Trabajo Dawazon 2.0
2º DAW

Por Samuel Gómez, Carlos Cortés, Victor Marín y Adrián Herrero

Índice

Introducción

Objetivo del proyecto

Este proyecto tiene como fin el desarrollo de una moderna plataforma de tienda en línea, que emplea C# 25 y .NET 10.0, cuyo propósito es brindar un sistema sólido, escalable y sencillo de extender para administrar y comercializar productos. La aplicación ofrece sus funcionalidades a través de una API REST para las operaciones CRUD que se basan en HTTP y una página web dinámica construida con Razor Pages, MVC y algunos componentes Blazor.

El sistema integra el empleo de PostgreSQL, una base de datos relacional que es dinámica y potente, se emplea para gestionar de manera estructurada el catálogo de productos y otros elementos con relaciones definidas. Está preparada para la concurrencia, es decir, para manejar sin errores peticiones simultáneas que tienen el potencial de causar inconsistencias en los datos.

Por último, la aplicación está totalmente lista para ser implementada por medio de Docker, lo que simplifica su distribución, ejecución en entornos aislados y recreación del entorno de desarrollo o producción sin configuraciones manuales complicadas.

La contenerización garantiza que todos los servicios, como ASP.NET y Postgres, y otros posibles servicios de soporte, puedan funcionar de manera consistente.

Alcance

- Alimentar un catálogo de productos que incluya nombre, precio, categoría, descripción e imagen.
- Permitir operaciones CRUD sobre los productos (crear, leer, actualizar, eliminar).
- Validar datos de entrada (por ejemplo: nombre no en blanco, precio mayor que cero, precio obligatorio, categoría obligatoria).
- Exponer la funcionalidad tanto vía REST como vía web dinámica.
- Contenerización mediante Docker para facilitar el despliegue.
- Uso de Postgre como base relacional concurrente.

Tecnologías empleadas

- Lenguaje: C#.
- Framework principal: ASP.NET .
- Bases de datos: Postgre (relacional concurrente).
- API: REST (endpoints HTTP).
- Contenerización: Docker / docker-compose.
- Entorno de desarrollo: JetBrains Rider.
- Otros: (Redis, NUnit, Playwright, Bruno, GH Actions).

Arquitectura del sistema

La arquitectura del sistema se fundamenta en un enfoque que es modular y escalable, el cual fusiona diversas tecnologías y paradigmas con la finalidad de proporcionar un ambiente resiliente, adaptable y de fácil mantenimiento. El proyecto está basado en .NET, que es el núcleo principal, y se apoya en diversos elementos adicionales que funcionan de manera integrada para satisfacer las exigencias técnicas y comerciales.

1. Estructura general del sistema

El sistema sigue una arquitectura cliente–servidor donde:

- El Servidor (Backend)
  - Gestiona la lógica de negocio.
  - Expone los servicios mediante REST y web dinámica.
  - Administra conexiones con la base de datos Postgres.
  - Proporciona una capa uniforme de validaciones, seguridad básica y gestión de datos.

- El Cliente
  - Consume la API REST.
  - Procesa los datos para mostrar el catálogo de productos u otras vistas.
  - Renderiza la web para que sea accesible por los usuarios con una interfaz gráfica.

2. Capas del sistema

La aplicación sigue una arquitectura por capas típica de .NET:

Capa de Controladores (Controllers)

Gestiona las peticiones externas. Aquí se definen:

- Endpoints REST
- Gestión de códigos HTTP
- Endpoints web.

Ejemplo:

- ProductRestController

Capa de Servicios (Services)

Contiene la lógica de negocio. Aquí se encuentran los procesos como:

- Validación adicional de datos
- Transformación avanzada de objetos
- Gestión de transacciones
- Mapeo de DTOs ↔ Entidades
- Cacheo de elementos

Ejemplo:

- ProductoService

Capa de Repositorios (Repositories)

Se encarga de la interacción con la base de datos:

- Postgre → repositorios

Ejemplo:

- ProductoRepository

Capa de Modelo (Entities / Documents / DTOs)

Incluye:

- Entidades EF para Postgres
- DTOs para entrada y salida en REST y web dinámica
- Ejemplo:
    - Producto (EF)
    - POSTandPUTProductoRequestDTO

3. Componentes principales de la arquitectura

API REST

Permite operaciones CRUD tradicionales sobre productos mediante:

- GET /productos
- POST /productos
- PUT /productos/{id}
- DELETE /productos/{id}

Incluye validaciones automáticas con:
- [Range]
- [Required]

Bases de datos

Se usa la base de datos Postgre, ideal para datos estructurados y relaciones simples.

Docker

- El sistema está completamente preparado para contenedores:
    - Un contenedor para .NET
    - Un contenedor para Postgres
    - Un contenedor para Nginx con el reporte de tests generado por NUnit
    - Un contenedor que corre los test de PlayWright y genera el reporte
    - Un contenedor para Nginx con el reporte de tests estándar de PlayWright
    - Un contenedor que corre los test de Bruno y genera el reporte
    - Un contenedor para Nginx con el reporte de Bruno
    - Un contenedor Nginx con el proxy inverso que filtra todas las conexiones a la aplicación y el resto de servicios
    - Un contenedor para la caché Redis
- mediante el docker-compose.yml, que permite orquestar todos los servicios.

4. Flujo general de funcionamiento

- El cliente realiza una petición (REST) al servidor.
- El controlador procesa la entrada y realiza las validaciones iniciales incluídas en los DTO.
- El servicio ejecuta la lógica de negocio y convierte los DTO a los modelos de datos. También interacciona con la caché.
- El repositorio trabaja con los modelos proporcionados por el servicio e interactúa con la base de datos.
- Todo el flujo anteriormente descrito se realiza en la dirección inversa hasta que finalmente el controlador devuelve al cliente la información correspondiente mediante un DTO de salida.

Patrones Utilizados

En el desarrollo de este proyecto se han aplicado varios patrones de diseño y estructuración que facilitan la organización del código, la mantenibilidad del sistema y la escalabilidad de la aplicación. Algunos de los patrones utilizados son los siguientes:

1. Patrón CRUD

El proyecto implementa de forma clara el patrón CRUD (Create, Read, Update, Delete), ya que la aplicación gestiona un catálogo de productos.

Ejemplos en el proyecto:

- Crear productos
- Leer productos
- Actualizar productos
- Eliminar productos

2. Patrón DTO (Data Transfer Object)

Un DTO (Data Transfer Object) es un objeto simple utilizado para transportar datos entre distintas capas o procesos de una aplicación sin contener lógica de negocio, solo propiedades.

Sirve para reducir el acoplamiento, evitar exponer directamente las entidades internas (modelos) y optimizar el intercambio de información.

Ejemplos en el proyecto:

- POSTandPUTProductoRequestDTO
- ProductoResponseDTO

Tecnologías

Este es un listado de todas las tecnologías de las que se ha hecho uso a lo largo de la práctica además del porqué han sido usadas:

- Rider: entorno de desarrollo principal usado para escribir y ejecutar el código. Se ha hecho uso de este entorno de desarrollo debido a que es el que se ha usado a lo largo de todo el curso, en parte por su facilidad a la hora de usar proyectos colaborativos y porque garantiza muy bien la calidad del código.
- Visual Studio Code: entorno de desarrollo secundario para revisar el código de los compañeros. Es el entorno más rápido si lo que se busca es simplemente visualizar código.
- Git: control de versiones para la gestión de cambios en el código. Se ha usado, además de porque es el estándar, ya que permite guardar un historial de cambios muy útil.
- GitHub: ha servido para alojar el repositorio del proyecto y poder gestionar los cambios desde ahí. Se ha usado ya que ofrece una integración directa con Git.
- Windows Terminal: para ejecutar los comandos y gestionar el proyecto desde terminal. Usada debido a que es la predeterminada y cubre todas las necesidades del proyecto.
- Trello: para el reparto de tareas. Se exigió en la anterior práctica y es muy cómodo en cuanto a gestionar qué parte hace cada miembro del equipo.
- Word: para poder redactar la documentación del proyecto. Usado ya que es el estándar en cuanto a procesadores de texto.
- Docker: para crear servicios garantizando su independencia y escalarlos de forma sencilla, mediante contenedores de virtualización ligera.

- Nginx: como servidor web y proxy inverso.
- Redis: como servicio de caché.

Requisitos funcionales y no funcionales

Requisitos Funcionales

RF1	Operaciones CRUD (productos, categorías y pedidos).
RF2	Carrito de compra.
RF3	Id único a cada entidad.
RF4	API REST.
RF5	Interfaz gráfica dinámica (Razor Pages, MVC y Blazor).
RF6	Pago con tarjeta (Stripe).
RF7	Usuarios, seguridad (Identity), JWT y HTTP Session.
RF8	Tarea programada.

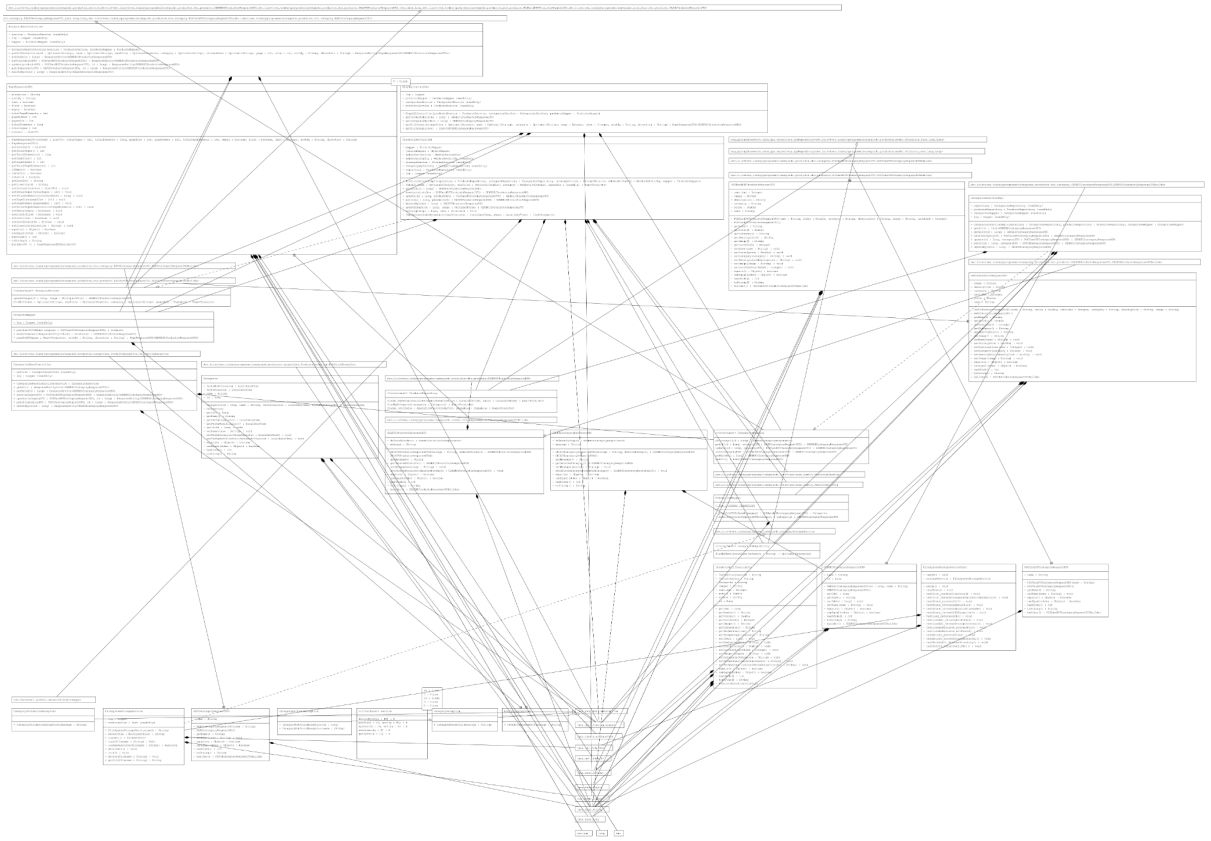
Requisitos No Funcionales

RNF1	Caché Redis.
RNF2	Testeo del código (unitarios + integración + E2E Playwright + Bruno).
RNF3	Docker.
RNF4	GitHub.
RNF5	Proxy inverso con Nginx.

Requisitos de Información

RI1	Modelo + DTOs de usuario.
RI2	Modelo + DTOs de cliente.
RI3	Modelo + DTOs de producto.
RI4	Modelo de categoría.
RI5	Modelo + DTOs de pedido.
RI6	Modelo + DTOs de comentario.
RI7	Modelo + DTOs de carrito.

Diagrama de clases



Principios SOLID aplicados

Principio de responsabilidad única (SRP)

Cada clase en el proyecto tiene una única responsabilidad. Por ejemplo, la clase ProductoService se encarga de la lógica de negocio de lo relativo a productos y categorías y ProductController se encarga de procesar peticiones HTTP.
Esto asegura que cada clase cumpla con su función específica sin mezclarse con otras responsabilidades.

Principio de abierto/cerrado (OCP)

Las clases están diseñadas para ser extendidas sin necesidad de modificar su código original. Por ejemplo, en ProductService, si se desea agregar un nuevo formato de archivo para guardar imágenes de productos, podemos extender la funcionalidad sin alterar el código existente, respetando así este principio.

Principio de sustitución de Liskov (LSP)

Las subclases pueden reemplazar a sus superclases sin afectar el funcionamiento del sistema.

Principio de segregación de interfaces (ISP)

Se prefieren interfaces específicas y enfocadas en lugar de grandes interfaces generales. Por ejemplo, la interfaz ProductoService se centra únicamente en las operaciones CRUD para Producto, mientras que la interfaz IService define operaciones CRUD genéricas para cualquier servicio. Esto evita interfaces infladas y mejora la mantenibilidad.

Principio de inversión de dependencias (DIP)

El proyecto depende de abstracciones en lugar de implementaciones concretas. Por ejemplo, ProductService depende de la interfaz IProductService, permitiendo cambiar las implementaciones sin afectar el código cliente.

Diseño de la base de datos

El sistema utiliza un diseño que incorpora una base de datos relacional habilitada para la fiabilidad en entornos de alta concurrencia . Esta arquitectura permite aprovechar las ventajas de ambos modelos según el tipo de información que maneja la aplicación.

1. Base de Datos Relacional: PostgreSQL

La base de datos Postgre se utiliza para almacenar información estructurada y altamente relacionada. En este proyecto, se emplea principalmente para gestionar la información principal del catálogo de productos, categorías y usuarios.

La entidad sigue una estructura estándar EF:

- PK auto-generada mediante [DatabaseGeneratedTypeValue].
- Validaciones con [Required], [Range].
- Enum mapeado como String para mayor claridad en la BD.

Relación con la API

Esta tabla es consultada y modificada a través de:

- Endpoints REST (/productos)
- Servicios y repositorios basados en EF

API REST

Este proyecto es un backend en C# para gestionar una tienda, incluyendo productos, categorías y pedidos. Proporciona acceso a los datos mediante API REST.

Puntos clave:

- Gestión de la tienda: CRUD de Producto, Categoría y Pedido.

- REST: Endpoints para operaciones estándar como crear, listar, actualizar o eliminar productos y pedidos.

- Arquitectura modular: Separación en capas (modelos, servicios, repositorios), siguiendo principios SOLID para mantener el código limpio y extensible.

- Abstracciones: Interfaces y clases abstractas desacoplan la lógica de negocio de la persistencia, facilitando cambios en la base de datos o en la implementación de los repositorios.

Docker

El proyecto se ejecuta dentro de contenedores Docker, lo que facilita el despliegue, la portabilidad y las pruebas automatizadas.

Puntos clave:

- Aislamiento y portabilidad: Cada componente (backend, base de datos) corre en su contenedor independiente, funcionando igual en cualquier sistema con Docker.
- Configuración simplificada: Uso de Dockerfile y docker-compose.yml para levantar contenedores fácilmente.
- Testcontainers: Se utilizan contenedores temporales para pruebas automatizadas, garantizando que los tests se ejecuten en un entorno limpio y controlado.

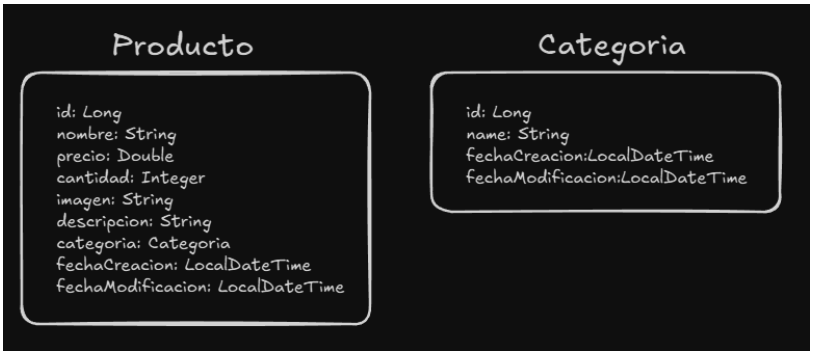
- Despliegue consistente: El proyecto funciona igual en desarrollo, pruebas y producción, evitando conflictos con el entorno local.

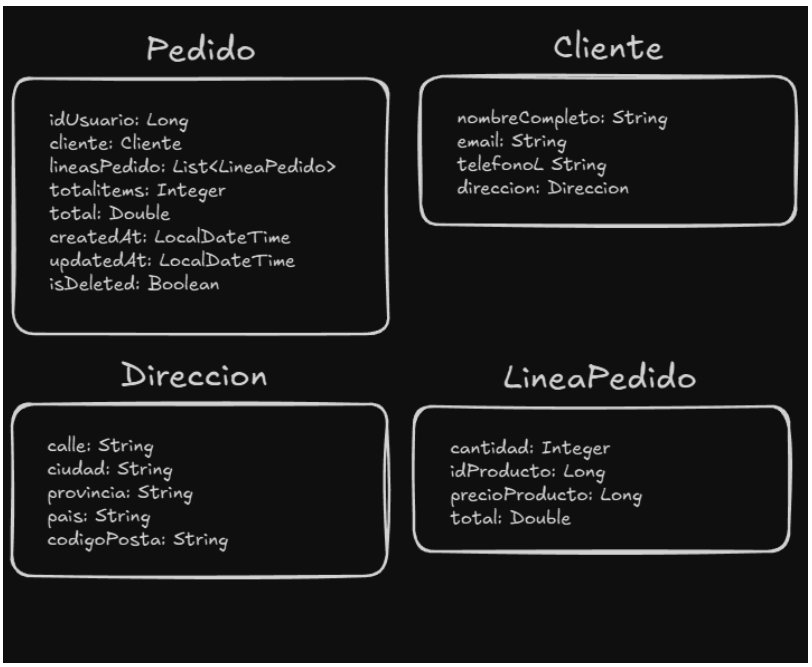
Casos de uso típicos:

- Levantar la aplicación y la base de datos con un solo comando (docker-compose up).
- Ejecutar tests automatizados con bases de datos temporales mediante Testcontainers.
- Escalar o replicar servicios de forma sencilla.

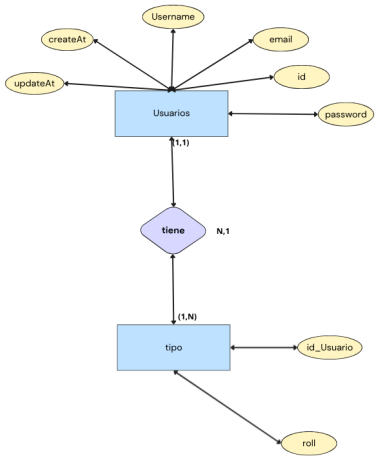
Modelos

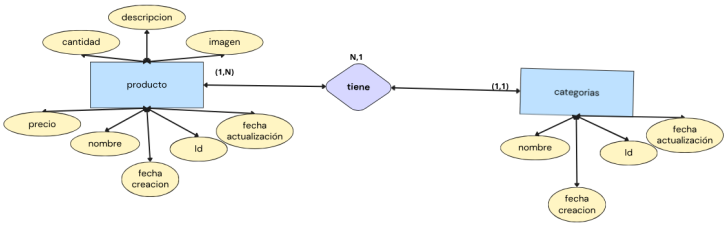
RELACIONAL:





Diagramas entidad-relación





Gitflow


Merge pull request #63 from Aragorn7372/main Víctor Marín Escribano
Merge branch 'G-Corp-YA:main' into main Víctor Marín Escribano
Merge pull request #62 from Aragorn7372/main Víctor Marín Escribano
mas intentos del docs y render Víctor Marín Escribano
Merge branch 'G-Corp-YA:main' into main Víctor Marín Escribano
Merge pull request #61 from Aragorn7372/main Víctor Marín Escribano
arreglo dosigen para que haga los reportes y aparte arreglo el dockerfile render Víctor Marín Escribano
Merge branch 'G-Corp-YA:main' into main Víctor Marín Escribano
Merge pull request #60 from Aragorn7372/main Samuel Gómez Gutiérrez
Merge remote-tracking branch 'origin/main' Víctor Marín Escribano
Merge branch 'G-Corp-YA:main' into main Víctor Marín Escribano
Merge pull request #59 from Aragorn7372/main Samuel Gómez Gutiérrez
arreglo dosigen para que haga los reportes y aparte arreglo el dockerfile render Víctor Marín Escribano
arreglo el dockerfile-render Víctor Marín Escribano
arreglo docs y aparte hago dockerfile-render Víctor Marín Escribano
Merge pull request #58 from G-Corp-YA/dev Víctor Marín Escribano
Merge pull request #57 from AdrianHerSac/dev Víctor Marín Escribano
Test completos AdrianHerSac
Merge pull request #56 from Sggs221/dev Samuel Gómez Gutiérrez
Arreglado html de /pwte y configuro para gh actions Sggs221
Playwright FUNCIONA!!! Sggs221
Merge branch 'G-Corp-YA:dev' into dev Samuel Gómez Gutiérrez
Merge pull request #55 from Aragorn7372/dev Carlos Cortés
quito el deploy original Víctor Marín Escribano
modifico actions Víctor Marín Escribano
mando a la verga el upload para que no me toque las carpetitas y estalle -_- Víctor Marín Escribano
arreglo git actions Víctor Marín Escribano
añado el env de bruno que se me olvido xdd Víctor Marín Escribano
arreglo env Víctor Marín Escribano
añado deploys Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Sggs221
Merge pull request #54 from Aragorn7372/dev Carlos Cortés
cambios en el gitignore Víctor Marín Escribano
despliegue en docker terminado, orgullo sanado Víctor Marín Escribano
añado parte del despliegue Víctor Marín Escribano
añado dockerfile Víctor Marín Escribano
Playwright con video Sggs221
Cambio de pc Samuel Gómez
Merge pull request #53 from charleecy/dev Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Carlos Cortés
Merge pull request #52 from charleecy/dev Samuel Gómez Gutiérrez
Añadido servicio de PDF Carlos Cortés
Merge branch 'G-Corp-YA:dev' into dev Carlos Cortés
Merge pull request #51 from charleecy/dev Samuel Gómez Gutiérrez
arreglamos identity y jwt Carlos Cortés
arreglamos carrito, creamos tarea programada Carlos Cortés
Merge pull request #50 from Sggs221/dev Víctor Marín Escribano
Merge branch 'G-Corp-YA:dev' into dev Samuel Gómez Gutiérrez
Merge pull request #49 from charleecy/dev Carlos Cortés
Segunda tanda de imágenes incluidas Carlos Cortés
Dashboard de stats de administrador Samuel Gómez
Merge pull request #47 from charleecy/dev Carlos Cortés
Imágenes de los productos subidas Carlos Cortés
Merge pull request #46 from Sggs221/dev Víctor Marín Escribano
Componente blazor de notificacion de compra manager Sggs221
Carrito y checkout terminados.Admin puede cambiar roles ahora. Login corregido. Sggs221
Merge pull request #45 from charleecy/dev Víctor Marín Escribano
Cagadita jejej Carlos Cortés
Avances en las vistas de carrito, falta Stripe Carlos Cortés
Merge pull request #43 from Aragorn7372/dev Samuel Gómez Gutiérrez
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #42 from charleecy/dev Víctor Marín Escribano
Manager finiquitao Carlos Cortés
Merge pull request #41 from Sggs221/dev Samuel Gómez Gutiérrez
arreglao Sggs221
Arregladas parte de las imagenes entre otros Sggs221
Merge pull request #40 from charleecy/dev Samuel Gómez Gutiérrez
Avances en las funcionalidades de Admin Carlos Cortés
arreglo el componente blazor Víctor Marín Escribano
Merge pull request #39 from Sggs221/dev Víctor Marín Escribano
Avances en las vistas Sggs221
Merge pull request #38 from charleecy/dev Víctor Marín Escribano
Avances en las vistas de usuario y carrito Carlos Cortés
Merge pull request #37 from Aragorn7372/dev Carlos Cortés
añado favService y modifico users Víctor Marín Escribano
Merge pull request #36 from Sggs221/dev Víctor Marín Escribano
gitignore arreglado Sggs221
Avances en vista general de productos e inicio de sesion Sggs221
Merge pull request #35 from charleecy/dev Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Carlos Cortés
Merge pull request #34 from charleecy/dev Víctor Marín Escribano
Corregidos algunos bugs en la API y en tests Bruno Carlos Cortés
Solucionados varios bugs de la API, se incluyen los test Bruno y readme.md Carlos Cortés
Merge pull request #32 from Aragorn7372/dev Carlos Cortés
añado documentacion Víctor Marín Escribano
añado configuracion de sesion y cookie de sesion Víctor Marín Escribano
Merge pull request #31 from charleecy/dev Samuel Gómez Gutiérrez
Cositasjoja Carlos Cortés
Cart controller implementado, bendisiones Carlos Cortés
Merge pull request #30 from Aragorn7372/dev Carlos Cortés
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #29 from Sggs221/dev Víctor Marín Escribano
Creado controller de usuarios Sggs221
Merge pull request #28 from Aragorn7372/dev Samuel Gómez Gutiérrez
arreglo permisos y roles en products controller Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #27 from Aragorn7372/dev Carlos Cortés
añado el controller de producto Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #26 from AdrianHerSac/dev Samuel Gómez Gutiérrez
Plantillas con las vistas de Layout, Index, Productos y estilos css AdrianHerSac
termino infraestructuras, cacheservice generica y arreglo la cache de productService Víctor Marín Escribano
Merge pull request #25 from charleecy/dev Víctor Marín Escribano
Stripe Carlos Cortés
Creado servicio de Stripe, inyectado en el CartService Carlos Cortés
Merge pull request #24 from Aragorn7372/dev Samuel Gómez Gutiérrez
añado el servicio de userService Víctor Marín Escribano
Merge pull request #23 from charleecy/dev Víctor Marín Escribano
Inyectado repo de usuarios en el servicio de carrito, corregidas algunas funciones Carlos Cortés
Add job names to GitHub Actions workflow Víctor Marín Escribano
Merge pull request #22 from Aragorn7372/dev Víctor Marín Escribano
Merge branch 'G-Corp-YA:dev' into dev Víctor Marín Escribano
Merge pull request #21 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #20 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #19 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #18 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #17 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #16 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #15 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #14 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #13 from Aragorn7372/dev Víctor Marín Escribano
Merge pull request #12 from Aragorn7372/dev Víctor Marín Escribano
espero qeu sea la ultima Víctor Marín Escribano
soy imbecil quite sin querer la configuracion Víctor Marín Escribano
espero terminar con esto Víctor Marín Escribano
termino de una vez el workflow Víctor Marín Escribano
añado librerias de logger para poder hacer reporte de test Víctor Marín Escribano
modifico y arreglo el deploy Víctor Marín Escribano
modifico el action Víctor Marín Escribano
cambio version libreria Víctor Marín Escribano
añado workflows y cambio de version las librerias Víctor Marín Escribano
añado workflows y cambio de version las librerias Víctor Marín Escribano
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #11 from charleecy/dev Carlos Cortés
Creado Mapper de Carrito e implementado en el servicio para devolver DTOs Carlos Cortés
Merge remote-tracking branch 'origin/dev' into dev Víctor Marín Escribano
Merge pull request #10 from Sggs221/dev Samuel Gómez Gutiérrez
Identity implementado Samuel Gómez
añado workflows y cambio de version las librerias Víctor Marín Escribano
cambio versiones y añado actions Víctor Marín Escribano
Merge pull request #9 from charleecy/dev Carlos Cortés
Avances en el servicio de carrito, a la espera de Stripe y Seguridad Carlos Cortés
Merge pull request #8 from charleecy/dev Carlos Cortés
Servicio de email implementado Carlos Cortés
Merge pull request #7 from Sggs221/dev Samuel Gómez Gutiérrez
Servicio de productos + Mapper Sggs221
Merge pull request #6 from charleecy/dev Carlos Cortés
Storage implementado Carlos Cortés
Repositorio de carrito finalizado Carlos Cortés
Merge pull request #5 from Aragorn7372/dev Víctor Marín Escribano
casi termino el repositorio de carrito Víctor Marín Escribano
Merge pull request #4 from Sggs221/dev Carlos Cortés
CartRepository + algunos modelos Sggs221
Merge pull request #3 from charleecy/dev Samuel Gómez Gutiérrez
cambios en el modelo cart Carlos Cortés
Avances en el carrito Carlos Cortés
Merge pull request #2 from Aragorn7372/dev Víctor Marín Escribano
añado script de librerias Víctor Marín Escribano
añado repositorios modelos y algunas cosas comunes Víctor Marín Escribano
Merge pull request #1 from Aragorn7372/main Víctor Marín Escribano
añado proyectos Víctor Marín Escribano
Initial commit Víctor Marín Escribano

Estimación de costes

HORAS DE TRABAJO ESTIMADAS		
Requisitos funcionales		
RF	**Nombre**	**Horas estimadas**
RF1	Operaciones CRUD (productos, categorías y pedidos).	40
RF2	Carrito de compra.	40
RF3	Id único a cada entidad.	1
RF4	API REST.	10
RF5	Interfaz gráfica dinámica (Razor Pages, MVC y Blazor).	120
RF6	Pago con tarjeta (Stripe).	5
RF7	Usuarios, seguridad (Identity), JWT y HTTP Session.	40
RF8	Tarea programada.	3
		259
Requisitos no funcionales		
RNF	**Nombre**	**Horas estimadas**
RNF1	Caché Redis.	5
RNF2	Testeo del código (unitarios + integración + E2E Playwrigt + Bruno).	40
RNF3	Docker.	20
RNF4	GitHub.	5
RNF5	Proxy inverso con Nginx.	2
		72

Requisitos de información

RI	Nombre	Horas estimadas
RI1	Modelo + DTOs de usuario.	1
RI2	Modelo + DTOs de cliente.	1
RI3	Modelo + DTOs de producto.	1
RI4	Modelo de categoría.	1
RI5	Modelo + DTOs de pedido.	1
RI6	Modelo + DTOs de comentario.	1
RI7	Modelo + DTOs de carrito.	1
		7

HORAS DE TRABAJO TOTALES
338

PRECIO POR HORA DE TRABAJO
135,00 €

PRECIO POR HORA EXTRA DE TRABAJO
160,00 €

COSTE TOTAL DE LAS HORAS DE TRABAJO
45.630,00 €

ESTIMACIÓN DE COSTES				
Nº	Concepto	Desglose		Coste
		Precio por hora	Horas	
1	Horas de trabajo estimadas por requisitos	135,00 €	338	45.630,00 €
		Precio por hora extra	15% de las horas totales	
2	Horas extra para cubrir imprevistos	160,00 €	50,70	8.112,00 €
		Costes antes de margen	17% sobre los costes	
3	Margen de beneficios	53.742,00 €	0,17	9.136,14 €
				62.878,14 €

Conclusiones

El proyecto demuestra una arquitectura modular y mantenible, siguiendo los principios SOLID, lo que facilita la extensión y el mantenimiento del código.

La utilización de Docker y Test Containers asegura portabilidad, consistencia en el despliegue y entornos controlados para pruebas automatizadas.

En conjunto, estas tecnologías permiten un sistema de tienda robusto, escalable y fácil de probar, preparado tanto para desarrollo como para producción.