AKIKAZE

Daniel castillejo Álvarez

Información sobre la página web:

Esta página web hecha para una tienda online de productos anime todo lo que hay relacionado a ello ha sido realizado con estas herramientas:

- *HTML*
- *CSS*
- *BOOSTRAP*
- *JAVASCRIPT*
- *MYSQL*
- *AJAX*
- *PHP*
- *SYMPHONY*

Descripción sobre la página:

***Akikaze**, es una tienda en línea temática que combina un diseño atractivo inspirado en el anime con una funcionalidad sencilla y eficiente. Ofrece a los fans del anime un espacio para explorar productos exclusivos, interactuar con la historia de la tienda, y realizar compras. A continuación, se detalla lo que contiene:*

1. Página Principal: Conoce Nuestra Historia

Introducción a la tienda: La página principal presenta una sección dedicada a la historia de **Akikaze**, explicando su origen, su misión, y el significado detrás del nombre "Akikaze" (viento de otoño).

Diseño inmersivo: Un fondo visualmente atractivo con imágenes temáticas de anime y cultura japonesa complementa el texto.
Mensaje comunitario: Refuerza la conexión emocional con los fans, destacando cómo la tienda busca ser más que un comercio, convirtiéndose en un espacio de comunidad.

Llamada a la acción: Un botón destacado invita a los usuarios a explorar la tienda y descubrir los productos disponibles.

2. Página de Productos

- **Catálogo interactivo:** Muestra una lista de productos relacionados con el anime, como figuras de colección, camisetas, pósters, accesorios, entre otros.
- **Detalles de producto:**
    - Cada producto tiene una página individual donde se puede ver:
        - Su descripción detallada.
        - Imágenes de alta calidad.
        - El precio.
        - La cantidad disponible en stock.
    - Desde esta página, los usuarios pueden añadir el producto al carrito de compras.
- **Búsqueda y exploración:** Los usuarios pueden navegar fácilmente por la tienda para encontrar lo que necesitan.

3. Perfil del Usuario

- **Zona personalizada:** Cada usuario registrado tiene acceso a su perfil, donde pueden gestionar su información personal:
    - Ver o editar sus datos (nombre, correo, etc.).
    - Cambiar su contraseña.
- **Gestión accesible:** Un diseño limpio y funcional facilita la navegación por su cuenta.

4. Carrito de Compras

- **Gestión de pedidos:** El carrito permite a los usuarios:
    - Revisar los productos seleccionados antes de proceder a la compra.
    - Ver el desglose del pedido (cantidades, precios unitarios, precio total).
    - Modificar la cantidad de productos o eliminarlos.
- **Proceso de compra:** Preparación para el checkout, simplificando la experiencia de compra.

5. Menú del Administrador

- **Acceso restringido:** Solo los administradores pueden acceder a esta sección especial.
- **Gestión de la tienda:**
    - **Productos:** Agregar, editar o eliminar productos del catálogo.
    - **Usuarios:** Supervisar o gestionar cuentas de usuario.
    - **Stock:** Controlar y actualizar las cantidades disponibles de cada producto.

6. Diseño Temático y Experiencia de Usuario

- **Estilo inspirado en el anime:** Con imágenes, colores y detalles visuales que evocan la estética del anime y la cultura japonesa.
- **Interfaz amigable:** Cada funcionalidad está diseñada para que sea intuitiva, tanto para fans del anime como para administradores.
- **Responsividad:** El diseño se adapta a diferentes dispositivos, desde computadoras hasta smartphones.

7. Funcionalidad en Detalle

- **Autenticación y Autorización:** Un sistema robusto que asegura el acceso de los usuarios registrados y diferencia entre roles (usuario y administrador).
- **Stock de productos:** Cada producto incluye un indicador de las unidades disponibles, asegurando que el inventario esté controlado.
- **Agregar al carrito:** Desde los detalles del producto, los usuarios pueden elegir cuántas unidades desean y añadirlas al carrito de compras.

Objetivo Principal

El propósito de **Akikaze** es ofrecer a los fans del anime una tienda en línea donde puedan adquirir productos de calidad relacionados con su pasión, mientras disfrutan de una experiencia personalizada y única. Al mismo tiempo, la tienda busca construir una comunidad de otakus que comparten el amor por el anime, el manga y la cultura japonesa.

MANUAL DE USUARIO

Navegación en la Página Web

Menú de Navegación

En la parte superior de la página, encontrarás la **barra de navegación** que te permite acceder fácilmente a las principales secciones del sitio:

- **Inicio:** Te lleva a la página principal de la tienda.
- **Productos:** Aquí podrás ver la lista de productos disponibles para su compra.
- **Iniciar Sesión:** Si ya tienes cuenta, accede con tu email y contraseña. Si aún no tienes cuenta, puedes registrarte desde aquí.
- **Registrarse:** Si no tienes una cuenta, haz clic en este enlace para crear una cuenta nueva.
- **Mi Perfil:** Si estás logueado, puedes acceder a tu perfil para ver o modificar tus datos personales.
- **Cerrar Sesión:** Cuando termines tu sesión, puedes cerrar sesión desde este enlace.
- **Carrito:** Accede a tu carrito para revisar los productos que has añadido antes de proceder al pago.

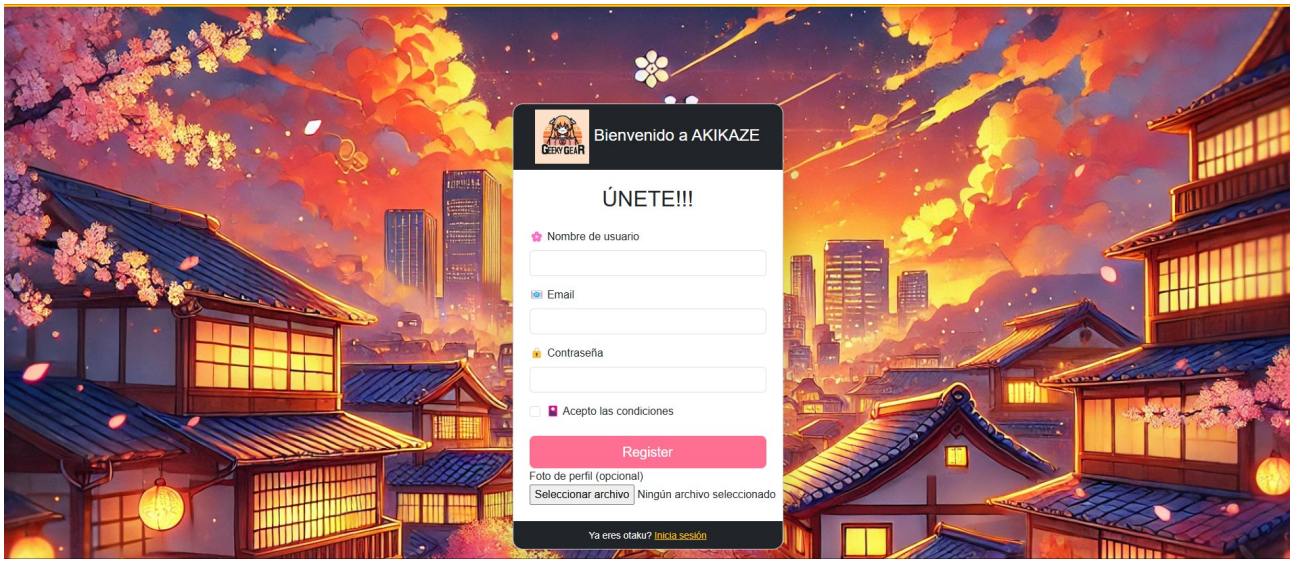
Si estas logueado:

Registro de Usuario

Si eres nuevo en AKIKAZE, puedes **registrarte** fácilmente:

1. Haz clic en el enlace **Registrarse** desde la barra de navegación.
2. Rellena los campos del formulario de registro:
    - **Nombre de usuario:** Escoge un nombre único para tu cuenta.
    - **Email:** Introduce un correo electrónico válido.
    - **Contraseña:** Establece una contraseña segura para tu cuenta.
    - **Acepta los términos y condiciones** marcando la casilla correspondiente.
3. Haz clic en **Registrar** para completar el proceso.

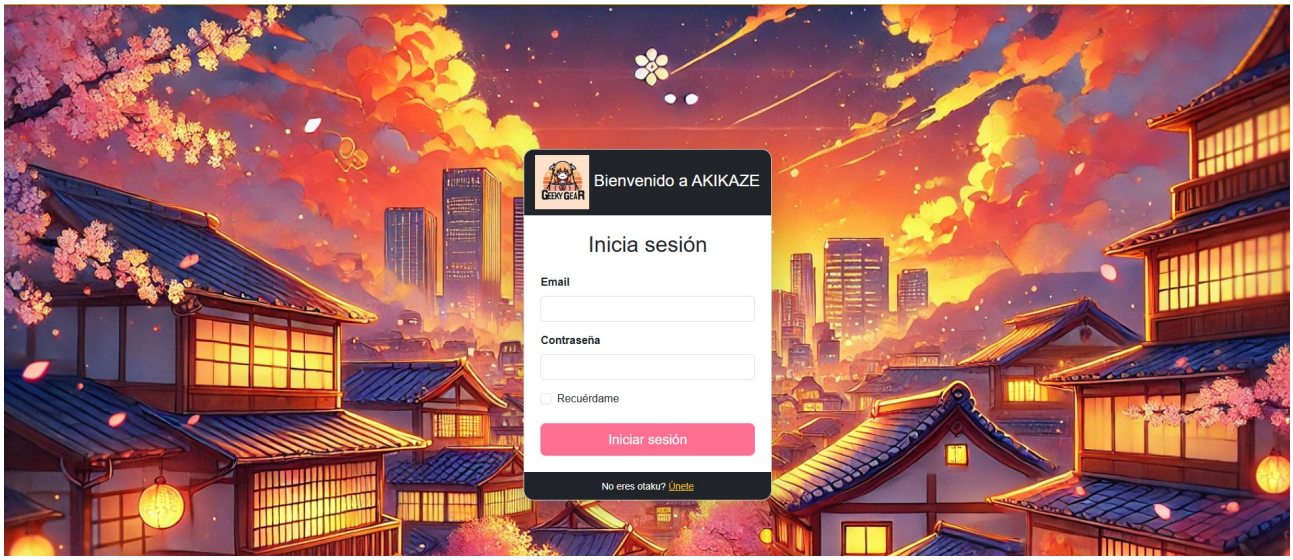
> **Nota:** Recibirás un correo de verificación para confirmar tu dirección de email si es necesario.



Inicio de Sesión

Si ya tienes una cuenta, puedes iniciar sesión de la siguiente manera:

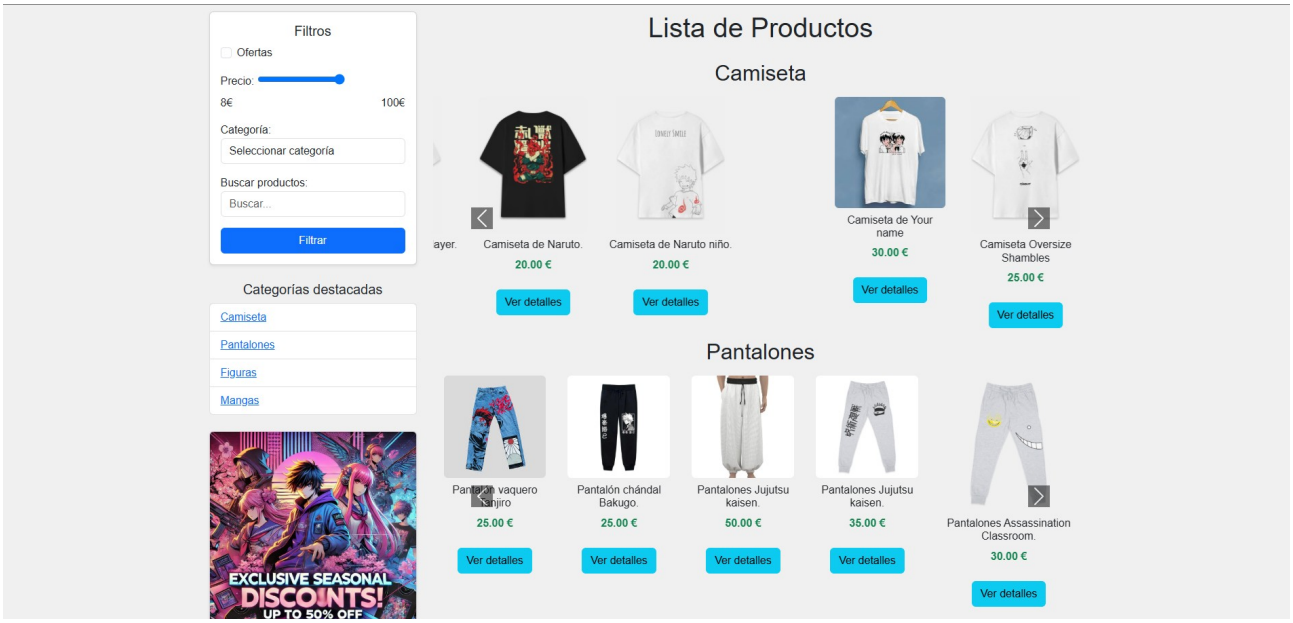
1. Haz clic en **Iniciar Sesión** desde la barra de navegación.
2. Introduce tu **email** y **contraseña** en los campos correspondientes.
3. Haz clic en el botón **Iniciar sesión**.
4. Si has activado la opción **"Recuérdame"**, podrás permanecer logueado en la web durante más tiempo sin necesidad de ingresar tus credenciales nuevamente.



Explorando los Productos

En la sección de **Productos**, podrás ver una lista de artículos disponibles para su compra. Esta página está organizada para que puedas ver productos categorizados, con imágenes y descripciones detalladas. Para agregar un artículo a tu carrito:

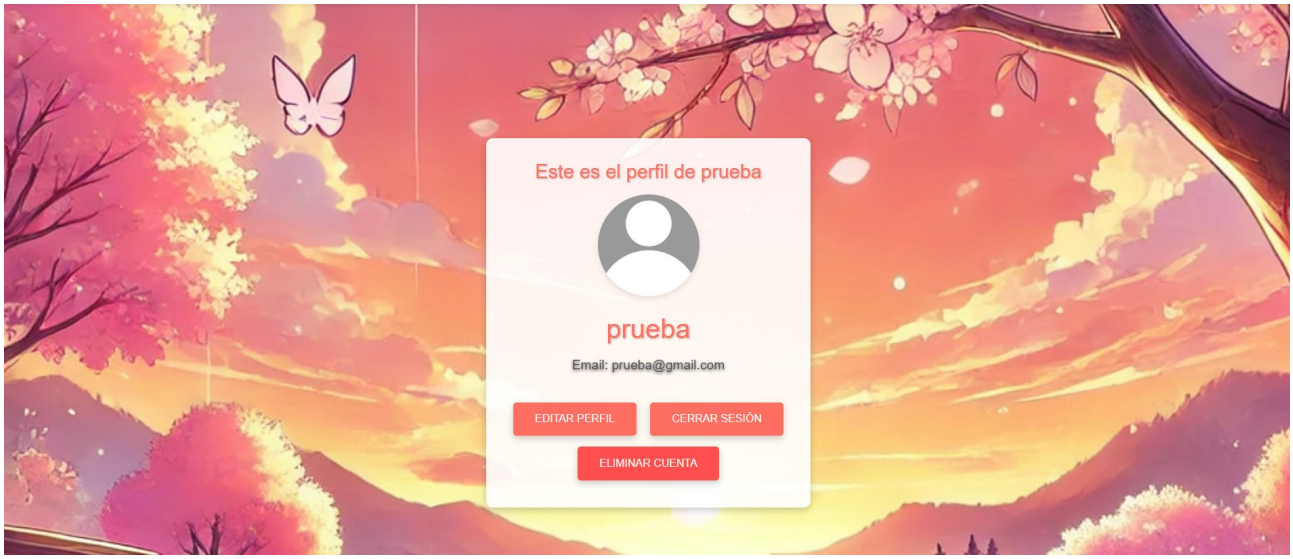
1. Navega por la tienda y selecciona el producto que te interesa.
2. Haz clic en el producto para obtener más detalles.
3. Haz clic en **Añadir al carrito**.



Mi Perfil

Si deseas modificar tu información personal, accede a tu **perfil** haciendo clic en el enlace correspondiente dentro de la barra de navegación. En tu perfil podrás:

- Ver tu información de contacto.
- Cambiar tu contraseña si lo deseas.
- Administrar las configuraciones relacionadas con tu cuenta.



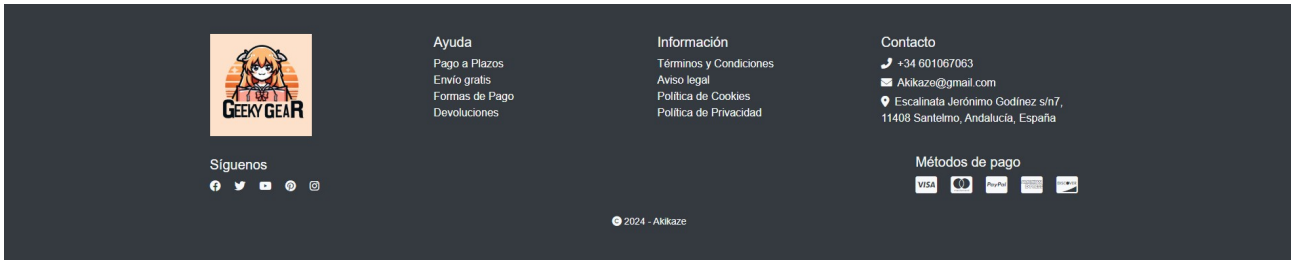
Cerrar Sesión

Para salir de tu cuenta, simplemente haz clic en **Cerrar Sesión** en la barra de navegación. Esto te desconectará de la web y te llevará de vuelta a la página de inicio.

Footer (Pie de Página)

En el **footer** de la página, encontrarás información adicional importante como:

- **Ayuda:** Acceso a información sobre pago a plazos, formas de pago y más.
- **Información:** Términos y condiciones, aviso legal, y políticas de privacidad.
- **Contacto:** Número de teléfono, correo electrónico y dirección física de la tienda.
- **Redes Sociales y Métodos de Pago:** Síguenos en Facebook, Instagram y más. Además, puedes ver los métodos de pago aceptados, como Visa, Mastercard, PayPal, etc.



Soporte Técnico

Si encuentras algún problema o tienes dudas, puedes ponerte en contacto con nuestro equipo de soporte a través del correo electrónico: **Akikaze@gmail.com** o llamando al **+34 601067063**. Además, en la página de contacto encontrarás más detalles sobre nuestra ubicación.

Redes Sociales

Síguenos en nuestras redes sociales para estar al tanto de las últimas actualizaciones, ofertas y novedades:

- **Facebook**
- **Twitter**
- **YouTube**
- **Pinterest**
- **Instagram**

---

2. Manual Técnico

2.1. Entorno Servidor

El entorno servidor de este proyecto está diseñado para gestionar las operaciones de backend, integrando la lógica de negocio y sirviendo datos al entorno cliente de manera eficiente.

2.1.1. Modelo de datos

El modelo de datos está construido sobre una base de datos relacional administrada por **MySQL**, gestionada mediante **PhpMyAdmin** sobre un servidor **Apache**.
La base de datos se estructura en tablas relacionadas que almacenan la información esencial para el funcionamiento de la aplicación, como usuarios, productos, pedidos y datos del carrito de compras.

Características principales:

- **MySQL** proporciona robustez y flexibilidad para manejar grandes volúmenes de datos.
- **PhpMyAdmin** permite una administración visual de las tablas, consultas SQL y configuraciones.
- Relaciones bien definidas entre tablas para garantizar consistencia e integridad referencial.

2.1.2. Estructura y funcionamiento.

El servidor está desarrollado utilizando **PHP** y el **Framework Symfony 7**, que implementa el patrón de diseño **Modelo-Vista-Controlador (MVC)**. En este proyecto, el entorno servidor se centra principalmente en los componentes de **Modelo** y **Controlador**, ya que no cuenta con una interfaz gráfica directa.

La estructura de este apartado es:

Bin: Contiene el ejecutable de la consola que se utiliza para ejecutar comandos de Symfony desde el terminal, como generar entidades, migraciones y ejecutar servidores locales.

Src: Carpeta principal del código fuente que incluye:

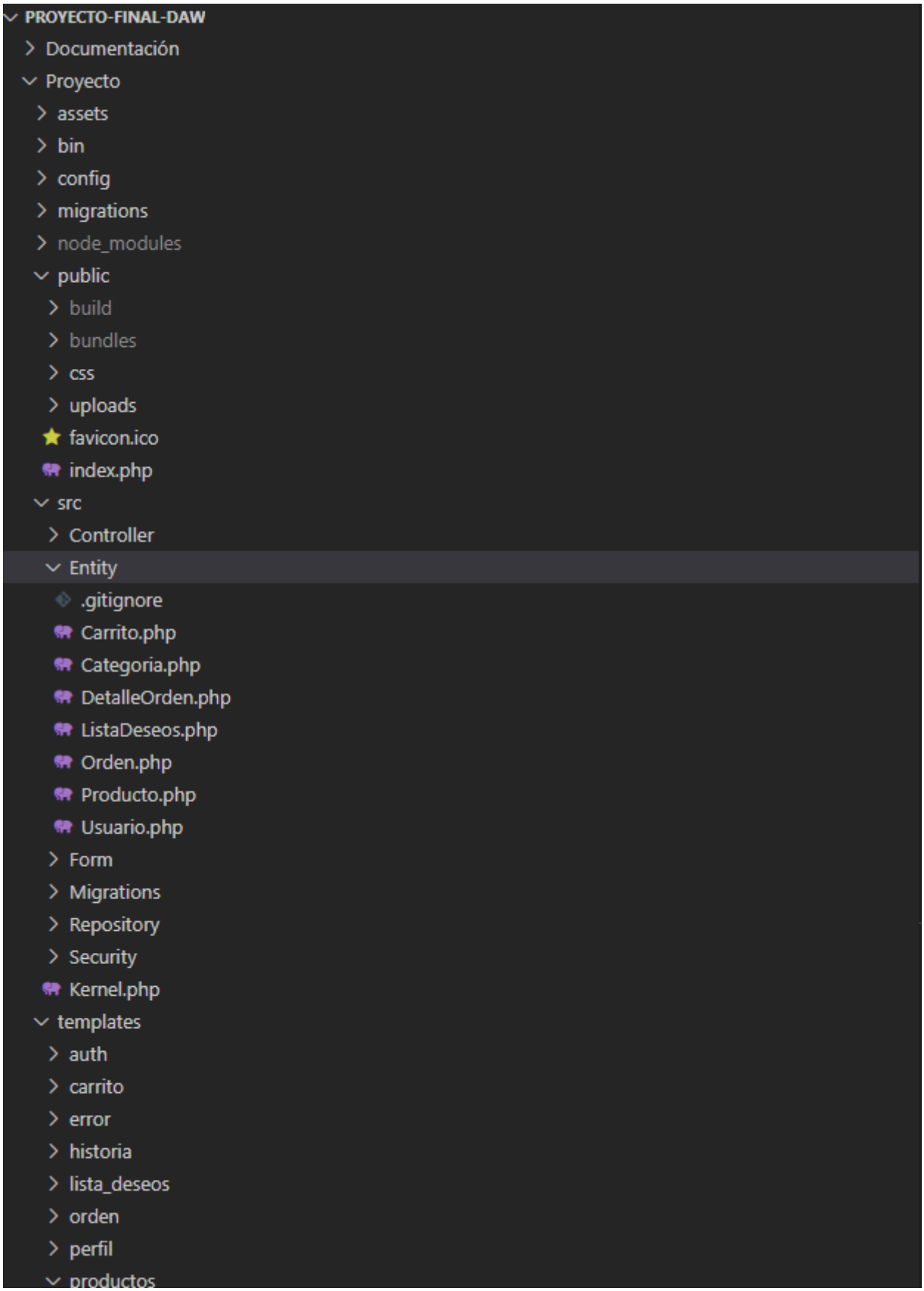
Entidades (`Entity`): Representaciones de las tablas de la base de datos en forma de clases PHP.

- **Controladores (`Controller`):** Lógica para manejar las solicitudes entrantes y devolver respuestas al cliente.
- **Servicios (`Services`):** Clases auxiliares que encapsulan lógica específica, como autorizaciones, cálculos, o interacción con la base de datos.

Test: Contiene los tests automatizados, estructurados como una réplica del directorio `src`. Sirve para validar que el código funciona correctamente.

Var: Carpeta temporal que almacena datos de caché, logs del sistema y sesiones de usuarios.

Vendor: Incluye todas las dependencias externas y el núcleo del framework Symfony, gestionadas por **Composer**.


PROYECTO-FINAL-DAW
Documentación
Proyecto
assets
bin
config
migrations
node_modules
public
build
bundles
css
uploads
favicon.ico
index.php
src
Controller
Entity
.gitignore
Carrito.php
Categoria.php
DetalleOrden.php
ListaDeseos.php
Orden.php
Producto.php
Usuario.php
Form
Migrations
Repository
Security
Kernel.php
templates
auth
carrito
error
historia
lista_deseos
orden
perfil
productos

Funcionamiento del Entorno Servidor:

El servidor procesa las solicitudes del cliente y realiza las siguientes operaciones:

1. **Controlador:**
    - Atiende las peticiones entrantes y actúa como intermediario entre el cliente y la lógica del modelo.

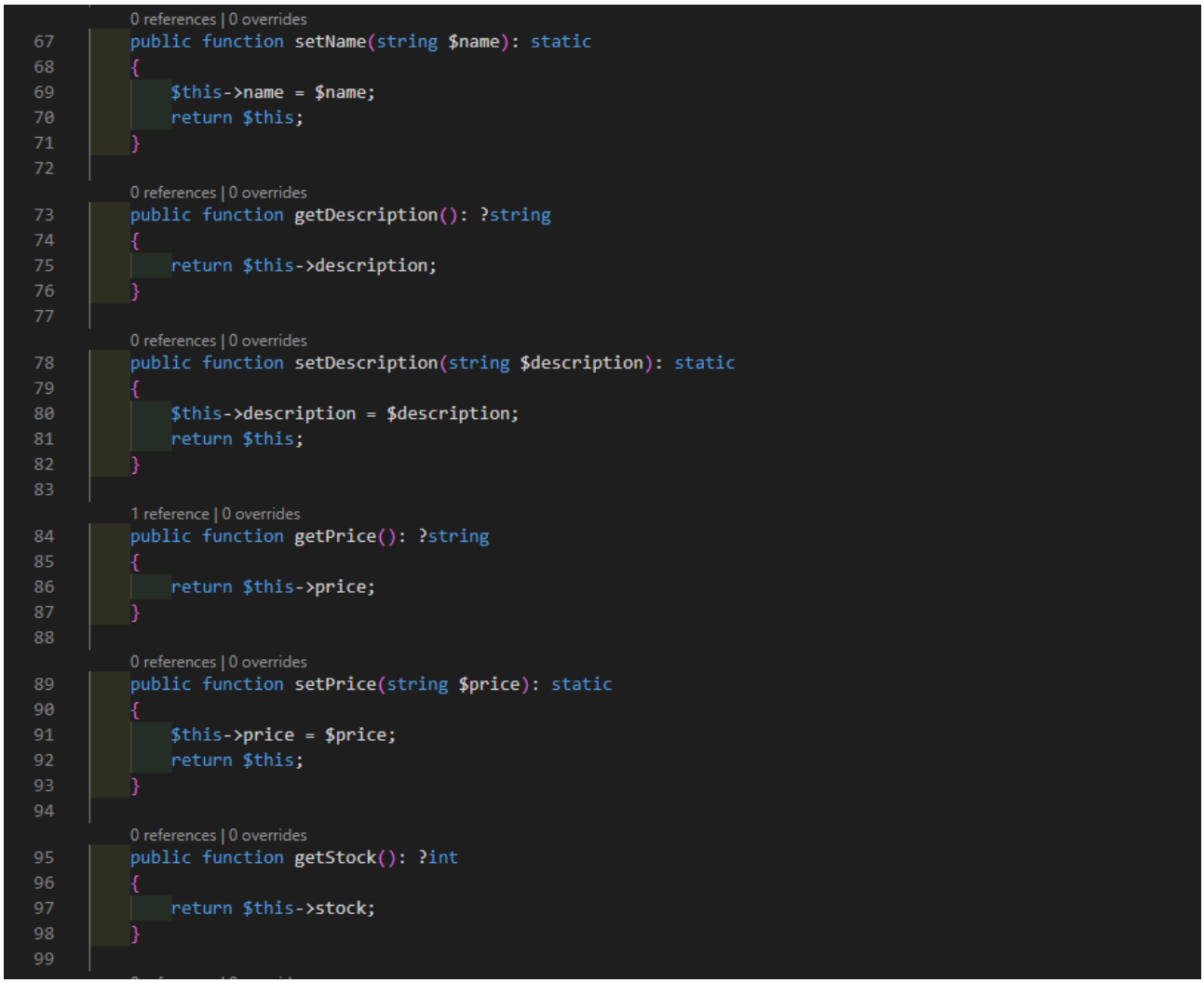
Ejemplo de controladores:

```
    <!-- Botón añadir al carrito -->
    <button id="add-to-cart" class="btn-primary" data-product-id="{{ producto.id }}">
        Añadir al carrito
    </button>
</div>
{% endblock %}

{% block javascripts %}
    {{ parent() }}
    {{ encore_entry_script_tags('DetallesProducto') }}
    <script>
        document.addEventListener("DOMContentLoaded", function () {
            const addToCartButton = document.getElementById('add-to-cart');
            const quantityInput = document.getElementById('product-quantity');

            addToCartButton.addEventListener('click', function () {
                const quantity = parseInt(quantityInput.value, 10);  // Cantidad del producto
                const productId = addToCartButton.getAttribute('data-product-id');  // ID del producto

                // Enviar la solicitud a la ruta app_carrito_agregar
                const url = `/carrito/agregar/${productId}`;

                fetch(url, {
                    method: 'GET', // Método GET
                })
                .then(response => response.json())
                .then(data => {
                    // Opcional: Muestra un mensaje al usuario indicando que el producto fue añadido
                    alert(`Producto añadido al carrito: ${data.productName}`);
                })
            });
        });
    </script>
{% endblock %}
```

1. **Mostrar los detalles del producto**
    - Obtiene datos de un producto específico desde la base de datos y los envía al cliente para su renderizado.
2. **Añadir un producto al carrito**
    - Procesa la solicitud para añadir un producto al carrito, valida la existencia del producto y actualiza los datos relacionados en la base de datos.

2. Modelo:

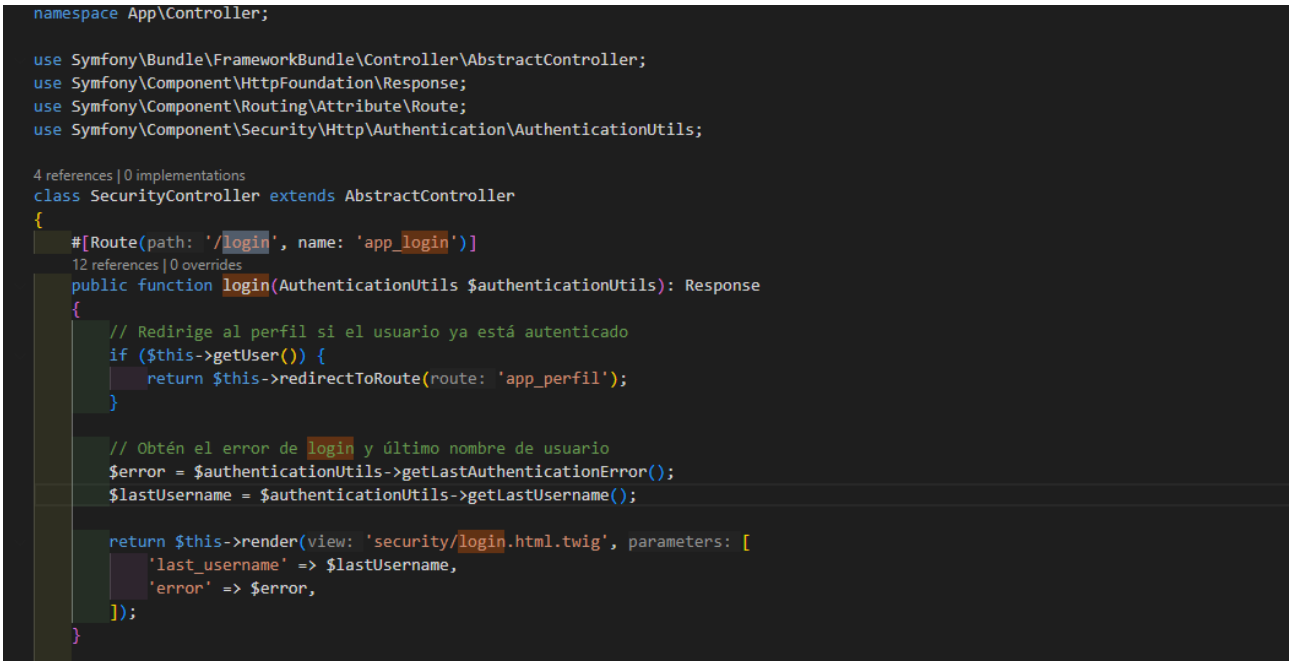
- Representa la lógica de negocio y administra las interacciones con la base de datos.
- Utiliza **Doctrine ORM** para mapear las entidades PHP con las tablas de la base de datos y manejar las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) de manera eficiente.



```
0 references | 0 overrides
public function setName(string $name): static
{
    $this->name = $name;
    return $this;
}

0 references | 0 overrides
public function getDescription(): ?string
{
    return $this->description;
}

0 references | 0 overrides
public function setDescription(string $description): static
{
    $this->description = $description;
    return $this;
}

1 reference | 0 overrides
public function getPrice(): ?string
{
    return $this->price;
}

0 references | 0 overrides
public function setPrice(string $price): static
{
    $this->price = $price;
    return $this;
}

0 references | 0 overrides
public function getStock(): ?int
{
    return $this->stock;
}
```

3. Servicios:

- Gestionan tareas específicas, como la autorización de usuarios mediante autenticación y validación de credenciales.



```php
namespace App\Controller;

use Symfony\Bundle\FrameworkBundle\Controller\AbstractController;
use Symfony\Component\HttpFoundation\Response;
use Symfony\Component\Routing\Attribute\Route;
use Symfony\Component\Security\Http\Authentication\AuthenticationUtils;

4 references | 0 implementations
class SecurityController extends AbstractController
{
    #[Route(path: '/login', name: 'app_login')]
    12 references | 0 overrides
    public function login(AuthenticationUtils $authenticationUtils): Response
    {
        // Redirige al perfil si el usuario ya está autenticado
        if ($this->getUser()) {
            return $this->redirectToRoute(route: 'app_perfil');
        }

        // Obtén el error de login y último nombre de usuario
        $error = $authenticationUtils->getLastAuthenticationError();
        $lastUsername = $authenticationUtils->getLastUsername();

        return $this->render(view: 'security/login.html.twig', parameters: [
            'last_username' => $lastUsername,
            'error' => $error,
        ]);
    }
```

2.2. Entorno cliente.

El entorno cliente corresponde a la interfaz que los usuarios finales utilizan para interactuar con la aplicación. Ha sido diseñado con un enfoque en la usabilidad y la experiencia del usuario, proporcionando un diseño responsivo y dinámico.

2.2.1. Tecnologías Utilizadas

HTML5 y Twig:
La estructura de las vistas dinámicas es renderizada en el servidor utilizando Twig (motor de plantillas de Symfony).

1. **CSS3 (SASS):**
   Utilizado para el diseño visual, con un enfoque en la adaptabilidad a diferentes dispositivos (responsive design).
   Se apoya en **Bootstrap** para acelerar el desarrollo del diseño.
2. **JavaScript (ES6):**
   Implementado para funcionalidades dinámicas, como validación en tiempo real y actualizaciones sin recargar la página.

3. **Frameworks y Librerías:**
    - **Bootstrap:** Asegura una estructura de diseño consistente.
    - **Lightbox:** Mejora la experiencia visual para la galería de imágenes.
    - **Fetch API:** Permite realizar solicitudes asincrónicas al servidor (AJAX).

2.2.2. Estructura General del Cliente

1. **Página de Inicio:**
   Presenta información clave, productos destacados y enlaces a las principales secciones.
2. **Catálogo de Productos:**
   Página interactiva que permite a los usuarios buscar, filtrar y explorar los productos.
3. **Vista de Producto:**
   Página dedicada con información detallada del producto, incluyendo:
    - Selector de cantidad.
    - Actualización dinámica del precio según la cantidad seleccionada.
    - Imágenes en formato galería.

   4. Carrito de Compras:
   Permite a los usuarios gestionar su selección antes de confirmar la compra. Incluye:
    - Funcionalidad para modificar cantidades.
    - Visualización del precio total en tiempo real.

   5. Formulario de Registro/Login:
   Implementa autenticación para garantizar la seguridad de los usuarios.

   6. Perfil de Usuario:
   Los usuarios registrados pueden acceder a sus datos personales y al historial de pedidos.

2.2.3. Funcionalidades Implementadas

1. **Interactividad Dinámica:**
   Actualización del contenido sin recargar la página mediante AJAX.
2. **Carrito de Compras en Tiempo Real:**
   El precio total se recalcula dinámicamente según la cantidad seleccionada por el usuario.
3. **Filtros Dinámicos:**
   En el catálogo de productos, los usuarios pueden aplicar filtros en tiempo real (rango de precios, categorías, etc.).
4. **Feedback Visual:**
   Notificaciones emergentes informan al usuario sobre acciones realizadas, como añadir productos al carrito.

2.2.4. Integración con el Servidor

El cliente y el servidor están sincronizados mediante solicitudes HTTP.

- **Método GET:** Para obtener información, como productos o detalles del carrito.
- **Método POST:** Para enviar datos, como credenciales de autenticación o pedidos confirmados.
- **Fetch API:** Se utiliza para operaciones asincrónicas, optimizando la experiencia del usuario.

3. Proceso del Proyecto

Semana 1: Boceto inicial

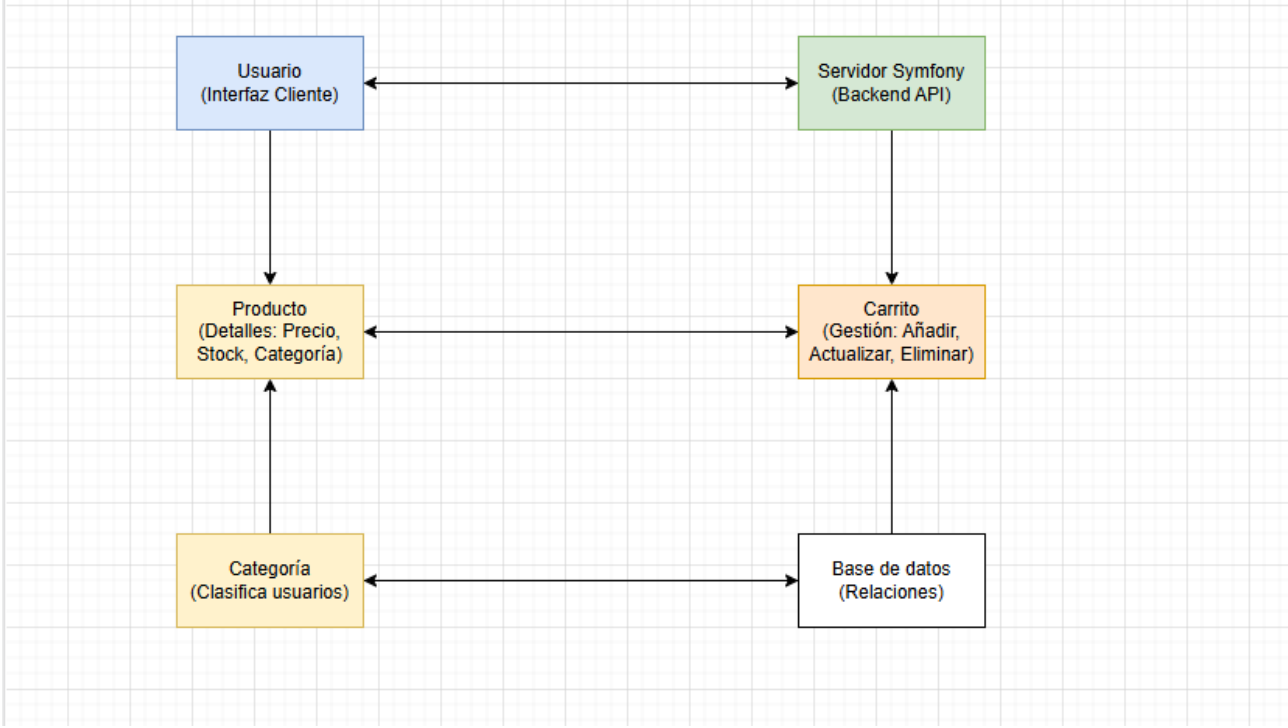
- **Objetivo:** Tener una idea clara de cómo lucirá y funcionará la página web.
- **Actividades realizadas:**
    1. Utilización de **Balsamiq** para crear un prototipo visual inicial del sitio web.
    2. Diseño de las páginas principales:
        - Página de inicio.
        - Catálogo de productos.
        - Detalle de producto.
        - Carrito de compras.
        - Registro/Login.
        - Perfil de usuario.
    3. Feedback inicial sobre el diseño para identificar mejoras en la experiencia del usuario.
- **Resultados:** Un boceto funcional que sirvió como guía para el desarrollo posterior.

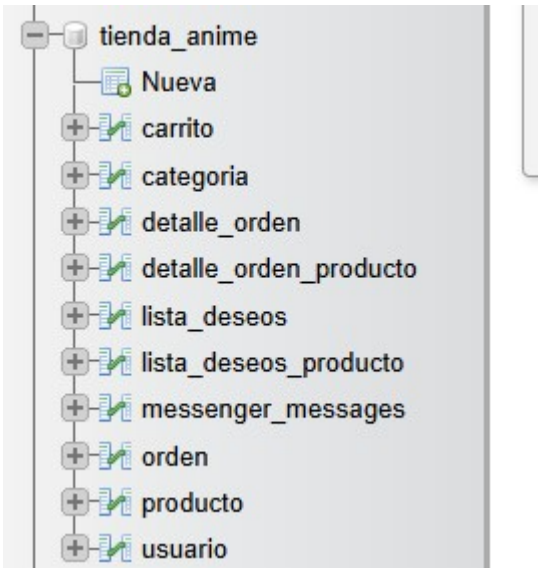
.

Semana 2: Diseño del modelo de datos

- **Objetivo:** Definir la estructura de la base de datos para soportar las funcionalidades principales.
- **Actividades realizadas:**
    1. **Identificación de entidades clave:**
        - Usuarios.
        - Productos.
        - Categorías.
        - Pedidos.
        - Carrito de compras.
    2. Definición de las relaciones entre las tablas (por ejemplo, "un usuario puede tener varios pedidos" o "un producto pertenece a una categoría").
    3. Creación del modelo E/R (Entidad-Relación) con herramientas como **MySQL Workbench** o diagramas en línea.
    4. Configuración inicial de la base de datos en **PhpMyAdmin**.

- **Resultados:** Un modelo de datos bien definido y listo para implementarse.





Semana 3: Configuración del entorno de desarrollo

- **Objetivo:** Preparar la infraestructura técnica para empezar a codificar.
- **Actividades:**
    1. Instalación de herramientas necesarias:
        - **Symfony 7** como framework principal.
        - **Composer** para la gestión de dependencias.
        - **PhpMyAdmin** para administrar la base de datos.
    2. Creación de un proyecto base con Symfony.
    3. Configuración del entorno local con un servidor Apache o Symfony CLI.
    4. Conexión del proyecto Symfony a la base de datos MySQL.
    5. Pruebas iniciales de configuración para asegurar que el entorno esté operativo.
- **Resultados:** Entorno configurado y funcional, listo para comenzar el desarrollo.

Semana 4: Desarrollo inicial del backend

- **Objetivo:** Implementar las funcionalidades básicas del servidor.
- **Actividades:**
    1. Creación de las entidades con **Doctrine ORM**:
        - Generación de las clases `Usuario`, `Producto`, `Categoría` y `Carrito`.
        - Configuración de las relaciones entre entidades.
    2. Migración de las entidades a la base de datos.
    3. Implementación de los controladores básicos:
        - Controlador para listar productos (`ProductController`).
        - Controlador para detalles de producto (`ProductController::show`).

4. Pruebas de las rutas del backend para asegurar la correcta conexión con la base de datos.

- **Resultados:** Backend inicial con funcionalidad básica para manejar productos.

Semana 5: Desarrollo de la interfaz (Frontend básico)

- **Objetivo:** Implementar las vistas principales y conectarlas al backend.
- **Actividades:**
  1. Uso de **Twig** para renderizar las páginas dinámicas.
  2. Diseño básico de las páginas principales:
     - Página de inicio con productos destacados.
     - Catálogo de productos con paginación.
     - Detalles de un producto.
  3. Implementación de un diseño responsivo con **Bootstrap**.
  4. Pruebas iniciales para asegurar que las vistas se carguen correctamente con los datos del backend.
- **Resultados:** Interfaz funcional y conectada al backend.

Semana 6: Funcionalidades dinámicas del cliente

- **Objetivo:** Mejorar la interactividad del cliente con JavaScript y Fetch API.
- **Actividades:**
  1. Implementación de funcionalidades dinámicas:
     - Actualización en tiempo real del precio según la cantidad seleccionada.
     - Añadir productos al carrito sin recargar la página.
  2. Uso de **Fetch API** para enviar y recibir datos del servidor de forma asincrónica (AJAX).

3. Validaciones dinámicas de formularios (por ejemplo, validación de campos de login).
4. Pruebas para validar que las interacciones funcionan correctamente.

- **Resultados:** Una interfaz más dinámica e interactiva.

Semana 7: Implementación de autenticación y perfil de usuario

- **Objetivo:** Asegurar que los usuarios puedan autenticarse y gestionar su perfil.
- **Actividades:**
  1. Configuración de **Symfony Security**:
     - Registro de nuevos usuarios.
     - Inicio de sesión (login) y cierre de sesión (logout).
  2. Implementación del controlador y la vista para el perfil del usuario:
     - Edición de información personal.
     - Visualización del historial de pedidos.
  3. Restricción de acceso a ciertas rutas según roles de usuario.
  4. Pruebas de seguridad y validación de credenciales.
- **Resultados:** Sistema de autenticación funcional.

Semana 8: Finalización y pruebas completas

- **Objetivo:** Pulir detalles finales y realizar pruebas exhaustivas.
- **Actividades:**
  1. Implementación de las últimas funcionalidades, como:
     - Gestión del carrito de compras (añadir, eliminar, modificar productos).
     - Confirmación de pedidos.
  2. Realización de pruebas completas:
     - Pruebas funcionales del backend y frontend.

    - Validación de la integración entre cliente y servidor.
    - Resolución de errores encontrados.
3. Optimización del rendimiento:
    - Uso de caché.
    - Minimización de archivos CSS y JS.
4. Documentación del proyecto (modelo de datos, estructura, funcionamiento).

- **Resultados:** Proyecto terminado, probado y documentado, listo para presentación o despliegue.

Errores encontrados durante la realización del proyecto.

1. Error: `quantityInput` y `priceDisplay` no definidos en la función de actualización del precio

Descripción del error:
Al intentar actualizar el precio total cuando el usuario interactúa con los botones de incremento o decremento de cantidad, me encontré con el problema de que algunos elementos en el DOM, específicamente los elementos con los IDs `quantityInput` y `priceDisplay`, no estaban definidos correctamente en el script. Este tipo de error ocurre cuando los IDs especificados en el JavaScript no coinciden con los elementos en el HTML, o bien cuando el DOM no está completamente cargado cuando se intenta acceder a esos elementos.

Solución:

Para evitar este error, me aseguré de que los elementos de los que dependía el script estuvieran presentes en el DOM y con los IDs correctos antes de acceder a ellos. Esto se logró mediante la validación de que los IDs coincidan entre el HTML y el JavaScript. Además, el script fue ejecutado dentro del evento `DOMContentLoaded` para asegurarme de que el DOM estuviera completamente cargado antes de interactuar con los elementos.

```
const quantityInput = document.getElementById('product-quantity');
const priceDisplay = document.getElementById('product-price');
```

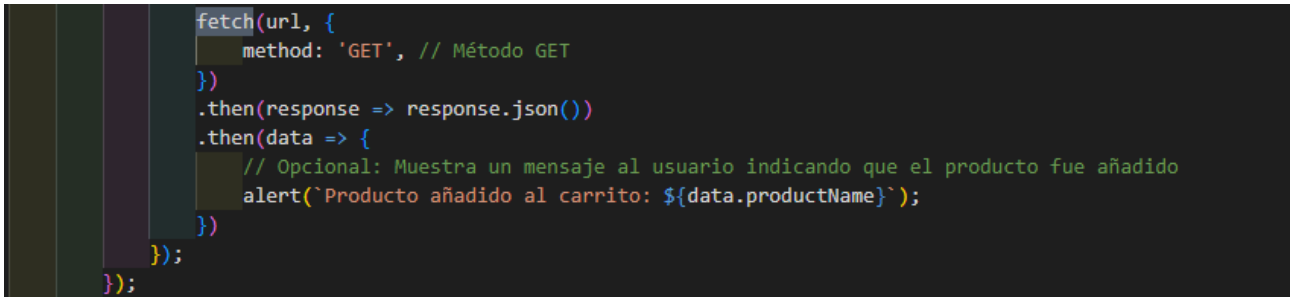
2. Error: El botón "Añadir al carrito" no maneja correctamente la respuesta del servidor.

Descripción del error:

Al hacer clic en el botón de "Añadir al carrito", se enviaba una solicitud al servidor, pero no se manejaba adecuadamente la respuesta, lo que generaba errores si el servidor no devolvía un JSON válido o si ocurría algún fallo en el proceso. Además, no se estaba mostrando un mensaje claro al usuario sobre el estado de la solicitud.

Solución:

Para solucionar este problema, implementé una gestión adecuada de errores utilizando `catch` para manejar posibles fallos en la solicitud `fetch()`. Además, aseguré que el servidor devolviera una respuesta en formato JSON con la información relevante, como el nombre del producto añadido al carrito. También incluí un mensaje de confirmación para que el usuario supiera que el producto se había añadido correctamente al carrito.



```
        fetch(url, {
            method: 'GET', // Método GET
        })
        .then(response => response.json())
        .then(data => {
            // Opcional: Muestra un mensaje al usuario indicando que el producto fue añadido
            alert(`Producto añadido al carrito: ${data.productName}`);
        })
    });
});
```

3. Error: `precioRange` no inicializado correctamente en algunos navegadores.

Descripción del error:

En el código se utiliza el `addEventListener` para manipular el control deslizante de precio (`precioRange`). Sin embargo, al cargar la página, el valor de este control no se inicializa correctamente si el DOM no está completamente cargado. Esto puede ocasionar que no se muestren los valores adecuados en las etiquetas de precio mínimo y máximo.

Solución:

El problema se soluciona asegurando que el código JavaScript solo se ejecute una vez que el DOM esté completamente cargado. Para ello, se debe encapsular el código dentro de un evento `DOMContentLoaded`, como ya está implementado en el script. Sin embargo, es recomendable revisar que el control deslizante esté presente en el DOM antes de intentar acceder a él, para evitar posibles errores si el control no está cargado o se omite.

```
document.addEventListener("DOMContentLoaded", function () {
    const precioRange = document.getElementById("precio");
    const precioMinValue = document.getElementById("precio_min_value");
    const precioMaxValue = document.getElementById("precio_max_value");

    // Función para actualizar el rango de precios en la interfaz y en la URL
    function updatePriceRange() {
        const precio = precioRange.value;

        // Actualizar la etiqueta del precio máximo
        precioMaxValue.innerText = `${precio}€`;

        // Actualizar la URL con el nuevo valor
        const url = new URL(window.location.href);
        url.searchParams.set('precio', precio);
        window.history.replaceState({}, '', url);
    }

    // Inicializar el control deslizante con el valor actual
    if (precioRange) {
        const initialValue = precioRange.value;
        precioMaxValue.innerText = `${initialValue}€`;
        precioMinValue.innerText = `${precioRange.min}€`;

        // Escuchar cambios en el control deslizante
        precioRange.addEventListener("input", updatePriceRange);
    }
});
```

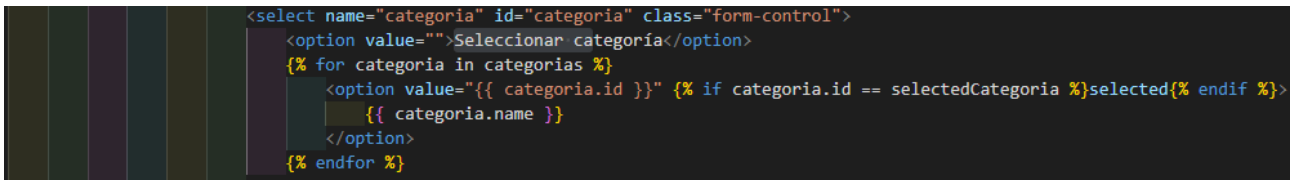
4. Error: El valor del filtro de categoría no se mantiene al aplicar los filtros.

Descripción del error:

Cuando el usuario selecciona una categoría en el filtro, el valor de la categoría seleccionada no se conserva si la página se recarga o se aplica un filtro. Esto se debe a que el valor seleccionado no se envía correctamente con la solicitud del formulario y no se restablece en el `select`.

Solución:

Para solucionar este problema, es necesario que el valor seleccionado se mantenga en el `select` después de que se apliquen los filtros. Esto se puede lograr asegurándose de que el valor del `select` esté marcado como `selected` en función de los parámetros de la URL o los datos disponibles en el servidor.



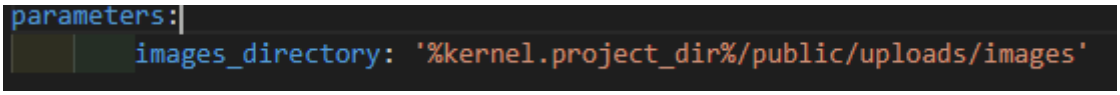
5.Problema al guardar las imágenes.

Descripción del error:

En el controlador `register`, al intentar guardar la imagen de perfil del usuario, me di cuenta de que usaba un parámetro llamado `images_directory` para indicar dónde mover la imagen. Esto puede causar errores si este parámetro no está configurado en el proyecto o si la carpeta destino no tiene permisos de escritura.

Solución:

Primero, revisé que el parámetro `images_directory` estuviera configurado en `services.yaml`. Esto lo solucioné añadiendo:



6. Problemas de permisos al mover archivos

Descripción del error:

En el controlador `edit`, al intentar mover un archivo subido al directorio `uploads/images`, se generaba una excepción porque el servidor no tenía permisos de escritura en la carpeta.

Solución:

Verifiqué que la carpeta existiera:

mkdir -p public/uploads/images

Le asigné permisos de escritura al servidor:

```
chmod -R 775 public/uploads/images
```

7. Fallo en la seguridad de rutas sensibles

Descripción del error:

Usuarios no autenticados podían acceder a rutas sensibles, como `/perfil/editar`, escribiendo la URL directamente en el navegador.

Solución:

Añadí restricciones en el archivo de configuración de seguridad para que solo los usuarios autenticados pudieran acceder: