Crafty

Mayra A. Mosquera Gironza

C.F.G.S. Desarrollo de Aplicaciones Web

Curso 22 - 23

Periodo marzo/junio

Instituto

Centre Específic d'Educació a Distància de la Comunidad Valenciana

Tutor

Guillermo Garrido Portes

Crafty

Resumen.

El Proyecto consiste en una plataforma web intermediaria entre consumidores finales y vendedores de productos artesanales hechos a mano y/o artesanales industriales. He podido desarrollar una tienda online funcional que permite registrar usuarios como compradores y permite que estos se registren posteriormente como vendedores. El proyecto utiliza tecnologías como React y Laravel, desplegadas en contenedores Docker. Se han aprovechado herramientas adicionales como Bootstrap, Formik y Sass para mejorar la experiencia de usuario y el diseño de la interfaz. A lo largo del desarrollo, se ha experimentado un aprendizaje constante y se han superado desafíos técnicos, logrando una web completa y cumpliendo en gran medida los objetivos planteados.

Crafty

Resumen.

El Projecte consistix en una plataforma web intermediària entre consumidors finals i venedors de productes artesanals fets a mà i/o artesanals industrials. He pogut desenvolupar una botiga en línia funcional que permet registrar usuaris com a compradors i permet que aquests es registren posteriorment com a venedors. El projecte utilitza tecnologies com React i Laravel, desplegades en contenidors Docker. S'han aprofitat eines addicionals com Bootstrap, Formik i Sass per millorar l'experiència de l'usuari i el disseny de la interfície. Al llarg del desenvolupament, s'ha experimentat un aprenentatge constant i s'han superat desafiaments tècnics, aconseguint una web completa i complint en gran mesura els objectius plantejats.

Crafty

Summary

Can you generate a version of this text in English and Valencian? The Project consists of a web platform that acts as an intermediary between end consumers and sellers of handmade and/or industrially crafted products. I have been able to develop a functional online store that allows users to register as buyers and later register as sellers. The project utilizes technologies such as React and Laravel, deployed in Docker containers. Additional tools like Bootstrap, Formik, and Sass have been utilized to enhance the user experience and interface design. Throughout the development process, there has been continuous learning and overcoming of technical challenges, resulting in a comprehensive website that largely fulfills the set objectives.

Contenido

I. Introducción.

I.1 Presentación del proyecto.

El proyecto consiste en el desarrollo de una plataforma web intermediaria entre consumidores finales y vendedores de productos artesanales industriales, hechos a mano y de forma sostenible.

La tienda se denomina Crafty y es una construcción extraída del idioma inglés. Proviene del sustantivo Craft que puede significar ‘el arte’, ‘la artesanía’, ‘el oficio’ y se refiere a la habilidad manual de crear algo. En adelante me referiré al proyecto como **‘Crafty’** o el **‘Proyecto’**.

El Proyecto es en esencia una tienda online que permite a los usuarios comprar y vender productos directamente a los clientes. La idea de la que se partía era ser una simple plataforma intermediaria que permitiese el contacto entre estos dos perfiles en el mercado pero conforme avanzaba el proyecto fue haciéndose evidente la necesidad de asegurar y comprobar que tanto los productos que se ponen en venta como los vendedores cumplen un mínimo de requisitos y garantías indispensables para poder formar parte de la plataforma que acredite la producción artesanal y sostenible (forma de producción, cumplimiento de leyes laborales y ambientales, etc).

Actualmente no es posible realizar estas comprobaciones a través de la web por lo que la idea es realizar este paso de forma externa y que los usuarios que solicitan el alta como vendedores deberían haber sido previamente admitidos, así como los productos que publican.

Crafty permite el registro de cualquier usuario que lo solicite como comprador y una vez dado de alta, según lo explicado en párrafos anteriores, se le permitirá darse de alta como vendedor.

I.2 Justificación y contexto en el que se desarrolla el Proyecto Integrado.

Crafty pretende ser no sólo una web de compra y venta de productos artesanales sino también un espacio en el que pudiese crearse una comunidad de usuarios preocupados por el medio ambiente y por el uso sostenible de los recursos naturales.

Para el desarrollo e implementación de este Proyecto he utilizado la mayoría de las herramientas vistas durante el actual curso 22 - 23 del Centre Específic d'Educació a Distància de la Comunidad Valenciana (en adelante, el "**CEED**"), entre ellos:

- Laravel, framework de PHP,
- React, biblioteca de Javascript,
- Bootstrap, framework de diseño,
- MySQL, sistema de gestión de base de datos relacionales MySQL,
- Sass, preprocesador de CSS,
- Gulp, el sistema de automatización de tareas
- Docker, plataforma de contenedores para ejecutar aplicaciones en entornos aislados y reproducibles.

I.3 Objetivos del Proyecto Integrado.

Los objetivos del Proyecto al proponerlo eran crear con las herramientas vistas en clase, tanto con habilidades adquiridas durante el C.F.G.S. Desarrollo de Aplicaciones Web (en adelante, el "**Curso**") como aprendiendo otras nuevas, una web online que permite realizar compras y ventas a los usuarios.

Los objetivos del Proyecto son:

- Permitir el registro de usuarios por defecto como compradores y poder darse de alta como vendedores. Una vez registrado un usuario puede solicitar su alta como vendedor dentro de la propia web, en una página preparada para ello.

- La idea era que los artesanos tuvieran que estar registrados en la web como usuarios para poder anunciar y vender sus productos. El registro de los artesanos en la plataforma tenía que pasar dos tipos de filtros. El primero estaría automatizado y el segundo sería una comprobación de datos fiscales, mercantiles

y laborales, según cada caso, que sería realizada por un ser humano. Como ya he indicado, este segundo filtro se pretende realizar por otros medios externos a los de la web.

- Crafty tenía como objetivo dividir los productos en venta por categorías habilitadas.
- Crafty tenía como objetivo tener áreas de usuario personal tanto para los vendedores como para los compradores.

El área de perfil se está dividida en tres áreas:

- **Información de usuario**: en esta sección se podrá visualizar la información general de los usuarios registrados y ver las direcciones. Se podrán modificar estos datos.
- **Área de compras**: En esta zona debería mostrarse una sección con los productos adquiridos por el usuario con la información del estado de los pedidos. Siempre que el pedido no se encuentre en estado de completado se puede cancelar.
- **Área de Venta**: Esta zona estará habilitada con un formulario para darse de alta como comprador y en caso de estar ya dado de alta se verán los productos en venta, añadir nuevos, ver las ventas realizadas y el método de cobro.
- **Gestión de ventas**: Los vendedores pueden actualizar cualquier campo de los productos y eliminarlos.
- Crafty tenía como objetivo contar con una pasarela de pagos integrada y segura con la que los compradores puedan gestionar sus compras.

II. Estado del Arte.

En los últimos años el uso de las plataformas online por parte de las empresas y emprendedores ha incrementado significativamente, transformando no solo nuestros hábitos de compra y venta sino también la forma de interactuar entre vendedores y consumidores. Cada vez es más frecuente no solo el comercio online como un área más de las empresas, sino que se opte sólo por esta vía como único medio de venta.

Fuente: CNMC

[1]

Hoy en día, los vendedores pueden conocer los hábitos de compra de sus consumidores, sus preferencias, gustos e incluso predecir sus necesidades. Asimismo, los compradores tienen al alcance de su mano el acceso a toda clase de productos, la información sobre los mismos y poseen el poder y la capacidad de compartir sus experiencias con dichos productos y los vendedores. Un solo comprador puede influir positiva o negativamente en las decisiones de otros a la hora de realizar una compra.

[1] Comisión Nacional de los Mercados y la Competencia. (2022). Evolución trimestral del volumen de negocio de los comercios electrónicos Recuperado de https://www.cnmc.es/prensa/ecommerce-i-trimestre-2022-cnmc-20221007

Como mencioné anteriormente, la forma de interacción entre consumidores y vendedores ha experimentado cambios significativos gracias al uso de tecnologías que permiten una conexión más cercana entre ambas partes. Las empresas y emprendedores buscan establecer una conexión casi personal con sus consumidores, a través de las redes sociales y plataformas de comercio que facilitan estas interacciones.

Al mismo tiempo, es notable el aumento en la conciencia social sobre el medio ambiente y los recursos naturales en los últimos años. Esto se ha convertido en un factor importante para algunos consumidores al momento de realizar sus compras, ya que buscan alternativas al *fast fashion* y a productos que contienen ingredientes perjudiciales tanto para el medio ambiente como para las personas.

De esta manera, surge la idea y la necesidad de crear una plataforma intermediaria que permita a compradores y vendedores alineados con intereses y preocupaciones similares conectar, vender, comprar e interactuar entre sí. Además, existe una creciente necesidad de proteger y preservar el arte, la confección y la creación de obras artesanales, así como los métodos de elaboración utilizados.

En el estado actual de la tecnología, se identifica una oportunidad para aprovechar estas tendencias y ofrecer una alternativa atractiva a los consumidores que buscan productos sustentables. Crafty se posiciona como una solución a la creciente demanda de productos sostenibles, brindando un espacio en línea que fomenta los contactos directos entre vendedores y compradores, así como la interacción y el intercambio de conocimientos en torno a la sustentabilidad y la protección ambiental.

III. Estudio de viabilidad. Método DAFO.

El estudio de viabilidad se puede encontrar en el Anexo I. En este se exponen las debilidades, amenazas, fortalezas y oportunidades de Crafty haciendo uso de las herramientas DAFO que la web “https://dafo.ipyme.org” dispone de forma gratuita y online. El archivo con el análisis DAFO de Crafty puede verse en el **Anexo I** de esta memoria.

IV. Análisis de requisitos.

El análisis de los requisitos funcionales y no funcionales en el desarrollo de software es fundamental, siendo la falta de estos o su inexactitud el origen de muchos de los problemas en la ingeniería de software.

Así pues, realizar este análisis es esencial en los proyectos de desarrollo de software para el funcionamiento de la plataforma, para identificar las necesidades de diseño, para su implementación y realizar las pruebas necesarias.

Descripción de requisitos

A continuación, expondré los requisitos funcionales y no funcionales identificados durante el desarrollo de la plataforma Crafty.

IV.1.1 **Requisitos funcionales:**

IV.1.1.1 Registro de usuarios:

El sistema debe permitir el registro de usuarios.

IV.1.1.2 Alta de vendedores:

El sistema debe permitir el alta de los usuarios como vendedores una vez que estén registrados, no antes.

IV.1.1.3 Gestión de datos y perfiles:

El sistema debe permitir a los usuarios gestionar sus datos personales y de contacto en su área de perfil. El sistema debe habilitar un área específica dentro de la zona de perfil para gestionar las compras, y ventas en caso de que el usuario esté dado de alta como vendedor.

IV.1.1.4 Inicio de sesión:

El sistema debe permitir a los usuarios iniciar sesión en la plataforma con el correo electrónico de registro y su contraseña.

IV.1.1.5 Gestión de productos:

El sistema debe permitir a los vendedores gestionar sus productos desde su área de vendedor. Deben poder visualizar sus productos, agregar nuevos, editarlos y eliminarlos.

IV.1.1.6 Carrito de la compra:

El sistema deberá permitir a los usuarios agregar productos al carrito de la compra, eliminarlos, visualizar la cantidad de productos y el precio total, así como completar la compra.

IV.1.1.7 Pasarela de pago:

El sistema debe proporcionar una pasarela de pago segura para que los compradores realicen sus compras de forma segura.

IV.1.1.8 Gestión de pedidos:

El sistema debe permitir a los usuarios gestionar sus pedidos y actualizar su estado

IV.1.2 **Requisitos no funcionales:**

IV.1.2.1 Seguridad:

El sistema debe garantizar la seguridad de los datos de los usuarios adoptando medidas de protección contra amenazas cibernéticas.

IV.1.2.2 Rendimiento:

El sistema debe ser capaz de gestionar una gran cantidad de usuarios y transacciones de la web, de forma rápida y fluida.

IV.1.2.3 Usabilidad:

La interfaz de usuario debe ser intuitiva y fácil de usar en todos los dispositivos para los que haya sido pensada. Por eso la interfaz es sencilla y cómoda, siguiendo patrones y diseños recurrentes actualmente en el mercado.

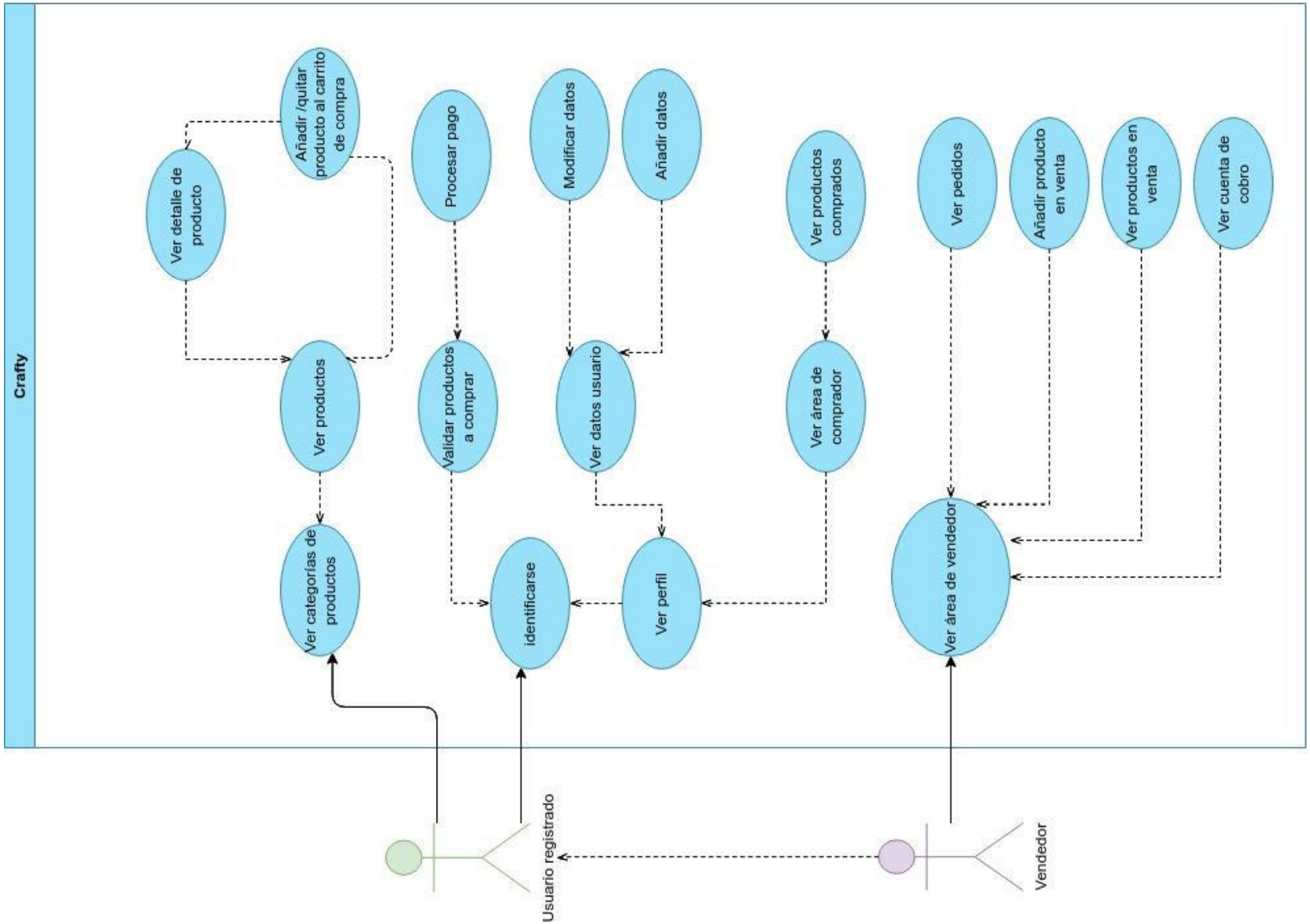
IV.1.2.4 Mantenibilidad:

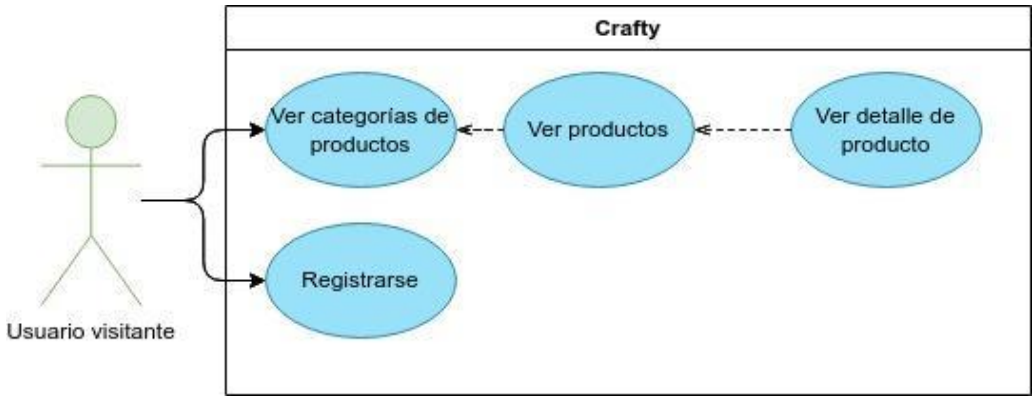
La plataforma debe estar diseñada para ser fácil de mantener y actualizar. La elección de frameworks como Laravel y React permiten que este mantenimiento sea más fácil, sencillo y ágil.

IV.2 Diagramas de caso de uso

Diagrama de caso de uso general para usuarios logueados: los usuarios logueados pueden ver además del área pública de la tienda, su área de perfil. Pueden modificar sus datos y revisar sus compras. Los usuarios logueados que además sean vendedores pueden ver sus productos, añadir productos, gestionar los pedidos, modificar datos y eliminar productos.

Diagrama de caso de uso general para usuario no logueados: los usuarios visitantes pueden ver la página de inicio de la tienda, ver los productos y los detalles de los productos y pueden registrarse.

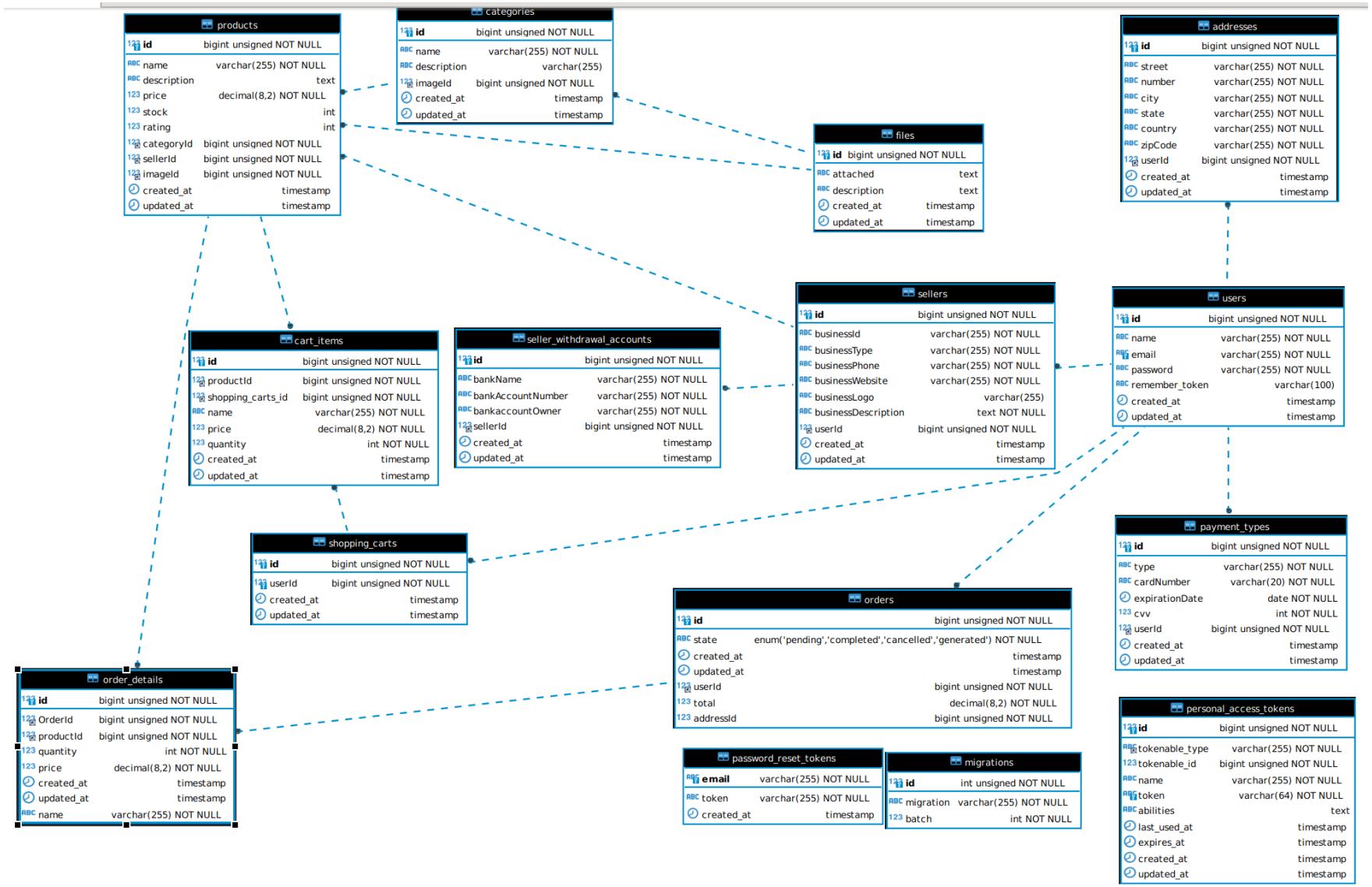

Crafty
Añadir /quitar producto al carrito de compra
Ver detalle de producto
Procesar pago
Modificar datos
Añadir datos
Ver productos comprados
Ver pedidos
Añadir producto en venta
Ver productos en venta
Ver cuenta de cobro
Ver productos
Validar productos a comprar
Ver datos usuario
Ver área de comprador
Ver categorías de productos
identificarse
Ver perfil
Ver área de vendedor
Usuario registrado
Vendedor



V. Diseño.

V.1 Diseño conceptual Entidad Relación

Crafty


products
id bigint unsigned NOT NULL
name varchar(255) NOT NULL
description text
price decimal(8,2) NOT NULL
stock int
rating int
categoryId bigint unsigned NOT NULL
sellerId bigint unsigned NOT NULL
imageId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
categories
id bigint unsigned NOT NULL
name varchar(255) NOT NULL
description varchar(255)
imageId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
files
id bigint unsigned NOT NULL
attached text
description text
created_at timestamp
updated_at timestamp
addresses
id bigint unsigned NOT NULL
street varchar(255) NOT NULL
number varchar(255) NOT NULL
city varchar(255) NOT NULL
state varchar(255) NOT NULL
country varchar(255) NOT NULL
zipCode varchar(255) NOT NULL
userId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
sellers
id bigint unsigned NOT NULL
businessId varchar(255) NOT NULL
businessType varchar(255) NOT NULL
businessPhone varchar(255) NOT NULL
businessWebsite varchar(255) NOT NULL
businessLogo varchar(255)
businessDescription text NOT NULL
userId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
users
id bigint unsigned NOT NULL
name varchar(255) NOT NULL
email varchar(255) NOT NULL
password varchar(255) NOT NULL
remember_token varchar(100)
created_at timestamp
updated_at timestamp
cart_items
id bigint unsigned NOT NULL
productId bigint unsigned NOT NULL
shopping_carts_id bigint unsigned NOT NULL
name varchar(255) NOT NULL
price decimal(8,2) NOT NULL
quantity int NOT NULL
created_at timestamp
updated_at timestamp
seller_withdrawal_accounts
id bigint unsigned NOT NULL
bankName varchar(255) NOT NULL
bankAccountNumber varchar(255) NOT NULL
bankaccountOwner varchar(255) NOT NULL
sellerId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
shopping_carts
id bigint unsigned NOT NULL
userId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
payment_types
id bigint unsigned NOT NULL
type varchar(255) NOT NULL
cardNumber varchar(20) NOT NULL
expirationDate date NOT NULL
cvv int NOT NULL
userId bigint unsigned NOT NULL
created_at timestamp
updated_at timestamp
orders
id bigint unsigned NOT NULL
state enum('pending','completed','cancelled','generated') NOT NULL
created_at timestamp
updated_at timestamp
userId bigint unsigned NOT NULL
total decimal(8,2) NOT NULL
addressId bigint unsigned NOT NULL
order_details
id bigint unsigned NOT NULL
OrderId bigint unsigned NOT NULL
productId bigint unsigned NOT NULL
quantity int NOT NULL
price decimal(8,2) NOT NULL
created_at timestamp
updated_at timestamp
name varchar(255) NOT NULL
personal_access_tokens
id bigint unsigned NOT NULL
tokenable_type varchar(255) NOT NULL
tokenable_id bigint unsigned NOT NULL
name varchar(255) NOT NULL
token varchar(64) NOT NULL
abilities text
last_used_at timestamp
expires_at timestamp
created_at timestamp
updated_at timestamp
password_reset_tokens
email varchar(255) NOT NULL
token varchar(255) NOT NULL
created_at timestamp
migrations
id int unsigned NOT NULL
migration varchar(255) NOT NULL
batch int NOT NULL

V.2 Diseño Físico o Diagrama Mysql

V.2.1 **Eloquent**

La base de datos se llama Crafty al igual que el Proyecto e hice uso del ORM (Object-Relational Mapping) Eloquent incorporado en Laravel para crear y gestionar de forma sencilla la base de datos. A continuación, explicaré brevemente cómo cree las tablas y sus relaciones con Eloquen, pondré un ejemplo de la creación de una tabla y un modelo y luego añadiré capturas de la creación de toda la base de datos con MySQL.

Eloquent proporciona clases, métodos, herramientas y funcionalidades que permiten interactuar con la base de datos desde el proyecto de forma ágil, sencilla y directa, ahorrando y simplificando el trabajo del desarrollador. Dentro del proyecto con el comando “php artisan” podremos consultar todas las funcionalidades de eloquent y obtener una breve descripción o explicación de lo que son.

Cada modelo en Eloquent es una clase que representa una tabla de la base de datos. Junto con los modelos también son de gran importancia las migraciones en Eloquent, que se utilizan para crear y modificar las tablas, así como sus columnas de manera controlada y fácil.

Además, Eloquent genera una tabla en la base de datos denominada “migrations” que se compone de tres campos “id, migration (nombre de los archivos en los que se crean las tablas en el proyecto) y batch (número de lote que indica el orden de ejecución de las migraciones)”. En esta tabla se registran las migraciones que se ejecutan en el proyecto.

V.2.2 MySQL

Sin perjuicio de lo expuesto en el apartado anterior, MySQL sigue siendo el sistema de gestión de base de datos de mi Proyecto, por lo que a continuación presentaré el diseño físico Mysql de mi base de datos. Para no perjudicar la extensión de la memoria ni la redundancia en la explicación de algunas tablas me limitaré a exponer solo alguna de ellas.

Sin contar la tabla “migraciones” la base de datos Crafty cuenta con un total de trece tablas. En la carpeta “migrations” del Proyecto se pueden apreciar más migraciones que tablas totales, esto sucede porque se crean migraciones para modificar o eliminar campos o tablas.

Algunas cuestiones generales a tener en cuenta de las tablas son las siguientes:

- Todas las claves primarias se denominan “id” y son de tipo bigInt unsigned y se generan de forma automática y de forma incremental para cada nuevo registro en las tablas.
- Todas las tablas cuentan con el campo “created_at” y el campo “updated_at” y son de tipo timestamp y registran la fecha de creación y modificación de cada registro en las tablas

Tabla users

```
-- crafty.users definition

CREATE TABLE `users` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `name` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `email` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `password` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `remember_token` varchar(100) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `users_email_unique` (`email`)
) ENGINE=InnoDB AUTO_INCREMENT=11 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- Como he venido explicado los usuarios registrados reciben automáticamente el rol de compradores y al no exigir ningún requisito diferente para poder obtener este rol. Por lo que todos los usuarios tienen automáticamente el rol de comprador.

- Todos los campos de la tabla son requeridos y el campo email debe ser único en toda la tabla. Por lo que no pueden existir dos usuarios registrados con el mismo correo electrónico.

- El campo “remember_token” guardará los tokens que se generan cuando un usuario inicie sesión.

Tabla Sellers

```
-- crafty.sellers definition

CREATE TABLE `sellers` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `bussnessId` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `bussnessType` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `bussnessPhone` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `bussnessWebsite` varchar(255) COLLATE utf8mb4_unicode_ci NOT NULL,
  `bussnessLogo` varchar(255) COLLATE utf8mb4_unicode_ci DEFAULT NULL,
  `bussnessDescription` text COLLATE utf8mb4_unicode_ci NOT NULL,
  `userId` bigint unsigned NOT NULL,
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  PRIMARY KEY (`id`),
  KEY `sellers_userid_foreign` (`userId`),
  CONSTRAINT `sellers_userid_foreign` FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- La tabla Sellers contiene los registros de los vendedores de Crafty. Como adelantaba, todos los usuarios son compradores y si solicita el alta pueden obtener el rol del vendedor. En este caso, la tabla sellers tiene una relación de “pertenencia” con la tabla User.

Tabla Products

- Esta tabla contiene los registros de los productos que pueden venderse en Crafty. Además de los campos habituales podemos ver un campo rating y un campo stock. Cuando un vendedor añade un producto nuevo a la Crafty debe indicar el stock de cada producto y puede modificarlo en cualquier momento. Sobre el campo rating actualmente no puede modificarse ni añadir rating a los productos nuevos, pero es un desarrollo en la lista de pendientes y mejoras.

- Esta tabla tiene definidos tres tipos de relaciones. Por un lado, tiene como clave ajena el id del vendedor que no puede ser nulo y nos indicará a qué vendedor pertenece, el

id de categoría para relacionarlo con alguna de las categorías disponibles en la tienda y el id de imagen que lo relaciona con la imagen almacenada en la tabla files, donde se almacena.

Tabla Orders

```
-- crafty.orders definition

CREATE TABLE `orders` (
  `id` bigint unsigned NOT NULL AUTO_INCREMENT,
  `state` enum('pending','completed','cancelled','generated') COLLATE utf8mb4_unicode_ci NOT NULL DEFAULT 'generated',
  `created_at` timestamp NULL DEFAULT NULL,
  `updated_at` timestamp NULL DEFAULT NULL,
  `userId` bigint unsigned NOT NULL,
  `total` decimal(8,2) NOT NULL,
  `addressId` bigint unsigned NOT NULL,
  PRIMARY KEY (`id`),
  KEY `orders_userid_foreign` (`userId`),
  CONSTRAINT `orders_userid_foreign` FOREIGN KEY (`userId`) REFERENCES `users` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=64 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_unicode_ci;
```

- Para poner en marcha la tienda decidí mantener solo el registro de un tipo de pedido. La tabla Orders registra los pedidos de compra y estará asociado al usuario que la realice. En un principio había estructurado la tabla para que con un campo de tipo enum se pudiese elegir entre el registro “seller” o el registro “buyer” y pudiese usarse para registrar tanto las órdenes de compra como las de venta, pero tuve que modificarlo para permitir que un usuario pudiese comprar varios productos en un único proceso sin tener que hacerlo por productos que pertenecieran al mismo vendedor.

- Actualmente para poder mostrar a los vendedores las órdenes de compra de sus productos, se accede a ellas a través de las órdenes de detalle en la que podemos ver los id de los productos y relacionarlos con los vendedores.

- Por otro lado, nos encontramos con un campo de tipo enum para registrar los estados del pedido. Una orden se crea por defecto con el estado de generada y tras confirmar el pago del pedido pasa al estado de pendiente, pero esto es algo que explicaré más adelante en el apartado de controladores.

- Por su parte en la tabla orders_details nos encontramos con los campos necesarios para identificar el producto a adquirir, su nombre, la cantidad y el precio total.

Tabla ShoppingCart

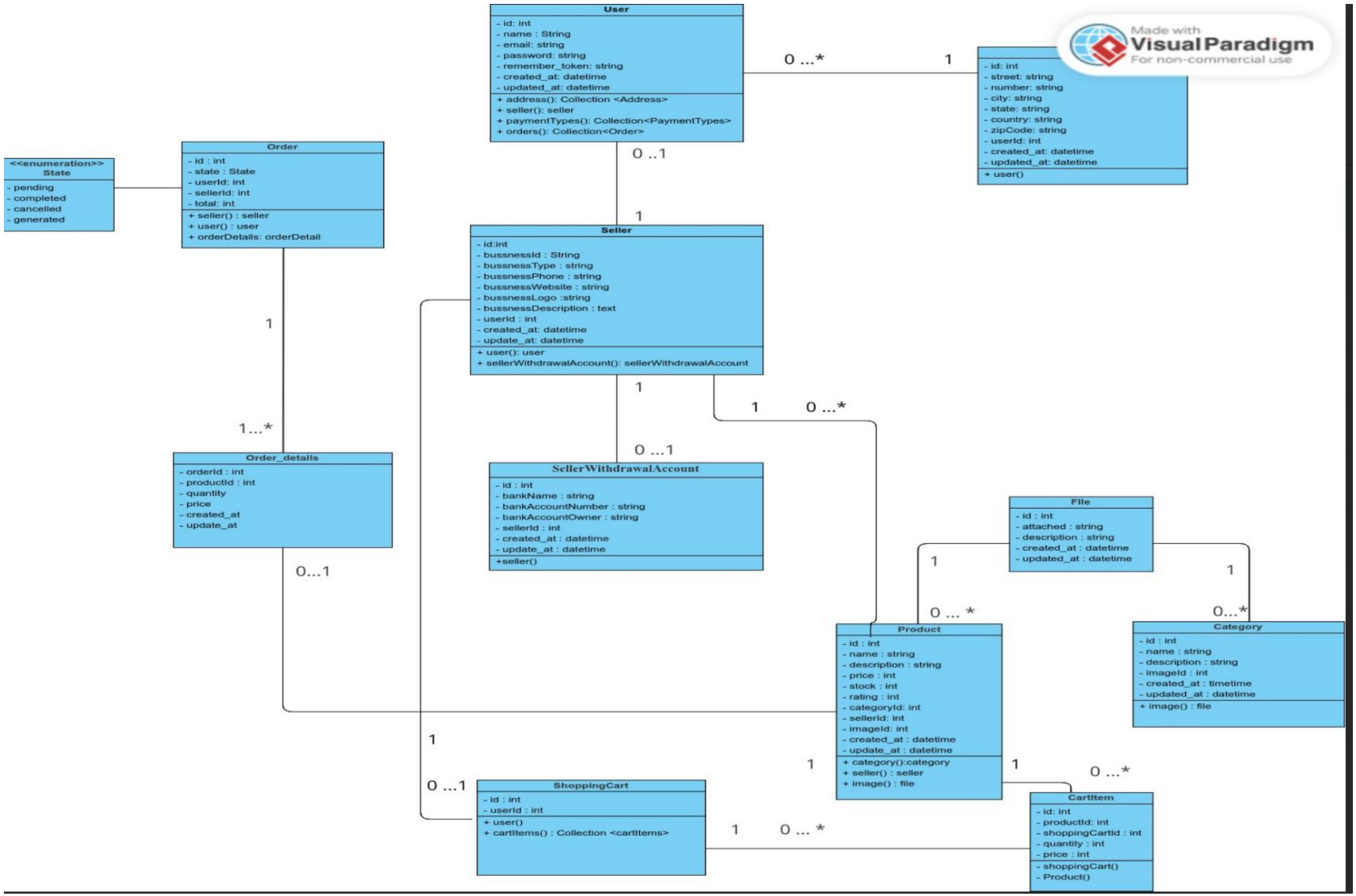
- Los registros en esta tabla se generan automáticamente cuando un usuario solicita añadir un producto al carrito y no se elimina ni se modifica mientras el usuario esté dado de alta. El que sufre las modificaciones es la tabla de cart_Items donde es posible eliminar un registro completo en caso de tener a cero la cantidad de productos o porque se haya completado el pago. También se puede modificar para añadir o quitar la cantidad del producto.

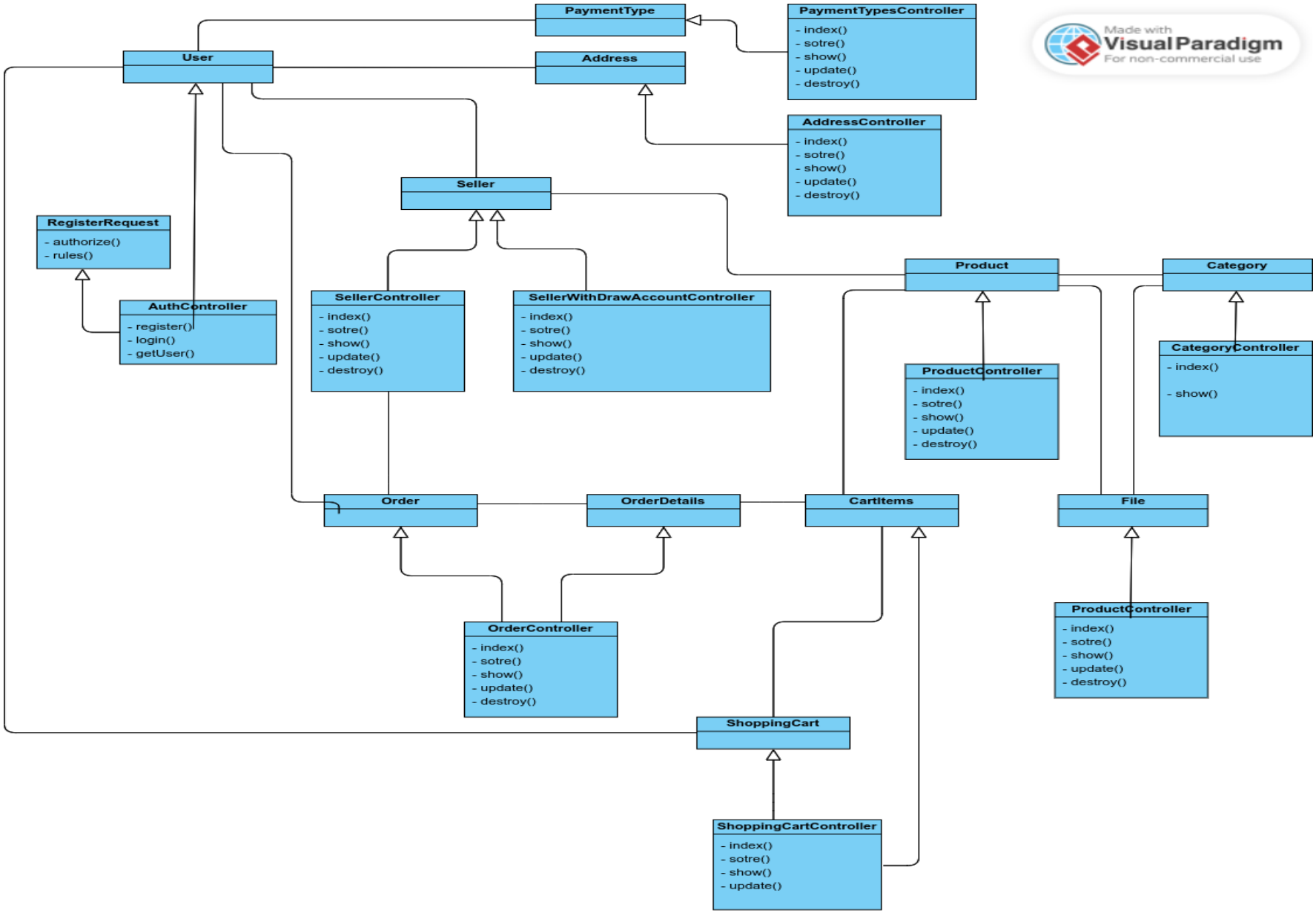
V.2.3 **Orientación a objetos.**

V.2.4 **Diagramas de clases.**

V.2.4.1 clases que extienden de la clase Model de Eloquent

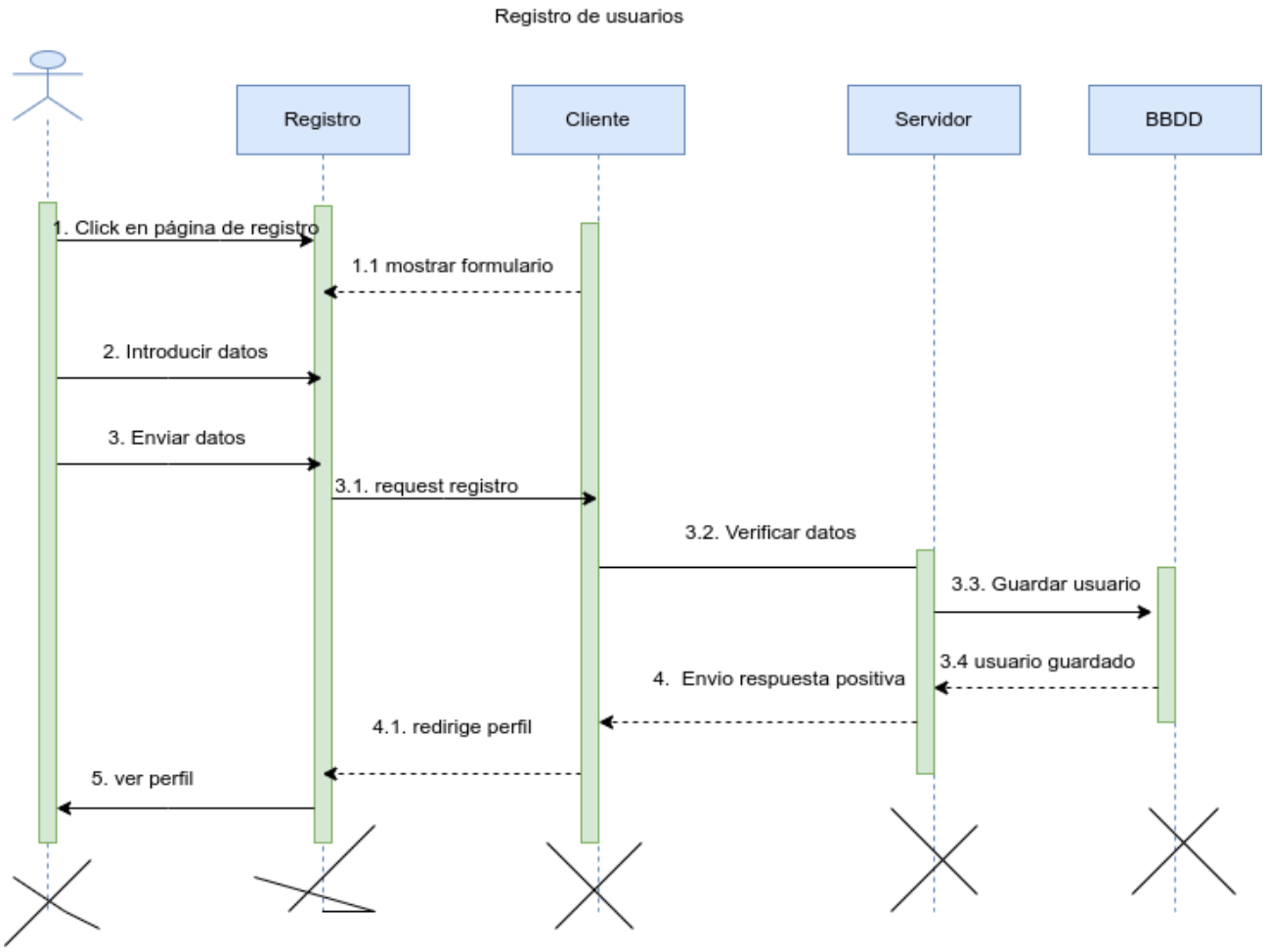
V.2.4.2 clases que extienden de la clase Controller de Eloquent


User
- id: int
- name : String
- email: string
- password: string
- remember_token: string
- created_at: datetime
- updated_at: datetime
+ address(): Collection <Address>
+ seller(): seller
+ paymentTypes(): Collection<PaymentTypes>
+ orders(): Collection<Order>
0 ...*
1
- id: int
- street: string
- number: string
- city: string
- state: string
- country: string
- zipCode: string
- userId: int
- created_at: datetime
- updated_at: datetime
+ user()
Made with VisualParadigm For non-commercial use
<<enumeration>> State
- pending
- completed
- cancelled
- generated
Order
- id : int
- state : State
- userId: int
- sellerId: int
- total: int
+ seller() : seller
+ user() : user
+ orderDetails: orderDetail
0 ..1
1
Seller
- id:int
- bussnessId : String
- bussnessType : string
- bussnessPhone : string
- bussnessWebsite : string
- bussnessLogo :string
- bussnessDescription : text
- userId : int
- created_at: datetime
- update_at: datetime
+ user(): user
+ sellerWithdrawalAccount(): sellerWithdrawalAccount
1
1...*
Order_details
- orderId : int
- productId : int
- quantity
- price
- created_at
- update_at
1
0 ...1
SellerWithdrawalAccount
- id : int
- bankName : string
- bankAccountNumber : string
- bankAccountOwner : string
- sellerId : int
- created_at : datetime
- update_at : datetime
+seller()
1
0 ...*
File
- id : int
- attached : string
- description : string
- created_at : datetime
- updated_at : datetime
1
0 ... *
1
0...*
Product
- id : int
- name : string
- description : string
- price : int
- stock : int
- rating : int
- categoryId: int
- sellerId: int
- imageId: int
- created_at : datetime
- update_at : datetime
+ category():category
+ seller() : seller
+ image() : file
Category
- id : int
- name : string
- description : string
- imageId : int
- created_at : timetime
- updated_at : datetime
+ image() : file
1
0 ...1
1
0 ...*
ShoppingCart
- id : int
- userId : int
+ user()
+ cartItems() : Collection <cartItems>
1
0 ... *
CartItem
- id: int
- productId: int
- shoppingCartId : int
- quantity : int
- price : int
- shoppingCart()
- Product()

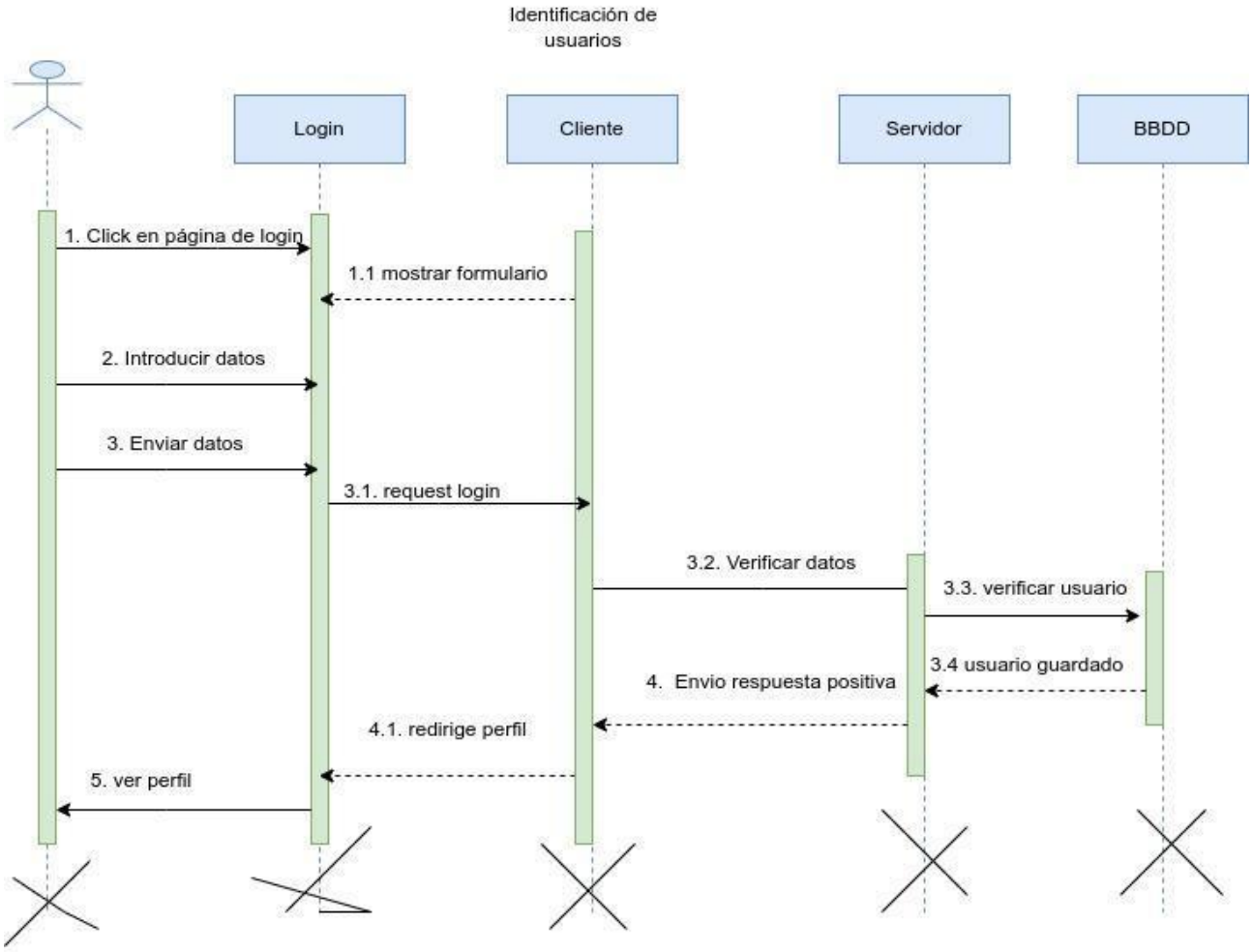

PaymentType
PaymentTypesController
- index()
- sotre()
- show()
- update()
- destroy()
User
Address
AddressController
- index()
- sotre()
- show()
- update()
- destroy()
Seller
RegisterRequest
- authorize()
- rules()
Product
Category
SellerController
- index()
- sotre()
- show()
- update()
- destroy()
SellerWithDrawAccountController
- index()
- sotre()
- show()
- update()
- destroy()
AuthController
- register()
- login()
- getUser()
ProductController
- index()
- sotre()
- show()
- update()
- destroy()
CategoryController
- index()
- show()
Order
OrderDetails
CartItems
File
OrderController
- index()
- sotre()
- show()
- update()
- destroy()
ProductController
- index()
- sotre()
- show()
- update()
- destroy()
ShoppingCart
ShoppingCartController
- index()
- sotre()
- show()
- update()
Made with
Visual Paradigm
For non-commercial use

V.2.5 **Diagrama de secuencias.**

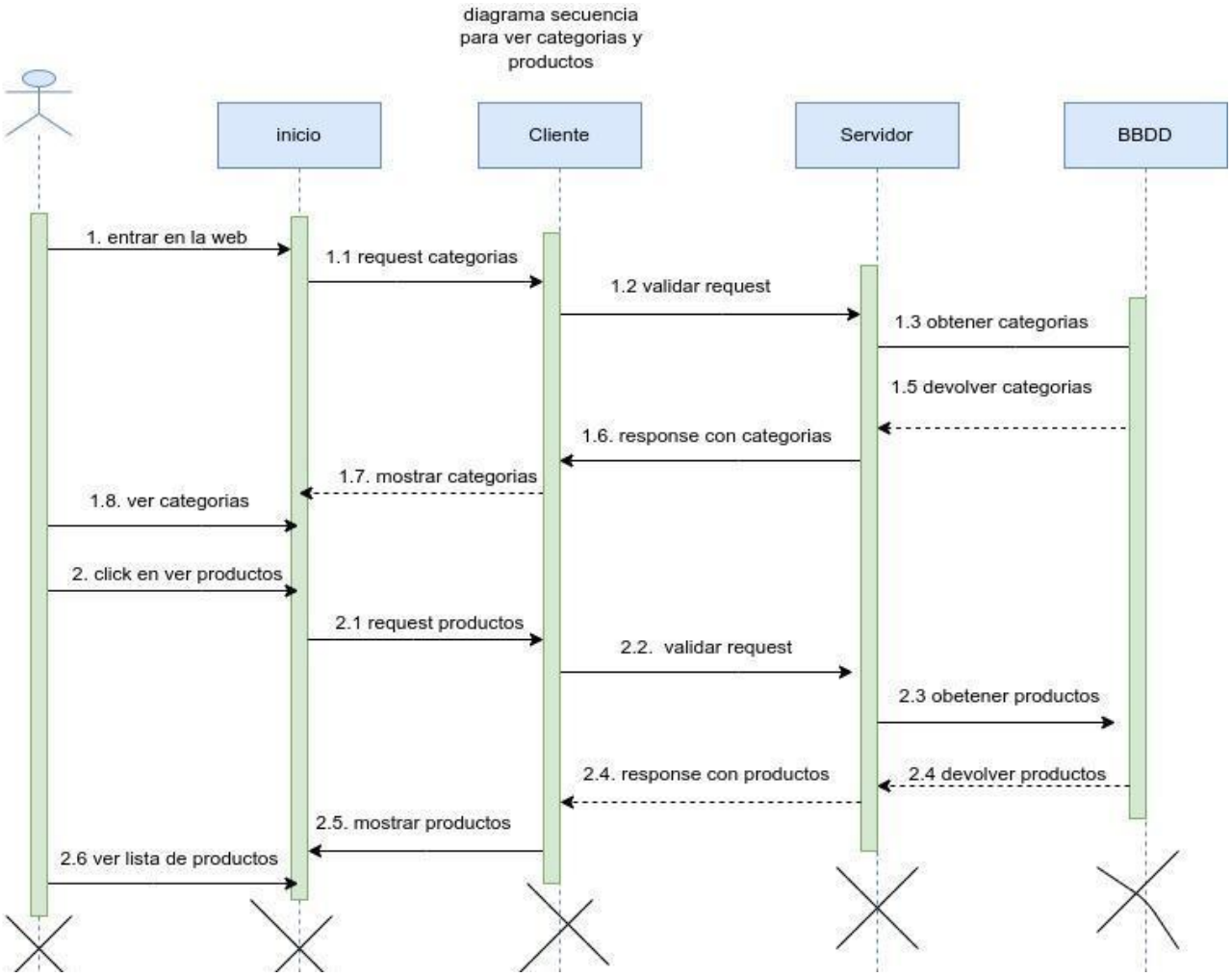
✓ Diagrama de secuencia de proceso para registro de usuarios



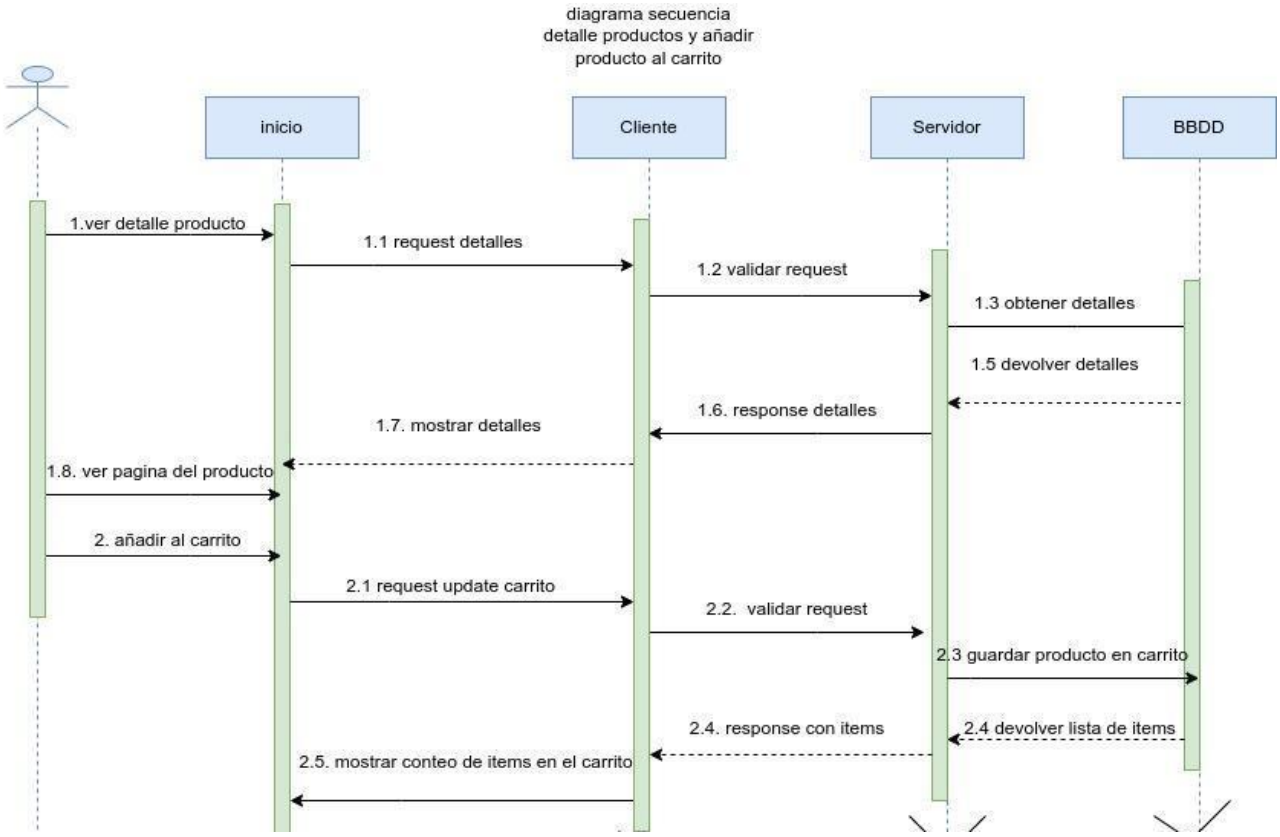
✓ Diagrama de secuencia de proceso para loguear usuario



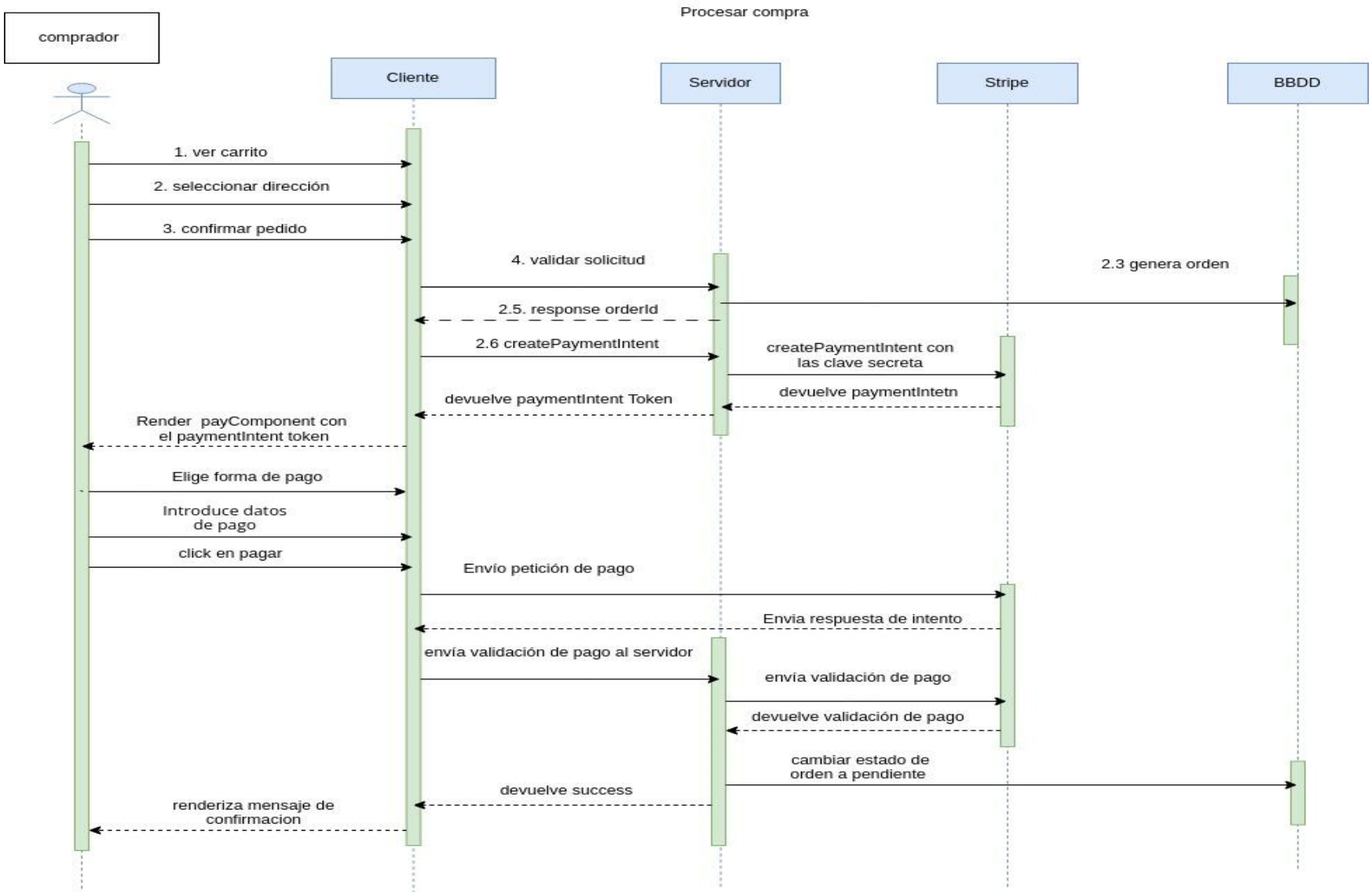
✓ Diagrama de secuencia proceso ver categorías y productos



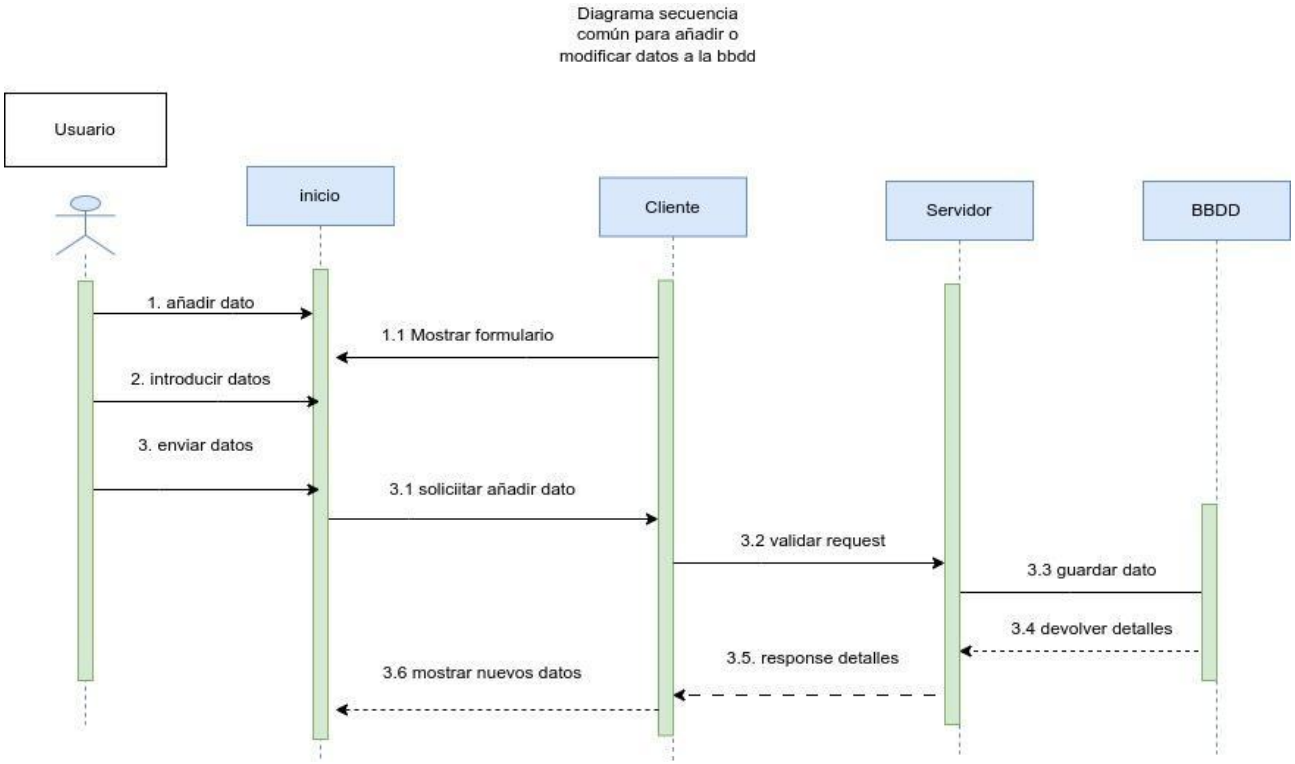
✓ Diagrama de secuencia proceso ver detalle productos y añadir al carrito



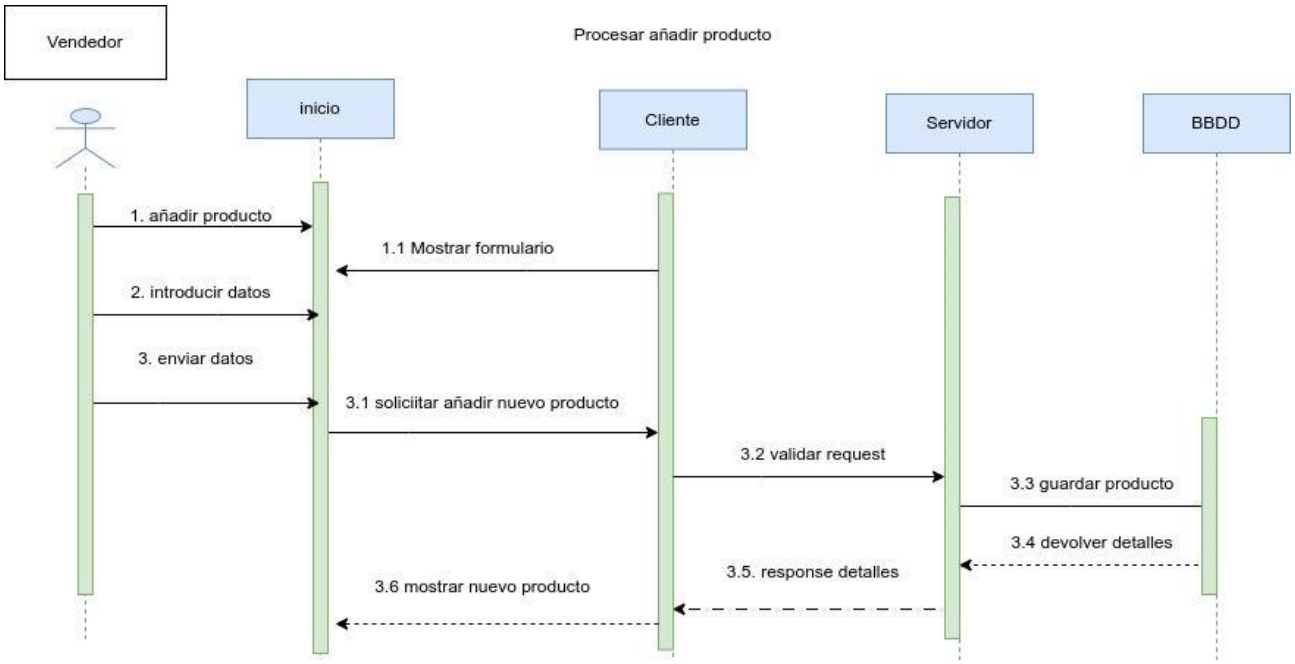
✓ Diagrama proceso gestionar compra usuario logueado


Procesar compra
comprador
Cliente
Servidor
Stripe
BBDD
1. ver carrito
2. seleccionar dirección
3. confirmar pedido
4. validar solicitud
2.3 genera orden
2.5. response orderId
2.6 createPaymentIntent
createPaymentIntent con las clave secreta
devuelve paymentIntetn
devuelve paymentIntent Token
Render payComponent con el paymentIntent token
Elige forma de pago
Introduce datos de pago
click en pagar
Envío petición de pago
Envia respuesta de intento
envía validación de pago al servidor
envía validación de pago
devuelve validación de pago
cambiar estado de orden a pendiente
devuelve success
renderiza mensaje de confirmacion

✓ Diagrama proceso gestionar añadir o modificar datos/información



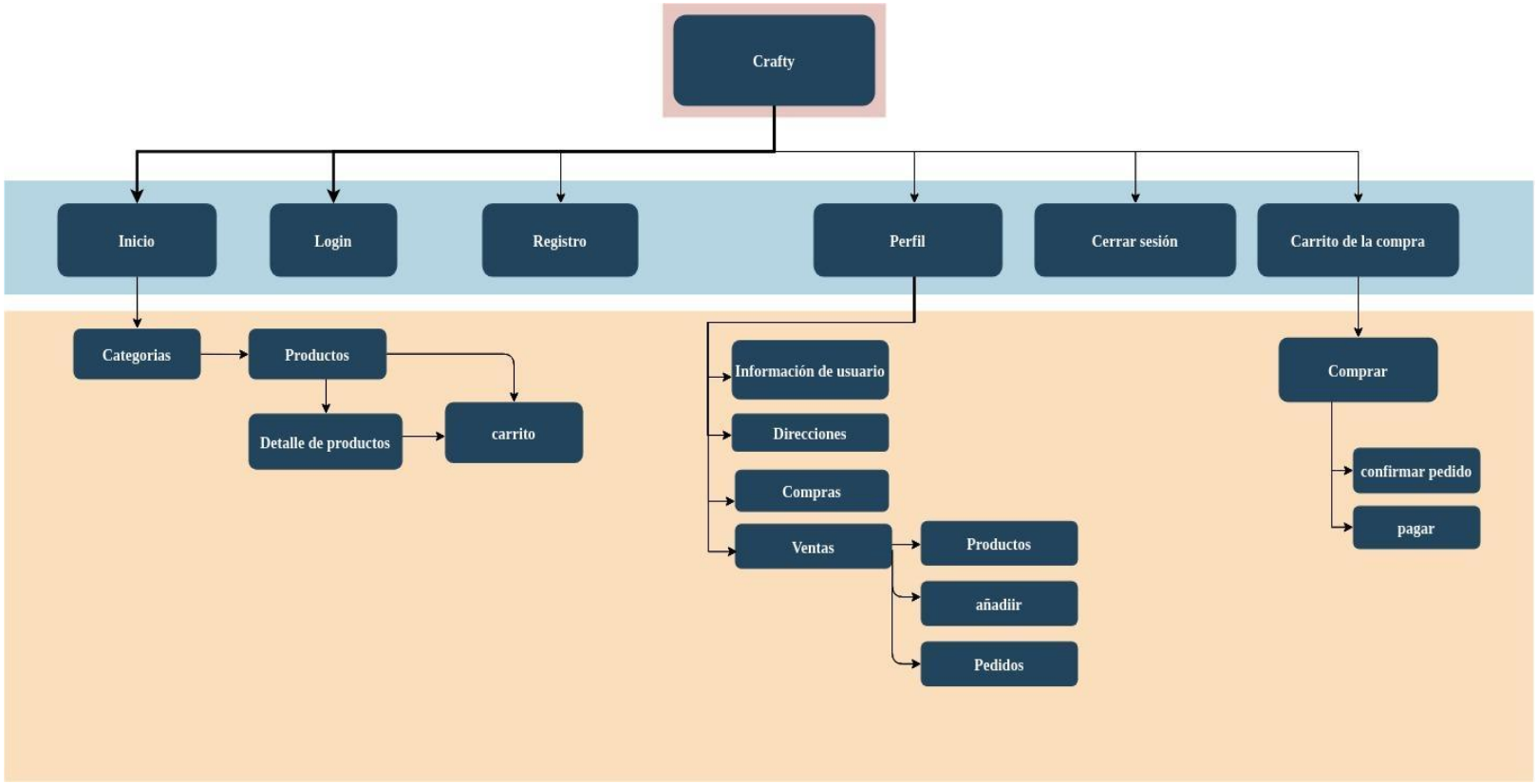
✓ Diagrama proceso gestionar añadir productos nuevos



V.2.6 **Mapa Web.**

El siguiente mapa web muestra de la forma más sencilla posible los enlaces y menús con los que cuenta la web y en los que los usuarios pueden navegar.

Crafty


Crafty
Inicio
Login
Registro
Perfil
Cerrar sesión
Carrito de la compra
Categorias
Productos
Detalle de productos
carrito
Información de usuario
Direcciones
Compras
Ventas
Productos
añadiir
Pedidos
Comprar
confirmar pedido
pagar

V.2.7 **Mockups**

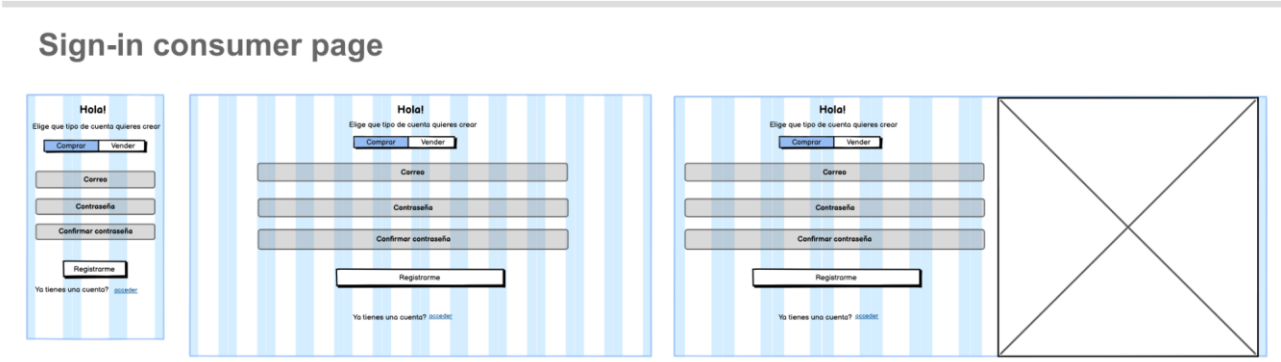
El diseño de la interfaz web ha ido variando conforme avanzaba el desarrollo. En algunos casos se ha simplificado el diseño para restar complejidad y adaptarlo a la información que se quería y se podía transmitir al usuario.

En post de no saturar este documento con capturas de los diseños me limitaré a mostrar y comparar solo algunos. En el **Anexo II** puede verse la comparación completa del diseño final de la interfaz y los mockups de los que se partió. Por último, los mockups preveían un diseño para distintas pantallas, pero para no llenar este apartado de capturas mostraré algunas pantallas en formato móvil, otras en tablet y otras con pantallas más grandes.

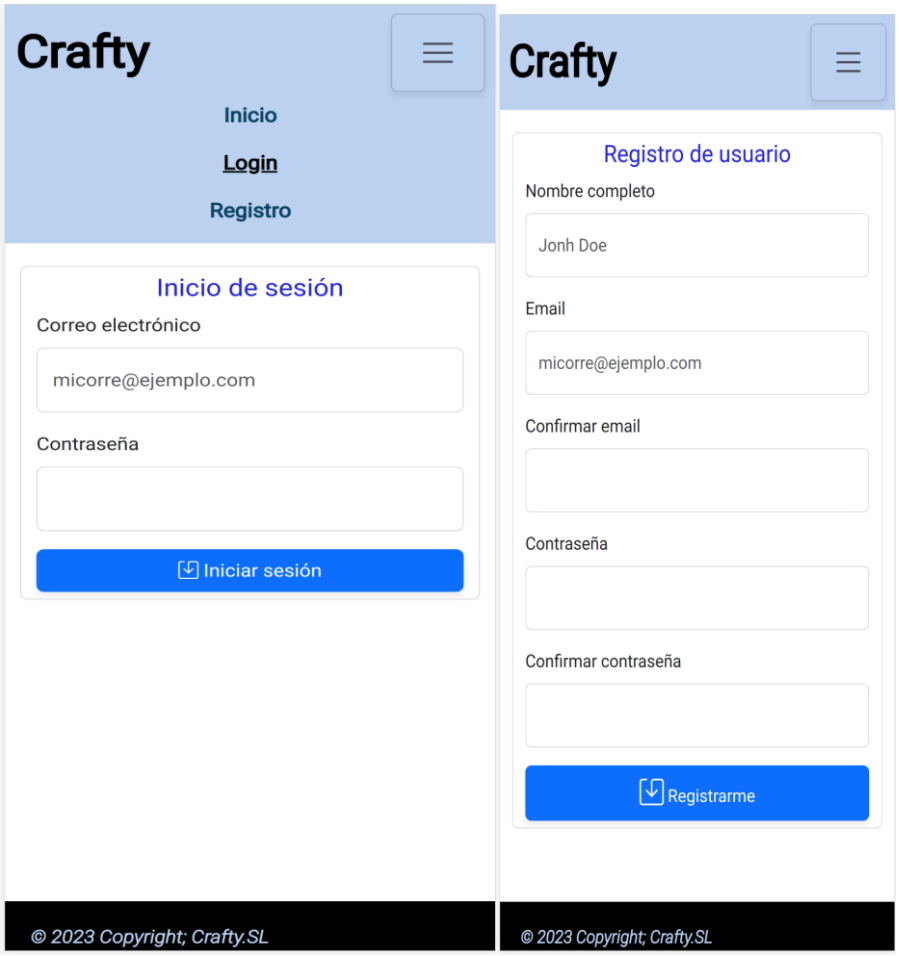
- **Registro y login de usuarios**.

La idea principal para esta interfaz era mostrar un registro y un login diferente según fuese el usuario un comprador o un vendedor. Pero conforme fue evolucionando el Proyecto entendí que no era necesario separar estas pantallas.

Resultado planeado para la página de registro:



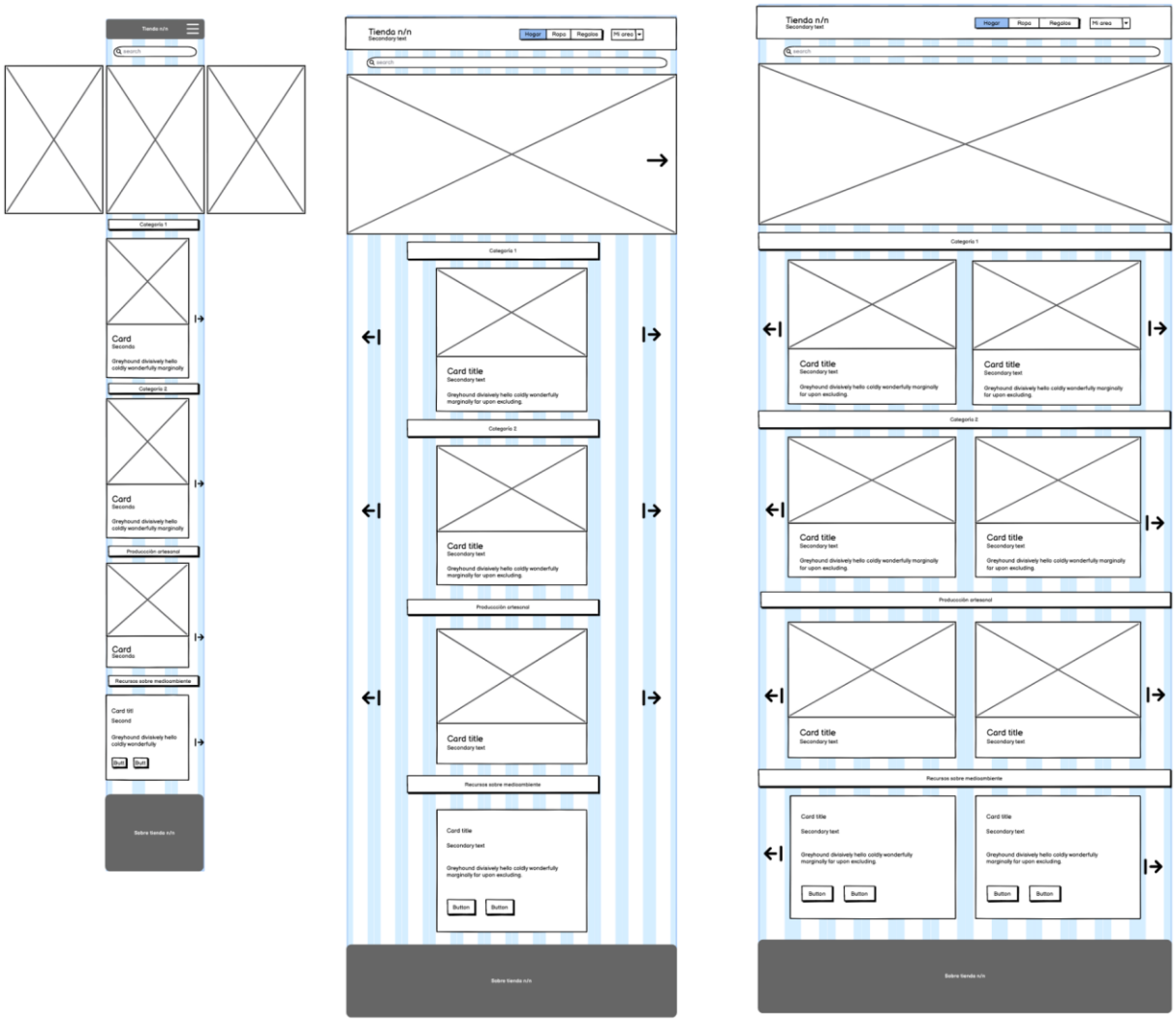
Resultado del registro y login:



- **Inicio**

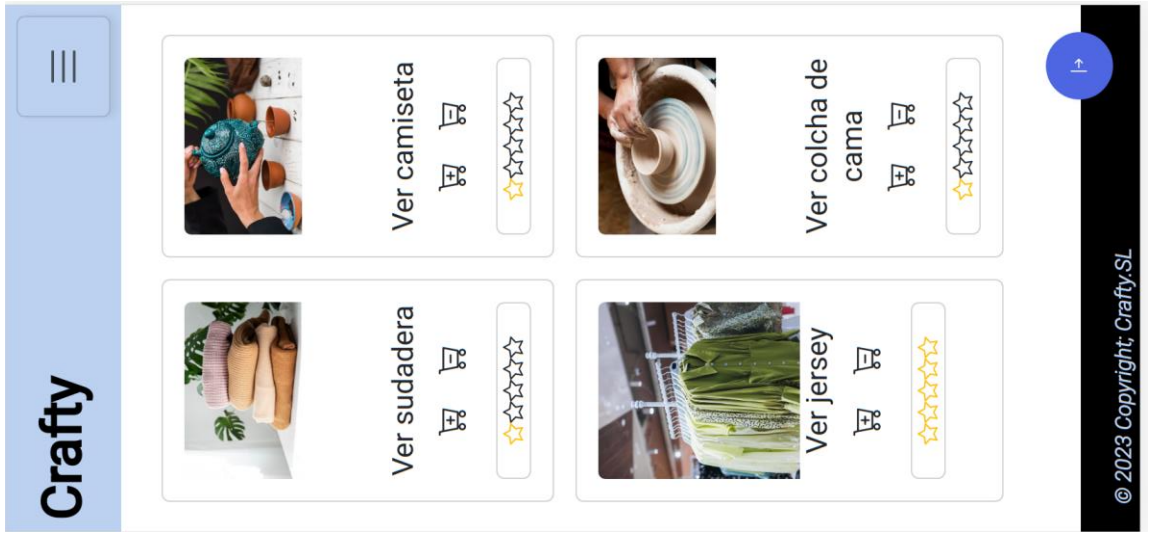
Para el inicio la idea inicial era mostrar un carrousel con información interesante sobre la web y las categorías de productos en cards. Casi en la recta final del Proyecto decidí cambiarlo y eliminar el carrousel porque no estaba aportando nada a la web en ese momento, quizá en un futuro lo vuelva a habilitar. Las categorías decidí añadirlas en un menú lateral de fácil acceso para que los usuarios puedan navegar entre categorías de una forma más sencilla y accesible.

Resultado planeado para el inicio:



Resultado obtenido para la página de inicio:


Crafty
Inicio Login Registro
Ropa
zapatos
Bolsos
Accesorios
Decoración
Higiene Personal
otros
¡Bienvenido a Crafty!
Tu lugar de encuentro para artículos artesanales únicos
¡Únete a nuestra comunidad!
Como comprador o vendedor, tenemos un lugar para ti!
Ya sea que estés buscando un regalo único o quieras compartir tus creaciones con el mundo, Crafty es el lugar ideal para ti. Explora nuestro marketplace, descubre nuevas pasiones y sumérgete en el maravilloso mundo de la artesanía.
¡Contáctanos!
Estamos aquí para ayudarte. Si tienes alguna pregunta, no dudes en ponerte en contacto con nuestro equipo de atención al cliente.
Teléfono: +34 622657878
Correo electrónico: info@crafty.com
Dirección: Calle Artesanía 123, Ciudad Artesana, España


Crafty
Ver camiseta
Ver colcha de cama
Ver sudadera
Ver jersey
© 2023 Copyright; Crafty.SL

- Área de compras

Estaba previsto que en una única página se mostrase toda el are a personal con los pedidos, listas de deseos, productos favoritos, etc. Sin embargo, solo pueden verse las órdenes de pedido, los datos personales, las direcciones y la posibilidad de modificarla.

Resultado esperado del área de compras:

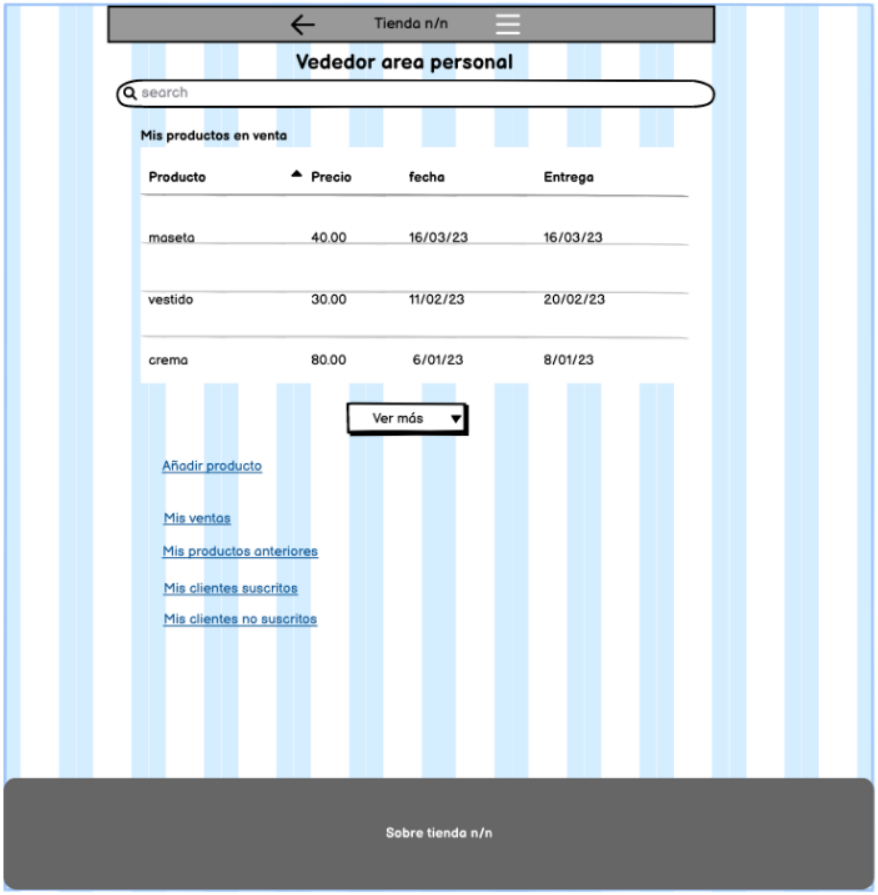
Resultado obtenido en el área de compras:


Crafty
Inicio
Perfil
Mi perfil
Mis direcciones
Mis compras
Area de venta
Mis pedidos
Referencia Total pedido Productos Cantidad Precio Acciones
1 62 20.00 adornos 1 20.00 € Cancelar Recibido
3 60 10.00 jabon 1 10.00 € Cancelar Recibido
4 59 450.00 zapatos 5 90.00 € Recibido
5 58 10.00 jabon 1 10.00 € Recibido
9 54 180.00 jersey 2 90.00 € Cancelar Recibido
10 53 60.00 sudadera 2 30.00 € Cancelar Recibido
11 52 60.00 sudadera 2 30.00 € Cancelado
12 49 200.00 jersey 1 90.00 € Recibido
jersey 1 90.00 €
adornos 1 20.00 €

- **Área de ventas**

El diseño del área de ventas es con diferencia uno de los aspectos que más cambió con la implementación. Se había planeado mostrar un panel de ventas completo con todos los datos y las herramientas necesarias para que los vendedores gestionan sus compras desde aquí, pero de momento estás gestiones tan complejas deberán llevarse con otras herramientas externas.

Resultado esperado del área de ventas:



Resultado obtenido:

La página es similar a la que se muestra en el área de compras.

VI. Codificación

VI.1 Tecnologías elegidas y su justificación.

Como ya he mencionado en varias ocasiones a lo largo de esta memoria, las tecnologías principales utilizadas en este Proyecto son los frameworks de desarrollo web Laravel y React.

Durante los dos años que dura el Curso he usado varios lenguajes de programación, entre ellos Java, PHP, Python y Javascript. Para el desarrollo de mi Proyecto el uso de PHP fue de las decisiones que tenía clara desde el principio. Con este lenguaje de programación empecé a entender, comprender el desarrollo web, conectar con mayor profundidad las herramientas vistas y aprendidas durante el primer año del Curso. Me dio claridad mental.

Por otro lado, para la elección de tecnologías para desarrollo del entorno cliente estuve indecisa ya que Laravel facilita también el desarrollo en dicho entorno porque incorpora herramientas y funcionalidades como el motor de plantillas Blade, sistemas y configuración de rutas. Elegir Laravel habría hecho el desarrollo de mi Proyecto mucho más fácil y ágil para mí e incluso podría haber completado todos los objetivos que tenía para el Proyecto, pero sentía constantemente la necesidad de incorporar y hacer uso del Framework de React.

Utilizar React en el entorno cliente ha sido un reto constante, porque, aunque lo usamos en el Curso, el tiempo que tuvimos para verlo, aprender a usarlo e implementarlo fue bastante corto y no pude alcanzar del todo los objetivos planteados por la guia de estudios y quería aprovechar el Proyecto para continuar aprendiendo. He tenido momentos de mucha frustración, pero también de mucha satisfacción personal por lo aprendido y por los pequeños logros, aunque me habría gustado poder haber conseguido más, ya que no llegué a mis objetivos.

Además, uno de mis objetivos personales era aprender a crear una API y practicar aún más la realización de llamadas a la API desde el front-end utilizando React para recoger y enviar datos desde y hacia el servidor. A Pesar de que en clase practicamos muchísimo con ejercicios y proyectos la utilización de funciones asíncronas y llamadas a APIs, así como el uso de Redux-Toolkit y sus herramientas para facilitar estas tareas, en mis

prácticas no he hecho uso de estas herramientas ni de ningún framework parecido y quería seguir practicando y adquiriendo conocimientos nuevos.

VI.1.1 **Entorno servidor**

VI.1.1.1 Descripción general.

Como he ido adelantando, para el entorno servidor utilicé Laravel como una API. Creía que sería mucho más complejo, pero Laravel facilita está tarea proporcionando una sintaxis clara para definir y gestionar los endpoints de la API.

Además, con la documentación oficial de Laravel y el uso del sistema de ayuda Copilot desarrollado por GitHub, he adquirido conocimientos, la capacidad y la confianza para utilizar eficazmente Laravel como una API.

VI.1.1.2 Creación de la base de datos:

Como herramienta para la administración de la base de datos he utilizado DBeaver es una aplicación de software cliente. El primer paso para el desarrollo de la API fue crear la base de datos desde esta herramienta y conectarla con mi proyecto.

VI.1.1.3 Configuración del proyecto:

Una vez creado el proyecto de Laravel, la primera tarea fue modificar el archivo “. env” y añadir el nombre del proyecto, de la base de datos, etc.

VI.1.1.4 Creación de tablas con el sistema de migraciones de Eloquent / Laravel:

Las tablas en Eloquent se guardan en la carpeta “migrations” ruta “. /database/migrations” y deben crearse con cierto orden, si una tabla tiene una clave ajena, la tabla a la que hace referencia debe crearse primero o de lo contrario dará error y no podrán migrarse.

En el apartado 5.d) ya he hecho referencia a la creación de las tablas y a estas mismas, por lo que pasaré a hablar un poco más de los modelos y la población de datos.

VI.1.1.5 Creación de modelos de Eloquent / Laravel:

Ya he explicado la creación de modelos, pero voy a profundizar un poco más en el contenido de estos. Los modelos se guardan en la carpeta “Models” en la ruta “.

/app/Models" y son clases que extienden de la clase Model administrada por Eloquent y representan a las tablas de la base de datos.

Cuando se genera un modelo mediante los comandos de laravel se generan automáticamente algunas líneas de códigos entre las que encontramos la declaración "use HasFactory" para hacer uso del trait[2] "HasFactory".

HasFactory permite generar instancias de modelos de manera fácil y rápida mediante la utilización de los factories, que sirven para generar datos de prueba o semilleros de datos para poblar las tablas con registros.

Por otro lado, en los modelos de mi Proyecto se puede apreciar la declaración "protected $guarded = []". Esta es una propiedad que sirve para especificar los campos de las tablas que no pueden ser asignados de forma masiva. En el caso de mi proyecto, al estar el array vacío no he restringido ningún campo. Esta configuración puede presentar riesgos de seguridad en caso de estar en un entorno real, pero al ser un proyecto lo he dejado así por facilidad y poder tener datos de prueba en el desarrollo del frontend.

En un entorno real, sería recomendable y seguro usar la declaración "protected $filable = []" y especificar dentro del array los campos que específicamente pueden asignarse de manera masiva.

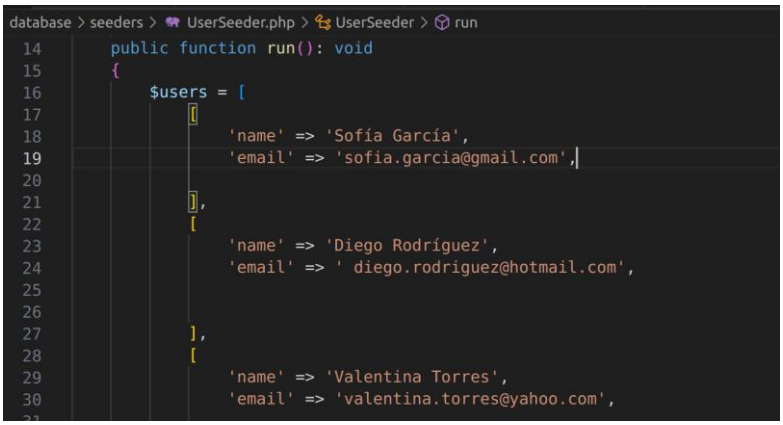
VI.1.1.6 Creación de semilleros y asignación masiva de datos:

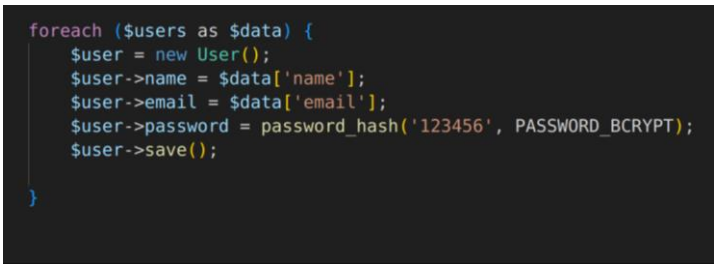
Tal y como he indicado antes, mediante los *factories* y seeders Laravel permite poblar la base datos con datos de prueba. Con los *factories* podemos crear datos aleatorios y con los semilleros se introducen los datos de forma controlada. Para el Proyecto utilicé principalmente los semilleros y dentro de estos utilicé factory en algún caso para generar descripciones aleatorias.

En la figura que veremos a continuación tenemos el código del semillero UserSeeder en el que se introducen a la base de datos cinco usuarios. Se puede apreciar un array de usuarios con arrays en los que se introduce de forma controlada el nombre de usuarios y

[2] Un trait es una forma de reutilizar código en Laravel al agregar funcionalidades predefinidas a una clase

los correos electrónicos[3]. Posteriormente, mediante un bucle forEach () se introducen los datos del array de usuarios a la tabla users a través de la clase User. La contraseña la introduzco directamente en el campo haciendo uso del password_has () para encriptarla. Como se puede apreciar, la contraseña se repetirá para todos los usuarios, esto para facilitar la realización de pruebas en el frontend.





VI.1.1.7 Creación de controladores:

Los controladores en Laravel son clases que se usan para manejar las solicitudes HTTP y definir la lógica del Proyecto. La mayoría de los controladores de mi Proyecto cuentan con un CRUD completo.

En el controlador AuthController donde se recoge la lógica del registro y login de usuarios se encuentra también un método getUser () que utilizó en el front para indicar cual es el usuario autenticado en todo momento y si este cuenta con el rol de vendedor. Aquí también podemos destacar el uso de un request personalizado usando la clase FormRequest. Esta clase establece las reglas necesarias para validar los datos que vienen desde el frontend

[3] Los nombres y correos electrónicos fueron creados de forma aleatoria por la herramienta de Inteligencia Artificial ChatGPT

antes de pasarlos a la base de datos. En este caso le estamos diciendo que todos los campos son requeridos, que el email debe ser único y que la contraseña debe tener mínimo una extensión de seis caracteres.

Otro controlador interesante es la clase CartShoppingController en el que se aprecia un CRUD casi completo a falta de un método delete que no será implementado de momento. La función más interesante en este controlador puede ser la de update () ya que se encarga de realizar varias gestiones al mismo tiempo. Si el usuario no tiene carrito de la compra, es decir que es la primera vez que añade un item al carrito, entonces se genera por primera vez un registro para ese usuario en la tabla ShoppingCart. Posteriormente comprueba que en en los registros de la tabla cart_items exista alguno que esté relacionado con el ShoppingCart del usuario. Si no existe lo crea y si existe entonces lo modifica teniendo en cuenta los datos enviados en la request. Al finalizar envía al entorno cliente una respuesta positiva junto con los datos del carrito de la compra del usuario actualizados.

Por otro lado, el controlador OrderController cuenta con una funcionalidad parecida a la de CartShoppingController, pero en lugar de ser en el método update () la encontramos en el método store () pues además de crear la nueva orden va generando órdenes de detalle por cada ítem que se le haya pasado por parámetro.

También resulta especialmente destacable el método index () que envía una respuesta diferente al cliente en función de la request recibida. Si se envía el userId entonces devuelve los pedidos realizados por ese cliente junto con los detalles. Si en la request se ha enviado el sellerId entonces se busca en cada orden los detalles de pedido que tengan productos de este vendedor y se reenvía al entorno cliente las órdenes de compra, pero solo con los detalles de los productos de ese vendedor, excluyendo el resto.

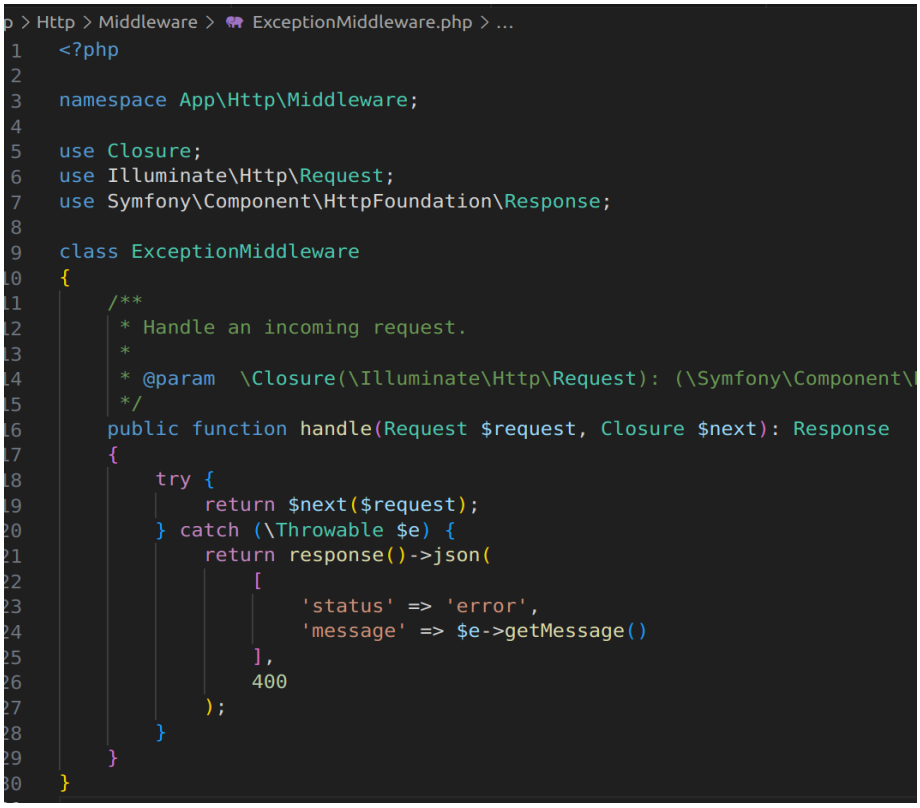
VI.1.1.8 Evitar o capturar errores y warnings

Al igual que en los supuestos explicados en líneas anteriores, en la captura y manejo de errores se pueden apreciar diferencias dentro de la lógica de los controladores.

En los primeros controladores se aprecia el uso de “try … catch ()” para manejar los errores. Sin embargo, durante las siguientes semanas descubrí otra forma de manejar los

errores mucho más eficiente y casi silenciosa creando un Middleware. En este caso creé una clase llamada “ExceptionMiddleware” para manejar las excepciones que pudieran producirse al recibir las peticiones HTTP y cuenta con un único método llamado “handle” que espera como parámetros la Request y un Clouser. En el caso del Proyecto, este último alude al controlador que va a procesar las solicitudes recibidas.

Por último, dentro del método podemos encontrar un bloque "try-catch" para capturar las excepciones. Si no se produce ninguna excepción se llama al controlador y se devuelve la Request, pero si se produce una excepción se envía una respuesta JSON con un código de estado de 400 con el mensaje de error.



Este Middleware debe agregarse dentro un array en el Kernel del Proyecto. Este Kernel es una clase incorporada en Laravel que extiende de la clase HttpKernel y que incorpora los middlewares que se ejecutan durante cada una de las peticiones hechas a la aplicación, en mi caso a la API.

Por último, para configurar las rutas hice uso de la función Route: group con prefijos adecuados para cada uno, por lo que todas las rutas definidas dentro de ese grupo empezaran con el prefijo. Lo explicaré más adelante.

Algunos ejemplos de estas rutas:

```php
Route::group(['prefix' => 'auth'], function () {
    Route::post('/register', [AuthController::class, 'register']);
    Route::post('/login', [AuthController::class, 'login']);
    Route::get('/user', [AuthController::class, 'getUser'])->middleware('auth:sanctum')
});

Route::group(['prefix'=> 'product'], function(){
    Route::get('/index', [ProductController::class, 'index']);
    Route::get('/show/{id}', [ProductController::class, 'show']);
    Route::post('/store', [ProductController::class, 'store']);
    Route::put('/update/{id}', [ProductController::class, 'update']);
    Route::delete('/destroy/{id}', [ProductController::class, 'destroy']);
});

Route::group(['prefix'=> 'category'], function(){
    Route::get('/index', [CategoryController::class, 'index']);
    Route::get('/show/{id}', [CategoryController::class, 'show']);
});

Route::group(['prefix' => 'file'], function () {
    Route::get('/index', [FileController::class, 'index']);
    Route::get('/show/{id}', [FileController::class, 'show']);
    Route::get('/print/{id}', [FileController::class, 'print']);
    Route::post('/store', [FileController::class, 'store']);
    Route::put('/update/{id}', [FileController::class, 'update']);
    Route::delete('/destroy/{id}', [FileController::class, 'destroy']);
});
```

VI.1.2 **Entorno cliente**

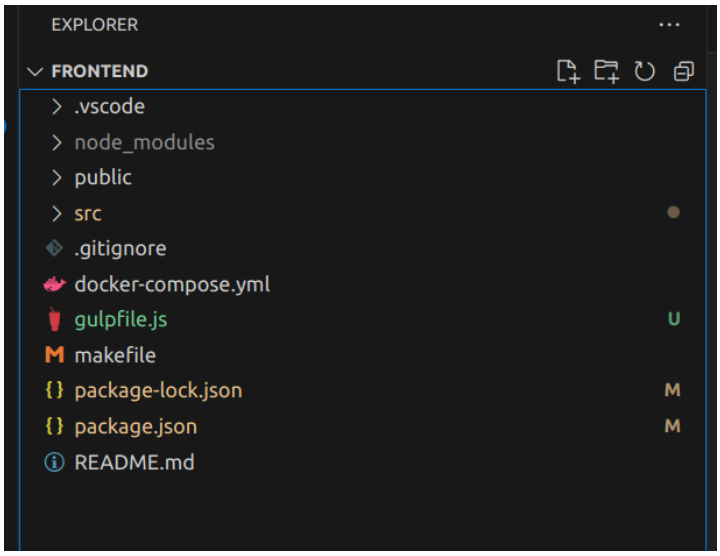
VI.1.2.1 Descripción general

El uso de React y Redux-toolkit en mi Proyecto ha sido un completo reto del que he aprendido muchísimo, no solo de React y javascript sino a manejar la frustración y mantener el foco, a buscar soluciones alternativas y uso de todas las herramientas y ayudas que existen en google, foros, la documentación oficial de React, de Redux-toolkit par React, de Copilot, React para Bootstrap, los videos de clase, entre otros.

Era consciente de que sería difícil incorporar React en mi Proyecto sobre todo porque tenía muchísimo que aprender y era consciente de ello, pero la dificultad que he tenido que afrontar no era ni cerca lo que esperaba. Algunos conceptos que creía tener claros tuve que volver a estudiarlos y algunos otros de los que ni siquiera era consciente. Por ejemplo, cuando pensaba que sabía cómo utilizar las rutas o enlaces dentro de React me tope con un problema durante muchas horas porque en el fondo sólo había descubierto una mínima parte de react-router-dom. Con suerte y mucha paciencia puede ir solucionando los

problemas que surgían o buscar alternativas, pero esto requirió mucho tiempo y al final no he podido conseguir todos los objetivos marcados.

La estructura del entorno cliente de mi Proyecto es la siguiente:



Algunos de los archivos los explicaré más adelante así que me limitaré en las siguientes líneas a explicar la estructura interna de las carpetas “public” y “src”.

La carpeta “public” contiene los archivos estáticos del Proyecto en el lado cliente. Esta carpeta se crea automáticamente al crear el proyecto de React. Dentro podemos encontrar varios archivos y recursos, aunque elimine algunos de los que venían por defecto ya que no iba a usarlos. En la carpeta se encuentra el archivo index.html y es aquí donde se monta la aplicación de React.

En la carpeta src encontramos casi todo el desarrollo del proyecto. Resaltan los archivos App.jsx con extensión de React y el index.js. El archivo index.js es el punto de entrada principal del Proyecto y se usa para renderizar las aplicaciones en los navegadores. Como podemos ver en la siguiente imagen se usa la línea:

Para simplificar la explicación del entorno cliente expondré el flujo de funcionalidad de una de las partes del Proyecto y es la que se corresponde con la visualización de los productos. Empezamos por el componente <MainLayout/> que se encuentra en la ruta “. /src/layout”.

Este componente es uno de los únicos componentes que se importan en el componente principal App.jsx.

La aplicación cuenta con tres menús diferentes como se puede observar en el apartado 5 de esta memoria. EL menú principal de la aplicación, que se encuentra en la cabecera se renderiza desde el componente <MainLayout/> en el que se define las rutas correspondientes, las condiciones para mostrarlas al usuario (como que este logueado o no o que tenga ítems en el carrito).

El siguiente menú es el de la barra lateral y su lógica y renderización se encuentra dividida en dos partes. Las rutas se encuentran y se renderizan en el componente <MainLayout/> y el código html que lo muestra también, pero los datos de cada parte del menú se envían desde App en función de si queremos ver las categorías, el perfil o la página inicial.

La ruta “/products” redirige al componente <ProductsPage/>. Este componente es bastante sencillo. En las primeras líneas de código tenemos la constante Products que extrae con el hook de react-redux useSelector () todos los productos del store. Con las constantes “error” y “isLoading” hago lo mismo.

Luego con la función useDispatch () envío las acciones al store, en este caso como se puede ver se envía la acción de la función fetchProducts (). También se aprecia el uso de la función useSearchParams (), un hook de React Router para acceder y manipular los parámetros de búsqueda de las URLs. En la siguiente figura podemos ver como se usa este hook para extraer el CategoryId de los productos que se quieren mostrar.

```
export const ProductsPage = () => {

    const Products = useSelector(selectAllProducts);
    const error = useSelector(selectProductError);
    const isLoading = useSelector(selectProductLoading);
    const dispatch = useDispatch();
    const firstExecution = useRef(true);
    const [searchParams, setSearchParams] = useSearchParams();
    const categoryId = searchParams.get("categoryId");

    const cartItems = useSelector((state) => state.cart.cartItems);

    useEffect(() => {
        if (firstExecution.current) {
            dispatch(fetchProducts({categoryId}))
            dispatch(loadCartItems());
            firstExecution.current = false;
        }
    }, [dispatch, firstExecution]);
```

En el resto del componente encontramos el return, el html y datos que visualizamos en el navegador.

```
<div className="mainPage  ">

    <div className="mainPage__carousel">
        <MainCarousel />

    </div>


    <div className="align-product">
        <div className="mainProducts d-flex justify-content-around py-4 px-4 row gap-2">
            {
                Products.map((product) => (<CardProductComponent key={product.id} props={product}  linkTo={`/products/${product.id}`}
            }
        </div>
    </div>
    {isLoading && <Spinner />}
    {error && <div className="alert alert-danger">{error}</div>}
</div>
```

Aquí se utiliza el componente <CardProductComponent /> personalizado y creado específicamente para ver los productos en formato card. Los productos se pintan haciendo uso del método. map() entre los parámetros que se pasan a este componente se encuentra el id del producto para posteriormente usarlo para pintar los detalles de cada producto al clicar en el botón ver.

Continuando con el flujo de ejecución vamos a ver el store del Proyecto. La estructura de la carpeta “/store” es un poco irregular ya que a mitad del desarrollo decidí cambiarla.

En un principio había decidido tener una carpeta por cada elemento del store como se ve en las carpetas “. /login” y “. /register” donde tengo separados los slices y los thunk. Pero luego decidí simplificarlo y dejarlo todo en un solo archivo. Simplificarlo desde mi punto de vista como estudiante y por el plazo que tenía que cumplir, ya que considero mucho más simple tener los archivos separados ya que el código queda mucho más limpio y entendible.

La carpeta “. /store” tiene un directorio llamadas “. /slices”, un archivo llamado “craftyStore.jsx” y “indexStore.jsx”. En el archivo “craftyStore.jsx” importo todos los slices de la aplicación. En el archivo inedxStore.jsx solo hay una línea de código que se encarga de asegurar la exportación de todo el contenido de la carpeta “. /store”.

```
export const store = configureStore({
    reducer: {
        register: registerUserSlice.reducer,
        login: loginUsersSlice.reducer,
        category: CategoryReducer,
        product: ProductReducer,
        authUser: authUserSlice.reducer,
        FileProduct: ProductReducer,
        address: AddressReducer,
        paymentType: PaymentTypeReducer,
        order: OrderReducer,
        sellerAccount:SellerAccountReducer,
        cart: CartSlice.reducer
    },
    middleware: (getDefaultMiddleware) => getDefaultMiddleware().concat(logger),
});
```

En la figura anterior vemos la configuración del store. Voy a usar el slice que maneja la lógica para la gestión de los productos de la web. Este slice se encuentra en el archivo ProductSlice.js donde primero encontramos funciones createAsyncThunk () que utilizo para definir acciones asíncronas de forma mucho más simple con Redux-Toolkit.

```
export const fetchProducts = createAsyncThunk(
    'product/fetchProducts',
    async (filter) =>  getProducts(filter)
);

export const fetchShowProduct = createAsyncThunk(
    'product/fetchOneProduct',
    async (productId) =>  showProduct(productId)
);

export const createProduct = createAsyncThunk(
    'product/createProduct',
    async (newProduct) =>  createProductApi(newProduct)
);

const productAdapter = createEntityAdapter({});
```

Podemos ver funciones como fetchProducts () que ejecutan la función getProducts (). fetchProducts () es usado en el componente <PoductsComponente /> que ya ha sido expuesto. La función getProducts () será explicada más adelante, pero es la que se encarga de ejecutar la petición HTTP para obtener los productos que se corresponden con el filtro que recibe como parámetro. Este parámetro es un id y puede ser tanto el id de una categoría o el id de un usuario (o vendedor).

En este slice en concreto hago uso de la función "createEntityAdapter ({})" que "*genera un conjunto de reductores y selectores preconstruidos para realizar operaciones CRUD en una estructura de estado normalizada que contiene instancias de un tipo particular de objeto de datos*[4]*"*.

En la siguiente figura vemos la estructura de la función ProductSlice (). Prácticamente todos los slices del Proyecto tienen una estructura parecida, aunque algunos tienen reducers definidos y otros no por no ser necesarios.

En la siguiente figura vemos también cómo se simplifica el manejo de los diferentes estados de una llamada asíncrona a las funciones *fetchProducts* () y showProducts (). Los extrareducers se ejecutarán en función del resultado de las peticiones HTTP.

[4] https://redux-toolkit.js.org/api/createEntityAdapter

```
const ProductSlice = createSlice({
    name: 'product',
    initialState: productAdapter.getInitialState({
        loading: false,
        error: null,
    }),
    reducers: {},
    extraReducers: {
        [fetchProducts.pending]: (state, action) => {
            state.loading = true;
            return state;
        },
        [fetchProducts.fulfilled]: (state, action) => {
            state.loading = false;
            state.error = null;
            productAdapter.setAll(state, action.payload);
            return state;
        },
        [fetchProducts.rejected]: (state, action) => {
            state.loading = false
            state.error = action.error.message
            return state;
        },
        [fetchShowProduct.pending]: (state, action) => {
            state.loading = true;
            return state;
```

Continuando con el flujo del Proyecto pasaré a explicar el contenido del directorio “. /api” donde se encuentra la lógica de las llamadas a la API. Estos archivos empecé a crearlos con la extensión .jsx, pero en realidad no son componentes por lo que no deberían llevar esta extensión.

Destaca la función asíncrona craftyApi () que recibe como parámetros el método de llamada, la url, el body y la query en caso de que se envíen datos. Este método está hecho para ser reutilizado en las llamadas y no tener que repetir cada vez las mismas líneas, aunque lo implementé un poco tarde por lo que algunas llamadas se realizan sin él. Además, se gestionan aquí las respuestas según el status.

```
src > api > craftyApi.jsx > craftyApi
1
2   export const CRAFTY_BASE_URL = 'http://localhost:8080/api';
3
4
5   export const craftyApi = async (
6       { method, uri, body, query }
7   ) => {
8       const endpoint = `${CRAFTY_BASE_URL}/${uri}` + (query ? `?${new URLSearchParams(query)}` : '');
9
10      const response = await fetch(endpoint, {
11          method: method,
12          headers: {
13              'Content-Type': 'application/json'
14          },
15          body: typeof body === 'object' ? JSON.stringify(body) : body
16      });
17      const data = await response.json();
18      if (response.status  < 200 || response.status >= 300) {
19          throw new Error(data.message);
20      }
```

Un buen ejemplo del uso del método craftyApi () y de llamadas a la API se puede encontrar en los slice que vimos en líneas anteriores.

```
src > api > productsApi.jsx > ...
1   import { craftyApi } from "./craftyApi";
2
3
4   export const getProducts = async(filter) => {
5       const response = await craftyApi({
6           method: 'GET',
7           uri: 'product/index',
8           query: filter
9       });
10
11      return response.data;
12      }
13  export const showProduct = async(productId) => {
14      const response = await craftyApi({
15          method: 'GET',
16          uri: `product/show/${productId}`
17
18      });
19
20      return response.data;
21      }
```

VI.1.2.2 Estilos de la web.

En el proyecto se incorpora la versión 5 de la biblioteca de bootstrap y la biblioteca React-Bootstrap. Esta última herramienta fue creada expresamente para trabajar con componentes de React y reemplaza el javascript de Bootstrap. Por agilidad y falta de tiempo estas son las herramientas que prevalecen en la web para dar estilo y forma al Proyecto.

Sin perjuicio de lo anterior, también se pueden encontrar en el proyecto algunos estilos personalizados incorporados al proyecto con el uso del preprocesador de estilos SASS y del transpilador Gulp.

Para ejecutar la tarea Gulp “default” que transpira el código sass se escribe el comando “npx gulp default”.

VI.1.3 **Integración con stripe**

Crafty se integra con la tecnología de Stripe para gestionar los pagos y cobros de productos. Stripe es una reconocida empresa de desarrollo de software cuyas soluciones *“posibilitan los pagos para minoristas que operan en Internet y en persona, empresas que trabajan con suscripciones, plataformas de software y marketplaces y mucho más (…) ayudando a las empresas a luchar contra el fraude, enviar facturas, emitir tarjetas virtuales y físicas, reducir las fricciones en el proceso de compra, obtener financiación, gestionar los gastos de la empresa y mucho más [5]”.*

La integración de Stripe con proyectos React y Laravel es fácil, segura y eficiente. Stripe ofrece una librería React y una amplia variedad de componentes que permiten implementar diversas formas de pago.

Entre las diferentes funcionalidades que ofrece Stripe para gestionar los pagos se encuentra el Payment Intent que permite gestionar pagos de forma segura. Cuando el cliente confirma el pago se envía una solicitud a Stripe con los detalles del pedido, la cantidad y la moneda en la que se realiza el pago. Tras recibir esta solicitud, Stripe devuelve un “client secret” o un token de autenticación y confirmación del intento de pago, tras los que se puede proceder al pago y si este se completa Stripe actualiza el estado del Payment Intent a "succeeded" y la plataforma o aplicación recibe la confirmación del pago y realiza las acciones necesarias, como enviar una confirmación al cliente y actualizar la base de datos.

Con el Payment Intent Stripe permite gestionar pagos con diversas características y ofrece funcionalidades avanzadas, como gestión de riesgos y prevención de fraudes. Con esta solución, Stripe simplifica el proceso de pago y brinda seguridad en las transacciones financieras online. Esta funcionalidad fue la elegida para ser integrada en el Crafty.

[5] Información extraída de la página web de https://stripe.com/es

Para llevar a cabo esta integración era necesario crear una cuenta en Stripe. Con esta cuenta se proporciona un entorno de desarrollo y gestión de pruebas para los desarrolladores. Stripe proporciona una clave pública y una clave privada que deben agregarse al Proyecto. La clave pública se almacena en ambos proyectos (Laravel y React) mientras que la clave privada se guarda en el entorno servidor en una variable de entorno.

En mi caso, tuve que realizar la integración tanto en el proyecto de Laravel como en el de React. La clave pública se encuentra almacenada en ambos proyectos y la privada solo en el entorno servidor como una variable de entorno.

VI.1.3.1 Integración de Stripe en el entorno cliente.

Tras instalar React Stripe tuve que agregar algunos de los componentes que proporciona Stripe para crear el formulario de pago de la aplicación e integrarlo con los componentes que ya tenía preparados para esta gestión, como el que se encarga de recuperar el carrito de la compra, las direcciones del usuario para gestionar la compra o para crear la orden de pedido.

El componente encargado de renderizar el carrito de la compra y la dirección espera a que el usuario elija una dirección y confirme que quiere realizar la compra para enviar una petición al backend para que genere una orden de compra. Cuando se recibe la respuesta positiva el cliente solicita esta vez al backend que solicite un intento de pago a Stripe. Esta parte la explicaré en el siguiente apartado.

El entorno cliente espera una respuesta exitosa por parte del servidor junto con un token y es entonces cuando renderiza el formulario de pago y permite al usuario ingresar sus datos de pago e intentar pagar. Tras lo cual, el entorno cliente realiza nuevamente una petición al entorno servidor con el pago y finalizará el proceso de compra con un mensaje al usuario mostrando el estado de la compra y redirigiendo a su área personal.

Los componentes proporcionados por Stripe para procesar la compra permite al usuario elegir diferentes métodos de pago y controla la gestión de errores directamente (validaciones de métodos de pago, fechas, etc).

VI.1.3.2 Integración de Stripe en el entorno servidor.

Por último, quería explicar brevemente la integración de Stripe en Laravel. Esta explicación será completada con la parte que se desarrolla en el entorno cliente. Como podemos ver en la siguiente imagen, toda la gestión backend que debe hacerse para proporcionar una pasarela de pagos sencilla se concentra en dos métodos añadidos en el controlador StripeController.

```
    public function createPayment(Request $request)
    {
        \Stripe\Stripe::setApiKey(getenv('STRIPE_SECRET_KEY'));
        $order = Order::find($request->get('orderId'));
        $paymentIntent = \Stripe\PaymentIntent::create([
            'amount' => $order->total * 100,
            'currency' => 'eur',
            'automatic_payment_methods' => [
                'enabled' => true,

            ],
            'metadata' => [
                'orderId' => $order->id,
            ]

        ]);

        return response()->json([
            'status' => 'success',
            'data' => ['clientSecret' => $paymentIntent->client_secret]
        ]);
    }
}
```

El método “createPayment”, con la clave secreta que me proporcionó Stripe realiza una llamada a Stripe utilizando su SDK (Software Developer Kit). Si todo se realiza correctamente, se envía una respuesta positiva al entorno cliente junto con el token generado y almacenado en la variable "$paymentIntent". Desde el entorno cliente, este token permite al usuario proceder con el pago.

Posteriormente, si el usuario continúa con el pago, se recibe otra petición desde el entorno cliente que será procesada por el método “process”. En este método, utilizando la clave secreta y el SDK de Stripe, se valida el pago. Si se recibe una respuesta positiva, se realizan tres acciones: se cambia el estado de la orden a "pendiente" para su recepción,

se eliminan los ítems del carrito de compras del usuario y se envía una respuesta positiva al entorno cliente.

VI.1.4 **Asegurar la funcionalidad en los navegadores más usados**

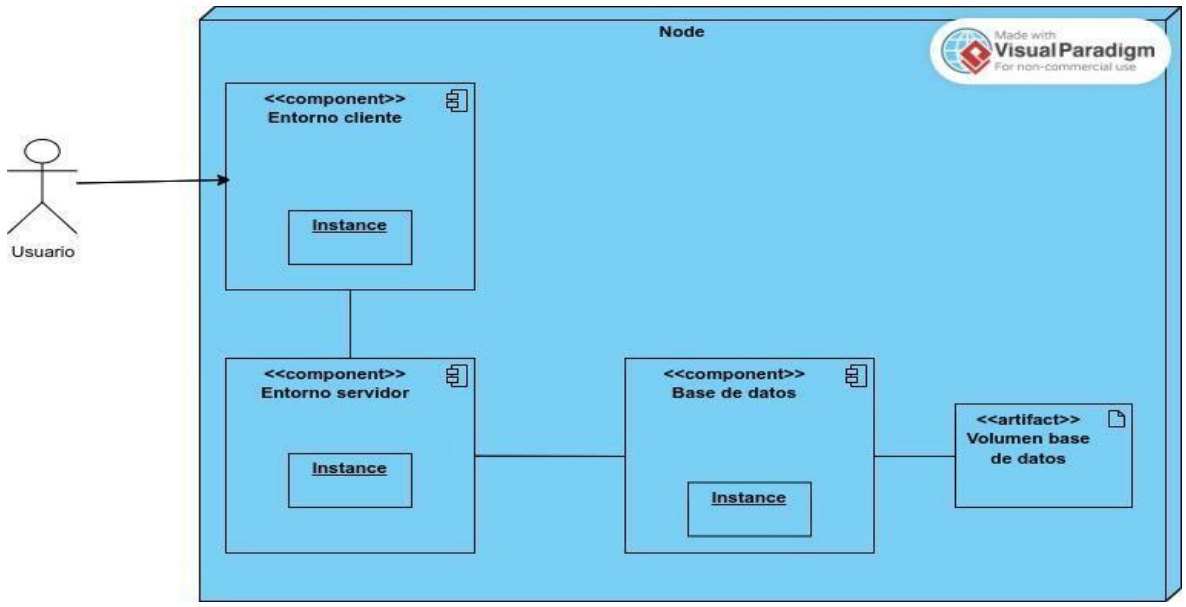
Se puede asegurar la funcionalidad del Proyecto tanto en el navegador de Chrome como en FIrefox. (ver imágenes del apartado de 5).

VI.1.5 **Documentación interna de código**

La documentación interna del código se ha realizado con JsDOC y aunque no se encuentra completa se puede ver en el **Anexo III**.

VII. Despliegue

VII.1 Diagrama de despliegue



VII.2 **Descripción de la instalación o despliegue**

VII.2.1 Fichero de configuración

Mi Proyecto se encuentra dividido en dos subproyectos que se encuentran en las carpetas “. /backend” y “. /frontend”. Ambos subproyectos se despliegan en contenedores docker.

VII.2.2 Entorno servidor:

Para el entorno servidor creé una carpeta “docker” para almacenar los servicios “db” y “php” en sus respectivas carpetas. Para el servicio “php” se utiliza una imagen propia basada en “php:8.2-alpine3.17” e incorpora drivers para el uso de MySQL.

```
FROM php:8.2-alpine3.17
RUN docker-php-ext-install mysqli pdo pdo_mysql && docker-php-ext-enable mysqli pdo pdo_mysql
WORKDIR /var/www/html
```

Para el servicio “db” la tenemos un volumen para persistir los datos y en este caso no se basa en una imagen propia, sino que se utiliza la imagen oficial de mysql como se puede apreciar en el documento docker-compose.yml del Proyecto.

```
services:
  php:
    build: ./docker/php
    user: 1000:1000
    depends_on:
      - db
    volumes:
      - ./:/var/www/html
    ports:
      - "8080:8000"
    networks:
      - fct
    command: "php artisan serve --host 0.0.0.0"

  db:
    image: mysql
    restart: always
    ports:
      - "8081:3306"
    environment:
      MYSQL_ROOT_PASSWORD: crafty
      MYSQL_DATABASE: crafty
      MYSQL_USER: crafty
      MYSQL_PASSWORD: crafty
    volumes:
      - "./docker/db/data:/var/lib/mysql"
    networks:
      - fct

networks:
  fct:
    name: fct
```

En este archivo podemos ver los servicios antes mencionados, la dependencia existente entre ellos, los volúmenes y los puertos donde se escucha a cada contenedor, las variables de entorno de la base de datos y se establece una red común para los contenedores llamada “ftc” que será la misma red donde se ejecuta el contenedor del entorno cliente.

VII.2.3 Entorno cliente:

Para el entorno cliente preparé un único archivo docker-compose.yml con un único servicio que se basa en una imagen de “node:18.15.0-alpine3.17”, al igual que en los contenedores

del entorno servidor, tenemos un volumen, puertos de escucha y una red común con el entorno servidor.

Con la red común aíslo los contenedores de la red principal y permito la comunicación entre los contenedores.

```
1
2 services:
3   node:
4     image: node:18.15.0-alpine3.17
5     volumes:
6       - ./:/app
7     working_dir: /app
8     ports:
9       - "3000:3000"
10    networks:
11      - fct
12    command: "npm start"
13 networks:
14   fct:
15     external: true
16
```

VII.2.4 Start-up:

Para arrancar el Proyecto, cada una de sus partes, utilizo los comandos propios de docker, pero a través de un archivo MakeFIle que facilita el despliegue de los contenedores. Un archivo MakeFile en cada subproyecto con los comandos de creación, arranque, para entrar la terminal, detener los contenedores y para ver los logs.

VII.2.5 Manual de instalación

Instalar y desplegar el Proyecto es bastante sencillo gracias al uso de contenedores dockers. Los pasos para instalar el Proyecto son los siguientes:

1. Instalar Docker
2. Clonar el repositorio
3. Configurar el entorno
4. Construir y desplegar los contenedores Docker para Node.js, PHP y MySQL: ejecutar el comando `make start` para iniciar los contenedores Docker en segundo plano y recrearlos si es necesario.
5. Acceso a la plataforma web: ingresando la URL del servidor Node.js se podría ver la interfaz de la plataforma, lista para su uso.
6. Verificar el historial de registros: con el comando “make logs” puede seguirse el registro de los contenedores.
7. Acceso a la terminal de los contenedores con el comando “make shell”.
8. Detener o eliminar los contenedores con el comando “make down”

VIII. Herramientas de apoyo.

VIII.1 Control de versiones

La herramienta de control de versiones utilizada durante el desarrollo del Proyecto fue GitLab. A través del siguiente link puede accederse al repositorio (con aprobación previa):

https://gitlab.com/mayra28mos/pfc.git.

a. Visual Studio Code

El editor de código utilizado para el desarrollo de la aplicación fue Visual Studio Code (VSC). Además, también aproveché alguna de sus extensiones como ayuda. Entre estas extensiones destacan:

- “SQL Server CLient(mssql)”: extensión que permite visualizar y manejar la base de datos desde VSC de forma sencilla y ágil.
- Source Controll
- Laravel Extra Intellisense
- PHP Intelephense

- GitHub Copilot

b. Bootstrap 5
c. Postman: herramienta de desarrollo de API que permite realizar solicitudes HTTP, probar y depurar APIs de forma eficiente.
d. Stripe: plataforma de pagos en línea que facilita la integración de pagos seguros en aplicaciones y sitios web.
e. Formiik: plataforma de gestión de formularios y procesos que simplifica la captura de datos, automatiza flujos de trabajo y mejora la eficiencia operativa.
f. JsDoc: herramienta de generación de documentación automática para proyectos de código abierto en JavaScript, que facilita la documentación del código y su comprensión por parte de los desarrolladores.
g. Visual Paradigm Online: Con esta aplicación se genera de forma gratuita alguno de los diagramas de la memoria.
h. draw.io: Con esta aplicación se genera de forma gratuita alguno de los diagramas de la memoria.
i. Balsamiq Wireframes: Con esta aplicación creé los bocetos del Proyecto.
j. Google
k. OpenIA (2023) Chat GPT.

IX. Conclusiones.

IX.1 Conclusiones sobre el trabajo realizado

El proyecto cumple con la mayoría de los objetivos planteados, aunque no de forma completa. Pero, en conclusión, he podido completar y entregar una web funcional completa e integrada con varias tecnologías. La implementación de funcionalidades como el registro de usuarios y vendedores, el área de perfil para ambos roles, la publicación y venta de

artículos, y la integración de una plataforma de pagos demuestran el alcance y la complejidad del proyecto.

IX.2 Conclusiones personales

Durante el desarrollo de este proyecto, mi objetivo principal era aplicar los conocimientos y habilidades adquiridas durante el Curso, así como familiarizarme con las herramientas utilizadas en el mismo. Además, tenía en mente que estas tecnologías son altamente demandadas en el mercado laboral actual, por lo que quería aprovechar la oportunidad para fortalecer mis conocimientos, ampliarlos y mejorar mis habilidades en general.

Durante el proceso, pude experimentar y comprender a fondo el funcionamiento de los contenedores Docker, algo que previamente solo había explorado en clase. Descubrí el poder y la utilidad de los archivos "MikeFile", los cuales simplificaron enormemente el despliegue del proyecto. En este sentido, me siento satisfecha con mi progreso en el manejo de estas tecnologías.

En cuanto a la división del proyecto en backend y frontend, realicé un análisis del mercado laboral y pude confirmar que las empresas demandan el uso conjunto de ambas tecnologías, lo cual permite una mejor organización de los equipos, mayor agilidad y un mejor rendimiento en el backend al no tener que manejar las peticiones de renderizado, ya que esto queda a cargo de React y su virtual DOM. Por lo tanto, mi objetivo de mejorar mis habilidades en ambos aspectos se vio cumplido.

En relación al backend, uno de mis principales objetivos era implementar mi propia API y seguir ampliando mis conocimientos en el uso del framework Laravel y el lenguaje PHP.

Aunque enfrenté algunas dificultades propias de trabajar con una API, estoy satisfecha con el resultado alcanzado, ya que logré cumplir con mi objetivo.

En cuanto al manejo de bases de datos utilizando Eloquent, considero que he mejorado significativamente mis habilidades en este ámbito, por lo que puedo dar por cumplidos los objetivos que me había planteado.

En el ámbito del frontend, opté por utilizar la librería de React, lo cual me permitió practicar el uso de JavaScript en una aplicación de página única (SPA), así como familiarizarme con el manejo de funciones asíncronas y las solicitudes a la API, junto con el manejo de las respuestas. Estoy satisfecha con mi progreso en estos aspectos, ya que pude practicar y mejorar estas habilidades al desarrollar una aplicación completa y funcional.

La implementación específica de React fue mi objetivo más ambicioso y, a la vez, el más desafiante. Invertí una gran cantidad de tiempo tanto en aprender a utilizarlo como en solucionar los errores que surgieron durante el proceso, ya sean relacionados con el funcionamiento del código o con problemas de compilación debido al uso del eslint de React. A pesar de las dificultades encontradas, estoy satisfecha con el resultado final.

Por último, es importante mencionar que no pude cumplir todos los objetivos que me había propuesto en términos de la funcionalidad de la web. Esto se debe principalmente a la complejidad y extensión del proyecto, así como a las dificultades inherentes a la integración de múltiples herramientas con las que tenía poca experiencia previa. Gran parte de mi tiempo se destinó a estudiar e investigar cómo utilizar estas herramientas, comprender sus particularidades y resolver errores.

En conclusión, puedo destacar que durante todo el desarrollo del Proyecto estuve en constante aprendizaje y mejora. En varias ocasiones actualicé y refactoricé mi código a medida que adquiría nuevos conocimientos y descubría formas más eficientes de trabajar. Aunque no logré alcanzar todos los objetivos planteados en cuanto a la funcionalidad de la web, considero que el desarrollo del Proyecto fue una experiencia enriquecedora que me permitió consolidar mis habilidades y ampliar mis conocimientos en las tecnologías utilizadas.

IX.3 **Posibles ampliaciones y mejoras**

Varios de los objetivos iniciales de Crafty no han podido cumplirse antes de finalizar el plazo de entrega del proyecto, entre ellos, la supervisión previa en el proceso de alta de los vendedores dentro de la propia web, así como los filtros necesarios para publicar productos en venta, el desarrollo completo del área de compradores y algunas de las funciones del área de vendedores como las de actualizar sus productos, variar el precio según la oferta y la demanda, realizar campañas de rebajas individuales y se verán reflejadas y mejor posicionadas en la web.

Aunque no estaba previsto en la propuesta del proyecto, también pretendía tener espacios en la web donde se pudiese obtener y compartir información sobre medioambiente y sostenibilidad, e incluso áreas de "*Do Your Self*" que enseñen a los usuarios a hacer productos ellos mismos.

También pretendía permitir a los usuarios dejar comentarios en los productos después de comprarlos e incluso que pudiesen enviar mensajes a los vendedores de forma previa a la compra para poder informar o resolver dudas sobre los productos y su fabricación.

Por lo que, teniendo en cuenta lo anterior, las ampliaciones y mejoras previstas serían las siguientes:

- Incorporar en Laravel un Panel de Administración que permita gestionar los trámites previos y posteriores al registro de vendedoras de forma más eficiente y automatizada. En el que se pueda supervisar mejor a este tipo de usuario.
- Incorporar los filtros previos para darse de alta como vendedor.
- Establecer directamente en la web los filtros necesarios para publicar productos.
- Permitir a los usuarios dejar comentarios y calificaciones en los productos adquiridos.
- Permitir a los usuarios modificar los productos que se encuentran en la cesta de la compra en la página de pago.
- Habilitar un chat entre compradores y vendedores.

- Mejorar las áreas de perfil, principalmente las de los vendedores para que puedan ser visitadas por los compradores.
- Mejorar el panel con las órdenes de compra y venta. Que sea más dinámico y facilita mayor información y acciones a los usuarios.
- Permitir que los vendedores gestionen sus ventas directamente sin la intervención de Crafty.
- Habilitar el registro de métodos de pago a los compradores y que puedan elegir entre estas opciones a la hora de pagar.
- Mejorar el diseño de la página de los detalles de los productos.
- Permitir a los usuarios no registrados o no logueados añadir productos a la cesta de la compra y poder realizar pagos.

X. Bibliografías

- Laravel (2023). Laravel documentation. Recuperado de https://laravel.com/docs/10.x
- PHP (2023). PHP documentation. Recuperado de https://www.php.net/docs.php
- Redux-Toolkit (2023). Redux-Toolkit documentation. Recuperado de https://redux-toolkit.js.org/
- Formik React (2023). Documentación recuperada de https://formik.org/docs/guides/react-native
- React (2023). React documentation. Recuperado de https://react-bootstrap.github.io/
- OpenAI (2023). Respuestas proporcionadas por ChatGPT.