

VILLAGE STORE



Deivyth Elian Sarchi Mena

2º DAW 2021 - 2022

IES La Vereda

Contenido

1 INTRODUCCION

1.1 MODULOS A LOS QUE IMPLICA

1.2 BREVE DESCRIPCION DEL PROYECTO

2 MANUAL TECNICO

3 ESTUDIO PREVIO

3.1 ESTUDIO DE SOLUCIONES EXISTENTES

4 PLAN DE TRABAJO

4.1 CALENDARIO DE OBJETIVOS

4.2 MODELO ENTIDAD RELACION

4.3 MOCKUP

7 DISEÑO

8 IMPLANTACION

9 RECURSOS

10 CONCLUSIONES

10.1 GRADO DE CONSECUCION DE OBJETIVOS

10.2 PROBLEMAS ENCONTRADOS

10.3 MEJORAS

11 ANEXOS Y DOCUMENTOS COMPLEMENTARIOS

12 REFERENCIAS / BIBLIOGRAFIA

1 INTRODUCCION

1.1 MODULOS A LOS QUE IMPLICA

En un principio iba a realizar la aplicación con lo aprendido en Desarrollo Web en Entorno Servidor utilizando el Framework de Symfony el cual te facilita mucho la obtención de datos de la base de datos. También al crear la interfaz con los ficheros Twig se puede hacer vistas dinámicas, heredar código entre las diferentes paginas HTML para ahorrar en la escritura de código que algo que es bastante interesante.

Por lo que en este caso tendría tanto el Front como el Back en una misma aplicación. Lo cual me parece una mala práctica, porque a la hora de querer modificar partes del código según como lo hayas planteado podría ser un trabajo bastante tedioso. También tenía pensado en hacer una API con Symfony pero al no tener tantos conocimientos ni soltura descarte la idea.

Por lo que me decante en utilizar Java Spring y Angular, así en el Front a la hora de modificar aspectos de mi página me despreocupo por si puedo tocar algún archivo de la parte del Back y en la parte del Back pues solo me tengo que preocupar de como enviar los datos a la parte del Front y gestionar los datos que me pase el Front, en caso de querer añadir más funcionalidad lo tendría todo mucho más focalizado.

Por la parte de Diseño de Interfaces Web, he utilizado la famosa biblioteca de Bootstrap para darle responsive a la página web y además con la ayuda de SASS he podido personalizar los estilos predeterminados de esta biblioteca.

En desarrollo Web en Entorno Cliente he utilizado Angular con su respectivo lenguaje de TypeScript. Con este módulo consigo tener un código mejor estructurado, no realizar tantas cargas de la página web y utilizar las diferentes herramientas que te ofrece Angular para facilitar la creación del código.

Por último, me decante por utilizar Java Spring ya que tiene un servidor de Tomcat ya integrado, si quieres te puede generar un .war o .jar para poder desplegarlo. También puedes implementar diferentes módulos para hacer más robusta tu aplicación, aparte de facilitarte mucho la creación de código.

1.2 BREVE DESCRIPCION DEL PROYECTO

En esta aplicación trata de una tienda online en la que cualquier persona puede vender sus productos siendo tanto un pequeño negocio el cual quiere dar a conocer alguno de sus productos en la tienda o una persona que quiera deshacerse de algún artículo que no le de uso.

En la página una vez registrados podremos crear objetos, para así ponerlos en venta. También, tendremos un apartado el cual listara en una tabla todos los objetos subidos a la página y así tener una gestión de los diferentes artículos.

Por otra parte, tendremos un carrito de la compra el cual se guarda para que en caso de entrar desde otro dispositivo podamos continuar nuestra compra. Una vez tengamos decididos los artículos que queremos comprar, nos creara un pedido del cual tendremos una factura. Todo esto lo podremos mirar en los diferentes apartados de la aplicación.

2 MANUAL TECNICO

La parte del Front se ha realizado con Visual Studio Code utilizando el Framework de Angular y la librería de PrimeNg para utilizar componentes como Autocomplete o Toast Module. Después, para realizar el responsive y los estilos de la página me he decantado por coger el Framework de estilos Bootstrap. Con la ayuda de SASS he podido modificar los valores por defecto de Bootstrap y así darle un toque más personalizado a la página web.

Para poder trabajar con la parte del Front lo único que necesitaremos es irnos a la página de <https://nodejs.org/es/> e instalarnos la versión LTS de Node. A continuación, lo que aremos es escribir en nuestro cmd `npm install -g @angular/cli`. Con estos dos pasos ya podríamos trabajar con Angular, en caso de no tener disponible la parte del Back. Nos ayudaremos la herramienta de Mockoon en <https://mockoon.com/>, para simular nuestra API.

En el Back se ha realizado con IntelliJ IDEA Community utilizando Java Spring con las dependencias de Spring Web, Spring Data JPA, Spring Security, Json Web Token, Mapstruct, Validation. Con todas ellas he creado una API con diferentes Uris para acceder a las diferentes entidades y funciones.

Para poder trabajar con nuestra API se recomienda utilizar la IDE anteriormente mencionada, ya que una vez de la clonas de Git Hub. Al abrirla con nuestra IDE de descarga todos los paquetes necesarios para trabajar la API. En caso de no disponer de la parte del Front lo que podemos utilizar para ver qué la API nos muestra lo deseado es utilizar la herramienta de Postman en <https://www.postman.com/>, aquí

podremos llamar a las diferentes Uris definidas en nuestra API. Tenemos que tener en cuenta que al utilizar Spring Security no podremos acceder a partes de API, para ello tendremos que obtener un token

3 ESTUDIO PREVIO

En esta página web queremos dar a la gente un Marketplace de fácil entendimiento y uso. Teniendo una interfaz sencilla, para que a la hora de subir un producto o a la hora de modificarlo no sea una tarea tediosa. También tendremos bastantes funciones que pueden ser útiles a la hora de gestionar tu almacén o buscar los productos que cada uno desee.

3.1 ESTUDIO DE SOLUCIONES EXISTENTES

Para realizar lo mencionado anteriormente ha habido un estudio previo sobre el Framework de Angular, en el cual tenemos diferentes componentes con sus respectivos servicios los cuales harán que por la parte del código este todo mejor estructurado. Y para la parte del cliente a la hora de cambiar de página o incluso para mostrar información de un artículo no estemos recargando toda la página, si no el componente que hemos declarado con anterioridad. Así habrá más fluidez para el usuario.

En caso del Back por lo que he podido ver, la mejor opción es hacer un API REST con el Framework de Java Spring que nos facilita la creación de factorías, controladores, entidades, servicios, el mapeado de una entidad a un DTO y las diferentes relaciones que hay en nuestra base de datos. Así podemos declarar una URI, que con los diferentes verbos de HTTP (GET, POST, PUT, PATCH, DELETE, OPTIONS) podremos ejecutar las diferentes reglas de negocio para las diferentes entidades que se declaren en la parte del Back.

Para probar dicha API he utilizado el programa Postman, guardando una serie de URIs para probar las diferentes funcionalidades de la API ya sea para crear objetos, obtener una lista de ellos o para la modificación de los mismos.

4 PLAN DE TRABAJO

4.1 CALENDARIO DE OBJETIVOS

CALENDARIO DE MARZO

VILLAGE STORE

SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 5
	PLANTEAR IDEAS. ORGANIZAR TIEMPOS			
		CREACION DE API REST		
			IMPLEMENTACIÓN DE DIFERENTES FUNCIONES EN API REST	
				DESARROLLO DE INTERFAZ PRINCIPAL

1

¹ Plan de trabajo, primera semana

CALENDARIO DE ABRIL

VILLAGE STORE

SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 5
DESARROLLO DE INTERFAZ PRODUCTOS Y USUARIO				
	DESARROLLAR METODOS DE CREACION OBJETOS			
		CREACION DE CARRITO Y FUNCIONES		
			CREACION DE FUNCIONES PARA GESTION DE PRODUCTOS	
				COMPROBACIÓN DE LAS DIFERENTES FUNCIONES

2

² Plan de trabajo segunda semana

CALENDARIO DE MAYO

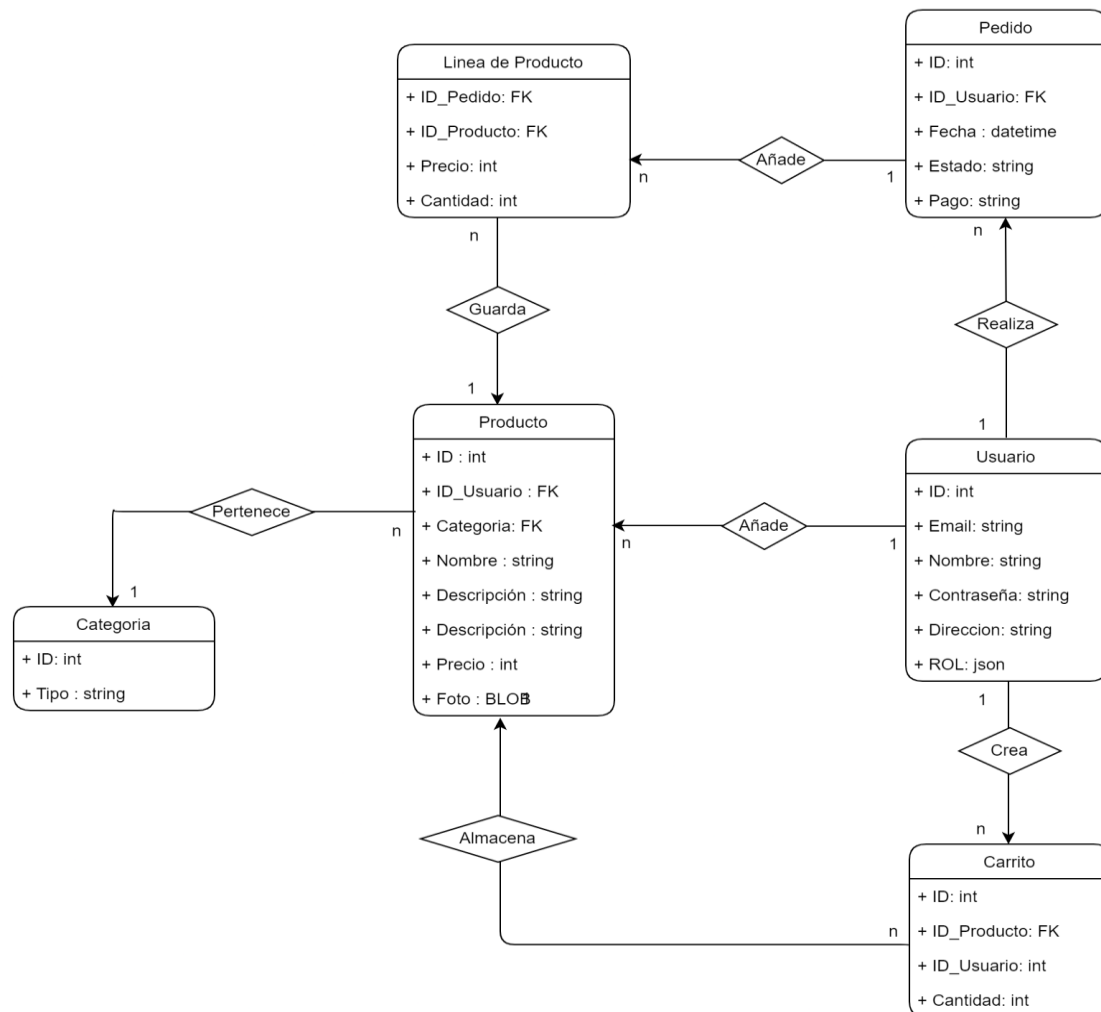
VILLAGE STORE

SEMANA 1	SEMANA 2	SEMANA 3	SEMANA 4	SEMANA 5
CREACION DE PEDIDOS Y FACTURAS				
	MEJORA DE INTERFAEZ			
		MEJORA DE METODOS EN FRONT Y BACK		
			REVISIÓN DE DOCUMENTACION	
				COMPROBACIÓN DE ERRORES

3

³ Plan de trabajo tercera semana

4.2 MODELO ENTIDAD RELACION



4

4.3 MOCKUP

⁴ Diseño de la base de datos



5 DISEÑO

Navbar sin sesión iniciada



6

Formulario de registro de usuario

Email
@ Email

Nombre
Nombre

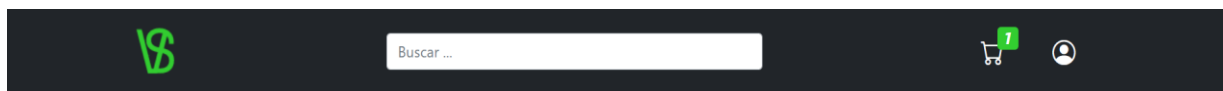
Contraseña
Contraseña

Repetir contraseña
Repetir contraseña

Registrarse

7

Navbar con sesión iniciada



8

⁶ Navbar sin sesión iniciada

⁷ Formulario de registro

⁸ Navbar con sesión iniciada

Formulario de inicio de sesión







Email
@ Email

Contraseña
Contraseña

Entrar

9

LISTA DE PRODUCTOS

 <p>La que mas mola Una y na mas 1,000,00 €</p>	 <p>Lechuga Muy fresca 10,00 €</p>	 <p>Pato NFT Mola mogollon 1,00 €</p>	 <p>Yo que se A 2,00 €</p>
 <p>Zapatillas bonitas Unas bambas muy guapas 15,00 €</p>	 <p>Zapatillas bonitas 2.0 Unas bambas muy guapas guapas</p>		

10

DETALLES DEL PRODUCTO

⁹ Formulario de inicio de sesión

¹⁰ Lista de productos



Lechuga
Muy fresca

Precio:	10,00 €
Cantidad:	<input type="text" value="0"/>
Comprar	
Añadir a la cesta	

11

Menú de usuario

Mi cuenta

[Datos](#)

[Pedidos](#)

[Cerrar sesion](#)

Productos

[Crear](#)

[Listar](#)

12

Datos del usuario

¹¹ Detalles del producto

¹² Menú del usuario

Nombre:	123AlEsconditeIngles	Cambiar
Email:	123@gmail.com	Cambiar
Contraseña:	*****	Cambiar

13

Cambio de nombre

Nombre



Cambiar

14

Cambio de email

Email



Cambiar

15

Cambio de contraseña

¹³ Datos del usuario

¹⁴ Formulario de cambio de nombre

¹⁵ Formulario de cambio de email

Contraseña actual


Contraseña

Repetir contraseña

Cambiar

16

Formulario añadir producto



Imagen

Seleccionar archivo Ninguno archivo selec.

Categoría

Nombre

Descripción

Precio
















Crear

17

Lista de productos del usuario

¹⁶ Formulario de cambio de contraseña



¹⁷ Formulario de creación de producto

Id	Imagen	Articulo	Precio	Acciones
161		Articulo	2	 
97		La que mas mola	1000	 
65		Pato NFT	1	 
163		Zapatillas bonitas	15	 
99		Zapatillas bonitas 2.0	15	 

Anterior Pagina 1 de 1 - Elementos totales: 5 Siguiente

18

Lista de pedidos

Realizado: 02-06-2022 1 articulo(s)	
Nº de pedido: 417	Factura 
<div>  <div> Lechugon 10,00 € x 10 </div> </div>	

19

6 IMPLANTACION

El desarrollo de esta aplicación se ha dividido en dos partes:

Para la parte del Back he utilizado el entorno de desarrollo de IntelliJ. En la realización del proyecto lo he dividido en diferentes entidades: los usuarios, los productos, las categorías, el carrito de la compra, línea del producto y el pedido.

En cada entidad he creado una estructura de tres carpetas:

Dominio

- Estaría la clase de la entidad y sus diferentes variables. Esta se encargará de tener la relación con la base de datos, poniendo la forma el nombre de la tabla, el nombre de las columnas y las relaciones que hay en la base de datos.

¹⁸ Lista de productos del usuario

¹⁹ Lista de pedidos del usuario


```

@Entity
public class ProductLine {

    2 usages
    | @Id
    | @GeneratedValue(strategy = GenerationType.AUTO)
    | @JoinColumn(name = "product_line_id")
    | private Long id;
    2 usages
    | @ManyToOne
    | @JoinColumn(name= "order_id", nullable = false)
    | private Order order;
    2 usages
    | @ManyToOne
    | @JoinColumn(name = "product_id", nullable = false)
    | private Product product;
    2 usages
    | @JoinColumn(nullable = false)
    | private Integer price;
    2 usages
    | @JoinColumn(nullable = false)
    | private Integer quantity;

    1 Dey
    | public Long getId() { return id; }

    1 Dey
    | public void setId(Long id) { this.id = id; }

    1 Dey
    | public Order getOrder() { return order; }

    1 Dey
    | public void setOrder(Order order) { this.order = order; }

    1 Dey
    | public Product getProduct() { return product; }

    1 Dey
    | public void setProduct(Product product) { this.product = product; }

    1 usage 1 Dey
    | public Integer getPrice() { return price; }

    1 usage 1 Dey
    | public void setPrice(Integer price) { this.price = price; }

    1 usage 1 Dey
    | public Integer getQuantity() { return quantity; }

    1 usage 1 Dey
    | public void setQuantity(Integer quantity) { this.quantity = quantity; }
    | }

```

20

Aplicación

- DTO que son los datos que queremos mostrar al cliente. Aquí se crea una clase con los datos que queremos enviar desde la API a nuestra interfaz, también podemos recibir datos de otras entidades que lo especificaremos en el mapper

```
32 usages  ⚡ Dey
public class ProductLineDTO implements Serializable {

    2 usages
    private Long id;
    2 usages
    private Long orderId;
    2 usages
    private Long productId;
    2 usages
    private String productName;
    2 usages
    private byte[] productImage;
    2 usages
    private Integer price;
    2 usages
    private Integer quantity;

    1 usage  ⚡ Dey
    public ProductLineDTO() {
    }

    ⚡ Dey
    public Long getId() { return id; }

    ⚡ Dey
    public void setId(Long id) { this.id = id; }

    1 usage  ⚡ Dey
    public Long getOrderId() { return orderId; }

    1 usage  ⚡ Dey
    public void setOrderId(Long orderId) { this.orderId = orderId; }

    1 usage  ⚡ Dey
    public Long getProductId() { return productId; }
```

```

1 usage  ± Dey
} public void setProductId(Long productId) { this.productId = productId; }

1 usage  ± Dey
} public Integer getPrice() { return price; }

± Dey
} public String getProductName() {
    return productName;
}

± Dey
} public byte[] getProductImage() {
    return productImage;
}

1 usage  ± Dey
} public void setProductImage(byte[] productImage) {
    this.productImage = productImage;
}

1 usage  ± Dey
} public void setProductName(String productName) { this.productName = productName; }

1 usage  ± Dey
} public void setPrice(Integer price) { this.price = price; }

1 usage  ± Dey
} public Integer getQuantity() { return quantity; }

1 usage  ± Dey
} public void setQuantity(Integer quantity) { this.quantity = quantity; }
}

```

21

- Mapper que hace la conversión de Entidad a DTO i viceversa. En esta clase podemos declarar las relaciones que tenemos en entre entidades. Diciendo el nombre de la variable de nuestra DTO y a que objeto iría vinculado en la entidad.

²¹ Entidad línea de producto DTO

```

7 usages 1 implementation  Dey
@Mapper(componentModel = "spring", uses = { ProductMapper.class, OrderMapper.class })
public interface ProductLineMapper extends EntityMapper<ProductLineDTO, ProductLine> {

    1 implementation  Dey
    @Override
    @Mapping(source = "orderId", target = "order.id")
    @Mapping(source = "productId", target = "product.id")
    ProductLine toEntity(ProductLineDTO dto);

    1 implementation  Dey
    @Override
    @Mapping(source = "order.id", target = "orderId")
    @Mapping(source = "product.id", target = "productId")
    @Mapping(source = "product.name", target = "productName")
    @Mapping(source = "product.image", target = "productImage")
    ProductLineDTO toDto(ProductLine entity);

    Dey
    default ProductLine fromId(Long id) {
        if(id == null) return null;
        ProductLine productLine = new ProductLine();
        productLine.setId(id);
        return productLine;
    }
}
}

```

22

- Servicio donde tendríamos las funciones que haría nuestra entidad.

```

Dey
public interface ProductLineService {

    1 usage 1 implementation  Dey
    ProductLineDTO addProductLine(ProductLineDTO productLineDTO);

}

```

23

- Implementación del servicio donde estableceremos la conversión del DTO explicado con anterioridad y, además ejecutaremos las diferentes funciones que nos proporciona el repositorio JPA.

²² Mapper de la entidad línea producto

²³ Servicio de línea de producto

```

± Dey
@Service
public class ProductLineServiceImpl implements ProductLineService {

    2 usages
    private final ProductLineRepository productLineRepository;
    3 usages
    private final ProductLineMapper productLineMapper;

    ± Dey
    public ProductLineServiceImpl(ProductLineRepository productLineRepository, ProductLineMapper productLineMapper) {
        this.productLineRepository = productLineRepository;
        this.productLineMapper = productLineMapper;
    }

    1 usage ± Dey
    @Override
    @Transactional
    public ProductLineDTO addProductLine(ProductLineDTO productLineDTO) {
        ProductLine productLine = productLineRepository.save(productLineMapper.toEntity(productLineDTO));
        return productLineMapper.toDto(productLine);
    }
}

```

24

Infraestructura

- Un repositorio que implementa el JpaRepository, en esta clase se establecen las diferentes funciones de la base de datos. Como la búsqueda por ID, el guardado de los datos, la eliminación entre otras muchas consultas.

```

3 usages ± Dey
public interface ProductLineRepository extends JpaRepository<ProductLine, Long> {

}

```

25

- Un controlador que crea las URI y ejecuta las funciones de nuestro servicio

²⁴ Implementación del servicio de linea de producto

²⁵ Repositorio de linea de producto

```

± Dey
@RestController
@CrossOrigin
public class ProductLineController {

    2 usages
    private final ProductLineService productLineService;

    ± Dey
    @Autowired
    public ProductLineController(ProductLineService productLineService) {
        this.productLineService = productLineService;
    }

    ± Dey
    @PostMapping(value = "/users/orders/product-line", produces = "application/json")
    public ResponseEntity<ProductLineDTO> addUserOrders(@RequestBody ProductLineDTO productLineDTO) {
        ProductLineDTO productLine = productLineService.addProductLine(productLineDTO);
        return new ResponseEntity<>(productLine, HttpStatus.CREATED);
    }
}

```

26

A demás, tenemos una carpeta specs donde se ha creado un constructor para crear un filtro de los diferentes valores que tenemos en nuestra entidad de forma anexadas. Para utilizar este filtro, tendríamos que realizar una URI la cual recibirá un objeto Page y nuestro filtro

Para la parte de Front he utilizado el entorno de desarrollo de Visual Studio, ya que ahí puedes añadir extensiones para facilitar la creación de código en Angular. En este caso he hecho algo parecido a lo anterior.

He creado diferentes carpetas una para las vistas compartidas, como puede ser el footer y el navbar. También hay otra que guarda PIPES, en este caso el pipe lo que haría es dar formato al dinero.

En otra carpeta llamada como su entidad tenemos guardado:

Modelo

Entidad - en caso de necesitar añadir funcionalidad o seguridad a nuestro objeto definiríamos una clase con el nombre de la entidad. También añadiríamos sus respectivos getter y setters para poder interactuar con la clase.

²⁶ Servicio de línea de producto

```

export class Product{

    private id: number | undefined;
    private name: string | undefined;
    private price: number;
    private supplierId?: number | undefined;
    private categoryId?: number;
    private supplierName?: string;
    private categoryName?: string;
    private description?: string;
    private image?: string;

    constructor(
        id: number | undefined,
        name: string,
        price: number,
        supplierId?: number,
        categoryId?: number,
        supplierName?: string,
        categoryName?: string,
        description?: string,
        image?: string
    ) {
        this.id = id
        this.name = name
        this.price = price
        this.supplierId = supplierId
        this.categoryId = categoryId
        this.supplierName = supplierName
        this.categoryName = categoryName
        this.description = description
        this.image = image
    }
}

```

27

Interfaz - con esta interfaz cogeremos los datos del servicio para así cuando lo cojamos en el TypeScript del componente podamos introducirlos en nuestra clase.

```

export interface IProduct{
    id: number;
    name: string;
    price: number;
    supplierId?: number;
    categoryId?: number;
    supplierName?: string;
    categoryName?: string;
    description?: string;
    image?: string;
}

```

28

²⁷ Clase producto en Angular

²⁸ Interfaz de producto en Angular

Servicio - en esta clase obtendremos los datos de nuestra API, haciendo la llamada a las diferentes URIs que hayamos creado en ella.

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { IProduct } from './product.interface';
import { Product } from './product.model';
import { environment } from 'src/environments/environment';

@Injectable({
  providedIn: 'root'
})
export class ProductService {

  productURL = environment.productURL;

  constructor(private http: HttpClient) { }

  getUserProducts(page: number, size: number, sort: string, filters?: string): Observable<IProduct[]> {
    let urlEndPoint: string = this.productURL+"?page=" + page + "&size=" + size + "&sort=" + sort;
    if(filters) {
      urlEndPoint = urlEndPoint + "&filter=" + filters;
    }
    return this.http.get<IProduct[]>(urlEndPoint);
  }

  getProduct(idProduct: number): Observable<IProduct>{
    return this.http.get<IProduct>(this.productURL+"/"+ idProduct);
  }

  insertItem(product: Product) {
    return this.http.post<Product>(this.productURL, product);
  }

  updateItem(product: Product) {
    return this.http.patch<Product>(this.productURL, product);
  }

  public deleteItem(productIdToDelete: number): Observable<any> {
    return this.http.delete<any>(this.productURL+"/"+ productIdToDelete);
  }
}
```

29

componente.ts


```
import { Component, OnInit } from '@angular/core';  
import { IProduct } from '../product.interface';  
import { Product } from '../product.model';  
import { ProductService } from '../product.service';  
  
You, 3 weeks ago • Create Village Store structure  
  
You, last week | 1 author (You)  
@Component({  
  selector: 'app-product-list',  
  templateUrl: './product-list.component.html',  
  styleUrls: ['./product-list.component.scss']  
})  
export class ProductListComponent implements OnInit {  
  
  products: Product[] = [];  
  
  userId?: number;  
  title: string = "";  
  
  page: number = 0;  
  size: number = 20;  
  sort: string = "name,asc";  
  
  first: boolean = false;  
  last: boolean = false;  
  totalPages: number = 0;  
  totalElements: number = 0;  
  
  nameFilter?: string;  
  priceFilter?: number;  
  
  itemIdToDelete?: number;  
  
  constructor(  
    | private productService:ProductService  
  ) { }  
  
  ngOnInit(): void {  
    | this.getAllItems();  
  }  
  
  public nextPage():void {  
    | this.page = this.page + 1;  
    | this.getAllItems();  
  }  
}
```

```

public previousPage():void {
  this.page = this.page - 1;
  this.getAllItems();
}

private getAllItems(): void {

  const filters:string | undefined = this.buildFilters();

  this.products = [];
  this.productService.getUserProducts(this.page, this.size, this.sort, filters).subscribe({
    next: (data: any) => {
      data.content.forEach((IPProduct : IPProduct)=> {
        let product = new Product([IPProduct.id,
          IPProduct.name,
          IPProduct.price,
          IPProduct.supplierId,
          IPProduct.categoryId,
          IPProduct.supplierName,
          IPProduct.categoryName,
          IPProduct.description,
          IPProduct.image]);
        this.products.push(product);
      })
      this.first = data.first;
      this.last = data.last;
      this.totalPages = data.totalPages;
      this.totalElements = data.totalElements;
    },
    error: (err) => {
      this.handleError(err);
    }
  })
}

private buildFilters():string|undefined {
  const filters: string[] = [];

  if(this.nameFilter) {
    filters.push("name:MATCH:" + this.nameFilter);
  }

  if (this.priceFilter) {
    filters.push("price:LESS_THAN_EQUAL:" + this.priceFilter);
  }

  if (filters.length >0) {
    let globalFilters: string = "";
    for (let filter of filters) {
      globalFilters = globalFilters + filter + ",";
    }
    globalFilters = globalFilters.substring(0, globalFilters.length-1);
    return globalFilters;
  } else {
    return undefined;
  }
}

```

30

³⁰ Lista del componente de producto

component.scss

```
.botoneraPaginacion {
  width: 100%;
  text-align: center;
  margin-top: 20px;
}

.botoneraPaginacion button, .botoneraPaginacion span {
  margin-left: 10px;
}
```

31

Component.html

```
<main class="container">
  <section class="row">
    <app-product-card class="col-lg-3 col-md-4 col-sm-6 col-12 mt-2" *ngFor="let product of products" [product]="product"></app-product-card>
  </section>

  <div class="botoneraPaginacion">
    <button [disabled]="first" class="btn btn-secondary btnAtras" (click)="previousPage()">Anterior</button>
    <span>Pagina {{(page+1)}} de {{totalPages}} - Elementos totales: {{totalElements}}</span>
    <button [disabled]="last" class="btn btn-secondary btnSiguiente" (click)="nextPage()">Siguiente</button>
  </div>
</main>
```

32

7 RECURSOS

Herramientas hardware

Fabricante del sistema: MSI

Modelo del sistema: MS-7A62

BIOS: BIOS Date: 11/30/16 13:21:44 Ver: V2.00 (type: BIOS)

Procesador: Intel(R) Core(TM) i5-6600K CPU @ 3.50GHz (4 CPUs), ~3.5GHz

Memoria: 8192MB RAM

Memoria del sistema operativo disponible: 8150 MB RAM

³¹ Estilos de la lista de producto

³² Componente HTML para listar productos

Herramientas de software

Sistema Operativo

- Windows 10 Pro 64-bit
- Ubuntu

IDEs

- IntelliJ IDEA Community
- Visual Studio

Herramientas

- Moockoon
- Postman

Frameworks

- Java Spring
- Angular
- PrimeNG
- Bootstrap

8 CONCLUSIONES

8.1 GRADO DE CONSECUCION DE OBJETIVOS

Los objetivos que se querían alcanzar son:

- Creación de usuarios, con una contraseña encriptada y la hora de navegar por la página web que la acredite un token para la máxima privacidad del usuario.
- Un usuario pueda cambiar de nombre, email o de contraseña.
- Creación de un carrito de la compra, el cual no se borre en caso de cerrar sesión o cambiar de dispositivo.
- Creación, modificación y listado de los artículos de un usuario.
- Crear una interfaz que sea paginable, con su respectiva información.
- Tramitar pedidos.
- Crear una línea producto para almacenar el precio de compra de cada artículo, la cantidad que ha comprado y ver los detalles del pedido.

8.2 PROBLEMAS ENCONTRADOS

En primer lugar, la única forma que nos enseñaron para crear usuarios y su posterior manejo fue a través del framework de symfony con PHP. Al no utilizar este Framework, he tenido varios problemas para la gestión de estos mismos.

A la hora de la creación del carrito he tenido que crear una relación Many to Many con clave compuesta. Ya que no tendría sentido que cada artículo de nuestro carrito tuviera un Id propio, ya que en un carrito una persona puede guardar varios productos, pero sin ser repetidos. Para esto tuve que estudiar un poco como sobre crear este tipo de relaciones en Java Spring.

También a la hora de crear una interfaz paginable, he tenido que buscar información de cómo hacer esto posible. Al principio parecía que iba a ser más complicado, pero gracias al Framework de JavaSpring esta tarea sería más sencilla. Lo único que me faltaba por aprender es como tratar la información que envió desde la API en mi interfaz.

8.3 MEJORAS

Tengo objetivos pendientes que por falta de tiempo y de conocimientos he tenido que dejar de lado para así poder realizar la entrega a tiempo.

Entre ellas esta añadir una forma de pago, para que la gente pueda comprar en la tienda sin tener acercarse al local para completar la compra. Esta idea la he tenido que descartar, ya que para poder implementarla tendría que estudiar las pasarelas de pago y después escribirlas en mi código

También me hubiera gustado implementar un sistema de envío de paquetes, en el cual quede constancia de cómo va tu pedido desde que realiza hasta cuando ya ha sido entregado. Pudiendo cancelar el pedido o tramitar una devolución. En este caso no he podido hacerlo por falta de tiempo.

9 ANEXOS Y DOCUMENTOS COMPLEMENTARIOS

La parte más compleja de mi aplicación es la forma de autenticación que implementado ya que consta de dos implementaciones de Java Spring las cuales he tenido que investigar por mi cuenta más y la forma de procesador los datos en desde el cliente. He elegido utilizar un generador de tokens generados por JWT <https://jwt.io/>, para crearlo he seguido la misma estructura de carpetas ya mencionada. He creado varias entidades que serían una para el crear el usuario, otro para el verificar el usuario y por último el token.

En este caso dentro del api crearíamos un intermediario que verificaría que las peticiones desde el cliente tengan un token verificado. En caso de no ser así, no podríamos acceder a la API. Para obtener el token lo que haríamos es enviar el email y la contraseña, en caso de ser correcto nos del volvería un token con su respectiva información.

Este token lo almacenamos en navegador con ayuda de TypeScript, para así cada vez que hagamos una petición a la API enviamos dicho token y así poder acceder a nuestro Back

Repositorios de GitHub donde se almacenan los archivos de la aplicación:

- <https://github.com/Deivyth/VillageStoreBack.git>
- <https://github.com/Deivyth/VillageStoreFront.git>

9.1 MANUAL DE USUARIO

Este proyecto consta de una pantalla principal la cual consta de tres partes:

Un navbar el cual tiene dos modos si no has iniciado sesión te mostrara dos botones en la parte derecha para iniciar sesión o crear una cuenta. Si ya hemos iniciado sesión, nos mostrara nuestro carrito de la compra y un icono para mostrar las diferentes opciones del usuario.

Entre ellas tendremos ver los datos del usuario, el cual nos enviara una ventana donde tendremos información del usuario. Como el nombre, el correo electrónico y la contraseña. Todos estos campos los podremos modificar haciendo clic en el botón de cambiar que tiene al lado de esta información. Si le damos clic nos llevara a un pequeño formulario donde podremos cambiar el valor de estos campos de forma individual.

La siguiente función es el poder listar los pedidos que hayamos hecho dentro de la tienda. Para así tener constancia de si se ha realizado bien el pedido y obtener más información de ella. Si le damos clic a ver factura, nos podremos descargar la factura en nuestro ordenador.

Otra de las funciones sería cerrar sesión, que lo que haría es borrar los datos de la sesión y devolvernos a la página de inicio.

La siguiente función es la de crear un producto para que así sea listado en la tienda. Si le damos clic a crear, se nos abrirá un formulario el cual nos pedirá una imagen, tendrá un desplegable para poder seleccionar la categoría del producto, el nombre, la descripción y su precio. En caso de tener algún dato mal la interfaz nos avisara para así corregir los campos mal puestos.

Por último, tendremos una opción de listar nuestros productos, en el cual si le damos clic se nos abrirá una ventana con una tabla donde podremos ver todos los productos que hemos creado, con su respectiva información. Al lado de cada producto tendremos dos opciones, la primera para modificar los datos de nuestro producto y la segunda para eliminarlo. En caso de elegir la primera opción nos llevará a un formulario que tendrá los datos del producto a modificar, si queremos modificar alguno de los campos lo único que tendremos que hacer es sobrescribir los datos del

formulario y darle al botón de modificar. Si elegimos la segunda opción nos saldrá una ventana emergente para confirmar el borrado del producto, si elegimos que, si el producto será eliminado de la base de datos, eso con lleva que si algún usuario tenía ese producto guardado en su carrito este se eliminará.

El main el cual ira cambiando según las necesidades que tengamos. Ya sean las mencionada anteriormente o bien cuando entramos a la página web nos saldría todos los productos que tenemos registrados en nuestra tienda. Si le damos clic a cualquiera de nuestros productos, no saldrá la información del artículo en el main. Aquí tendremos un formulario en el cual podremos elegir la cantidad de unidades que queramos el producto y dos opciones que están deshabilitadas. Para habilitarla solo tendremos que poner como mínimo 1 producto, una vez lo hayas hecho tendremos que elegir comprar o añadir el carrito. Si le damos a comprar nos redireccionaría al carrito para realizar la compra y si le damos añadir nos dejaría en la página que estamos para así seguir comprado los productos.

Una vez en la ventana del carrito nos saldría todos los productos que hemos añadido con anterioridad y un cálculo del precio que tendremos que pagar. Si le damos al botón de comprar, se nos guardaría los datos del carrito en la base de datos en forma de pedido y nuestro carrito se borraría. Si todo ha ido bien la aplicación nos lo notificaría.

El footer que mostrara una breve descripción de la página web y sus redes sociales con un link para poder acceder a cada una de ellas.

10 LEYENDA DE IMAGENES

1. Plan de trabajo, primera semana
2. Plan de trabajo segunda semana
3. Plan de trabajo tercera semana
4. Diseño de la base de datos
5. Mookup
6. Nabvar sin sesión iniciada
7. Formulario de registro
8. Nabvar con sesión iniciada
9. Formulario de inicio de sesión
10. Lista de productos
11. Detalles del producto
12. Menú del usuario
13. Datos del usuario
14. Formulario de cambio de nombre
15. Formulario de cambio de email
16. Formulario de cambio de contraseña
17. Formulario de creación de producto
18. Lista de productos del usuario
19. Lista de pedidos
20. Entidad línea de producto
21. Entidad línea de producto DTO
22. Mapper de la entidad linea producto
23. Servicio de línea de producto
24. Implementación del servicio de linea de producto
25. Repositorio de linea de producto
26. Servicio de línea de producto
27. Clase producto en Angular
28. Interfaz de producto en Angular
29. Servicio de producto en Angular
30. Lista del componente de producto
31. Estilos de la lista de producto
32. Componente HTML para listar productos

11 REFERENCIAS / BIBLIOGRAFIA

Angular

<https://angular.io/>

Bootstrap

<https://getbootstrap.com/>

Java Spring

<https://spring.io/>

PrimeNg

<https://www.primefaces.org/>

Mockoon

<https://mockoon.com/>

Postman

<https://www.postman.com/>

Repositorios

<https://github.com/Deivyth/VillageStoreBack.git>

<https://github.com/Deivyth/VillageStoreFront.git>