Grado Superior

Desarrollo Aplicaciones Web

Trabajo fin de Grado

Detalles de Patch
Web online para productos artesanales

Autor:Jordi Llopis Godino

Tutores: Fortunato Yekue/ Luís García

Mayo 2025

Declaración de autoría:

Yo, Jordi Llopis Godino, declaro bajo mi responsabilidad que el Trabajo de Fin de Grado titulado *Detalls de Patch* es autoría propia, y que el contenido presentado no infringe las leyes vigentes sobre propiedad intelectual ni derechos de autor. Asimismo, manifiesto que todo el material de terceros utilizado en este trabajo ha sido citado y referenciado adecuadamente, atribuyéndose a sus legítimos autores.

Valencia, mayo de 2025.

Firmado:

Jordi Llopis Godino

Introducción

El proyecto, titulado *Detalls de Patch*, consiste en el desarrollo de una página web personal con una tienda online dedicada a la venta de productos artesanales. Esta plataforma tiene como finalidad ofrecer un espacio digital donde se pueda mostrar y comercializar una amplia variedad de artículos hechos a mano, entre los que se incluyen detalles para bodas, muñecos, ropa y otros complementos únicos.

Detalls de Patch comenzó su actividad participando en mercados locales de la zona, donde sus productos artesanales fueron ganando popularidad gracias a su calidad y personalización. Sin embargo, con el paso del tiempo, ha surgido la necesidad de adaptarse al entorno digital para ampliar su alcance y ofrecer una experiencia de compra más cómoda, directa y personalizada al gusto del cliente.

La idea de este proyecto surge precisamente de esa evolución: dar el salto a la venta online manteniendo la esencia del producto artesanal, hechos con amor y a medida. La web permite a los usuarios navegar por el catálogo, obtener información detallada de cada artículo, realizar pedidos y contactar fácilmente para solicitar productos personalizados.

Con este proyecto se pretende no solo facilitar el proceso de venta online, sino también crear una plataforma que refleje el estilo y la filosofía de *Detalls de Patch*, combinando funcionalidad, diseño y cercanía con el usuario.

Objetivos

El objetivo principal de este proyecto es desarrollar una plataforma web funcional y atractiva que permita a *Detalls de Patch* gestionar y vender sus productos artesanales de forma online, ofreciendo una experiencia personalizada , accesible y muy intuitiva para el cliente.

Además, se plantean los siguientes objetivos específicos:

- ➢ Crear una interfaz clara, intuitiva y visualmente atractiva que represente la identidad de la marca.

- ➢ Permitir la navegación por un catálogo de productos clasificados y con información detallada.

- ➢ Implementar un sistema de carrito de compra y gestión de pedidos.

- ➢ Facilitar la personalización de productos por parte del cliente, a través de formularios de contacto.

- ➢ Garantizar la correcta visualización y funcionamiento de la web en distintos dispositivos (responsive design).

- ➢ Proporcionar un panel de administración básico para gestionar productos, pedidos y clientes.

Metodología

Para la realización del proyecto se ha seguido una metodología de desarrollo ágil, organizando el trabajo en fases bien definidas que permiten avanzar de manera iterativa y controlada. A lo largo del proceso se ha tenido en cuenta tanto el diseño visual como la funcionalidad de la aplicación, buscando siempre el equilibrio entre estética y usabilidad.

Las principales fases del desarrollo han sido:

1. Análisis de necesidades: Se ha recogido la información relevante sobre el tipo de productos, el perfil del cliente y las necesidades concretas del negocio artesanal.

2. Diseño: Se han creado maquetas de la estructura de la web, teniendo en cuenta la identidad visual de *Detalls de Patch*, la experiencia de usuario y la atracción al mismo.

3. Desarrollo: Se ha implementado la web utilizando tecnologías modernas tanto en el frontend como en el backend, con especial atención a la seguridad, el rendimiento y la escalabilidad.

4. Pruebas: Se ha realizado una fase de pruebas funcionales y de usabilidad para asegurar el correcto funcionamiento de la web en diferentes dispositivos y navegadores.

5. Despliegue: Finalmente, la aplicación se ha desplegado en un entorno de producción, dejándola lista para su uso real y futura ampliación.

Agradecimientos

Quiero aprovechar esta parte para dar las gracias a todas las personas que me han acompañado y apoyado durante este camino, tanto en lo personal como en lo académico.

En primer lugar, a mi familia, por estar siempre ahí, por su paciencia infinita, su ánimo constante y por creer en mí incluso en los momentos en los que yo mismo dudaba y por confiar en mí cuando les dije que quería irme a Valencia a estudiar, lejos de casa. Sin ellos, este proyecto no habría sido posible.

A mis amigos, por ayudarme a desconectar gracias a su apoyo, consejos, chistes y grandes momentos.. Gracias por estar al otro lado siempre que lo he necesitado.

A mis compañeros de DAW, con los que he compartido tantos momentos tanto graciosos como duros. Gracias por la ayuda, la compañía que ha conseguido crear un gran grupo .

Un agradecimiento muy especial para mi compañero y asere Rodolfo, por todas esas tardes de estudio compartidas, por la paciencia, por las explicaciones, y sobre todo por la compañía que hemos pasado juntos. ¡Este trabajo también es tuyo!

Y cómo no, gracias también a mis profesores Luis y Fortu, por su dedicación, por transmitirnos sus conocimientos, y por estar siempre dispuestos a ayudar cuando lo necesitábamos.

Índice

1. Introducción

1.1 Introducción general

El proyecto, *Detalls de Patch*, consiste en el desarrollo de una página web personal con una tienda online dedicada a la venta de productos artesanales. Esta plataforma tiene como finalidad ofrecer un espacio digital donde se pueda mostrar y comercializar una amplia variedad de artículos hechos a mano, entre los que se incluyen detalles para bodas, muñecos, ropa y otros complementos únicos.

Detalls de Patch comenzó su actividad participando en mercados locales de su zona, donde sus productos ganaron popularidad por su calidad, originalidad y nivel de personalización. Con el paso del tiempo, la sensación era de que necesitaba ampliar su alcance y facilitar el acceso a los clientes y ofrecer una experiencia de compra más cómoda, directa y personalizada.

Este proyecto nace precisamente de esa necesidad: crear una web que mantenga la esencia del producto artesanal, hecho con dedicación y cariño, pero adaptada a las demandas del entorno online moderno. La web permitirá a los usuarios explorar el catálogo, conocer cada producto en detalle, realizar pedidos y contactar fácilmente para encargos personalizados.

1.2 Motivación del proyecto

La motivación detrás de este trabajo no es solo técnica, sino también personal, ya hace años que tenía pensado crear una web de este estilo para mi madre. *Detalls de Patch* representa el esfuerzo, la creatividad y el trabajo manual de una pequeña artesana, y ofrecerle una herramienta digital adecuada para mostrar sus creaciones supone una forma de apoyar este tipo de comercio local y tradicional.

Además, este proyecto es una oportunidad para aplicar de forma práctica los conocimientos adquiridos durante el ciclo formativo de Desarrollo de Aplicaciones Web, integrando diseño, desarrollo backend y frontend, usabilidad, accesibilidad y despliegue en producción. El reto de combinar todos estos aspectos en una solución completa y funcional resulta muy enriquecedor a nivel académico y profesional.

1.3 Objetivos

El objetivo principal de este proyecto es desarrollar una plataforma web funcional y atractiva que permita a *Detalls de Patch* gestionar y vender sus productos artesanales online, ofreciendo una experiencia personalizada y sencilla para el cliente.

Los objetivos específicos son:

- ➢ Crear una interfaz clara, intuitiva y visualmente atractiva que represente la identidad de la marca.

- ➢ Permitir la navegación por un catálogo de productos clasificados y con información detallada.

- ➢ Implementar un sistema de carrito de compra.

- ➢ Facilitar la personalización de productos por parte del cliente, mediante formularios de contacto.

- ➢ Garantizar la correcta visualización y funcionamiento de la web en distintos dispositivos (diseño responsive).

- ➢ Proporcionar un panel de administración para gestionar productos, pedidos y clientes de forma sencilla.

1.4 Organización de la memoria

Esta memoria está estructurada en varios capítulos, cada uno de los cuales aborda una parte concreta del desarrollo del proyecto:

- ➢ En el Capítulo 2, se realiza un análisis del estado del arte, explorando webs similares y las tecnologías más habituales.

- ➢ El Capítulo 3 detalla los requisitos funcionales y técnicos, especificaciones del sistema, estimación de costes y análisis de riesgos y viabilidad.

- El Capítulo 4 se centra en el análisis detallado del sistema, incluyendo casos de uso y modelo de datos.

- En el Capítulo 5 se presenta el diseño del sistema: arquitectura, interfaces y lógica de negocio.

- El Capítulo 6 describe la implementación técnica del proyecto y las pruebas realizadas.

- Finalmente, el Capítulo 7 recoge las conclusiones generales, la revisión de los objetivos y una propuesta de mejoras futuras.

2. Estado del arte

2.1 Análisis de tiendas online similares

El auge de los productos artesanales y personalizados ha dado lugar a numerosas plataformas online que permiten a los pequeños creadores vender sus productos. Entre las más destacadas se encuentran:

- ➢ Etsy: Es una de las plataformas más conocidas para la venta de productos artesanales. Ofrece una gran variedad de productos, sistema de pagos integrado y herramientas de análisis para vendedores.Pero tiene sus desventajas, las comisiones y la poca personalización que le puedes añadir son una de sus principales desventajas.

- ➢ Shopify: Es una solución completa para crear tiendas online, con plantillas y muchas funcionalidades integradas, aunque con costes mensuales y limitaciones en cuanto a personalización sin conocimientos técnicos.

- ➢ Prestashop / WooCommerce: Permiten crear tiendas online más personalizadas y auto-gestionadas, pero requieren conocimientos técnicos para su correcto mantenimiento.
- ➢ Eaudemar: Una Tienda online de perfumes, sencilla con estilos modernos y minimalistas pero con una interfaz intuitiva enfocada a la experiencia del usuario

Estas opciones muestran cómo los artesanos pueden digitalizar su negocio, aunque muchas veces con herramientas genéricas o poco adaptadas a sus necesidades concretas. Por ello, desarrollar una tienda a medida como *Detalls de Patch* permite cubrir requerimientos específicos de forma más eficiente, como un panel personalizado de administración, opciones de contacto directo para productos hechos por encargo, y una experiencia visual única.

2.2 Tecnologías utilizadas habitualmente

Las tecnologías más empleadas en el desarrollo de tiendas online y plataformas web modernas incluyen:

- Frontend:
    - *HTML5*, *CSS3* y *JavaScript* como tecnologías base.
    - *Vue.js*, *React* o *Angular* para construir interfaces interactivas y dinámicas.
    - Frameworks como *Bootstrap* para un diseño responsive y componentes reutilizables.
- Backend:
    - *PHP* (en plataformas como WordPress/WooCommerce o Prestashop).
    - *Python* con frameworks como *Flask* o *Django*, que permiten mayor control y flexibilidad.
    - *Node.js*, usado en muchas plataformas modernas por su rapidez y escalabilidad.
- Bases de datos:
    - *MySQL*, *PostgreSQL* y *SQLite* son las más habituales en proyectos web.
    - Se utilizan para almacenar productos, usuarios, pedidos y otros datos relacionados.
- Despliegue y gestión:
    - *Docker* para contenerizar aplicaciones y facilitar su despliegue.
    - Servidores como *Nginx* para servir contenido web.
    - Herramientas como *Supervisor*, *pm2* o *systemd* para mantener procesos activos en producción.

En el caso de *Detalls de Patch*, se ha optado por una arquitectura basada en:

- ➢ Frontend con *Vue.js* para una interfaz moderna, rápida y fácil de usar.
- ➢ Backend con *Flask* por su simplicidad y capacidad de integración con bases de datos.
- ➢ Base de datos MySQL, ideal para relaciones entre productos, usuarios y pedidos.

Esta combinación de tecnologías permite un desarrollo modular, escalable y completamente personalizado, lo que se adapta perfectamente a las necesidades de un negocio artesanal que busca diferenciarse.

3. Requisitos, especificaciones y viabilidad

3.1 Requisitos funcionales y no funcionales

Requisitos funcionales

Los requisitos funcionales describen lo que el sistema debe hacer:

- RF1: El usuario podrá navegar por el catálogo de productos clasificados por categoría.
- RF2: El usuario podrá ver el detalle de cada producto (nombre, descripción, precio, imágenes).
- RF3: El usuario podrá añadir productos al carrito de compra.
- RF4: El usuario podrá realizar un pedido rellenando un formulario de pago.
- RF5: El administrador podrá gestionar productos (añadir, editar, eliminar).
- RF6: El administrador podrá ver y gestionar pedidos recibidos.
- RF7: El sistema permitirá personalizar encargos mediante un formulario.
- RF8: La web enviará notificaciones por email tras la realización de un pedido.

Requisitos no funcionales

Los requisitos no funcionales definen cómo debe comportarse el sistema:

- RNF1: La web será responsive, adaptándose a móviles, tablets y ordenadores.
- RNF2: El sistema debe estar disponible al menos el 95% del tiempo.
- RNF3: La aplicación web deberá cargar en menos de 3 segundos.

- ➢ RNF4: La interfaz debe ser accesible y fácil de usar para personas sin conocimientos técnicos.

- ➢ RNF5: Se protegerán los datos personales de los usuarios mediante HTTPS y buenas prácticas de seguridad.

- ➢ RNF6: El sistema será modular, permitiendo futuras mejoras y ampliaciones.

3.2 Especificaciones del sistema

El sistema estará compuesto por los siguientes elementos:

- ➢ Frontend:
    - SPA desarrollada con *Vue.js*.
    - Bootstrap como base de diseño para asegurar una buena presentación responsive.
    - Integración con API REST para operaciones con productos y pedidos.
- ➢ Backend:
    - Desarrollado en *Flask* (Python).
    - API REST que permite las operaciones CRUD para productos, pedidos y usuarios.
    - Validación de formularios y gestión de sesiones administrativas.
- ➢ Base de datos:
    - *MySQL* como sistema gestor.

- Tablas principales: productos, pedidos, usuarios y reset_tokens.

➢ Despliegue:

- Contenedores *Docker*.
- Servidor *Nginx* como proxy inverso.
- *Supervisor* para mantener los servicios activos.
- Dominio propio y certificado SSL.

3.3 Estimación de costes

Este proyecto se ha realizado en un entorno de desarrollo propio, sin inversión económica directa. Pero hay una estimación de costes en caso de desplegar la solución a nivel real:

Elemento	Coste estimado anual
Dominio web (.com / .es)	12 €
Alojamiento VPS básico	60 – 100 €
Total aproximado	72 – 112 €

3.4 Identificación de riesgos

Durante el desarrollo del proyecto, se han identificado los siguientes posibles riesgos:

➢ R1: Dificultades técnicas al integrar distintas tecnologías (Vue, Flask, MySQL).

- *Solución*: realizar pruebas periódicas y consultar documentación oficial.

➢ R2: Problemas de seguridad al tratar con formularios y datos personales.

- *Solución*: validar y sanitizar entradas, usar HTTPS.

➢ R3: Falta de conocimientos específicos en administración de servidores.

- *Solución*: investigar, aprender con documentación y apoyo del profesorado y de compañeros.

3.5 Viabilidad técnica y económica

El proyecto es técnicamente viable, ya que se basa en tecnologías conocidas por el desarrollador, muchas de ellas estudiadas durante el ciclo DAW (HTML, CSS, JS, Python, bases de datos).

Además, se ha optado por herramientas y recursos gratuitos o de código abierto, lo que permite reducir el coste a prácticamente cero durante la fase de desarrollo y pruebas.

En cuanto a la viabilidad económica, el mantenimiento anual de una plataforma de este tipo puede mantenerse por debajo de los 100 €, haciéndolo muy asequible para un pequeño negocio artesanal. Esta inversión se compensa fácilmente con unos pocos pedidos mensuales.

4. Análisis

4.1 Análisis de usuarios

Se han identificado dos perfiles principales de usuario para la plataforma:

Usuario comprador

Es el visitante habitual de la web. Sus características y necesidades son:

- ➢ Busca productos únicos, artesanales y personalizados.
- ➢ Necesita una interfaz sencilla, visual y clara.
- ➢ Requiere ver productos, con imágenes, descripciones y precios.
- ➢ Desea una experiencia de compra fácil: añadir productos al carrito, hacer pedidos y contactar para encargos especiales.
- ➢ Se puede acceder desde dispositivos móviles o de escritorio.

Usuario administrador

Corresponde a la persona responsable de gestionar la tienda:

- ➢ Añade, edita y elimina productos.
- ➢ Consulta pedidos realizados por los clientes.
- ➢ Actualiza información del catálogo o la galería.
- ➢ Gestiona mensajes o solicitudes de productos personalizados.
- ➢ Puede crear más usuarios administradores.
- ➢ Recuperar la contraseña.

4.2 Casos de uso

A continuación se detallan los principales casos de uso para cada perfil:

Casos de uso del comprador:

- CU01 - Navegar por el catálogo: El usuario puede explorar todos los productos.
- CU02 - Ver producto: El usuario accede al detalle del producto con imágenes, precio y descripción.
- CU03 - Añadir al carrito: El usuario añade un producto al carrito para su posterior compra.
- CU04 - Realizar pedido: Finaliza el pedido, introduciendo datos de contacto y envío, tarjeta(no real) y confirmación.
- CU05 - Contactar para personalización: Envía un mensaje con una solicitud personalizada desde la ruta de contacto.

Casos de uso del administrador:

- CU06 - Iniciar sesión: El administrador accede al panel de control mediante autenticación.
- CU07 - Gestionar productos: Puede crear, modificar o eliminar productos.
- CU08 - Ver y gestionar pedidos: Consulta los pedidos recibidos.
- CU09 - Gestionar mensajes personalizados: Lee y responde solicitudes de personalización.
- CU10 - Recuperar la contraseña.

4.3 Estructura de la base de datos

La base de datos, denominada detalls, ha sido diseñada con el objetivo de almacenar de forma eficiente y estructurada la información relacionada con los usuarios, productos, pedidos y transacciones realizadas en la plataforma. Se han seguido principios de normalización y claridad para garantizar integridad y escalabilidad.

A continuación, se detallan las tablas implementadas:

Tabla	Descripción
users	Almacena la información de los usuarios registrados, incluyendo email y contraseña. Ideal para gestionar el acceso a la plataforma.
productos	Contiene los productos disponibles: nombre, descripción, precio, imagen, stock, etc.
pagos	Registra los pedidos realizados, incluyendo datos del cliente, lista de productos, dirección, importe total, etc.
reset_tokens	Guarda los tokens generados para el restablecimiento de contraseñas, junto con la fecha de expiración.

Modelo de datos y relaciones

- Users: cada usuario registrado tiene un email único y una contraseña encriptada. No se establece una relación directa con los pagos, ya que en la tabla pagos se almacena sólo el email del comprador como referencia.

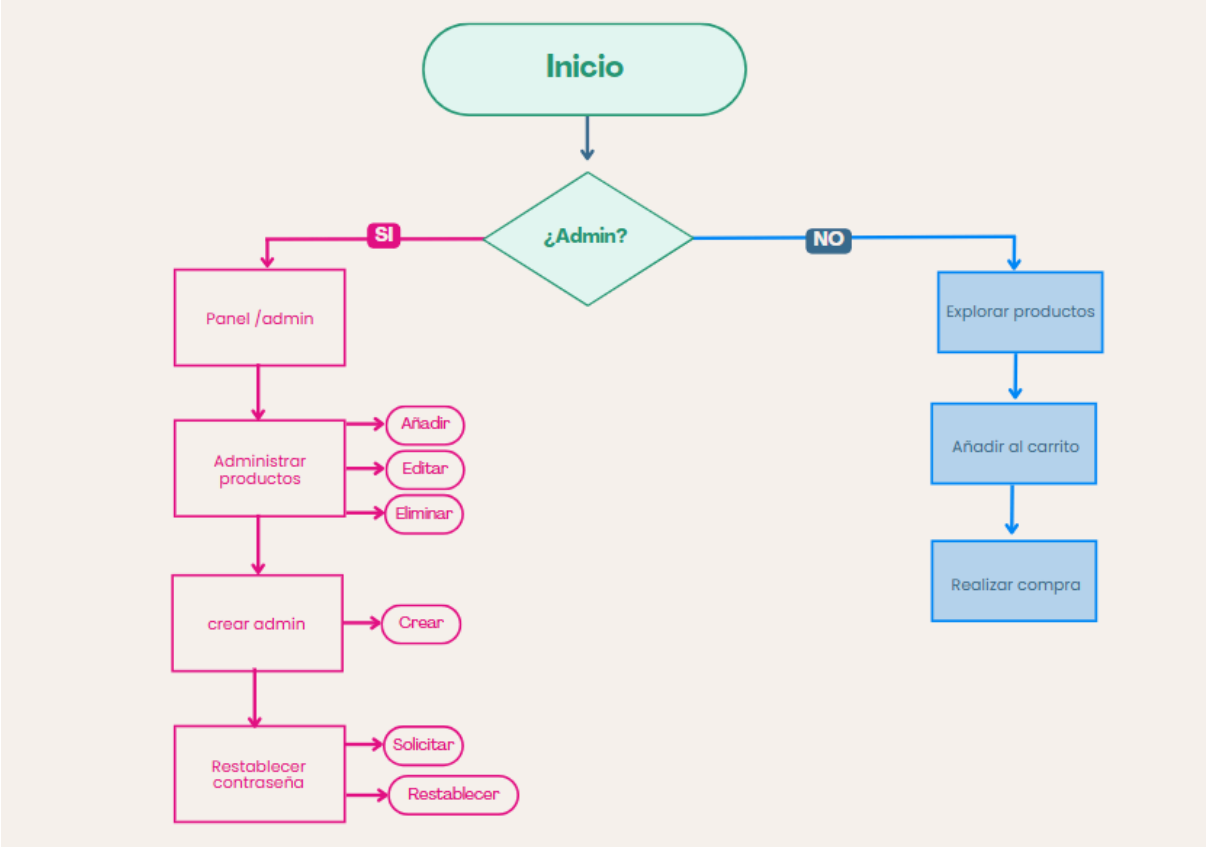
- Productos: tabla central que almacena los artículos disponibles. En esta versión, los productos se almacenan como un texto serializado dentro del campo productos de la tabla pagos, por lo que no se ha implementado una relación directa entre ambas tablas (aunque podría optimizarse con una tabla

intermedia para mejorar la integridad referencia, pero para que sea más sencillo visualmente desde la ruta /admin se ha hecho así).

- Pagos: almacena toda la información necesaria para procesar un pedido: productos seleccionados, dirección, ciudad, código postal, provincia y el total del pedido. Actúa como una tabla de pedidos.

- Reset_tokens: gestiona el sistema de recuperación de contraseñas mediante tokens temporales vinculados a emails que deben formar parte de la base de datos users.

Diagrama de flujo de uso



5. Diseño

5.1 Diseño de la arquitectura

La arquitectura del sistema sigue una estructura **cliente-servidor**, dividida en tres capas principales:

- **Frontend**: Desarrollado en Vue.js, proporciona una interfaz responsiva y dinámica para el usuario final. Se comunica con el backend mediante peticiones HTTP a través de fetch.

- **Backend**: Implementado con Flask (Python), expone una API REST que gestiona la lógica de negocio, operaciones con la base de datos y seguridad.

- **Base de datos**: Un sistema MySQL almacena información persistente, como usuarios, contraseñas cifradas y tokens de recuperación.

La aplicación se apoya en esta separación para facilitar el mantenimiento, la escalabilidad y el despliegue independiente de cada componente. La arquitectura también permite proteger rutas, validar accesos mediante tokens y almacenar temporalmente información sensible (como solicitudes de reseteo de contraseña) con caducidad controlada.

5.2 Diseño de la interfaz

La interfaz de usuario está diseñada para ser clara, accesible y funcional. Usa HTML semántico y estilos personalizados con CSS y clases reutilizables. Las vistas están organizadas como componentes Vue, lo que facilita su reutilización y mantenimiento.

Entre las vistas principales se encuentran:

- Login y registro: Formularios validados y con respuestas inmediatas del servidor.

- Página de recuperación de contraseña: Se accede mediante un enlace enviado por correo, incluye formulario seguro para restablecer la clave.

- ➢ Dashboard o vista principal: Mostrada tras el login, adaptada según el tipo de usuario si existiera diferenciación.

En caso de error (por ejemplo, un token inválido), se muestra un mensaje claro y se redirige automáticamente al usuario al inicio de sesión. Además, después de operaciones exitosas como restablecer la contraseña, se muestra un mensaje de éxito seguido de una redirección automática a la página principal.

5.3 Diseño del Backend

El backend de la aplicación está desarrollado con Flask y sigue una estructura RESTful para gestionar las distintas funcionalidades del sistema. A continuación se describen las rutas y características más relevantes:

Autenticación y autorización

- ➢ **POST /login**: Verifica las credenciales del usuario y, si son válidas, inicia sesión utilizando Flask-Login. Solo los administradores pueden acceder al área protegida /admin.

- ➢ **POST /register**: Crea un nuevo usuario administrador, validando que el correo no exista previamente.

- ➢ **POST /logout**: Cierra la sesión del usuario actual.

Recuperación de contraseña

- ➢ **POST /recuperar**: Genera un token seguro con expiración (1 hora) mediante secrets.token_urlsafe y lo envía al correo del usuario para permitir la recuperación de la contraseña.

- ➢ **GET/POST /resetear/<token>**: Verifica la validez y expiración del token. Si es válido, permite establecer una nueva contraseña, almacenándola de forma segura con generate_password_hash y eliminando el token tras su uso.

Gestión de productos (administrador)

- ➢ **POST /admin**: Añade un nuevo producto a la base de datos.

- ➢ **PUT /admin/<id>**: Actualiza la información de un producto existente.

- ➢ **DELETE /admin/<id>**: Elimina un producto por su identificador.

- ➢ **GET /admin/editar/<id>**: Recupera la información detallada de un producto para su edición.

Visualización de productos (usuario)

- ➢ **GET /**: Muestra productos destacados con mayor stock y un mensaje de bienvenida.

- ➢ **GET /carrito**: Muestra los articulos añadidos al carrito, nombre, precio imagen... y el total.

- ➢ **GET /tienda**: Muestra todos los productos ordenados por ID descendente (últimos añadidos).

Procesamiento de pagos

- ➢ **POST /pago**: Recibe el carrito, total y datos de envío. Almacena la compra en la base de datos y envía un correo de confirmación al comprador mediante smtplib.

Contacto

- ➢ **POST /contacto**: Permite al usuario enviar un mensaje al administrador mediante un formulario de contacto. El mensaje es enviado por correo electrónico.

Seguridad y buenas prácticas implementadas

- ➢ Las contraseñas se almacenan con hash mediante werkzeug.security.generate_password_hash() y se validan con check_password_hash().

- ➢ Los tokens de recuperación tienen expiración y se eliminan tras su uso para evitar reutilización.

- ➢ Se emplea Flask-CORS para permitir peticiones del frontend (Vue.js).

- ➢ Todos los errores se manejan de forma controlada, devolviendo códigos HTTP adecuados (400, 401, 404, 500) junto con mensajes en formato JSON legibles tanto por humanos como por el cliente.

- ➢ Se hace uso de Flask-Login para gestionar la sesión de usuarios autenticados.

- ➢ Se protege el acceso a ciertas rutas mediante @login_required.

6. Implementación y pruebas

6.1 Implementación

La implementación del sistema se realizó siguiendo una arquitectura cliente-servidor. Se desarrollaron dos componentes principales:

- ➢ Frontend: Construido con Vue.js. Se encarga de la interfaz de usuario, navegación y envío de solicitudes HTTP al backend mediante fetch.

- ➢ Backend: Implementado en Flask (Python), sirve como API REST. Gestiona la lógica de negocio, validación de datos, autenticación, envío de correos y acceso a la base de datos MySQL mediante la librería flask_mysqldb.

Durante la implementación se llevaron a cabo las siguientes tareas:

- ➢ Diseño de rutas en el backend para operaciones CRUD sobre productos (GET, POST, PUT, DELETE).

- ➢ Implementación de autenticación de usuarios con sesiones y Flask-Login.

- ➢ Registro y login de usuarios, con validación de credenciales mediante werkzeug.security.

- ➢ Recuperación de contraseña vía correo electrónico y tokens temporales.

- ➢ Procesamiento de pedidos y almacenamiento de la información de compra.

- ➢ Envío de correos de confirmación usando smtplib.

- ➢ Integración del backend con una base de datos MySQL, utilizando consultas SQL crudas mediante cursores (cursor.execute()).

- ➢ Manejo de errores y validaciones tanto del lado del cliente como del servidor.

6.2 Pruebas funcionales

Las pruebas funcionales se realizaron para asegurar que cada funcionalidad del sistema opera correctamente. Se probaron manualmente las siguientes acciones principales:

Caso de prueba	Acción	Resultado esperado
Crear producto	Rellenar formulario y enviar	Producto guardado en la base de datos y visible en la lista
Editar producto	Modificar un producto existente	Cambios reflejados en la lista y base de datos
Eliminar producto	Pulsar botón de eliminar	Producto eliminado de la lista y base de datos
Mostrar lista de productos	Acceder a la vista principal	Se muestran todos los productos disponibles
Validación de campos	Dejar campos vacíos en el formulario	Se muestra mensaje de error y no se procesa la acción

Además, se validaron:

- ➢ Accesos no autorizados a rutas protegidas.
- ➢ Comprobación de tokens de recuperación de contraseña.
- ➢ Errores del servidor al guardar o eliminar productos.

6.3 Pruebas de rendimiento

Para evaluar el rendimiento del sistema, se utilizaron herramientas como Postman, Apache Bench (ab) y las herramientas de desarrollo del navegador. Los resultados fueron los siguientes:

- ➢ Tiempo de respuesta del backend: promedio entre 80 ms y 150 ms para operaciones simples (CRUD).
- ➢ Carga de la vista inicial: menos de 1 segundo en red local.
- ➢ Pruebas de concurrencia: hasta 100 solicitudes simultáneas a GET /productos sin pérdida de datos ni caídas del servidor.
- ➢ Uso de recursos: en entorno de desarrollo, el servidor Flask mantuvo un uso moderado de CPU y bajo consumo de memoria.

6.4 Pruebas de usabilidad

Se realizaron pruebas de usabilidad con un pequeño grupo de usuarios no técnicos. Se evaluaron los siguientes aspectos:

- ➢ Intuición de la interfaz: qué tan fácil es comprender y navegar por el sistema.
- ➢ Tiempo de ejecución de tareas: agregar, editar o eliminar un producto.
- ➢ Retroalimentación visual: presencia de mensajes claros ante errores o acciones exitosas.

Resultados:

- ➢ Todos los participantes lograron realizar tareas básicas sin necesidad de instrucciones.
- ➢ Se sugirió mejorar el contraste de algunos botones y mostrar mensajes más visibles tras una acción.

7. Conclusiones

7.1 Evaluación de objetivos y resultados

Los objetivos planteados al inicio del proyecto fueron alcanzados satisfactoriamente:

- ➢ Desarrollar una aplicación web funcional con arquitectura cliente-servidor.
- ➢ Implementar un sistema completo de gestión de productos (añadir, eliminar, editar, listado).
- ➢ Desarrollar un módulo de autenticación de usuarios, incluyendo login, registro y recuperación de contraseña.
- ➢ Integrar el sistema con una base de datos MySQL, utilizando consultas directas desde Flask.
- ➢ Incorporar funcionalidades esenciales para un sistema de ventas, como carrito de compras, confirmación de pedidos y envío de correos electrónicos.
- ➢ Realizar pruebas funcionales, de rendimiento y usabilidad que confirmaron el correcto funcionamiento y la estabilidad del sistema.

7.2 Conclusiones

El desarrollo del sistema permitió aplicar múltiples conocimientos adquiridos en el área de desarrollo web, bases de datos, seguridad y experiencia de usuario. Se logró construir una aplicación robusta y escalable que cumple con los requerimientos funcionales propuestos.

El uso de herramientas modernas como Flask, Vue.js y MySQL permitió una integración efectiva entre frontend y backend, logrando una experiencia fluida y sencilla para el usuario final. Además, se prestó especial atención a la validación de datos y la gestión de errores, lo que contribuyó a la fiabilidad del sistema.

7.3 Trabajo futuro y mejoras posibles

Si bien el sistema desarrollado es funcional, existen varias áreas en las que se podría continuar trabajando para mejorar y ampliar sus capacidades:

1. **Incluir en el panel de administración** estadísticas y reportes.

2. **Implementar pasarelas de pago reales** para que los pedidos puedan concretarse completamente en línea.

3. **Añadir un chat con inteligencia artificial (IA):** Integrar un sistema de chat basado en IA para mejorar la atención al cliente, ofreciendo respuestas automáticas, soporte 24/7 y recomendaciones personalizadas en tiempo real.

4. **Implementación de un selector de idioma:** Añadir un botón desplegable para cambiar el idioma de la interfaz de usuario, permitiendo que los clientes seleccionen su idioma preferido. Esto mejorará la accesibilidad y el alcance global de la plataforma, facilitando la experiencia para usuarios internacionales .

Apéndice

En este apéndice se incluye información complementaria para el desarrollo del proyecto *Detalles de Patch*.

- **Código fuente completo:**
  El código completo del backend, incluyendo la configuración y las rutas para la gestión de usuarios, productos, pedidos y sistema de recuperación de contraseña, está disponible en el repositorio oficial de GitHub:
  https://github.com/JordiLlopis10/tfg

- **Tecnologías usadas:**

  - **Flask:** Framework web en Python para el desarrollo del backend.
  - **Flask-MySQLdb:** Para la conexión con la base de datos MySQL.
  - **Flask-Login:** Para la gestión de autenticación y sesiones de usuarios.
  - **Flask-CORS:** Para habilitar el acceso desde el frontend en un dominio distinto.
  - **Werkzeug Security:** Para la encriptación y verificación de contraseñas.

- **SMTP:** Para el envío de correos electrónicos, como confirmación de compra y recuperación de contraseña.

➢ **Estructura de la base de datos:**
La base de datos relacional contiene tablas para usuarios, productos, pagos, y tokens para recuperación de contraseña, garantizando la integridad y seguridad de la información.

Bibliografía

Para la realización de este proyecto se consultaron y utilizaron las siguientes fuentes de documentación oficial y recursos técnicos:

- **Documentación de Flask**
  https://flask.palletsprojects.com/en/latest/

- **Documentación de Flask-Login:**
  https://flask-login.readthedocs.io/en/latest/

- **Documentación de Flask-MySQLdb:**
  https://flask-mysqldb.readthedocs.io/en/latest/

- **Documentación de Werkzeug Security (hashing de contraseñas):**
  https://werkzeug.palletsprojects.com/en/latest/security/

- **Documentación de Python para envío de emails (smtplib y email.message):**
  https://docs.python.org/3/library/smtplib.html
  https://docs.python.org/3/library/email.message.html

- **Generación y manejo seguro de tokens en Python (secrets module):**
  https://docs.python.org/3/library/secrets.html

- **API Fetch (JavaScript):**
  Documentación oficial para realizar peticiones HTTP desde el navegador.
  https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API

- **CORS (Cross-Origin Resource Sharing):**
  https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

- **JSON (JavaScript Object Notation):**
  https://www.json.org/json-en.html