE DE DESARROLLO

3.2.2.1 BASE DE DATOS

3.2.2.1.1 Análisis de requisitos de datos de la aplicación

El objetivo del sistema que se va a desarrollar es gestionar la información de una tienda de ropa online. Para ello vamos a tener en cuenta las dos principales entidades de las cuales luego tendremos que gestionar los datos.

- Usuarios
- Productos

Dentro de los usuarios distinguimos a ojos del desarrollador 3 tipos del mismo, **Registrados – No Registrados – Administradores**, sabiendo que el desarrollador cuenta con que tiene que haber estos 3 tipos de usuarios, es importante recalcar que no es un dato que se quiera registrar, ya que solo va a haber un usuario Administrador y la diferencia entre los registrados y no es que los registrados se guardarán en la base de datos y los no registrados no.

Usuario Administrador:

- Al existir **un único administrador** y siendo una distinción a nivel de desarrollo, se debe asegurar su existencia en la base de datos con credenciales seguras.
- Este usuario tendrá acceso total a la gestión de productos, colecciones y visualización de todos los pedidos.

Usuarios Registrados:

- El sistema debe almacenar una ficha única por cliente con sus credenciales de acceso (**email y contraseña**) y datos personales (**nombre, apellidos, teléfono**).
- Se debe permitir almacenar una o varias direcciones predeterminadas vinculadas a este usuario para agilizar futuras compras.
- Las compras realizadas deben quedar vinculadas permanentemente a su ID de usuario para consulta histórica.

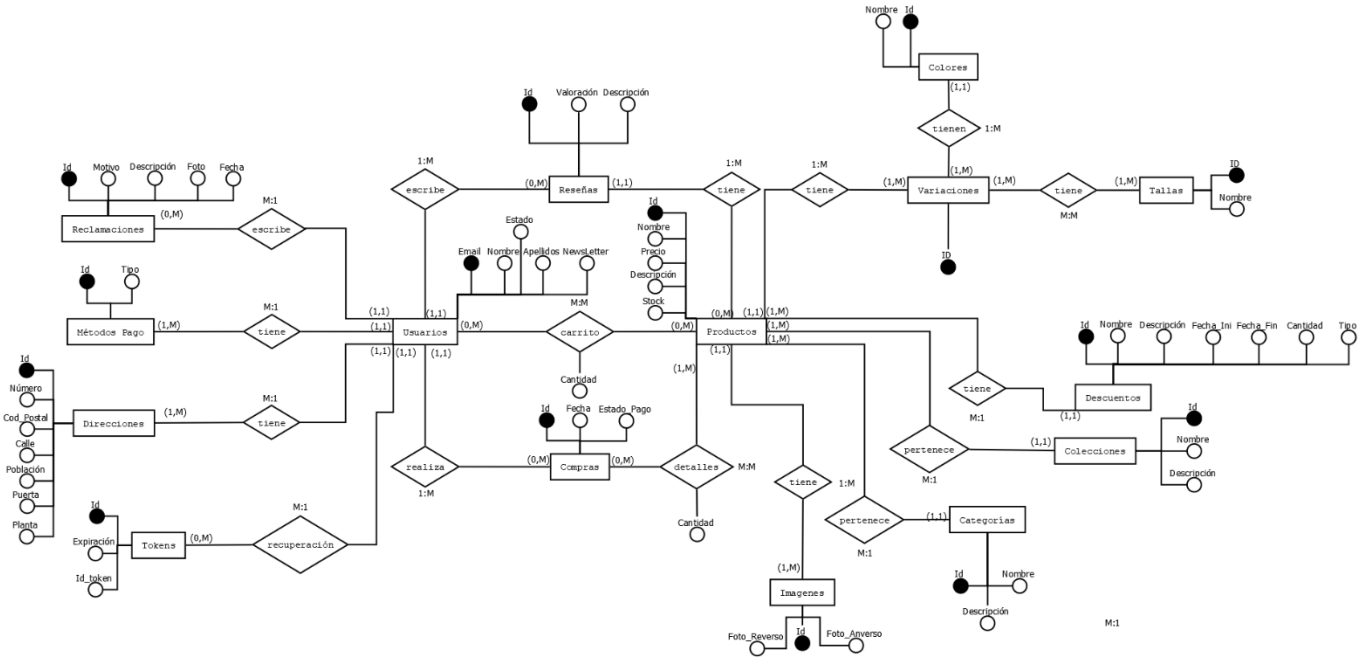
Usuarios No Registrados:

- Requisito de No-Persistencia: El sistema no creará un registro en la tabla de Usuarios para estos clientes.
- Sus datos personales (nombre, dirección, email) se recogerán exclusivamente en el momento de la compra y se guardarán como datos estáticos asociados al pedido, sin crear una cuenta reutilizable.

Los datos que vamos a guardar sobre los productos son los corrientes en la mayoría de tiendas online, como pueden ser **categorías, tallas, colores, imágenes**, etc.

3.2.2.1.2 Diseño lógico de datos

Una vez estudiados los requerimientos propuestos para el correcto funcionamiento de la aplicación, se ha desarrollado el siguiente modelo de base de datos:



3.2.2.1.3 Paso del modelo lógico al modelo relacional

Base de datos

Usuarios(Email, Nombre, Apellido1, Apellido2, Password, Teléfono, Estado, NewsLetter)

Productos(Id, Nombre, Descripción, Precio, CATEGORIAID, COLECCIONID, DESCUENTOID)

Stock(IDPRODUCTO, IDVARIACION, stock)

VariacionesProductos(Id, IDPRODUCTO)

Colores(Id, ColorPatron, ColorBase, IDVARIACION)

TallasProductos(Id, IDVARIACION, talla)

Compras(Id, Fecha, EstadoPago, IDMÉTODOPAGO, EMAILUSUARIO)

Reseñas(Id, Valoración, Descripción, EMAILUSUARIO, IDPRODUCTO)

Descuentos(Id, Nombre, Descripción, FechaIni, FechaFin, Cantidad, Tipo)

Colecciones(Id, Nombre, Descripción)

Categorías(Id, Nombre, Descripción)

Imágenes (Id, tipo, ruta, IDPRODUCTO)

Reclamaciones (Id, Motivo, Descripción, Foto, Fecha, EMAILUSUARIO)

Método Pago(Id, Tipo)

Direcciones(Id, Número, CodPostal, Calle, Población, Puerta, Planta, EMAILUSUARIO)

Carrito (Id, EMAILUSUARIO, IDVARIACION, Cantidad, FechaAgregado)

DetallesCompra (Id, IDCOMPRA, IDVARIACION, IDTALLA, Cantidad, PrecioTotal)

Tokens (Id, IdToken, Expiración, EMAILUSUARIO)

3.2.2.1.4 Aplicación de reglas de normalización al modelo relacional

- 1FN

Al comprobar las entidades y sus atributos, nos damos cuenta de que cumple la 1FN ya que todos los campos son atómicos.

- 2FN

No se cumple la 2FN ya que hay campos que no dependen de forma completa

Usuarios(Email, Nombre, Apellido1, Apellido2, Password, Teléfono, Estado, NewsLetter)

Productos(Id, Nombre, Descripción, Precio, CATEGORIAID, COLECCIONID, DESCUENTOID)

Stock(IDPRODUCTO, IDVARIACION, stock)

VariacionesProductos(id, IDPRODUCTO, IDCOLOR)

Colores(Id, ColorPatron, ColorBase)

TallasProductos(IDVARIACION, IDTALLA)

Tallas(id, Nombre)

Compras(Id, Fecha, EstadoPago, IDMÉTODOPAGO, EMAILUSUARIO)

Reseñas(Id, Valoración, Descripción, EMAILUSUARIO, IDPRODUCTO)

Descuentos(Id, Nombre, Descripción, FechaIni, FechaFin, Cantidad, Tipo)

Colecciones(Id, Nombre, Descripción)

Categorías(Id, Nombre, Descripción)

Imágenes (Id, tipo, ruta, IDPRODUCTO)

Reclamaciones (Id, Motivo, Descripción, Foto, Fecha, EMAILUSUARIO)

Método Pago(Id, Tipo)

Direcciones(Id, Número, CodPostal, Calle, Población, Puerta, Planta, EMAILUSUARIO)

Carrito (EMAILUSUARIO, IDVARIACION, Cantidad, FechaAgregado)

DetallesCompra (IDCOMPRA, IDVARIACION, IDTALLA, Cantidad, PrecioTotal)

Tokens (Id, IdToken, Expiración, EMAILUSUARIO)

- 3FN

Nuestra base de datos si que estaría en 3 forma nominal ya que no hay dependencias transitivas.

3.2.2.1.5 Tipos de datos para el sistema gestor seleccionado

Los tipos de datos que se van a utilizar en el sistema de gestión de base de datos son los siguientes:

- **VARCHAR:** gracias a este tipo de datos se nos permite guardar una cantidad variable de caracteres hasta un límite especifico.
- **TEXT:** este tipo de datos sirve para el almacenamiento de grandes cadenas de caracteres, utilizado sobre todo en las descripciones.
- **ENUM:** este tipo de datos sirve para guardar una lista de posibilidades, dentro de la cuál solo se almacenará uno de todos los datos de la lista.
- **BOOLEAN:** este tipo de datos sirve para guardar verdadero o falso y aunque a la hora de trabajar MySQL lo convierte en el tipo TINYINT, el cual nos permite guardar 0 o 1, en este caso trabajaremos con BOOLEAN para una mejor comprensión en el momento de redacción del script.
- **DATE :** este tipo de datos sirve para guardar la fecha necesaria, guardando la fecha en formato YYYY-MM-DD (año, mes, día)
- **TIMESTAMP:** este tipo de datos sirve para además de guardar la fecha registrar la hora siguiendo el formato anterior añadiendo la hora el minuto y el segundo (H-m-s) convirtiéndolos a UTC.

- **DATETIME:** este tipo de datos sirve para además de guardar la fecha registrar la hora siguiendo el formato anterior añadiendo la hora el minuto y el segundo (H-m-s) pero este no hace la conversión horaria que se aplica con TIMESTAMP
- **INT:** este tipo de datos sirve para almacenar números enteros, los cuales nos van a ayudar a identificar campos siendo utilizados principalmente como clave primaria.
- **DECIMAL:** este tipo de datos sirve para almacenar números con decimales pudiendo ajustar la precisión del mismo, es decir, la cantidad de dígitos y decimales que queremos guardar.

3.2.2.2 SERVIDOR

3.2.2.2.1 LISTA DE FUNCIONES EN PHP

Para hablar de la lista de funciones de PHP hay que tener en cuenta que el desarrollo del proyecto esta basado en la arquitectura MVC (modelo, vista, controlador) por lo que vamos a hacer un desglose de los distintos apartados y ficheros de los que se disponen.

Empezaremos hablando del fichero "*index.php*", dentro de este fichero lo que encontramos es la lógica de redirección de la aplicación para proteger así los datos y el funcionamiento de la aplicación desarrollada. Esta lógica empieza por recopilar los ficheros de control, en los cuales vamos a tener las distintas funciones que realiza la aplicación comprobando a través de este fichero a que controlador pertenece la función y si existe.

Una vez que hemos aclarado el funcionamiento del fichero "*index.php*" podemos pasar a explicar las funciones de cada uno de los ficheros, siendo los primeros los controladores:

- Controlador Seguridad
- Controlador Navegación
- Controlador Tablas

Controlador Seguridad	
Función	Descripción
Login()	Esta función recibe los datos codificados en json, llamando a la función del modelo de seguridad comprobación() que nos dice si el usuario está dentro de la base de datos o no. Dependiendo de la respuesta del modelo nos enviará a una vista u otra.
cerrarSesion()	Esta función llama a la función cerrarSesion() del modelo seguridad

Controlador Navegación: en este fichero tenemos funciones que son simplemente para mostrar las páginas del sitio web, siendo funciones de una línea en las que se requiere la vista que necesitamos. Menos en el caso de la página de usuario en la cuál se comprueba si el usuario está registrado o no, en el caso de que no se le envía al login.

Controlador Tablas	
Función	Descripción
Home ()	Esta función crea un modelo de productos y saca todos los datos necesarios de los productos gracias a la función mostrarProductosHome() y una vez que tiene los productos los muestra en la vista correspondiente.
Register()	Esta función nos permite registrarnos dentro del sistema de usuarios, para así guardar las compras y poder acceder al historial.
volvarApi()	Esta función nos permite generar un JSON con los datos del producto que queremos visualizar gracias a la función del modelo de productos generarJSON().
paginaUsuario()	Esta función busca los datos del usuario gracias a la función del modelo de usuario datosUsuario() y la función generarTablaUser() del modelo compras.
generarCarrito()	Esta función nos permite generar un carrito de compras. En la cual recibimos un fichero json que decodificamos y comprobamos que no está vacío, sabiendo que no está vacío preparamos la cadena de texto que necesitamos ya que está es un json con objetos y para generar los datos requeridos solo necesitamos los datos sin caracteres como : “{,},[,],\”. Gracias a la función mostrarCarrito() del modeloCarrito sacamos la estructura html necesaria para visualizar el carrito.
gestionarUsuarios()	Esta función saca los datos de todos usuarios de la base de datos y nos envía la respuesta a la vista correspondiente.
gestionUsuario()	Esta función sirve para gestionar los datos de un usuario, en este caso lo que nos permite modificar es el estado del usuario.
cambiarEstado()	Esta función recibe un json con los datos del usuario que vamos a modificar y con esos datos llamamos a la función modificarEstado() del modeloUsuarios, dependiendo de si el cambio ha sido exitoso o no devolveremos un mensaje indicando el estado del cambio.
gestionarProductos()	Esta función nos permite mostrar la vista de gestión de productos, al tener que acceder al modeloProductos para sacar los datos de los mismos, hace que la

	tengamos que tener dentro del controlador de tablas y no en el de navegación. Para generar la tabla con los datos necesarios llamamos a la función del modeloProductos generarDatosTabla().
modificarProducto()	Esta función nos permite mostrar la vista de modificación de un producto en específico, al necesitar sacar datos de las distintas tablas involucradas en este proceso (Productos, Descuentos, Categorías y Colecciones) es necesario que este dentro de este controlador; gracias a las funciones sacarDatosMod() del modeloProductos, sacarDescuentos() del modeloDescuentso, sacarCategorias() del modeloCategorias, sacarColecciones(), del modeloColecciones y sacarStock() del modeloProductos obtenemos los datos necesarios para mostrar la vista.
actualizarProducto()	Esta función nos permite actualizar los datos del producto que hemos modificado. Esta recibe un fichero json con los datos del producto se decodifican a un array asociativo y se pasan por la función modificarProducto() del modeloProductos una vez se ha ejecutado esa función se devuelve el estado de la acción ejecutada.
addProduct()	Esta función nos permite sacar la vista del formulario de adición de producto, al necesitar de los modelos: Productos, Descuentos, Categorias, Colecciones y Tallas esta función se encuentra en este controlador y no en el de navegación. Gracias a las funciones sacarDescuentos(), sacarCategorias(), sacarColecciones() y sacarTallas() obtenemos los datos necesarios para rellenar el formulario de creación de producto.
addProducto()	Esta función nos permite añadir el producto creado a la base de datos. Para ello obtiene un fichero json por parte del cliente que se decodifica a un array asociativo y se pasa a la función addProduct() del modeloProductos una vez que se ha ejecutado esa función se

	guarda el estado final y se devuelve al cliente.
eliminarProd()	Esta función nos permite eliminar productos de la base de datos, para ello recibimos un json de parte del cliente con los datos del producto a eliminar, se decodifica en un array asociativo y se pasa a la función eliminarProd() del modeloProductos una vez ejecutada esta función se devuelve el estado de la acción al cliente.
gestionarColecciones()	Esta función nos genera la vista necesaria para gestionar las colecciones de nuestra base de datos, al necesitar acceso al modeloColecciones, se guarda en este controlador y no en el de navegación. Para generar la vista llamamos a la función datosTabla() del modeloColecciones y guardamos los datos en una variable que se usará en la vista para generar la tabla.
eliminarColeccion()	Esta función nos permite eliminar colecciones de la base de datos, para ello recibimos un json por parte del cliente con los datos de la colección que vamos a eliminar y llamando a la función eliminarColeccion() del modeloColecciones obtenemos una respuesta del estado de la acción que devolvemos al cliente.
actualizarColeccion()	Esta función nos permite actualizar una colección en específico de nuestra base de datos, para ello obtenemos un json por parte del cliente con los datos de la colección que vamos a actualizar, los cuales decodificamos a un array asociativo. Con los datos almacenados en un array llamamos a la función actualizarColeccion() del modeloColecciones y una vez ejecutada esta función guardamos el estado de la acción en una variable para devolvérsela al cliente.
crearColeccion()	Esta función nos permite crear una colección dentro de la base de datos, para ello recibimos un json con los datos de la nueva colección los cuales decodificamos y guardamos en un array asociativo, para luego llamar a la función crearColeccion() del modeloColeccion

	guardando el estado de la acción en una variable que luego codificamos en json y devolvemos al cliente.
gestionarDescuentos()	Esta función nos obtener los datos necesarios para mostrar la vista de gestión de descuentos, al tener que hacer uso del modeloDescuentos para sacar los datos de requeridos para la vista se encuentra dentro de este controlador y no del navegador. Gracias a la función generarTabla() del modeloDescuento sacamos los datos necesarios y los pasamos a vista correspondiente.
crearDescuento()	Esta función nos permite crear un descuento, para ello obtenemos los datos del formulario enviado por el cliente en formato json, el cual decodificamos y pasamos a la función crearDescuento() del modeloDescuentos, el resultado de esta acción es guardado en una variable y devuelto en forma de json al cliente.
eliminarDescuento()	Esta función nos permite eliminar un descuento dentro de la base de datos, para ello obtenemos los datos enviados por parte del cliente en formato json los cuales son decodificados y almacenados en una variable, la cual se pasa a la función borrarDescuento() del modeloDescuento el resultado de esta acción se guarda en una variable que se devuelve al cliente.
actualizarDescuento()	Esta función nos permite actualizar los datos de un descuento en específico de la base de datos, para ello obtenemos los datos del formulario enviado por parte del cliente en formato json, el cual se decodifica y guarda en variable con formato de array asociativo. Con la variable llamamos a la función actualizarDescuento() del modeloDescuento y el resultado de esta acción se almacena en una variable que se envía de nuevo al cliente.
gestionarCategorias()	Esta función nos permite mostrar los datos de las categorías de la base de datos dentro de la vista correspondiente. Al hacer uso del modeloCategorias dentro de esta función es necesario tenerla en este controlador en lugar de en el navegador, gracias a este modelo y a su

	función generarTabla() obtenemos los datos que se quieren mostrar en la vista y los cargamos en la misma.
eliminarCategoria()	Esta función nos permite eliminar categorías de la base de datos, obteniendo en formato json los datos enviados por el cliente, estos datos se decodifican a un array asociativo que se usa como argumento en la función eliminarCategoria del modeloCategorias el resultado de esta acción se guarda en una variable y se envía al cliente de nuevo.
actualizarCategoria()	Esta función nos permite actualizar una categoría en especifico de la base de datos, para ello obtenemos los datos del cliente mediante un json que se decodifica y se guarda en una variable como array asociativo esta variable se pasa a la función actualizarCategoria() del modeloCategorias, guardando el resultado de la acción en una variable que se vuelve a enviar al usuario.
crearCategoria()	Esta función nos permite crear categorías nuevas en la base de datos, para ello obtenemos los datos de la nueva categoría mediante un json enviado por el cliente, el cual se decodifica y guarda en una variable como array asociativo esta variable se pasa a la función crearCategoria() del modeloCategorias, la respuesta de esta acción se guarda en una variable que se codifica en json y se devuelve al cliente.
gestionarCompras ()	Esta función saca los datos necesarios de las compras de la base de datos gracias a la función generarTabla() del modeloCompras y se envían al cliente para su uso.
sacarDetalles()	Esta función nos permite sacar los detalles de una compra especificada por el cliente, obteniendo los datos mediante una json que se decodifica y se guarda en una variable como array asociativo la cual se pasa a la función sacarDestalles() del modeloCompras, los datos que nos devuelve se codifican otra vez a json y se envían al cliente.
actualizarCompra()	Esta función nos permite actualizar el estado de una compra dentro de la base

	de datos. Para ello obtenemos los datos del cliente en formato json que se decodifican y guardan en una variable como array asociativo. Estos datos se desestructuran en dos variable id y estado, estas variables se pasan a la función actualizarCompra() del modeloCompras y dependiendo del estado de la acción devolvemos un mensaje apropiado.
gestionarIncidencias()	Esta función nos permite cargar los datos necesarios para visualizar la vista de gestión de Incidencias del panel de administración para ello se utiliza la función sacarReclamaciones() del modeloReclamaciones estos datos se cargan en la vista necesaria.
cambiarIncidencia()	Esta función nos permite cambiar el estado de una incidencia dependiendo de si se ha trabajado con ella o no. Para ello obtenemos el estado mediante un json por parte del cliente, este se decodifica y guarda en una variable como array asociativo el cual se pasa a la función cambiarEstado() del modeloReclamaciones. El estado de esta acción se almacena en una variable y dependiendo de este se envía un mensaje al cliente.
enviarReclamacion()	Esta función nos permite registrar una reclamación dentro de la base de datos, para ello obtenemos los datos de la reclamación mediante un json por parte del cliente, este se decodifica y guarda en una variable como array asociativo, además, se crean dos variables más que son la fecha de la reclamación y la foto de la misma, con todos estos datos se llama a la función guardarReclamacion() del modeloReclamaciones y el estado de esta acción, se guarda en una variable, dependiendo del estado de esta variable se envía un mensaje al cliente.
modDireccion()	Esta función nos permite modificar nuestra dirección como usuario. Para ello se obtienen los datos del formulario a través de un json el cual se decodifica y guarda en una variable como array asociativo, esta variable se pasa a la función modDireccion() del

	modeloDireccion obteniendo el resultado de la acción realizada, este se almacena y dependiendo de su valor se envía un mensaje.
guardarCompra()	Esta función nos permite registrar una compra dentro de la base de datos. Para ello obtenemos los datos guardados en el carrito del usuario mediante un json que decodificamos y guardamos en un array asociativo llamado carrito, además, se crean dos variables precTotal y prodGuardados que se usaran más adelante. Primero se hace un bucle recorriendo el array carrito para calcular el precio total de la compra sacando el precio de cada producto y multiplicado por la cantidad del mismo obtenemos el precio total de ese producto, este se añade a la variable precTotal repitiendo esto hasta que no tengamos más productos en carrito. Con el precio total y el usuario que nos llega dentro del carrito, usamos la función addCompra() del modeloCarrito que nos devuelve el id de la compra añadida y el estado de la acción. Una vez obtenido el id de la compra se procede a guardar los detalles de la misma dentro de la base de datos. Volvemos a recorrer el carrito, ya que dentro de los detalles de la compra se guarda el id de la variación del producto, la cantidad, la talla y el precio total del producto por la cantidad. Dentro de este segundo bucle for, lo hacemos es sacar los colores del producto, una vez tenemos estos colores en dos variables los pasamos junto al id del producto a la función sacarVariacion() del modeloProducto y lo guardamos en una variable. Se calcula el precio total de la cantidad de producto que toque en cada ciclo y junto a los datos mencionados anteriormente se pasan a la función guardarDetalles() del modeloCompras. Según el estado de esta acción añadimos uno al contador prodGuardados. Por ultimo se verifica que el total de los productos del carrito sea igual al total de los productos guardados, si esto es así se

	envía un mensaje indicando que se ha guardado la compra, en caso contrario se envía un mensaje de error.
datosUsuario()	Esta función nos permite sacar los datos necesarios para generar la vista de usuario. Gracias a la función datosUsuario() del modeloUsuario obtenemos los datos del usuario que necesitamos y los enviamos al cliente para su visualización en la correspondiente vista.

Una vez terminado con los controladores del proyecto podemos pasar a explicar las funciones de los distintos modelos creados para el correcto funcionamiento de la aplicación.

Los distintos modelos que tenemos dentro de la aplicación son:

- conexion.php
- modelo_carrito.php
- modelo_categorias.php
- modelo_colecciones.php
- modelo_compras.php
- modelo_descuento.php
- modelo_direccion.php
- modelo_productos.php
- modelo_reclamaciones.php
- modelo_seguridad.php
- modelo_tallas.php
- modelo_usuarios.php

Modelo Conexión	
Función	Descripción
Conectar()	Función estática que sirve para iniciar la conexión con la base de datos. En esta declaramos los datos de la conexión (dsn, usuario y contraseña) y mediante el objeto PDO d php iniciamos una conexión a la base de datos, además modificamos el atributo de los errores del objeto PDO para que lance excepciones en caso de error. Una vez realizado todo esto devolvemos la conexión mediante una variable.

Gracias a este modelo hacemos que todos los siguientes modelos lo importen teniendo así la función estática conectar() que como se menciona anteriormente sirve para conectar con la base de datos, permite a los distintos modelos poder interactuar con esta. Para ello se declara en todos los modelos un atributo privado llamado conex, que en el constructor de la clase de cada modelo se inicia con la función conectar(), haciendo que todos los modelos vean la base de datos.

Cabe recalcar que todos los siguientes modelos comparten esta función constructora, añadir que todas las funciones que se conecten a la base de datos están controladas por una estructura de try catch, que aunque no sea especificado dentro de cada función se ha tenido en cuenta y desarrollado correctamente, por lo que una vez explicado todo esto se van detallar las distintas funciones correspondientes a cada modelo.

Modelo Carrito	
Función	Descripción
mostrarCarrito($jsonDecode)	Esta función requiere de un array asociativo como argumento para su correcto funcionamiento. Al principio se declaran 4 variables distintas, un array de productos, un producto que a su vez es un array, un contador de productos y una variable llamada html. Mediante un bucle for recorremos el array pasado por argumento a la función. Por cada iteración de este bucle se genera un identificador para el producto a mostrar y un array llamado celda en la que se guarda la clave y el valor de cada iteración. Se comprueba que la primera casilla del array celda se llame "*IdProd*" y además se comprueba que el contador del bucle sea distinto de 0, si se cumplen estas condiciones, se guarda el producto dentro del array de productos generando un

	array asociativo con el identificador del producto y los datos del mismo, se vacía el producto (ya que está guardado en el array) y añadimos 1 al contador de productos para generar el siguiente índice. Seguidamente se construye el producto con los datos del array celda para la siguiente iteración. Por cada iteración se comprueba si es la última para así asegurarnos de que se guarda el último producto. Una vez se construye el array de productos, esta función genera el html que se va a mostrar en la página de carrito, para ello se usa un bucle foreach en la que cada iteración sobre el array de productos se comprueba si la categoría es de accesorios o no ya que estos no tienen talla. Una vez terminados todos los procesos de esta función se devuelve la cadena del html que se ha construido. Esta función trabaja de esta manera, ya que a la hora de cargar la vista, se llama desde el controlador de navegador el cual no tiene acceso a los modelos, por lo que desde el cliente cuando se carga, se hace una petición al servidor en la que se llama al controlador de tablas y realiza esta función, al obtener el html de esta manera luego es el cliente el que lo carga dentro de la vista correspondiente.

Modelo Categorias	
Función	Descripción
sacarCategorias()	Esta función nos permite sacar las distintas categorías de la base de datos y devolverlas al controlador para que se muestren en la vista. Mediante una sentencia sql en la cual se piden el id y el nombre de todas las categorías de la base de datos, se obtienen los datos que se guardan en un array asociativo y se devuelve al

	controlador, en caso de que no se efectúe esta acción se devolverá falso.
generarTabla()	Esta función nos permite sacar los datos de todas las categorías de la base de datos que son necesarios para construir la tabla del panel de administrador que gestiona las mismas. Mediante una sentencia sql se obtienen todos los datos y se devuelven mediante un array asociativo al controlador, en caso de que la acción no sea exitosa se devolverá falso.
eliminarCategoria($id)	Esta función nos permite eliminar una categoría de la base de datos, requiriendo el id de la categoría a eliminar por argumento. Mediante una sentencia sql en la que se específica el id de la categoría a borrar ejecutamos la acción, en caso de que se borre la categoría se devolverá un mensaje indicando que ha sido exitosa y en caso contrario se devolverá false.
actualizarCategoria($datos)	Esta función nos permite actualizar los datos de una categoría en específico, para ello mediante argumento se le pasa un array en el cuál se encuentran los datos de la categoría a actualizar, (id, nombre y descripción). Lo primero que se hace es una desestructuración de los datos separando, en tres variables los distintos datos pasados por argumentos. Mediante una sentencia sql se actualizan el nombre y la descripción de la categoría especificada por el id. En caso de que la actualización sea correcta se devolverá un mensaje indicando este hecho, en caso contrario se devolverá false.
crearCategoria($datos)	Esta función nos permite crear una categoría dentro de la base de datos, para ello se requiere un array con los datos a introducir que se pasarán por argumento. Lo primero que se hace es una desestructuración de los datos separándolos en dos variables. Mediante una sentencia sql se introduce la nueva categoría, si esta acción ha sido exitosa, se guarda el id de la categoría creada mediante la función lasInsertId()

	de la clase PDO y se devuelve un array con el mensaje exitoso y la identificación de la nueva categoría, en caso contrario se devolverá false.

Modelo Colección	
Función	Descripción
sacarColecciones()	Esta función nos permite sacar el id y el nombre de las colecciones de la base de datos. Mediante una sentencia sql en la que se indican los datos a sacar, se realiza esta acción, se comprueba si ha sido exitosa o no y en caso de que si, se devolverá un array con todas las filas que devuelva esta acción, en caso contrario se devolverá false.
datosTabla()	Esta función nos permite sacar los datos necesarios para generar la tabla de la vista de gestión de colecciones del panel de administración. Mediante una sentencia sql se sacan todos los datos de la base de datos correspondientes a las colecciones y en caso de que esta acción sea exitosa se devolverá un array, en caso contrario se devolverá false.
eliminarColeccion($id)	Esta función nos permite eliminar una colección de la base de datos. Para ello se requiere el id de la colección a eliminar que se pasará por argumento y mediante una sentencia sql se eliminará. En caso de que sea exitosa esta acción se devolverá un mensaje indicando que así ha sucedido, en caso contrario se devolverá false.
actualizarColeccion($datos)	Esta función nos permite actualizar los datos de una colección en específico de la base de datos, para ello requieren los datos de la colección por argumentos. Lo primero que se hace es sacar los datos de la colección a actualizar a tres variables, (id, nombre y descripción). Mediante una sentencia sql se indica la colección que se va a modificar y los datos que se van a modificar de esta. En caso de que la acción sea exitosa se devolverá un mensaje indicando lo

	sucedido, en caso contrario se devolverá false.
crearColeccion($datos)	Esta función nos permite crear una nueva colección dentro de la base de datos. Para ello se requieren los datos por argumento, estos se sacan a dos variables distintas (nombre y descripción). Mediante una sentencia sql se insertará en la base de datos la nueva colección. Se comprueba que la inserción ha sido exitosa y en ese caso se devuelve un array con un mensaje y con el id de la nueva colección, en caso contrario se devuelve false.

Modelo Compras	
Función	Descripción
addCompra($usuario, $precio)	Esta función nos permite añadir una compra dentro de la base de datos, pidiendo por argumentos el usuario al que se va a enlazar la compra y el precio total de la compra a guardar. Creamos tres variables, $hoy, $estado, $metodo, a estas variables les asignamos un valor predeterminado para poder añadir los datos nuestra base de datos, a $hoy le asignamos el valor de la fecha del día que se realiza la compra, $estado se queda en pendiente y $metodo asignamos 1 que según lo que tenemos en la base de datos corresponde a pago con tarjeta. Una vez que tenemos estas variables establecidas, con una sentencia sql insertamos dentro de la base de datos, en caso de que la acción sea exitosa devolvemos true y el id de la compra que hemos añadido, en caso contrario devolvemos false.
guardarDetalles($idCompra, $idVariacion, $idTalla, $cantidad, $precio)	Esta función nos permite guardar los detalles de la compra realizada en nuestra base de datos. Gracias a los datos pedidos por argumentos y mediante una sentencia sql guardamos dentro de la base de datos y en caso de que la operación sea exitosa devolvemos true y en caso contrario devolvemos false.
generarTabla()	Esta función nos permite sacar los datos necesarios para generar la tabla de la

	vista de gestión de compras del panel del administrador. Mediante una sentencia sql sacamos los datos necesarios (id, fecha, estado del pago, usuario y preciototal) en caso de que sea correcta la ejecución se guardan los datos en un array asociativo y se devuelve, en caso contrario se devuelve false.
generarTablaUser($usuario)	Esta función nos permite, mediante una sentencia sql sacar los datos de las compras de un usuario en específico, usada en el panel de usuario para ver el historial de compras. Mediante una sentencia sql y gracias al usuario pasado por argumentos, sacamos los datos del historial del usuario en concreto. En caso de que la acción sea correcta se guardan los datos en un array asociativo y se devuelven, en caso contrario se devuelve false.
sacarDetalles($id)	Esta función nos permite, mediante una sentencia sql sacar los datos en concreto de una compra especificada por el id que se pasa por argumentos. En caso de que la acción sea correcta se guardan los datos en un array asociativo y el total de las filas devueltas, estos datos se devuelven, en caso contrario se devuelve false.
actualizarCompra($id, $estado)	Esta función nos permite actualizar el estado de una compra, gracias a los datos pedidos por argumentos y combinados por una sentencia sql se actualiza. En caso de que la acción sea correcta se devuelve true, en caso contrario se devuelve false.

Modelo Descuento	
Función	Descripción
sacarDescuentos()	Esta función nos permite sacar todos los descuentos que por fecha están activos dentro de la base de datos. Mediante una sentencia sql sacamos los datos necesarios filtrando por la fecha, en caso de que esta acción sea exitosa devolvemos un array asociativo con los

	datos, en caso contrario devolvemos false.
generarTabla()	Esta función nos permite sacar los datos para generar la tabla de la vista de gestión de descuentos del panel de administrador. Mediante una sentencia sql sacamos todos los descuentos, si esta acción se realiza correctamente devolvemos un array asociativo con los datos, en caso contrario devolvemos false.
crearDescuento($datos)	Esta función nos permite añadir un descuento a la base de datos. Para ello pasamos los datos en forma de array asociativo a la función por argumentos. Lo primero que se hace es sacar los datos del array mediante una desestructuración para tener los datos guardados en variables. Generamos el sql necesario para añadir el descuento y lo ejecutamos con los datos de las variables. En caso de que la acción haya sido realizada correctamente, se genera un array resultado, con el mensaje y el id del descuento creado, este array se devuelve, en caso contrario se devuelve false.
borrarDescuento($id)	Esta función nos permite borrar descuentos de la base de datos. Mediante una sentencia sql y el id que se le pasa por argumentos podemos seleccionar exactamente que descuento se quiere borrar de la base de datos. En caso de que la acción sea realizada correctamente devolvemos un mensaje indicándolo, en caso contrario devolvemos false.
actualizarDescuento($datos)	Esta función nos permite actualizar los datos de un descuento, para ello gracias a los datos que se pasan por argumentos. Lo primero que hacemos es sacar el id del descuento que vamos a actualizar y se guarda en una variable. Seguidamente se comprueba que es lo que se actualiza del descuento en cada comprobación se ejecuta una sql personalizada que actualiza ese dato si se requiere. Si la acción es realizada correctamente se devuelve un mensaje de éxito, en caso contrario se devuelve un mensaje de error. En caso de que ninguna de las

	comprobaciones sea correcta se devuelve un mensaje indicando que los datos son incorrectos y que sean revisados.

Modelo Dirección	
Función	Descripción
modDirección($numero, $codPostal, $calle, $poblacion, $puerta, $planta, $usuario)	Esta función nos permite modificar los datos de la dirección, en el panel de usuario. Para ello gracias a los datos que se pasan por argumentos actualizamos la dirección. Lo primero que hacemos es borrar las direcciones anteriores del usuario para asegurarnos de que los datos que se van a introducir sean los nuevos del usuario y únicos. Una vez que hemos borrado los datos del usuario añadimos a la base de datos los nuevos datos, si la acción ha sido correcta, se devuelve true, en caso contrario se devuelve false. En este caso, en lugar de hacer un update se ha optado por eliminar y añadir de nuevo los datos introducidos por el usuario, ya que en la modificación puede haber datos que no haya añadido el usuario y en solo uno de los casos cambian todos los datos pudiendo hacer un update.

Modelo Productos	
Función	Descripción
mostrarProductosHome()	Esta función nos permite sacar los datos necesarios para visualizar los productos en la página principal de web. Mediante una sentencia sql sacamos los datos necesarios, si esta acción es correcta se guardan los datos en un array asociativo y se devuelven, en caso contrario se devuelve un mensaje de error.
generarJson($id, $categoria)	Esta función nos permite construir un json con los datos de los productos para el carrito. Lo primero que se hace es crear las variables en las que se van a almacenar los datos. Lo primero que se hace es comprobar que la categoría del producto no sea

	"Accesorios", ya que al estos no tener tallas no habría que guardarlas. Pasado el filtro de accesorios o no, se crea una sentencia sql con la que se podrán sacar los datos del producto, si esta sentencia se ejecuta correctamente y devuelve datos, se guardan en un array asociativo. Con el array de los datos devueltos por la sentencia sql se hace un bucle foreach recorriendo este array. En cada iteración se comprueba si la talla que tenemos almacenada es distinta a la anterior, en caso de que lo sea se guarda la nueva en una variable, Se comprueba si la talla esta guardada junto a su identificador, si no es así se guarda en un array. Se comprueba si dentro de las fotos esta la foto del producto actual, si no esta se añade. Se guardan los colores en dos variables y se comprueban si existen dentro del array de stock, en caso de que no existan se crea esa nueva casilla y se le añade el valor del stock. Se comprueba si la talla no existe dentro del array que une los colores con las tallas y dentro del array de tallas, si es así se añade dentro del array de las tallas la talla y dentro del array de colores se guarda la talla y el color al que pertenece. En caso de que exista se comprueba que los colores no estén repetidos para así añadir más colores una misma talla. Se suma uno al contador de tallas. Por ultimo se genera el array asociativo con todos los datos y se codifica a json, este se devuelve. Para los accesorios el proceso sería el mismo, pero sin la parte de las tallas. En ambos casos si la sentencia sql falla se devuelve false.
generarDatosTabla()	Esta función nos permite sacar los datos necesarios para la vista de gestión de productos del panel de administrador. Mediante una sentencia sql se sacan los datos de los productos de la base de datos y se almacenan en un array asociativo el cual se devuelve si todo ha sido correcto,

	en caso de que no se encuentren datos se devuelve false.
sacarDatosMod($id, $categoria)	Esta función nos permite sacar los datos de un producto en específico para su modificación. Lo primero que se hace es comprobar la categoría del producto que se va a visualizar. Dependiendo si es un accesorio o no se declarará una sentencia sql para sacar los datos necesarios. Estos datos se guardan en un array asociativo. Seguidamente se sacan los datos a un array auxiliar para poder trabajar de forma más cómoda. Se realiza un bucle for hasta la longitud del array con los datos que nos devuelve la base de datos, en este bucle se hace lo siguiente por cada iteración: Se guarda la ruta de la foto. Se guarda el id de la talla. Se comprueba si la talla existe dentro del array de tallas, si no es así se guarda junto con los colores que lleva asignados, si por el contrario si lo está se crea un array auxiliar con los datos de los colores y la talla para poder comprobar si están guardados, en caso de que no estén se añaden los colores a la talla correspondiente. Se comprueba que si la foto está guardada y se añade al array con las demás fotos. Por ultimo se crea un array producto con los datos del mismo, los datos de las tallas y las fotos, este array es el que se devuelve. Para los accesorios es el mismo proceso, pero sin las tallas.
sacarStock($id)	Esta función nos permite ver el stock de los productos dentro de la base de datos. Mediante una sentencia sql y gracias al id que se pasa como argumento sacamos el stock del producto indicado, si la acción es correcta se guardan los datos dentro de un array asociativo y se devuelven.
modificarProducto($datos)	Esta función nos permite modificar los datos de un producto dentro de la base de datos.

	Para ello se crean tres variables para comprobaciones posteriores, con la dirección de subida de fotos, los tipos permitidos de fotos y un array para recolectar el resultado del procesamiento de las fotos. Lo primero que se hace es separar los datos del array del argumento en variables para su correcto tratamiento, una vez que tenemos los datos separados empieza una transacción de PDO. En esta transacción lo primero que se hace es comprobar las fotos, si la ruta es un array y no está vacío se sacan las fotos de la base de datos para comprobar que la foto que estamos viendo no está repetida y así evitar datos duplicados. Seguidamente se crean dos sentencias sql, ya que puede ser que se haya borrado alguna foto o que se haya añadido alguna. Para cada foto que nos devuelve la base de datos comprobamos que si la foto existe o no, comparando las rutas de guardado. En caso de que las rutas coincidan significa que la foto existe y que está guardada, en caso de no coincidan, significa que el administrador ha borrado esa foto del producto, por lo que se elimina de la base de datos y se borra de la carpeta de fotos. Comprobadas todas las fotos, se comprueban ahora las fotos que el administrador ha añadido, estas vienen con un nombre definido por el navegador, sabiendo esto podemos ver si es una foto que se haya añadido por el usuario o una foto que haya sido generada por el servidor. Si la foto es nueva, lo que tenemos que hacer es modificar el nombre, ya que la foto va vinculada al color que corresponde por el nombre de esta. Para ello se comprueba el tipo de la foto y la extensión que lleva y además, se guardan los datos binarios de la imagen para su posterior subida al servidor. Se comprueba si el texto alternativo de la imagen esta vació después de su depuración anterior, para asi guardar un nombre único. En caso de que no este

	vacío se guarda ese nombre junto a la extensión y en caso de que este vació se crea “manualmente” el nombre. Para terminar con las fotos se comprueba si el fichero existe dentro del servidor y en caso de que exista un fichero con ese nombre, se borra, esto se hace por que puede ser que se haya querido cambiar una foto porque estuviera vinculada a un color incorrecto y así además, se evitan duplicados. Se sube el fichero al servidor y se guarda la ruta dentro de la base de datos.
	En el caso de los colores se sacan todos de la base de datos y se guardan en un array asociativo. Para su tratamiento empezamos recorriendo los colores que nos ha pasado el usuario con un bucle for hasta la longitud del array de colores del usuario, dentro de cada color se hace un primer bucle foreach en el que se comprueba si el color que estamos comprobando por parte del usuario existe dentro de la base de datos, si existe se guarda su id y los colores correspondientes. En caso de que no exista, se separan los colores en color base y color patrón para añadirlos a la base de datos. Y se saca el id del nuevo color creado para guardarlo así en el array de colores.
	En el caso de las variaciones, se sacan los datos de las correspondientes al producto que se está modificando. Se guardan los datos que nos devuelve la base de datos en un array asociativo y se recorre con un bucle foreach. En cada iteración se comprueba si la variación existe, recordemos que las variaciones se componen del id de la talla, el id de los colores y el id del producto, dentro de la base de datos. Para ello recorremos el array construido anteriormente de colores y en caso de que coincidan los colores significa que la variación existe. En caso de que no exista significa que el administrador la ha borrado. Una vez que se han comprobado que variaciones se han eliminado se

	comprueban cuales se han añadido, para ello se recorre el array de colores creado anteriormente. Dentro de cada iteración se busca dentro de las variaciones para saber si ya existe una variación con esas características o no, en caso d que no exista se añade a la base de datos. Como es posible que se hayan añadido nuevas variaciones se vuelven a sacar los datos de las variaciones.
	En el caso del stock se saca el stock de cada variación. Para comprobar el stock se realiza mediante un bucle foreach. Primero recorremos mediante otro foreach las nuevas variaciones para comprobar si la variación tiene stock vinculado o no, en caso de que los id coincidan se guarda el stock en una variable, gracias a la cual podemos saber si se ha actualizado el stock del producto o no, en caso de que haya sido actualizado se cambia dentro de la base de datos. En caso de que no exista stock, significa que el administrador ha borrado esa variación por lo tanto no hay que guardar stock. Para comprobar si se ha añadido stock de una variación nueva se hace al revés, se recorren las nuevas variaciones y se comprueba con el stock existente, en caso de que no exista significa que el administrador ha añadido una nueva variación por lo que hay que guardar el nuevo stock.
	En el caso de las tallas sacamos los datos que nos dicen que tallas tienen las distintas variaciones. Con estos datos guardados dentro de un array asociativo se empieza a trabajar. Primero se recorre las tallas devueltas por el servidor y en cada talla se comprueba si existe con las nuevas variaciones, en caso de que no exista significa que se ha borrado la talla de la variación borrando así de la base de datos la vinculación de la talla a la variación. Seguidamente hace el proceso inverso para comprobar si se ha añadido alguna

	variación a una talla y en caso de que así sea se vinculan en la base de datos. Por último, se comprueba si se le ha aplicado algún descuento al producto o no. Con estos datos comprobados, se actualizan los datos propiamente del producto y se termina la transacción, devolviendo un mensaje de éxito, en caso de que algo falle se para la transacción y se devuelve un mensaje de error.
addProduct($datos)	Esta función nos permite añadir un producto a la base de datos. Para ello se crean tres variables para comprobaciones posteriores, con la dirección de subida de fotos, los tipos permitidos de fotos y un array para recolectar el resultado del procesamiento de las fotos. Lo primero que se hace es separar los datos del array del argumento en variables para su correcto tratamiento, una vez que tenemos los datos separados empieza una transacción de PDO. La transacción empieza comprobando si el se ha aplicado algún descuento al producto y en caso de que se haya aplicado o no se comprueba mediante una sentencia sql si los datos del producto no existen dentro de la base de datos y dependiendo de si se ha aplicado un descuento o no la sentencia cambia. Una vez que se han comprobado los datos, en el caso de que coincidan se guarda el id del producto con el que coincide. Se comprueba el id y en caso de estar vacío se añade el producto a la base de datos y se guarda el id, en caso de que exista se devuelve un mensaje indicándolo y se cancela la transacción. Una vez guardados los datos de la entidad producto dentro de la base de datos se empieza el proceso de depuración de datos para evitar duplicidad en la base de datos. Comenzando por las fotos, si la ruta es un array y no está vacío se sacan las fotos de la base de datos para comprobar que la

	foto que estamos viendo no está repetida y así evitar datos duplicados. Seguidamente se crean dos sentencias sql, ya que puede ser que se haya borrado alguna foto o que se haya añadido alguna. Para cada foto que nos devuelve la base de datos comprobamos si la foto existe o no, comparando las rutas de guardado. En caso de que las rutas coincidan significa que la foto existe y que está guardada, en caso de no coincidan, significa que el administrador ha borrado esa foto del producto, por lo que se elimina de la base de datos y se borra de la carpeta de fotos. Comprobadas todas las fotos, se comprueban ahora las fotos que el administrador ha añadido, estas vienen con un nombre definido por el navegador, sabiendo esto podemos ver si es una foto que se haya añadido por el usuario o una foto que haya sido generada por el servidor. Si la foto es nueva, lo que tenemos que hacer es modificar el nombre, ya que la foto va vinculada al color que corresponde por el nombre de esta. Para ello se comprueba el tipo de la foto y la extensión que lleva y, además, se guardan los datos binarios de la imagen para su posterior subida al servidor. Se comprueba si el texto alternativo de la imagen esta vació después de su depuración anterior, para asi guardar un nombre único. En caso de que no este vacío se guarda ese nombre junto a la extensión y en caso de que este vació se crea “manualmente” el nombre. Para terminar con las fotos se comprueba si el fichero existe dentro del servidor y en caso de que exista un fichero con ese nombre, se borra, esto se hace por que puede ser que se haya querido cambiar una foto porque estuviera vinculada a un color incorrecto y así además, se evitan duplicados. Se sube el fichero al servidor y se guarda la ruta dentro de la base de datos.
	En el caso de los colores se sacan todos de la base de datos y se guardan en un array asociativo.

	Para su tratamiento empezamos recorriendo los colores que nos ha pasado el usuario con un bucle for hasta la longitud del array de colores del usuario, dentro de cada color se hace un primer bucle foreach en el que se comprueba si el color que estamos comprobando por parte del usuario existe dentro de la base de datos, si existe se guarda su id y los colores correspondientes. En caso de que no exista, se separan los colores en color base y color patrón para añadirlos a la base de datos. Y se saca el id del nuevo color creado para guardarlo así en el array de colores.
	Una vez guardado y obtenido los datos sobre los colores, podemos guardar los detalles de las variaciones dentro de la base de datos, para ello recorremos el array de colores creado anteriormente y por cada color se añade el id del producto y del color a la base de datos, una vez que se ha terminado de añadir todas las variaciones se recogen los id de las nuevas variaciones y se guarda en un array asociativo.
	Con los datos de las nuevas variaciones se pasa a comprobar el stock de las mismas. Recorriendo las nuevas variaciones se comprueba que colores coinciden y se guarda el stock de dicho color, para añadirlo a la base de datos.
	Por último, se vinculan las tallas a las distintas variaciones, para ello se recorren los datos pasados por el usuario para comprobar que variación tiene cada talla. Por cada iteración del bucle foreach que recorre los datos del usuario se comprueba si existen colores dentro de la talla, en caso de que no existan colores se pasa a la siguiente iteración directamente, por el contrario se recorren los colores de la talla sacando el id del color gracias al array de colores creado anteriormente, con el id guardado en una variable se comprueban las variaciones, si el id del color coincide con el id del color de la variación se tiene que vincular esa variación a la talla.

	Una vez que se ha realizado este proceso con todas las tallas del producto a añadir, se termina la transacción y se devuelve un mensaje indicando que la ejecución ha sido correcta, en caso de que algo falle se para la transacción y se devuelve un mensaje de error.
eliminarProd($id)	Esta función nos permite eliminar un producto y sus imágenes del servidor y la base de datos. Mediante sentencias sql y gracias al id que se pasa por argumento podemos sacar todas las imágenes que tenemos que borrar. Una vez que tenemos las rutas de las fotos a borrar guardadas en un array, borramos el producto de la base de datos si el borrado es correcto, pasamos a eliminar las fotos del servidor y si todo sale bien, se devuelve un mensaje de éxito, en caso contrario se devuelve un mensaje de fallo.
sacarVariacion($id, $color1, $color2)	Esta función nos permite sacar el id de una variación sabiendo el color y el producto. Para ello lo primero que sacamos es el id del color de la variación mediante una sentencia sql y los datos color1 y color2. Con el id guardado en una variable y mediante una sentencia sql en la que se especifica el id del producto, que se ha pasado por argumento y el id del color extraído anteriormente, se saca el id de la variación que se está buscando, este se guarda en una variable y se devuelve. De forma predeterminada en la función se devuelve -1, para indicar así que no se ha encontrado la variación que se busca.

Modelo Reclamaciones	
Función	Descripción
guardarReclamacion($motivo, $descripcion, $foto, $fecha, $user)	Esta función nos permite guardar una nueva reclamación dentro de la base de datos, gracias a los datos que se pasan por argumento y mediante una sentencia sql se añade a la base de datos la nueva reclamación, si la acción ha sido realizada correctamente se devuelve true en caso contrario se devuelve false.

sacarReclamaciones()	Esta función nos permite sacar las reclamaciones para la vista de gestión de reclamaciones en el panel de administrador. Para ello sacamos los datos necesarios para construir esta tabla, (id, motivo, descripción, usuario y estado) mediante una sentencia sql. En caso de que esta acción se realice correctamente, se guardan los datos en un array asociativo y se devuelve, en caso contrario se devuelve false.
cambiarEstado($id, $estado)	Esta función nos permite cambiar el estado de una reclamación en específico. Para ello gracias a los datos que se pasan por argumentos podemos actualizar la reclamación al estado necesario mediante una sentencia sql. En caso de que la acción sea correcta se devuelve true, en caso contrario se devuelve false.

Modelo Seguridad	
Función	Descripción
Login()	Esta función permite el inicio de sesión dentro del sistema de usuarios, distinguiendo entre usuarios normales y administradores. Para ello lo primero que se hace es crear un modelo de usuario para poder acceder a sus funciones, con el modelo creado se comprueba que los datos que nos envían están siendo enviados a través de POST en caso de que sea así se guardan en variables, el email y la contraseña además, sacamos la contraseña que tenemos guardada del usuario mediante la función datosInicio() del modelo de usuario. Se crea un patrón para comprobar si la contraseña que se tiene guardada en la base de datos está cifrada o no. La primera comprobación que se hace es si tenemos algo dentro la contraseña de la base de datos, en caso de que no haya nada significa que el usuario no existe, en caso de que hayamos obtenido algo significa que el usuario existe. Pasado el primer filtro se renueva la id de la sesión de php y se comprueba si la

	contraseña coincide con la que se tiene guardada, en caso de que coincida se comprueba si el usuario que está intentando entrar es un administrador, en caso de que así sea se guarda dentro de la sesión que el usuario es administrador y se devuelve true, en caso de que sea un usuario normal, se guarda dentro de la sesión y se devuelve true, en caso de que o no exista o la contraseña sea incorrecta, se devuelve un array en el que se indica que el estado es false y un mensaje indicando porque.
comprobarAdminstrador($usuario)	Esta función nos permite saber si el usuario que está intentando iniciar sesión es un administrador o no, para ello se pasa el usuario por argumentos y gracias a un array en el que se indican los usuarios de los administradores, se comprueba y si coincide, significa que es un administrador y se devuelve true, en caso contrario no lo es y se devuelve false.
cerrarSesion()	Esta función nos permite cerrar sesión mediante la función session_destroy() de php.

Modelo Tallas	
Función	Descripción
sacarTallas()	Esta función nos permite sacar los datos de las tallas de la base de datos. Mediante una sentencia sql sacamos los datos, se guardan en un array y se devuelve. Esto nos permite saber que texto corresponde a cada talla, ya que en la base de datos, se guardan las tallas en una tabla y luego esta se relaciona con las variaciones de los productos mediante el id, haciendo que en el resto de tablas este guardado el id en lugar del texto correspondiente.

Modelo Usuarios	
Función	Descripción
datosInicio($email)	Esta función nos permite sacar la contraseña del usuario en el momento del inicio de sesión.

	Mediante una sentencia sql y gracias al argumento que le pasamos, podemos sacar la contraseña. Si recuperamos una contraseña se guarda en un array asociativo y se devuelve, en caso contrario se devuelve -1.
register($nombre, $apellido1, $apellido2 = null, $email, $password, $phoneNumber, $estado= true, $newsletter)	Esta función nos permite registrar nuevos usuarios dentro de la base de datos. Lo primero que se hace es una comprobación de que los datos que nos envía el usuario no estén vacíos, pasado ese filtro, creamos la sentencia sql y ciframos la contraseña, en caso de que la inserción sea correcta se devuelve true, en cualquier caso contrario se devuelve false.
mostrarUsuarios()	Esta función nos permite sacar los datos de los usuarios para mostrarlos dentro de la vista de gestión de usuarios del panel de administrador. Mediante una sentencia sql se sacan los datos necesarios, se comprueba si ha devuelto algún dato y en ese caso se guardan en un array asociativo que se devuelve, en caso contrario se devuelve un mensaje indicando que no hay usuarios.
sacarUsuario($email)	Esta función nos permite ver sacar los datos de un usuario en especifico. Mediante una sentencia sql y gracias al argumento que se le pasa, sacamos los datos del usuario, se guardan en un array asociativo y se devuelven, en caso contrario se devuelve un mensaje indicando que no se ha encontrado el usuario.
modificarEstado($email, $estado)	Esta función nos permite cambiar el estado de un usuario dentro de la base de datos. Mediante una sentencia sql y gracias a los argumentos que le pasamos se realiza la actualización, en caso de que está se realice correctamente se devuelve true, en caso contrario se devuelve false.
datosUsuario()	Esta función nos permite sacar los datos del usuario, para la visualización de su página personal. Gracias a una sentencia sql y utilizando el session para sacar el usuario se obtienen los datos guardados en la base

	de datos, si esta acción es correcta se guardan en un array asociativo y se devuelve, en caso contrario se devuelve false.
modPerfil($email, $nombre, $apellidos)	Esta función nos permite modificar los datos del usuario. Lo primero que hacemos es separar los apellidos en dos variables, ya que en la base de datos tenemos apellido1 y apellido2. Mediante una sentencia sql actualizamos los datos y en caso de que esta acción sea correcta se devuelve true, en caso contrario se devuelve false.

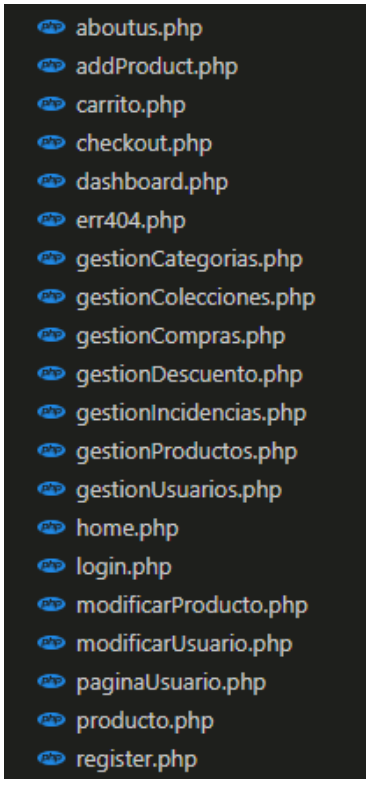
3.2.2.3 Cliente

3.2.2.3.1 Diseño de la interfaz

Para el diseño de la interfaz y siguiendo con la filosofía del proyecto de usar las tecnologías vanilla, se ha utilizado HTML y CSS ayudado de JavaScript para el correcto funcionamiento de esta.

Gracias al uso de estilos dentro de CSS y utilizando flexbox en la página se consigue el resultado esperado y facilita la adaptación a distintos terminales como pueden ser teléfonos móviles.

Dentro del proyecto se han desarrollado distintas páginas o vistas, que hacen que todo sea posible, siendo este el listado de las distintas vistas que se usan:



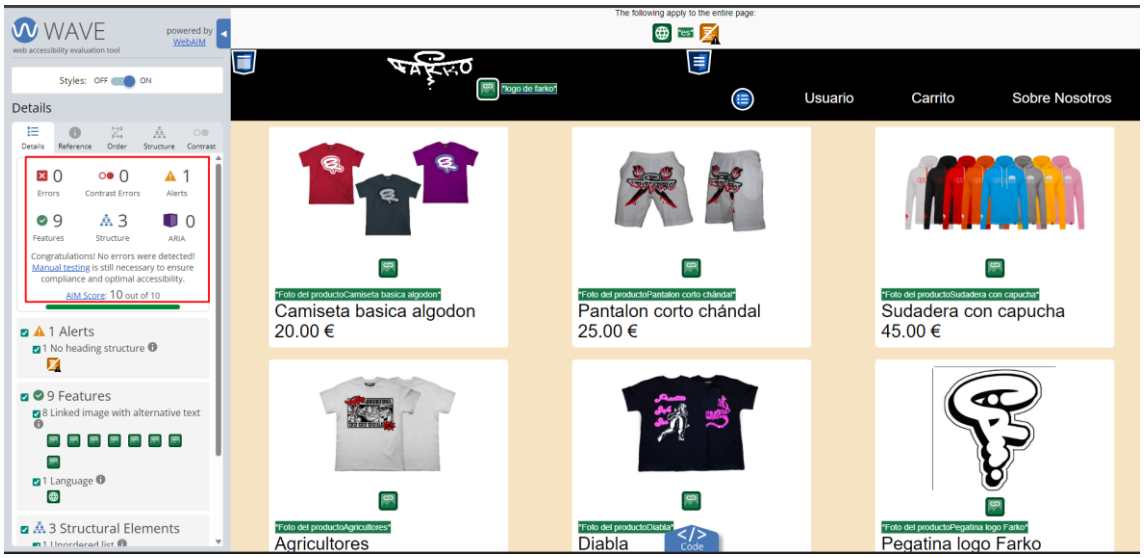
Los estilos que reciben estas vistas se encuentran dentro de la carpeta /estilos en la cual se encuentran todos los ficheros CSS utilizados en la estilización de la página. Estas vistas al compartir estilos como puede ser el del navegador o el de las alertas que saltan, estos se han guardado dentro del fichero estiloprincipal.css definiendo en este, estilos para el body, header o navegador entre otros.

```css
:root{
    --colorFondo: rgb(246, 227, 194);
    --colorFondoProductos: white;
}


body{
    background-color: var(--colorFondo);
    font-family: Arial, Helvetica, sans-serif;
    font-size: 14px;
    padding: 0;
    margin: 0;
    display: flex;
    flex-direction: column;
    align-items: center;
    overflow-x: hidden;
}

header{
    background-color: black;
    width: 100%;
    display: flex;
    flex-direction: row;
    justify-content: space-between;
    max-height: 10%;
    position: sticky;
    top:0;
}

nav{
    width: 50%;
    max-height: 100%;
}
```

Esto es un ejemplo de lo que se ha estilado dentro de los ficheros de estilos.

3.2.2.3.2 Accesibilidad

Para la comprobación de la accesibilidad se ha utilizado la herramienta WAVE, utilizando la extensión en el navegador Chrome.

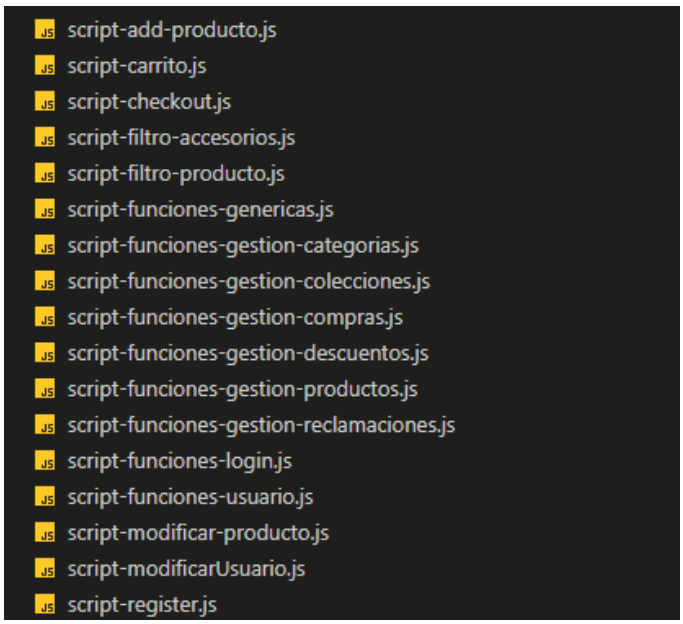


Como podemos ver en la captura de pantalla de la página principal de la aplicación esta no encuentra errores de legibilidad o fallos.
Al seguir una gama de colores común a lo largo de la aplicación se entiende que esta no va a tener problemas de accesibilidad, por ejemplo, con lectores de pantalla para personas con visión reducida.

3.2.2.3.3 Desarrollo web entorno cliente

Para el desarrollo del entorno cliente se ha utilizado la tecnología JavaScript en su estado base, sin librerías adicionales o frameworks que importen funcionalidades.

Esto conlleva que se tengan ficheros únicos, para cada página ya que estás tienen funciones específicas. El listado de todos los ficheros que se han utilizado para el desarrollo es el siguiente:



Todos estos ficheros comparten una función, que aun que haya sido definida en cada fichero se utiliza de la misma forma, que es la función activarListener(), esta función sirve para agrupar los listener de los distintos eventos que se han contemplado dentro de

las páginas, en esta función es donde se controla que los formularios estén completos antes de enviar, donde se recogen datos y donde se usan el resto de funciones de los ficheros.

Script-add-producto

- **cogerDatos()**: esta función nos permite recoger los datos del producto que se está creando.
- **coloresStock()**: esta función nos permite agrupar el stock de cada color.
- **coloresTallas()**: esta función nos permite agrupar las variaciones que se añaden a cada talla.
- **addProducto(datos)**: función asíncrona que nos permite hacer el envío de los datos que hemos recogido en el formulario y devuelve una alerta con el estado de la acción.
- **activarCruces()**: esta función sirve para activar el listener de las cruces de la página, ya que al momento de añadir datos se añaden cruces que antes no existían.

Script-carrito

- **enviarDatos()**: esta función nos permite enviar los datos del carrito guardados en el localStorage al servidor para generar la vista del carrito.
- **calcularTotal()**: esta función nos permite calcular el total del carrito de forma dinámica.
- **borrarCarrito()**: esta función nos permite vaciar el carrito tanto en la pantalla como en el localStorage.
- **actualizarCarrito()**: esta función nos permite actualizar las cantidades de los productos del carrito de forma dinámica.
- **eliminarProducto(id, div)**: esta función nos permite eliminar un producto en específico del carrito, tanto del localStorage como de la pantalla que se está viendo.
- **carritoVacio()**: esta función es para dar estilos al carrito en el momento de borrarlo o en caso de que este vacío cuando entremos a la vista.

Script-checkout

- **rellenarForm(datos)**: esta función asíncrona nos permite rellenar los datos del usuario en caso de que esté registrado a la hora de finalizar el pago.
- **pagar(datos)**: esta función asíncrona nos permite simular el pago de la compra.

Script-filtro-accesorios

- **rellenarProducto()**: esta función asíncrona nos permite recoger los datos el producto seleccionado y mostrar estos en la tarjeta de la vista del producto.
- **cambiarFoto(color, foto)**: esta función nos permite cambiar la foto dependiendo del color que se elija.

Script-filtro-producto

- **rellenarProducto()**: esta función asíncrona nos permite recoger los datos el producto seleccionado y mostrar estos en la tarjeta de la vista del producto.
- **generarSelects()**: esta función nor permite generar los desplegables de las tallas y colores.
- **cambiarCOLORES(talla)**: esta función nos permite mostrar los colores disponibles dependiendo de la talla que se haya seleccionado.
- **cambiarTALLAS(color)**: esta función nos permite mostrar las tallas disponibles según el color que se haya elegido.
- **cambiarFoto(color, fotos)**: esta función nos permite cambiar la foto dependiendo del color que se elija.

Script-funciones-genericas

- **addToCart()**: esta función nos permite añadir al carrito de localStorge el producto seleccionado.
- **limpiar()**: esta función nos permite borrar la selección de los productos que hayamos seleccionado, como pueden ser el color o la talla del producto.
- **sacarStock()**: esta función nos permite sacar el stock de la variación seleccionada.
- **generarAlerta(mensaje)**: esta función permite generar una alerta visible para el usuario, ayudando a mostrar información relevante.
- **comprobacionArticulo(productos, articulo)**: esta función nos permite controlar que el producto seleccionado no haya sido añadido previamente al carrito para así evitar datos duplicados.
- **cerrarSesion()**: esta función asíncrona nos permite cerrar la sesión.
- **capitalize(cadena)**: esta función nos permite poner una cadena de texto con la primera letra en mayúscula y el resto en minúsculas.
- **borrarAlerta()**: esta función nos permite borrar alertas.

Script-funciones-gestion-categorias

- **activarTabla()**: esta función nos permite activar los listener de la tabla en la vista de gestión de categorías.
- **borrar(id, evento)**: esta función asíncrona nos permite borrar la categoría seleccionada.
- **actualizar(datos)**: esta función asíncrona nos permite actualizar los datos de la categoría seleccionada.
- **crear(datos)**: esta función asíncrona nos permite crear una categoría nueva.

Script-funciones-gestion-colecciones

- **activarTabla()**: esta función nos permite activar los listener de la tabla en la vista de gestión de colecciones.

- **borrar(id, evento)**: esta función asíncrona nos permite borrar la colección seleccionada.
- **actualizar(datos)**: esta función asíncrona nos permite actualizar los datos de la colección seleccionada.
- **crear(datos)**: esta función asíncrona nos permite crear una colección nueva.

Script-funciones-gestion-compras

- **sacarDetalles(id, usuario, estado, precioTotal)**: esta función asíncrona permite sacar los detalles de una compra en específico y generar el cuadro de dialogo que nos permite visualizarlos.
- **actualizarEstado(datos)**: esta función asíncrona permite actualizar el estado de una compra.
- **actualizarEstadoTabla(id, estado)**: esta función refleja el cambio del estado de la compra en la tabla de la vista de gestión de compras.

Script-funciones-gestion-descuentos

- **activarTabla()**: esta función nos permite activar los listener de la tabla en la vista de gestión de descuentos.
- **borrar(id, evento)**: esta función asíncrona nos permite borrar el descuento seleccionado.
- **actualizar(datos)**: esta función asíncrona nos permite actualizar los datos del descuento seleccionado.
- **crear(datos)**: esta función asíncrona nos permite crear un descuento nuevo.

Script-funciones-gestion-productos

- **borrar(id, event)**: esta función asíncrona nos permite borrar el producto seleccionado.

Script-funciones-gestion-reclamaciones

- **cambiarEstado(datos)**: esta función asíncrona nos permite cambiar el estado de una reclamación.

Script-funciones-login

- **enviarFormulario(email, password)**: esta función nos permite enviar los datos de inicio de sesión del usuario al servidor.

Script-funciones-usuario

- **scarDetalles(id, usuario, estado, precioTotal)**: esta función asíncrona nos permite obtener los detalles sobre el usuario para mostrarlos en la página del usuario.
- **enviarReclamacion(datos)**: esta función asíncrona nos permite enviar reclamaciones.
- **modDireccion(datos)**: esta función asíncrona nos permite guardar los cambios en la dirección.

- **modDatos(datos)**: esta función nos permite cambiar los datos del usuario.

Script-modificar-producto

- **cogerDatos()**: esta función nos permite obtener los datos del producto que se está modificando.
- **coloresStock()**: esta función nos permite sacar el stock de las distintas variaciones de color.
- **coloresTallas()**: esta función nos permite sacar los colores que tiene cada talla del producto.
- **actualizar(datos)**: esta función asíncrona nos permite actualizar los datos del producto que se está modificando.
- **activarCruces()**: esta función sirve para activar el listener de las cruces de la página, ya que al momento de añadir datos se añaden cruces que antes no existían.

Script-modificarUsuario

- **aplicarCambios()**: esta función nos permite aplicar los cambios que se han hecho al usuario.
- **enviarCambios(estado)**: esta función asíncrona nos permite enviar el estado del usuario al servidor.
- **sacarEmail()**: esta función nos permite obtener el email del usuario seleccionado.

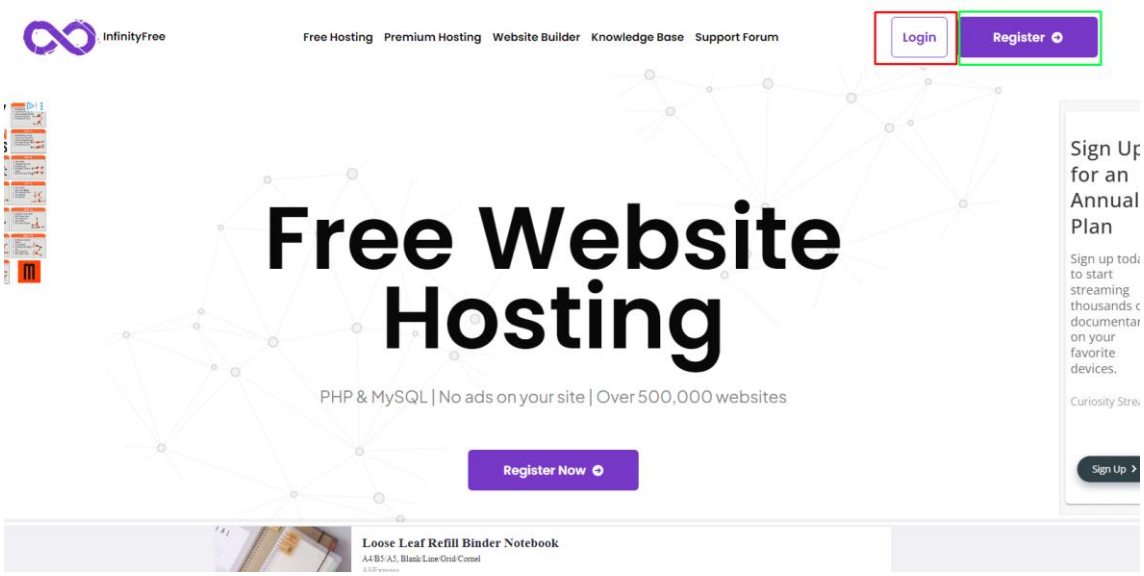
Script-register

- **redirigir()**: esta función nos permite enviar al usuario a la página de inicio una vez se ha registrado.
- **enviarFormulario(email, password)**: esta función nos permite enviar los datos de inicio de sesión del usuario al servidor.

3.2.3 Fase de despliegue

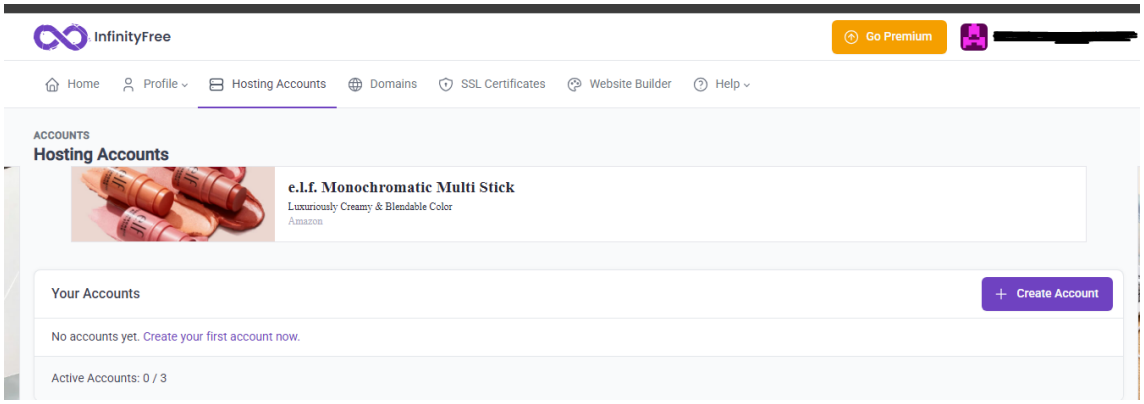
Para el despliegue de nuestra aplicación se ha seleccionado el servidor de hosting infinityfree, por su alta disponibilidad y su reducido precio, siendo este gratuito.

En el inicio del despliegue lo primero que hay que hacer es crear una cuenta dentro de la página de infinityfree.

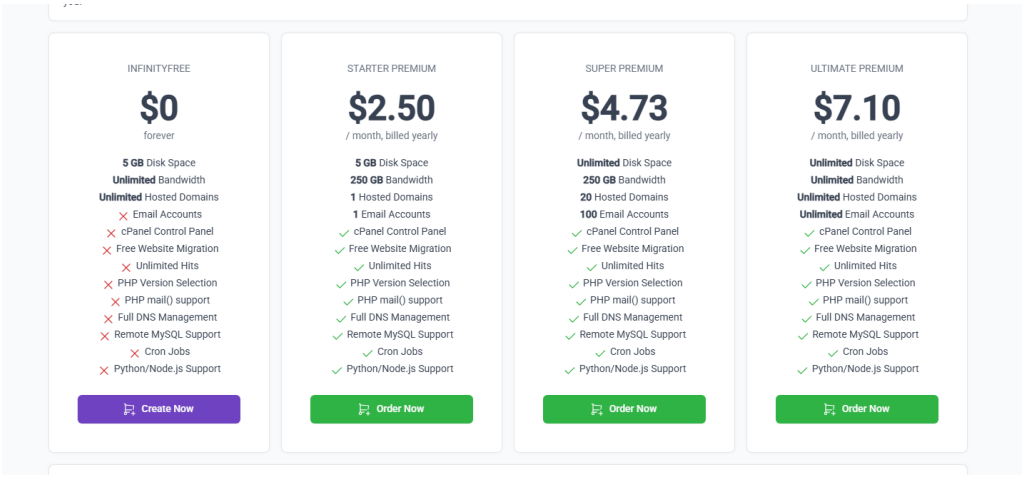


En caso de que tengamos cuenta ya creada dentro de la página con darle a login (cuadrado rojo) e iniciar sesión sería valido, en caso de no tenerla pulsaremos en register (cuadrado verde).

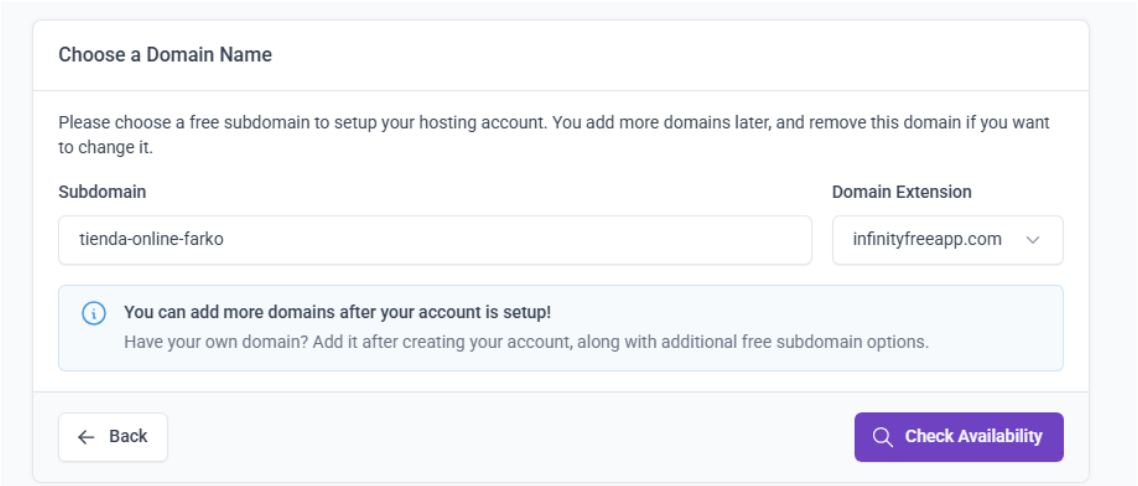
Una vez que ya hemos creado o entrado en nuestra cuenta, aparecerá el panel de control.



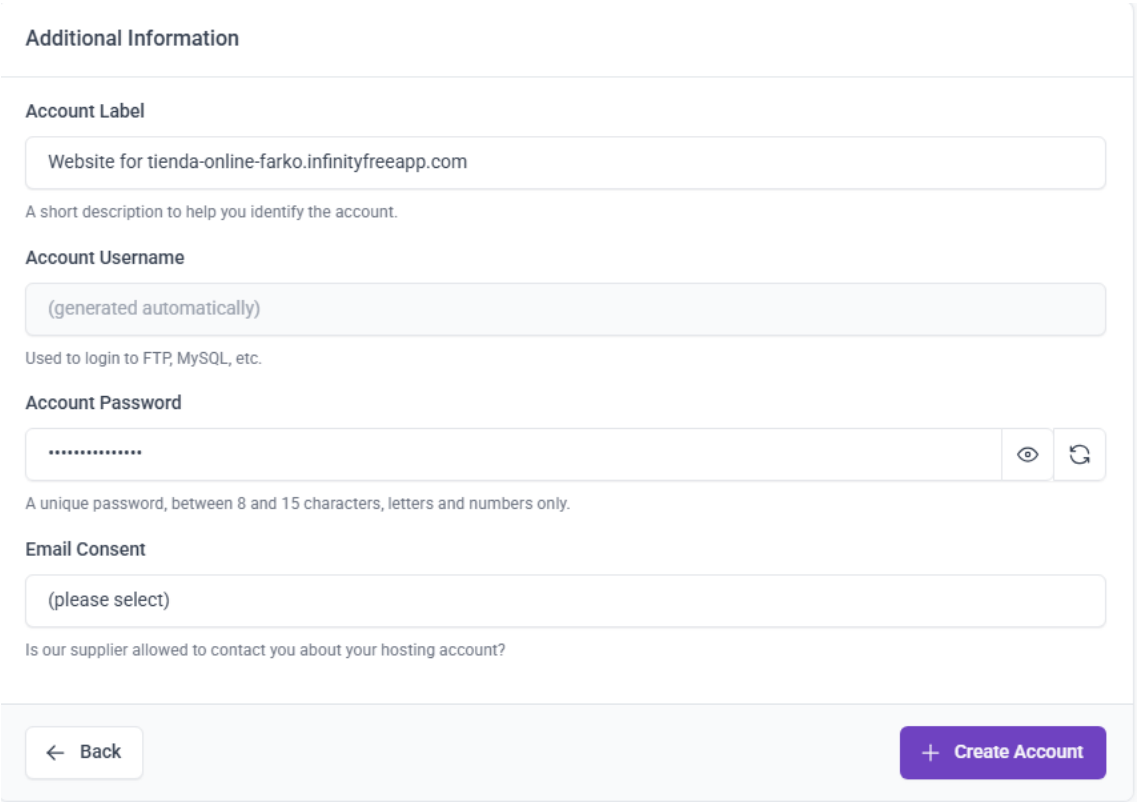
En esta pantalla podemos ver varias opciones, en nuestro caso usaremos el botón morado en el que pone create account, para crear un servidor de hosting personalizado, este botón nos llevará a la selección de servidores en el que nos dejará elegir el plan que queremos obtener.



En nuestro caso se escogerá el plan de 0 dólares, ya que cumple con los requisitos que necesitamos para mostrar nuestra página web. Una vez seleccionado el plan, tendremos que elegir el nombre del dominio además de la extensión que se le dará.



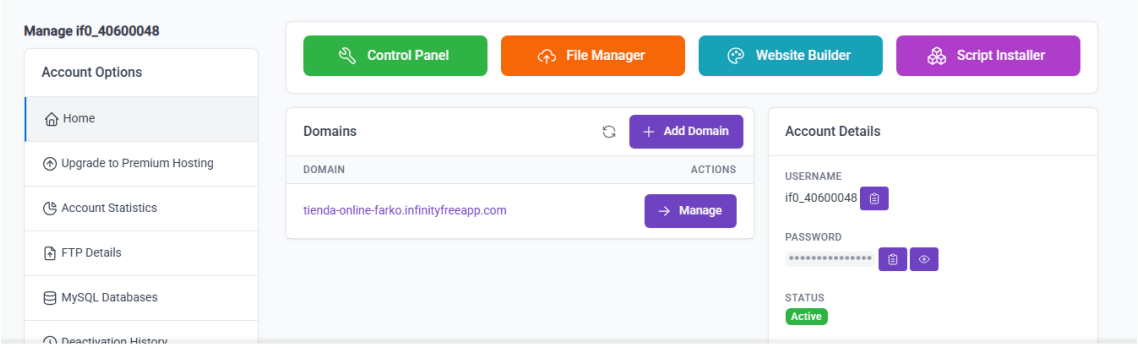
Estos son los datos elegidos para este desarrollo, como subdominio tienda-online-farko y extensión infinityfreeapp.com



Cuando la página haya comprobado que el dominio está libre y que es correcto, pedirá información extra, en la cual especificaremos la contraseña y la etiqueta identificatoria en el panel de control. "**Recomendación de apuntar la contraseña**" en este caso la que ha generado automáticamente es la siguiente: "*2pEeCoId0noCm4A*"

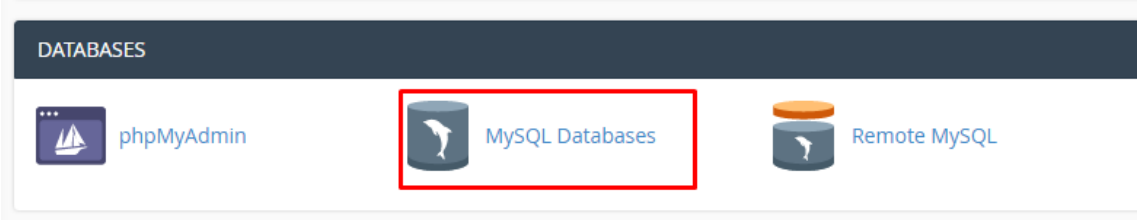
El usuario que nos ha generado: if0_40600048

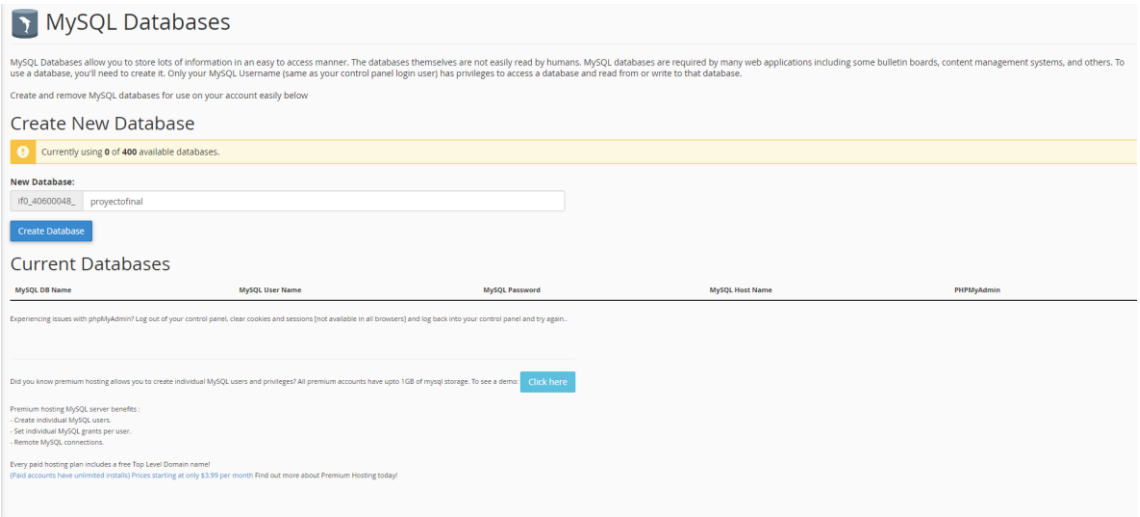
Una vez que todo haya cargado correctamente se nos mostrará la siguiente pantalla:

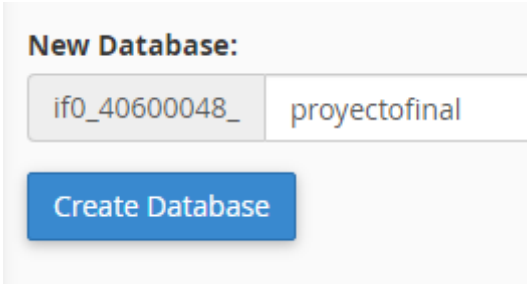


Esta es el dashboard de nuestro sitio web, dentro de este tenemos disponibles varias opciones, entre ellas abrir el panel de control o el gestor de archivos.

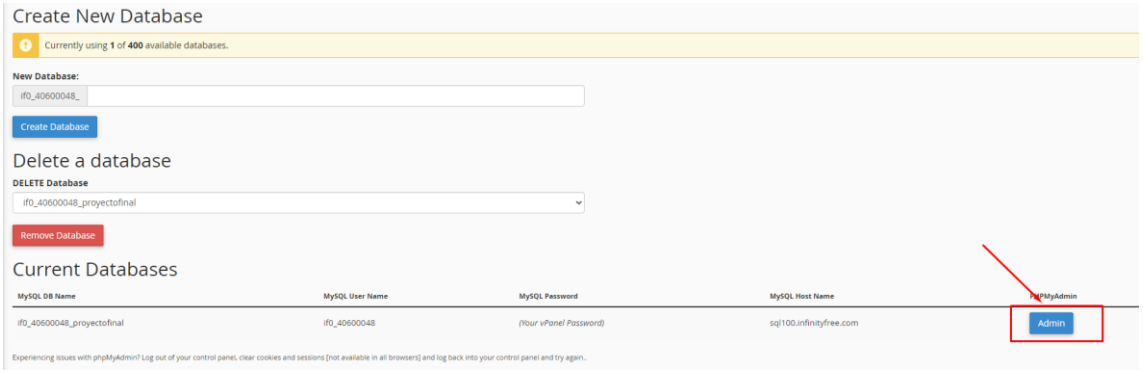
Si abrimos el panel de control, encontraremos una pantalla parecida al gestor de wordpress, en esta tendremos que buscar la opción de MySQL Databases, para crear la base de datos.

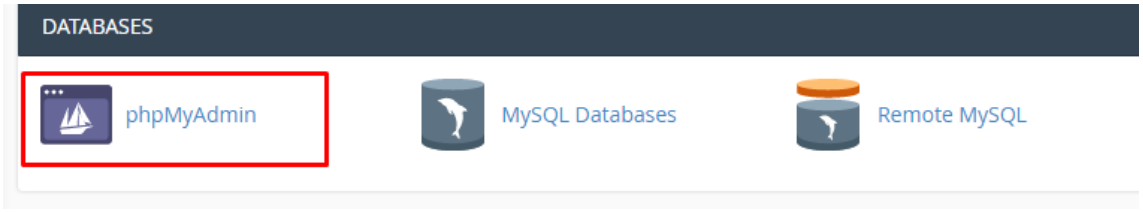




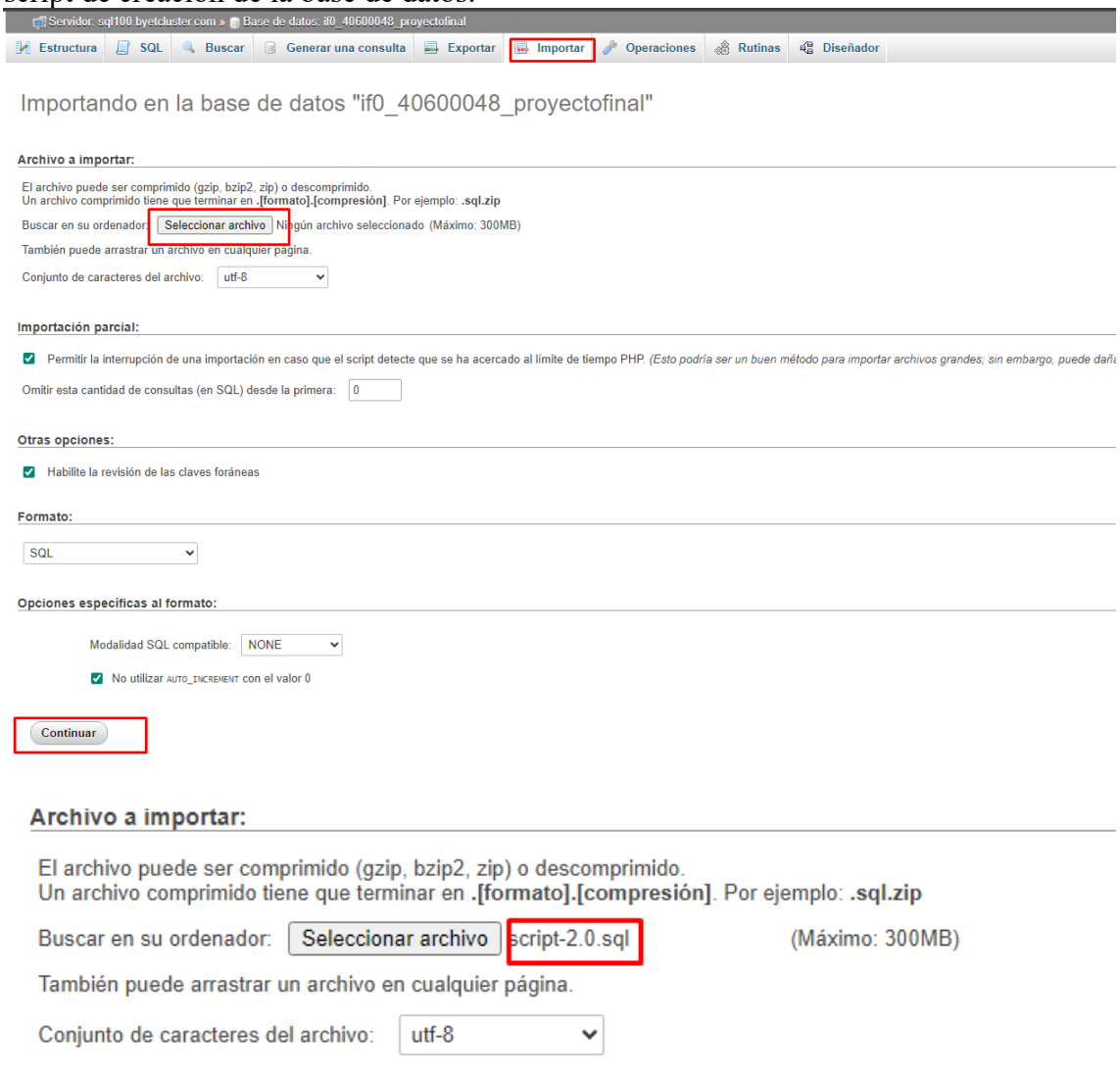


Una vez que hemos creado la base de datos ya podemos ir a phpMyAdmin, ya que aquí subiremos la base de datos de nuestro proyecto. Para ello tenemos dos formas o desde la propia pantalla de creación o volviendo al panel de control:

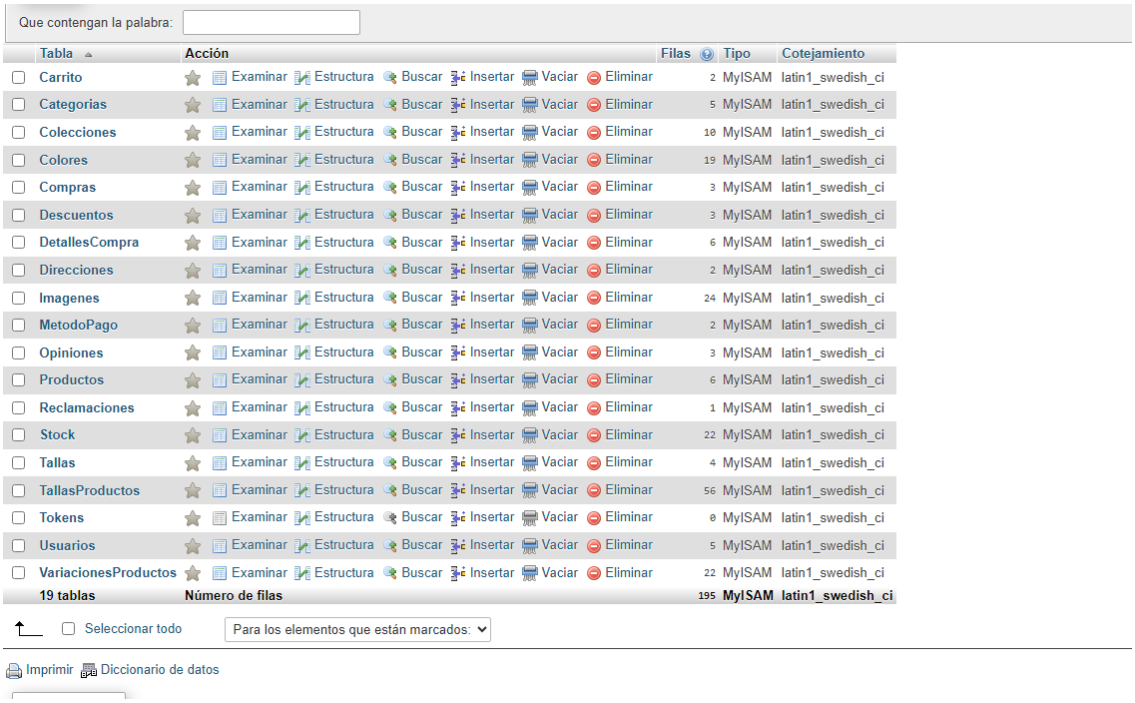




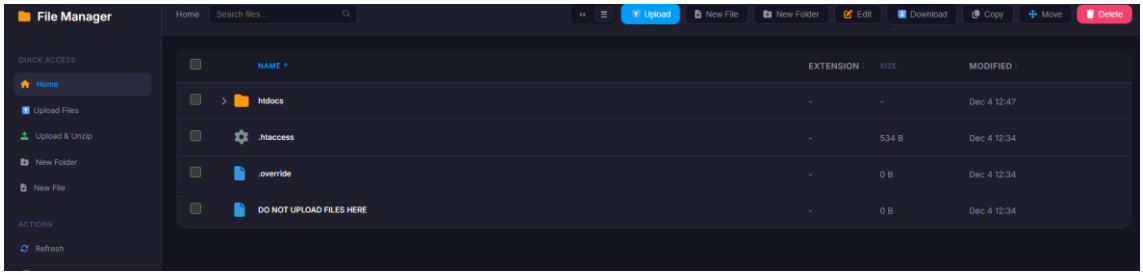
Una vez dentro del administrador, iremos a la pantalla de importar ya que tenemos un script de creación de la base de datos:



Una vez que hayamos importado el fichero, veremos que se han creado todas las tablas con los correspondientes datos:

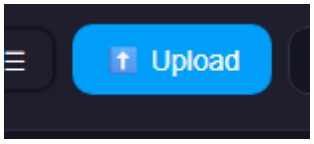


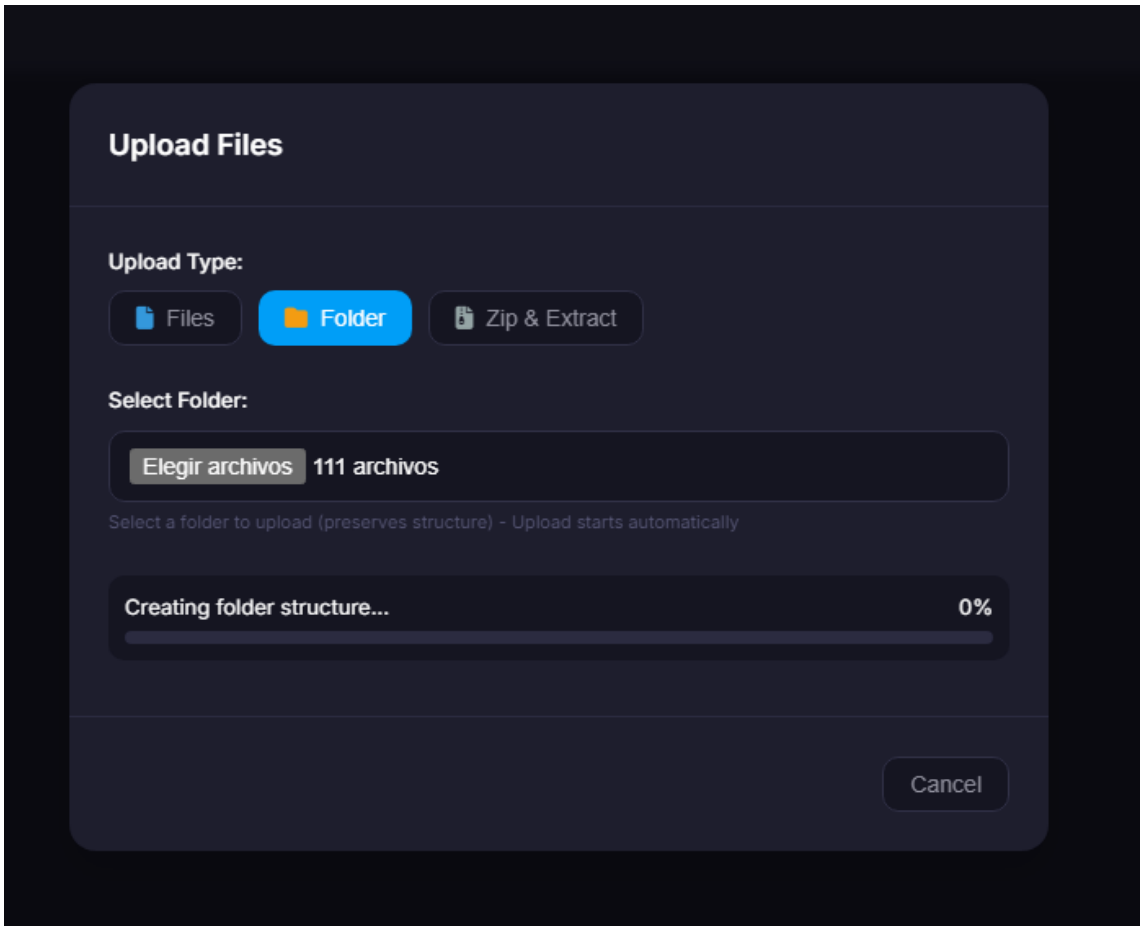
Una vez terminada esta parte se realiza la subida de ficheros al htdocs del domino, pulsando en el botón naranja del dashboard:



Entramos en la carpeta htdocs para subir nuestros ficheros.

Seleccionamos Upload y subimos los ficheros





IMPORTANTE CAMBIAR LOS DATOS DE LA CONEXIÓN DE LA BASE DE DATOS EN LOS FICHEROS PHP

```php
class ModeloConexion{
    public static function conectar(){
        try{
            //Datos de conexión
            $dsn = "mysql:host=sql100.infinityfree.com;dbname=if0_40600048_proyectofinal";
            $usuario = "if0_40600048";
            $password = "2pEeCoId0noCm4A";
            //Iniciamos la conexión con la base de datos
            $conex = new PDO($dsn, $usuario, $password);
            $conex->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            return $conex;
        }catch(PDOException $pdoEx){
            die ("Conexión fallida: " . $pdoEx->getMessage());
        }
    }

}
```

Se puede cambiar directamente desde la propia web.

Una vez que tengamos todo esto hecho, nuestra página web estará disponible.

3.3 SEGUIMIENTO Y CONTROL DE INCIDENCIAS

En la fase de desarrollo de la aplicación se encontraron distintos fallos y complicaciones que entorpecieron el flujo de trabajo como pueden ser, problemas a la hora de la

comunicación del cliente con el servidor, problemas con el cotejamiento de datos y un sinfín de problemas extras.

Para ello se hizo un seguimiento de las versiones estables con la herramienta GIT, subiendo todo a la nube cuando funciona correctamente, así en caso de que trabajando se estropee algo y hayamos realizado muchos cambios, siempre podremos volver a una versión anterior estable. Además de la herramienta GIT el formato tradicional de seguimiento a papel no ha pasado desapercibido ya que este ha jugado un papel crucial a la hora del desarrollo de la aplicación.

4 RECURSOS MATERIALES

4.1 INVENTARIO VALORADO DE MEDIOS

BIEN DE INVERSIÓN	FOTO	PRECIO/UND	Nº UNDS	SUBTOTAL
PcCom Ready		1299€	2	2589€
Office 365		67.20€	2	134.4€
Monitor LG 25MS500-B 24.5" LED IPS FullHD 100Hz		89€	2	178€
PcCom Essential MK20 Combo Teclado + Ratón con Copilot Negro		9.99€	2	18.98€

Estantería TROTTEN		687€	1	687€
MALM Escritorio, blanco, 140x65 cm		199€	2	398€
Silla de oficina Gravity Black		299€	2	598€
IMPORTE TOTAL				4603.38€

4.2 PRESUPUESTO ECÓNOMICO

ENTRADAS	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre	TOTAL
Ahorro personal / Préstamo familia	10.000€												10.000€
Préstamos	7.982,36 €												8.863,38 €
Ventas de bienes/servicios	5.477,18 €	1.925,49 €	2.888,24 €	2.888,24 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	51.612,03 €
Ayudas y subvenciones												1.381,02 €	1.381,02 €
Intereses de c/c e inversiones													
TOTAL ENTRADAS	23,459.54 €	1,925.49 €	2,888.24 €	2,888.24 €	4,813.73 €	4,813.73 €	4.813,73 €	4,813.73 €	4,813.73 €	4.813,73 €	4,813.73 €	6.194,75€	71,052.32 €
SALIDAS	**Enero**	**Febrero**	**Marzo**	**Abril**	**Mayo**	**Junio**	**Julio**	**Agosto**	**Septiembre**	**Octubre**	**Noviembre**	**Diciembre**	**TOTAL**
Devolución de préstamo e intereses	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	2.843,40 €
Alquileres (local. vehículos. etc,)	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	7.200€

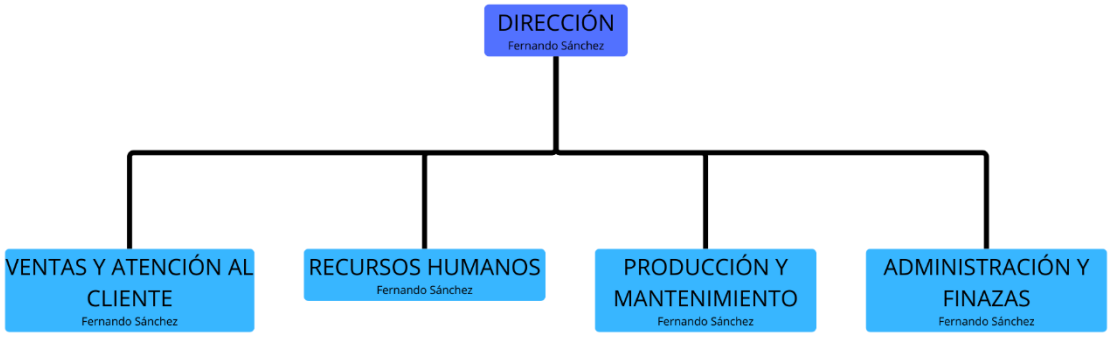
Maquinaria y herramientas (propiedad)													
Vehículos (propiedad)													
Mobiliario (propiedad)	1.683€												1.683€
Equipos Informaticos(Propiedad)	2.920,38 €												2.920.38 €
Reformas y acondicionamientos													
Gastos de constitución	500€												500€
Seguros (local y vehículo en propiedad)	200€												200€
Publicidad	1.800€												1.800€
Compra de mercaderías													
Servicios externos (gestoría)	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	600€
Personal (salarios. seg,social)	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03€	2.884,03 €	2.884,03€	2.884,03€	34.608,36€
Pagos a cuenta del IRPF o del IS							173,29€			519,88€		519,88€	1.213,06 €
Suministros (agua. telefono. luz etc,)	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	960 €
TOTAL SALIDAS	10.954,36 €	3.850,98 €	3.850,98 €	3.850,98 €	3.850,98 €	3.850,98 €	4.024,27 €	3.850,98 €	3.850,98 €	4.370,86 €	3.850,98 €	4.370,86 €	54.528,20 €

5 RECURSOS HUMANOS

5.1 ORGANIZACIÓN

La estructura organizativa de la empresa quedaría de la siguiente forma:

DEPARTAMENTO/ ÁREA FUNCIONAL DE TU IDEA DE NEGOCIO	FUNCION / TAREA
Dirección	Se dedica a la dirección de la empresa
Recursos Humanos	Se dedica a la gestión del personal
Producción/Mantenimiento	Se dedica a la gestión del producto de la empresa
Ventas/atención al cliente	Se dedica a la gestión de la venta del producto de la empresa
Administración y finanzas	Se dedica a la administración y financiación de la empresa



5.2 CONTRATACIÓN

Los dos puestos de trabajos que tendremos dentro de esta empresa son: **“jefe de áreas y servicios” y “profesional de oficios de 1ª ”.**

PROFESIOGRAMA			
PUESTO DE TRABAJO	Jefe de áreas y servicios		
DESCRIPCIÓN DEL PUESTO			
CATEGORÍA PROFESIONAL	Empleados	ACTIVIDAD DE LA CATEGORÍA PROFESIONAL	Personal que por sus conocimientos y/o experiencia realiza tareas administrativas, comerciales, organizativas, **de informática,** de laboratorio y, en general, las específicas de puestos de oficina, que permiten informar de la gestión, de la actividad económico-contable, coordinar labores productivas o realizar tareas auxiliares que comporten atención a las personas.
TAREAS DEL PUESTO DE TRABAJO	Tareas técnicas de codificación de programas de ordenador en el lenguaje apropiado, verificando su correcta ejecución y documentándoles adecuadamente. Actividades que impliquen la responsabilidad de un turno o de una unidad de producción que puedan ser secundadas por una o varias personas trabajadoras del Grupo profesional inferior.		
PERFIL PROFESIONAL			
TITULACIÓN	**Técnico superior en desarrollo de aplicaciones web**		
CONOCIMIENTOS	- Lenguajes de marcas y sistemas de información - Sistemas informáticos - Bases de datos - Programación - Entornos de desarrollo - Desarrollo web en entorno cliente - Desarrollo web en entorno servidor - Despliegue de aplicaciones web - Diseño de interfaces web - Inglés profesional para ciclos formativos de Grado Superior - Digitalización aplicada al sector productivo (GS) - Sostenibilidad aplicada al sistema productivo - Itinerario personal para la empleabilidad - Itinerario personal para la empleabilidad II - Proyecto inter modular de desarrollo de aplicaciones Web		
ACTITUDES PERSONALES	Capacidad de observación, de razonamiento y reflexión. Capacidad de organización, rigurosidad y disciplina.		

	Capacidad numérica. Capacidad para el trabajo en equipo, cooperación. Facilidad en el aprendizaje de idiomas. Capacidad para el autoaprendizaje y uso de las nuevas tecnologías.
EXPERIENCIA	Un año

CONDICIONES LABORALES						
GRUPO PROFESIONAL	CATEGORÍA PROFESIONAL	GRUPO SALARIAL	SALARIO BASE	COMPLEMENTOSALARIAL (Calidad y Cantidad)	CONTRATO	TIPO DE JORNADA
3	Empleado Jefe de área	3-A	1093.51	120.02	Autónomo con alta inicial en el reta	Completa

PROFESIOGRAMA			
PUESTO DE TRABAJO	Profesional de oficios de 1a		
DESCRIPCIÓN DEL PUESTO			
CATEGORÍA PROFESIONAL	Operario	ACTIVIDAD DE LA CATEGORÍA PROFESIONAL	Personal que por sus conocimientos y/o experiencia ejecuta operaciones relacionadas con la producción, bien directamente, actuando en el proceso productivo, o en labores de mantenimiento, transporte u otras operaciones auxiliares, pudiendo realizar, a su vez, funciones de supervisión o coordinación.
TAREAS DEL PUESTO DE TRABAJO	Tareas administrativas desarrolladas con utilización de aplicaciones informáticas. Tareas de mecanografía, con buena presentación de trabajo y ortografía correcta y velocidad adecuada que pueden llevar implícita la redacción de correspondencia según formato e instrucciones específicas, pudiendo utilizar paquetes informáticos como procesadores de textos o similares		
PERFIL PROFESIONAL			
TITULACIÓN	**Técnico superior en desarrollo de aplicaciones web**		
CONOCIMIENTOS	- Lenguajes de marcas y sistemas de información - Sistemas informáticos - Bases de datos - Programación - Entornos de desarrollo		

	- Desarrollo web en entorno cliente - Desarrollo web en entorno servidor - Despliegue de aplicaciones web - Diseño de interfaces web - Inglés profesional para ciclos formativos de Grado Superior - Digitalización aplicada al sector productivo (GS) - Sostenibilidad aplicada al sistema productivo - Itinerario personal para la empleabilidad - Itinerario personal para la empleabilidad II - Proyecto inter modular de desarrollo de aplicaciones Web
ACTITUDES PERSONALES	Capacidad de observación, de razonamiento y reflexión. Capacidad de organización, rigurosidad y disciplina. Capacidad numérica. Capacidad para el trabajo en equipo, cooperación. Facilidad en el aprendizaje de idiomas. Capacidad para el autoaprendizaje y uso de las nuevas tecnologías.
EXPERIENCIA	Un año

CONDICIONES LABORALES						
GRUPO PROFESIONAL	CATEGORÍA PROFESIONAL	GRUPO SALARIAL	SALARIO BASE	COMPLEMENTOSALARIAL (Calidad y Cantidad)	CONTRATO	TIPO DE JORNADA
5	Operario Profesional de oficios de 1a	5-D	979.19	114.52	Temporal Contrato para parados de larga duración	Completa

			A	B	C	D	E	F	
TRABAJADOR DE LA EMPRESA	PUESTO DE TRABAJO	TIPO DE CONTRATO	Nº pax	SALARIO MENSUAL (A x B)	SEGURIDA D SOCIAL MENSUAL (A x C)	SALARIO ANUAL (B x 12)	SEGURIDAD SOCIAL ANUAL (C x 12)	BONIFICACIÓN Y REDUCCIONES A LA SEGURIDAD SOCIAL 12 meses	COSTE TOTAL CON BONIFICACIÓN Autónomo (D + F) Contratados (D + E) - F
Promotor / Autónomo	**Jefe de Taller**	Gerente	1	1093.51€	273.10€	13122.12€	3277.2€	960.00€	14082.12€
Personal Contratado	Profesional de oficio de 1a	Temporal Contrato para parados de larga duración	1	1367.14€	471.39€	16405.68€	5656.68€	1536€	20526.32€
TOTAL			2	2460.65€	745.03€	29527.8€	8933.88€	2496€	34608.44€

5.3 ORGANIZACIÓN DE LA PREVENCIÓN

Todas las empresas están obligadas a organizar la gestión de la prevención de los riegos laborales para garantizar la seguridad y la salud de todos sus trabajadores/as.

Debido a esto la modalidad de prevención que vamos a seguir es la siguiente:

MODALIDAD	REQUISITOS
Asunción personal por el empresario de la actividad preventiva.	- Empresas de menos de seis trabajadores. - La empresa desarrolla actividades no incluidas en el anexo I del Reglamento de los Servicios de Prevención. - El empresario desarrolla su actividad profesional de forma habitual en el centro de trabajo. - El empresario debe tener la capacidad correspondiente a las funciones preventivas que va a desempeñar.

Se ha elegido esta modalidad por ser la que mejor se adapta a las necesidades de la empresa, ya que no implica un coste adicional y tiene una constitución rápida y sencilla.

6 VIABILIDAD TÉCNICA

6.1 ESTUDIO DE VIABILIDAD TÉCNICA

Para el desarrollo de este proyecto se ha seleccionado un conjunto de técnicas y herramientas accesibles y actualizadas. El total de las de las herramientas de trabajo utilizadas para el desarrollo del mismo son gratuitas, haciendo que cualquier usuario pueda utilizar las mismas herramientas sin afectar al rendimiento o a la calidad del mismo.

DESARROLLO BACKEND

Para el desarrollo del entorno Backend o entorno servidor, se ha utilizado el lenguaje PHP 8.2.12 en el entorno de Visual Studio Code.

DESARROLLO FRONTEND

Para el desarrollo del entorno Forntend o entorno cliente se ha utilizado HTML 5, CSS 3 y JavaScript.

En ninguno de los casos se han implementado librerías externas, todo se ha hecho manualmente o usando funciones que tienen los lenguajes nativamente.

CONTROL DE VERSIONES

Para el control de versiones de este proyecto se ha implementado Git utilizando Github como centro de repositorios, para asegurar el acceso al mismo siempre que tengamos conexión a internet, permitiendo tener un control y flujo de trabajo estables en cualquier entorno.

7 VIABILIDAD ECONÓMICO-FINANCIERA

7.1 INVERSIÓN Y GASTOS INICIALES

BIENES DE INVERSIÓN	COSTE			FUENTE DE FINANCIACIÓN
	Alquiler		Compra	
	€ / mes	€ / año		
Edificio / Local	600€	7.200€		Leasing
Reformas / Acondicionamiento del local				
Maquinaria / Equipos				
Herramientas / Utillaje				
Mobiliario			1.683€	Préstamo
Equipos informáticos			2.920,38€	Cuenta Ahorro-Empresa
Vehículos				
TOTAL. BIENES DE INVERSIÓN	600€	7.200€	4.603,38€	11.803,38 €

GASTOS INICIALES	COSTE	
	€ / mes	€ / año
Constitución de la empresa		500€
Capital social mínimo exigido legalmente		3.500€
Reservas (dinero primeros meses)		
Seguros. (local. vehículos. etc,)	16,67€	200€
Publicidad	150€	1.800€
Servicios externos. (gestoría. etc,)	50€	600€
Suministros. (agua. luz. teléfono. etc,)	80€	960€
TOTAL GASTOS INICIALES		7.560€

7.2 FINANCIACIÓN

TOTAL BIENES DE INVERSIÓN	11.803,38 €
TOTAL GASTOS INICIALES	7.560 €
INVERSIÓN NECESARIA	19.363,38 €

FINANCIACIÓN DE INVERSIONES Y GASTOS INICIALES	
FINANCIACIÓN PROPIA	10.000€
Ahorros personales,	5.000€

Préstamos familiares,	5.000€
FINANCIACIÓN AJENA	1.381,02€
AYUDAS Y SUBVENCIONES	1.381,02€
Ayuntamiento de Cuenca,	€
Diputación de Cuenca,	€
Junta de comunidades de Castilla-La Mancha,	€
Creación,	€
Inversión,	1.381,02€
Financiación,	€
Gobierno de España,	€
PRÉSTAMOS	7.982,36€
Préstamo bancario	7.982,36€
Líneas ICO	€
FINANCIACIÓN NECESARIA	19.363,38€

7.3 AMORTIZACIÓN DEL PRÉSTAMO

CONDICIONES DEL PRÉSTAMO	
Entidad financiera	Santander
Capital solicitado	7.982,36€
Tipo de interés (i) (Euribor + Diferencial)	1,25% + 2,143% 3,393%
Plazo de devolución	3 años

	A	B	C	D	E
Período	Cuota anual (1)	Interés anual (D x i)	Capital devuelto (A - B)	Capital pendiente (D - C)	Capital Amortizado (C1.2.3,, + E1.2.3.,,)
0				7.982,36€	

1	2.843,36 €	270,84€	2.572,51€	5.409,85€	2.572,51€
2	2.843,36 €	183,56€	2.659,80€	2.750,05€	5.232,31€
3	2.843,36 €	93,31€	2.750,05€	0	7.982,36€

7.4 AMORTIZACIÓN DE LOS BIENES DE INVERSIÓN

BIENES DE INVERSIÓN (Adquiridos en propiedad. los alquilados NO)	Valor de adquisición (A)	Coeficiente lineal máximo (B)	Cuota de amortización anual (A x B)
Edificio / Local			
Reformas / Acondicionamiento del local			
Maquinaria / Equipos			
Herramientas / Utillaje			
Mobiliario	1.683€	10%	168,3€
Equipos informáticos	2920,38€	26%	759,29€
Vehículos			
TOTAL	4.603,38€		927,59€

7.5 VIABILIDAD ECONÓMICA-FINANCIERA

ENTRADAS	Enero	Febrero	Marzo	Abril	Mayo	Junio	Julio	Agosto	Septiembre	Octubre	Noviembre	Diciembre	TOTAL
Ahorro personal / Préstamo familia	10.000€												10.000€
Préstamos	7.982,36 €												8.863,38 €
Ventas de bienes/servicios	5.477,18 €	1.925,49 €	2.888,24 €	2.888,24 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	4.813,73 €	51.612,03 €
Ayudas y subvenciones												1.381,02 €	1.381,02 €
Intereses de c/c e inversiones													
TOTAL ENTRADAS	23,459.54 €	1,925.49 €	2,888.24 €	2,888.24 €	4,813.73 €	4,813.73 €	4.813,73 €	4,813.73 €	4,813.73 €	4.813,73 €	4,813.73 €	6.194,75€	71,052.32 €
SALIDAS	**Enero**	**Febrero**	**Marzo**	**Abril**	**Mayo**	**Junio**	**Julio**	**Agosto**	**Septiembre**	**Octubre**	**Noviembre**	**Diciembre**	**TOTAL**
Devolución de préstamo e intereses	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	236,95 €	2.843,40 €
Alquileres (local. vehículos. etc,)	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	600€	7.200€

Maquinaria y herramientas (propiedad)													
Vehículos (propiedad)													
Mobiliario (propiedad)	1.683€												1.683€
Equipos Informaticos(Propiedad)	2.920,38 €												2.920.38 €
Reformas y acondicionamientos													
Gastos de constitución	500€												500€
Seguros (local y vehículo en propiedad)	200€												200€
Publicidad	1.800€												1.800€
Compra de mercaderías													
Servicios externos (gestoría)	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	50€	600€
Personal (salarios. seg,social)	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03 €	2.884,03€	2.884,03 €	2.884,03€	2.884,03€	34.608,36€
Pagos a cuenta del IRPF o del IS							173,29€			519,88€		519,88€	1.213,06 €
Suministros (agua. telefono. luz etc,)	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	80 €	960 €
TOTAL SALIDAS	10.954,36 €	3.850,98 €	3.850,98 €	3.850,98 €	3.850,98 €	3.850,98 €	4.024,27 €	3.850,98 €	3.850,98 €	4.370,86 €	3.850,98 €	4.370,86 €	54.528,20 €

RESULTADO (Entradas - Salidas)	12.505,18 €	-1.925,49€	-962,75 €	-962,75 €	962,75 €	962,75 €	789,45 €	962,75 €	962,75 €	442,86 €	962,75 €	1.823,88 €
SALDO EN CAJA	12.505,18 €	10.579,69 €	9.616,95 €	8.654,20 €	9.616,95 €	10.579,69 €	11.369,14 €	12.331,89 €	13.294,63 €	13.737,49 €	14.700,24 €	16.524,12 €

El plan de tesorería ofrece una visión dinámica de los cobros y pagos previstos, permitiendo anticipar la situación de liquidez de la empresa y sus necesidades financieras.

En el caso de mi empresa, percibimos liquidez ya que al revisar las cuentas a final de año vemos que el saldo no es negativo y aunque algunos meses presentemos saldos negativos, contamos con la liquidez necesaria para cubrir los gastos y asegurar los pagos requeridos a la empresa.

7.6 CUENTA DE RESULTADOS

(A) INGRESOS DE EXPLOTACIÓN	51.612,03 €	**(B) GASTOS DE EXPLOTACIÓN**	48.057,09€
Ventas	51.612,03 €	Amortización de inversiones	927,59€
		Alquileres *(local, vehículo, mobiliario…)*	7.200€
		Gastos de constitución	500€
		Seguros, (local, vehículos adquiridos…)	200€
		Publicidad	1.800€
		Compra de mercaderías/materia prima	
		Servicios externos *(gestoría ,limpieza...)*	600€
		Personal, (Salarios, Seg. Social)	34.608,44€
		Pagos a cuenta del IRPF o del IS	1.213,06€
		Suministros *(agua, teléfono, gas, etc.)*	960€
(C) INGRESOS FINANCIEROS		**(D) GASTOS FINANCIEROS**	270,84€
Intereses de c/c e inversiones		Intereses de préstamos	270,84€

Resultado de explotación	**(E) = (A – B)**	3.554,94€
Resultado financiero	**(F) = (C – D)**	-270,84€
RESULTADO ANTES DE IMPUESTOS (G) (E + F)		3.284,1€
Impuesto sobre Sociedades	**(H)** (G x 15%)	492,62€
Pagos a cuenta del IRPF o del I.S	**(I)**	1.213,06€
Liquidación del I.S	**(J)** = (H - I)	-720,44€
RESULTADO FINAL	**(K)** = (G - j)	4.004,54€

Al comprobar el ejercicio económico de la empresa y gracias a la Cuenta de Resultados, podemos afirmar que la empresa cuenta con un beneficio neto de 4.004,54€ una vez todos los impuestos han quedado liquidados, reflejando así una situación favorable para la empresa.

7.7 BALANCE PATRIMONIAL

ACTIVO *Muestra en qué ha invertido la empresa el dinero del pasivo.* *(Bienes y derechos en propiedad)*		**PASIVO** *Muestra de dónde obtiene el dinero la empresa para financiar el activo.* *(Deudas a devolver)*	
		PATRIMONIO NETO	8.885,56€
		Capital social mínimo exigido	3500€
		Cuenta de resultados *(+) si ganancias, (-) si pérdidas*	4.004,54 €
		Ayudas y subvenciones	1.381,02€
ACTIVO CORRIENTE (Derechos) (permanecen en la empresa menos de 1 año)	16.524,12 €	**PASIVO CORRIENTE** (Deudas a devolver en menos de 1 año)	5.904,5 €
Existencias.		Créditos pendientes de devolución.	
Facturas de clientes pendientes de cobro.	-	Deudas pendientes a proveedores.	5.904,5 €
Efectivo en Bancos, en caja.	16.524,12 €	Deudas pendientes a acreedores (Seg.Soc, Hacienda)	
ACTIVO NO CORRIENTE (Bienes) (permanecen en la empresa más de 1 año)	3.675,79€	**PASIVO NO CORRIENTE** (Deudas a devolver en más de 1 año)	5.409,85€
Bienes de inversión *(local,mobiliario,vehículo…)*	4.603,38€	Préstamo pendiente de devolución.	5.409,85€
Amortización de inversiones *(con signo "–")*	-927,59€		
TOTAL ACTIVO	20.199,91 €	**TOTAL PASIVO**	20.199,91€

Solvencia: Mi empresa si es solvente ya que mi activo total es mayor que la suma del pasivo corriente + el pasivo no corriente 17.104,05€ > 11.314,35€

Quiebra: Mi empresa no está en quiebra ya que el activo total no es menor que el pasivo corriente + el pasivo no corriente, es decir que tengo activos.

Liquidez: Mi empresa cuenta con liquidez ya que mi activo total es superior al pasivo corriente, haciendo que pueda pagar mis deudas gracias a la liquidez.

No liquidez: Mi empresa no está en situación de no liquidez ya que tengo más activo total que pasivo corriente es decir deudas.

Relación entre fondos propios y deudas: La relación entre los fondos se encuentra estable ya que los propios superan la cantidad de las deudas.

Proporción entre deuda a corto plazo y deudas a largo plazo: El pasivo a largo plazo de la empresa seria de 5.409,85€ y respecto a las deudas a corto plazo tenemos pendiente el pago a proveedores cuya cuantía sería 5.904,5 €.

8 CONCLUSIÓN

Para concluir el tema que nos concierne actualmente, se describirá el punto de vista personal que se ha obtenido a lo largo del desarrollo de la aplicación web de la Tienda de FARKO empezando a abordar el tema, empezaré hablando de la viabilidad, ya que es un proyecto enteramente desarrollado con tecnologías vanillas, es decir, sin frameworks o librerías externas, es un proyecto interesante y que puede ser escalado en caso de necesitar funcionalidades nuevas, haciendo que el proyecto desarrollado sea una opción viable en el presente y en el futuro.

Sobre la viabilidad a nivel económico se debe comentar que todas las herramientas que se han utilizado a lo largo del desarrollo son gratuitas, por lo que no habría problemas en replicarlo. Gracias a su bajo coste en el desarrollo y teniendo capacidad de llegar a ser una página en la cual se lleguen a realizar transacciones reales hace que el proyecto tenga un gran atractivo para futuros clientes.

Una vez que abordados los principales temas y volviendo a lo mencionado al principio del punto, la tienda online de farko, personalmente ha sido un reto, desafiando constantemente al ingenio del desarrollador y siendo una fuente de nuevos desafíos, pero no todo ha sido fácil y agradable habiendo días bastantes difíciles y de mucho trabajo, cosa que será así durante el resto de la carrera profesional que se espera desarrollar al terminar este curso.

Antes de terminar he de decir, que tengo muchas ganas de prosperar y encontrar nuevos retos profesionales, que espero, que superen el desarrollo de esta aplicación.

Y ya, por último, no puedo despedir el documento sin agradecer a todas las personas que han formado parte de este proceso, ya no tanto a nivel técnico, si no a nivel educativo, porque si no llega a ser por todas las personas que han puesto un poquito de ellos en mi con su conocimiento este proyecto no podría ser posible.

Bibliografía

1. Documento TENDENCIAS DEL MERCADO EN ESPAÑA
2. Reglamento General de Protección de datos
3. Ley de Servicios de la Sociedad de la Información

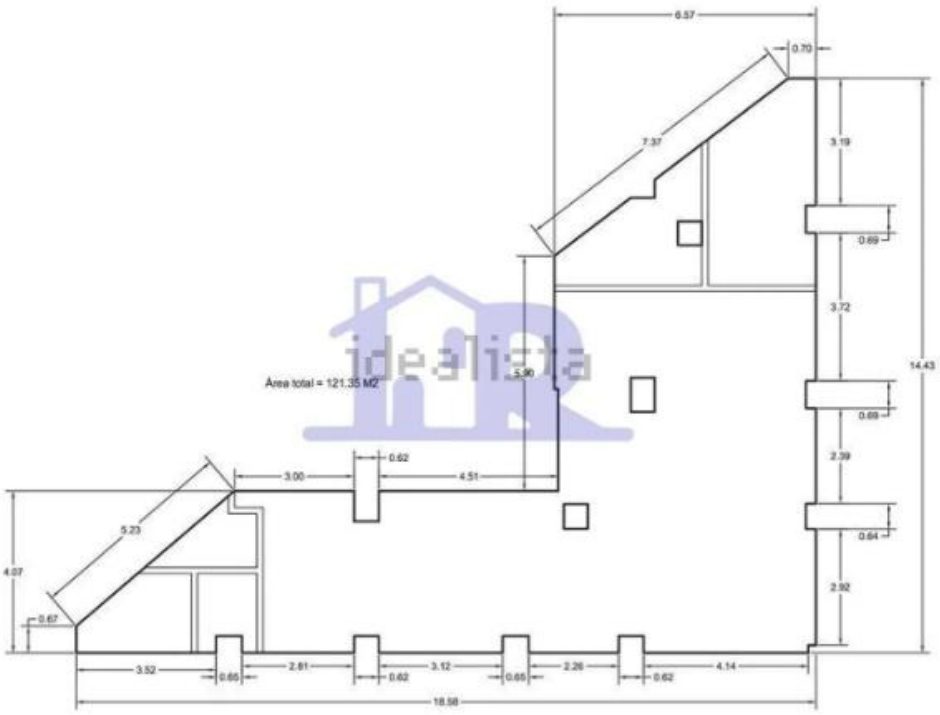
https://developer.mozilla.org/es/docs/Web/CSS/CSS_cascading_variables/Using_CSS_custom_properties

https://developer.mozilla.org/en-US/docs/Web/CSS/justify-content

https://owius.com/tendencias-2025-en-desarrollo-web-en-espana-pwas-ia-y-sostenibilidad/

Anexos

1. TECNOLOGÍAS Y HERRAMIENTAS DE DESARROLLO
    - PHP
    - JavaScript
    - MySQL
    - HTML
    - CSS
2. Local



Metros cuadrados: 140 m^2 (según anuncio) 121,35 m^2 (según plano)