

STOCK CONTROL **APLICACIÓN WEB**

PROYECTO FINAL
DESARROLLO DE APLICACIONES WEB



PAULA PAVÓN MONTAÑEZ

Desarrollo de la idea inicial	3
Tecnologías empleadas	3
Diseño	4
Bases de datos	4
Servidor	5
Cliente	6
Despliegue	6
Instalación	7
Control de versiones	9
Guías de estilo	10
Colores	12
Diseño	12
Funcionalidades	13
Entorno servidor	14
Modelos	14
Controladores	15
Repositorios	17
Seeder	17
Seguridad	18
Testeo	18
Entorno cliente	19
Manual de uso	21
Conclusiones	23
Mejoras futuras	23
Índice de imágenes	24
Bibliografía y referencias	24

Introducción

Desarrollo de la idea inicial

StockControl nació de la idea de proporcionar a pequeñas empresas y autónomos una aplicación de control de inventario que fuese fácil de usar, simple, intuitiva y adaptada a ordenadores, tablets y móviles, para que el usuario pudiese elegir en cada momento desde que dispositivo consultar el stock y que este estuviese siempre actualizado.

Bajo este pretexto se diseñó un prototipo de aplicación cuyo contenido incluiría control de stock de productos y un listado detallado de proveedores y tiendas, en caso de que la empresa tuviese inventario en más de un emplazamiento.

El listado detallado de los productos contendría datos específicos de los mismos, como podrían ser los lotes a los que pertenecen cada uno de ellos.

Para su uso, se proporcionó, aparte del listado en detalle, un formulario para añadir y otro para editar respectivamente cada apartado, es decir, productos, tiendas y proveedores.

Además, para proporcionar integridad y seguridad a la aplicación se diseñó un formulario de registro e inicio de sesión, en el que poder dar de alta a distintos trabajadores del negocio, si fuese necesario y que únicamente ellos tuviesen acceso a la lectura, edición y borrado de elementos.

Tecnologías empleadas

Para procurar que la aplicación cumpla con los estándares actuales se ha elegido realizar una API REST, esto quiere decir que tenemos una parte de la aplicación que controla el servidor y que se encarga de controlar los datos, mientras que otra parte, llamada cliente, se encarga de trabajar con las peticiones que realizan los usuarios. De esta forma, conseguimos separar ambas partes de forma que sean independientes entre sí, esto también implica que los lenguajes empleados en ambas capas sean también independientes entre sí, lo único necesario será que las respuestas que enviaremos desde cliente sean, en nuestro caso, en formato JSON.

Las API REST cuentan con una serie de métodos para trabajar, que son:

- GET: se emplea para recuperar recursos.
- POST: se utiliza para crear recursos o enviar datos a un recurso que ya existe previamente.
- PUT: utilizado para editar un recurso existente.
- DELETE: se utiliza para eliminar un recurso.

Diseño



Fig1. Logotipo Adobe



Fig2. Logotipo CSS3

Para el prototipado se ha utilizado la suite de Adobe, se han realizado wireframes del diseño provisional, así como un diseño de la página principal con la tabla de productos antes de realizar la adaptación al lenguaje CSS.

Bases de datos



Fig3. Logotipo MariaDB

La base de datos utilizada para el proyecto es MariaDB, la elección se basa en que es una de las bases de datos más utilizadas, creada por los desarrolladores originales de MySQL y es open source, es decir, de código abierto. Además, podemos indicar que tiene buenos valores en rendimiento y estabilidad.

Servidor



Fig4. Logotipo Spring Boot

La tecnología empleada para servidor es Spring Boot, se ha hecho esta elección porque Spring Boot provee de un sistema para crear aplicaciones muy sencillo, ya que es muy fácil de configurar, para ello únicamente es necesario configurar el proyecto en Maven o Gradle, Maven en nuestro caso, seleccionar el lenguaje que queramos, que en nuestro caso será Java, y añadir las dependencias que queramos emplear.

The Spring Initializr web interface for configuring a new project. It has a dark theme. On the left, there's a "Project" section with radio buttons for "Maven Project" and "Gradle Project" (selected), and "Language" with radio buttons for "Java" (selected), "Kotlin", and "Groovy". Below that is the "Spring Boot" version selection, with "2.5.4" selected. The "Project Metadata" section includes fields for "Group" (es.iesrafaelalberti), "Artifact" (stock-control), "Name" (stock-control), "Description" (Stock Control by Paula Pavon), and "Package name" (es.iesrafaelalberti.stock-control). At the bottom of this section are "Packaging" options (Jar, War selected) and "Java" version options (16 selected, 11 and 8 also visible). On the right, the "Dependencies" section lists several options: "Spring Web" (WEB), "MySQL Driver" (SQL), "Spring Data JPA" (SQL), "Lombok" (DEVELOPER TOOLS), and "Spring Security" (SECURITY). Each dependency has a brief description. At the top right of the dependencies section is a button "ADD DEPENDENCIES... CTRL + B". At the bottom of the interface are three buttons: "GENERATE CTRL + G", "EXPLORE CTRL + SPACE", and "SHARE...". A warning message at the top left states: "The following attributes could not be handled: Spring Boot 2.4.3.RELEASE is not available, 2.4.10 has been selected."

Fig5. Configuración Spring Boot

Además, Spring Boot dispone de una gran cantidad de documentación que poder consultar cuando sea necesario.

Cliente

En cuanto al cliente, se ha empleado el diseño SPA (Single Page Application), o lo que es lo mismo, aplicación de una única página, esto quiere decir que los datos se sobrescriben en nuestra aplicación de forma dinámica, en vez de tener que recargar la página cada vez que se actualizan los datos de la misma.



Fig6. Logotipo React

La tecnología empleada será React, que se define como una librería de JavaScript desarrollada por FaceBook y que nos facilitará la creación de nuestra Single Page Application.

React está basada en el uso de componentes, dichos componentes son piezas independientes que pueden ser reutilizadas las veces necesarias para crear nuestra interfaz.

React, como otras librerías similares, utilizan un componente padre, en el cual se anidaran otros componentes hijos, que a su vez, también pueden tener hijos.

Despliegue



Fig7. Logotipo Docker

El despliegue de la aplicación se realizará con Docker, ya que nos brinda la posibilidad de almacenar nuestro proyecto en diferentes contenedores, para que nuestra aplicación pueda ejecutarse en cualquier máquina con Docker

fácilmente, ya que permite crear, probar e implementar aplicaciones de forma rápida y sencilla..

Instalación

Para la instalación en Docker es necesario un archivo Dockerfile:

```
1 FROM openjdk:11
2 ADD ./server /usr/stock-control
3 WORKDIR /usr/stock-control/server
4 EXPOSE 8000
```

Fig8. Configuración Dockerfile

Este archivo contiene la versión de jdk y donde indicamos que nuestra carpeta server se copie a nuestro contenedor en la ruta especificada, a continuación nos movemos a esa carpeta para, por último, exponer el puerto 8000.

También es necesario un archivo Docker-Compose, en el que indicamos la versión, los datos de acceso a la base de datos y a phpMyAdmin.

```
1 version: '3.7'
2
3 services:
4   db:
5     image: mariadb
6     restart: always
7     ports:
8       - "5000:3306"
9     volumes:
10      - ./db-init:/docker-entrypoint-initdb.d/
11     environment:
12       MYSQL_DATABASE: db_stockcontrol
13       MYSQL_USER: root
14       MYSQL_PASSWORD: asd123
15       MYSQL_ROOT_PASSWORD: asd123
16
17   phpmyadmin:
18     image: phpmyadmin
19     restart: always
20     ports:
21       - "8081:80"
22     environment:
23       - PMA_ARBITRARY=1
24
```

Fig9. Configuración Dockercompose

```
server:
  build: .
  restart: always
  ports:
    - "8000:8000"
  depends_on:
    - "db"
  volumes:
    - "./server:/usr/stock-control/server"
  command: java -jar stock-control.jar
```

Fig10. Configuración Dockercompose Server

Dentro del server, el build se encarga seleccionar el dockerfile que hay en la ruta, a continuación indicamos que se reinicie el servidor, exponiendo los puertos 8000 local y 8000 de la máquina, indicamos también que dependa de la base de datos, si no está arrancada no arranca el servidor, por último crea un volumen en la carpeta server con la carpeta server de mi proyecto y por último ejecutamos la aplicación en java.

```
client:
  image: nginx
  restart: always
  ports:
    - "80:80"
  depends_on:
    - "server"
  volumes:
    - "./client:/usr/share/nginx/html"
```

Fig11. Configuración Dockercompose Client

Respecto al cliente, la imagen parte de Nginx, los puertos son 80:80, indicamos que dependa del servidor y creamos un volumen.

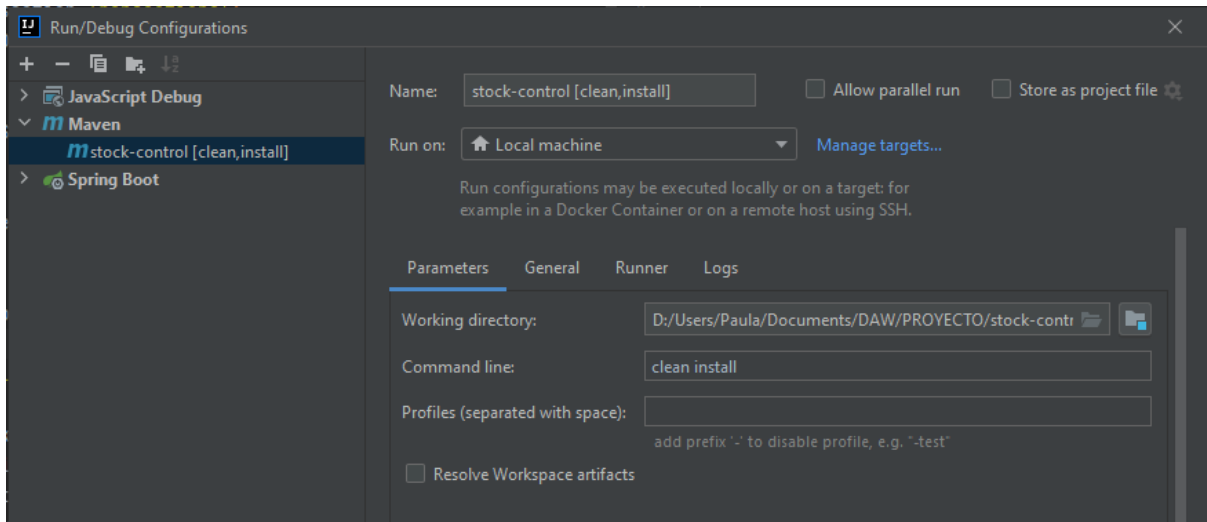


Fig12. Configuración IntelliJ para exportar JAR

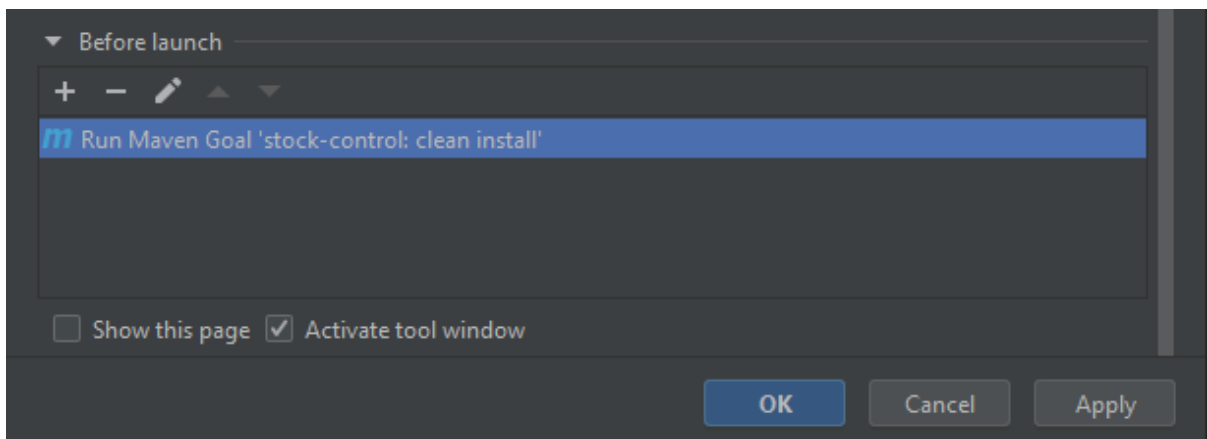


Fig13. Configuración IntelliJ para exportar JAR

Control de versiones

Respecto al control de versiones, se hará con GitHub, estando sincronizado con nuestros IDE, para ello se ha creado un repositorio nuevo en el que alojar el proyecto.



Fig14. Logotipo GitHub

GitHub es una plataforma que utiliza el sistema de control de versiones GIT, lo que nos permite trabajar y poder consultar y regresar a distintas versiones de nuestro proyecto sin perder los cambios.

Guías de estilo

Se diseñaron tres pantallas principales para la aplicación, la ventana de login y registro, el listado detallado de los productos, tiendas y proveedores y un formulario para añadir y editar productos, tiendas o proveedores.

- Formulario de login/registro

A wireframe of a login/register form. It features a header with a hamburger menu icon on the left and two window control icons on the right. The main content area contains a central form box with three input fields, a checkbox, and a submit button.

Fig15. Wireframe formulario login/registro

- Vista detallada de elementos con menú lateral

A wireframe of a table view with a sidebar. The sidebar on the left contains a hamburger menu icon and four window control icons. The main table area has a header row and 15 data rows, each starting with a checkbox. At the bottom of the table, there are two buttons.

Fig16. Wireframe tablas con menú lateral

- Formulario de añadir o editar elementos

Fig17. Wireframe formulario añadir/editar elementos

Se realizó también un diseño provisional de la pantalla principal de la aplicación, en la que se muestra una lista detallada de los productos disponibles, así como un menú para acceder a las diferentes categorías, un apartado de perfil de usuario y el modo noche integrado en la aplicación.

StockControl

Tiendas

Productos

Proveedores

Cerrar sesión

Productos											
<input type="checkbox"/>	Nombre	Marca	Peso	Stock	Fecha compra	Precio compra	IVA	Precio venta	Oferta	Descatalogado	Proveedor
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input checked="" type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA
<input type="checkbox"/>	Botines	Adidas	1kg	5	30/06/2021	20€	21%	60€	10€	No	Zapatos SA

Editar


Borrar

Fig18. Diseño de prototipo inicial

Colores

Se ha utilizado un tono azul para la aplicación, puesto que numerosos estudios demuestran que la mayoría de las personas asocian el color azul a inteligencia, estabilidad o serenidad, entre otros, así como también es utilizado por muchas marcas de tecnología.

Este tono azul se acompaña de un tono gris que, además de ser un color neutro, implica seguridad, madurez y fiabilidad.

El código hexadecimal del azul empleado es:  #1676e4

El logotipo diseñado es el siguiente:

The logo consists of the words "StockControl" in a bold, blue, sans-serif font. The word "Stock" is in a lighter blue shade, and "Control" is in a darker blue shade.

Fig19. Diseño Logotipo StockControl

Se trata de un logotipo simple que sigue la línea del diseño de la aplicación y que refleja el uso que se le da a la misma.

Se ha recurrido al uso del nombre de la aplicación directamente, en vez de trabajar con imágenes u otros recursos visuales.

Cabe destacar también que la aplicación ha sido diseñada para su uso en ordenadores pero está adaptada a otros dispositivos, como tablets o móviles.

Diseño

El diseño inicial de la aplicación y en el que se ha basado su posterior desarrollo es el siguiente:

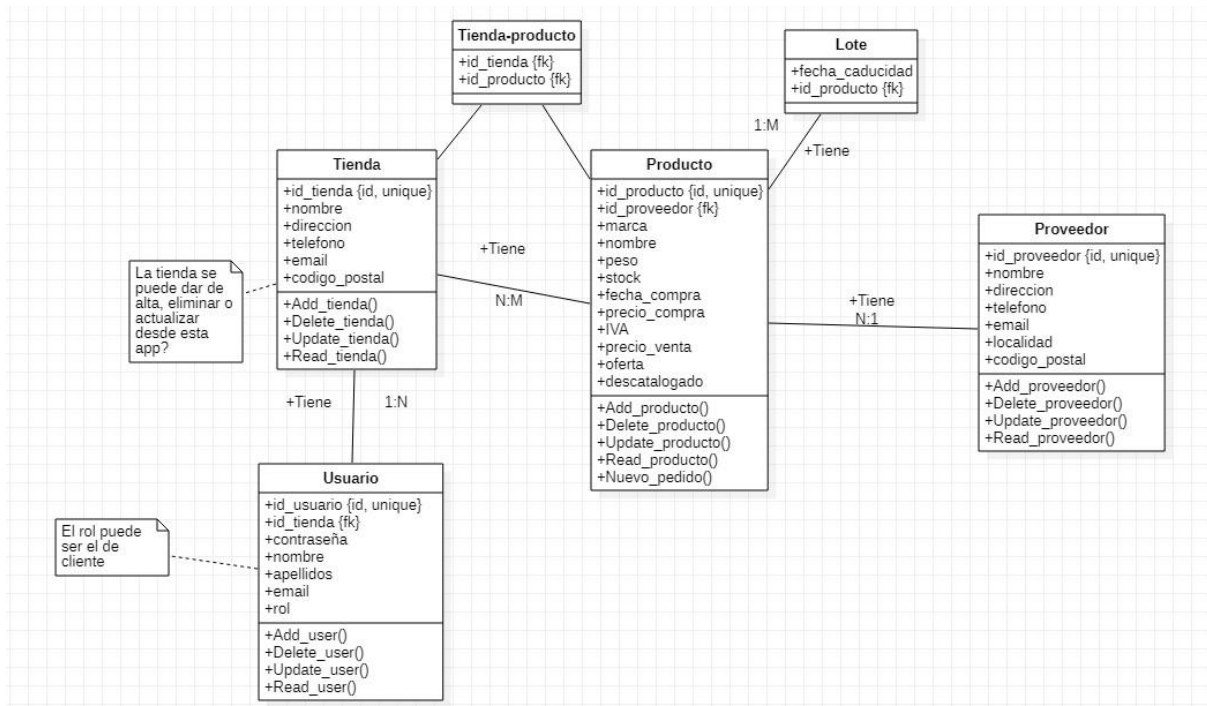


Fig20. Diseño de diagrama de clases

Descripción

Como resultado de todas las tecnologías empleadas, nos encontramos ante una aplicación funcional, pero en fase de desarrollo, donde el usuario puede registrarse, loguearse, dar de alta, eliminar, editar productos, tiendas, proveedores.

Funcionalidades

Las funcionalidades con las que contamos en la aplicación son:

- Inicio sesión
- Editar/Borrar/Añadir usuario
- Editar/Borrar/Añadir rol para usuario
- Editar/Borrar/Añadir producto
- Editar/Borrar/Añadir tienda
- Editar/Borrar/Añadir proveedor
- Editar/Borrar/Añadir lotes para productos
- Editar/Borrar/Añadir categorías para productos

Desarrollo

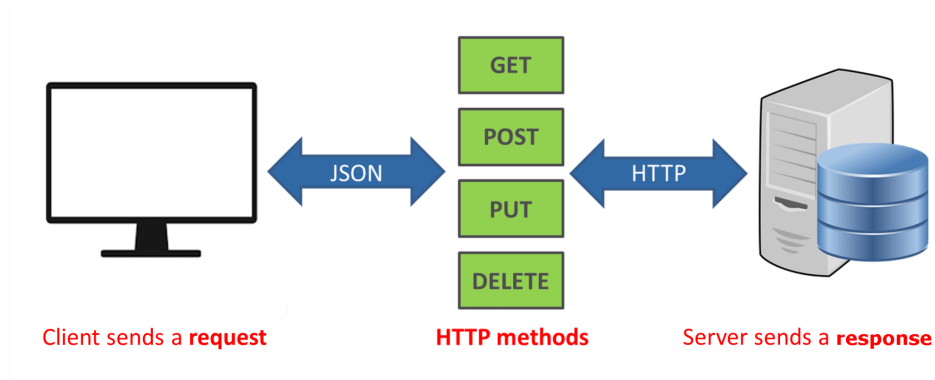


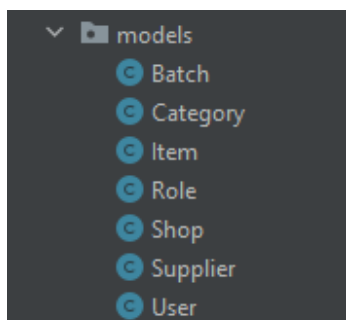
Fig21. Relación cliente servidor API REST

Como se ha mencionado anteriormente, se ha utilizado un diseño de API REST para la creación de esta aplicación. De esta forma, por un lado tenemos el servidor, por otro el cliente y la relación entre ambos se hace en forma de JSON. Como ya hemos mencionado, esto permite que la aplicación actúe de forma independiente y mejore la fiabilidad, flexibilidad, escalabilidad y reutilización.

Entorno servidor

En el entorno servidor se crearán diferentes paquetes con funcionalidades separadas, esta es la organización empleada:

Modelos



Los modelos contienen la información de los elementos de cada tabla. Es decir, para el modelo de usuario, se definen todos los parámetros que debe tener un usuario, como un email, una contraseña, nombre, etc. También se definen si esos parámetros deben ser únicos, pueden ser nulos, etc.

Fig22. Modelos

```
public class User {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
    @Column(unique = true, nullable = false)  
    private String email;  
    @Column(nullable = false)  
    private String password;  
    private String name;  
    private String surname;  
    private String token;  
}
```

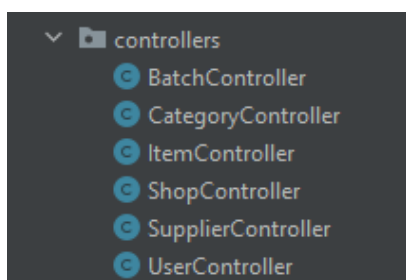
Fig23. Modelo de Usuario

Además, también se establecen las relaciones con otras tablas y su comportamiento con respecto a ellas. Por ejemplo, nuestros usuarios están relacionados con las tiendas, ya que necesitaremos saber a qué tienda pertenece cada usuario, así como también conocer el tipo de relación que tienen, si es de 1 a muchos, 1 a 1, muchos a muchos...

```
@JsonBackReference(value="user_shop")  
@ManyToOne  
private Shop shop;  
  
@ToString.Exclude  
@ManyToMany(fetch = FetchType.EAGER)  
@JoinTable(name = "usuarios_rol",  
    joinColumns = @JoinColumn(name = "id_usuario", referencedColumnName = "id"),  
    inverseJoinColumns = @JoinColumn(name = "id_rol", referencedColumnName = "id"))  
private Set<Role> roles = new HashSet<>();
```

Fig24. Ejemplo de relaciones 1:M, M:N

Controladores



Los controladores nos permiten definir la relación entre los métodos GET, PUT, POST y DELETE, la relación con su URL y los parámetros de entrada, recibir la respuesta, ejecutar su funcionalidad en el repositorio e indicar si ha sido satisfactoria.

Fig25. Controladores

```

@RestController
public class ItemController {
    @Autowired
    private ItemRepository itemRepository;

    @PostMapping("/producto/nuevo")
    public ResponseEntity<?> itemNew(@RequestBody Item newItem) {
        itemRepository.save(newItem);
        return new ResponseEntity<>(newItem, HttpStatus.OK);
    }

    @PutMapping("/producto/{id}")
    public ResponseEntity<?> itemUpdate(@PathVariable("id") long id,
                                       @RequestBody Item newItem) {
        itemRepository.findById(id).orElseThrow(EntityNotFoundException::new);
        newItem.setId(id);
        itemRepository.save(newItem);
        return new ResponseEntity<>(newItem, HttpStatus.OK);
    }

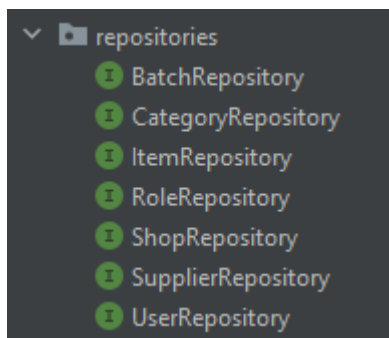
    @DeleteMapping("/producto/{id}")
    public ResponseEntity<?> itemDelete(@PathVariable("id") long id) {
        Optional<Item> oldItem = itemRepository.findById(id);
        if (oldItem.isEmpty()) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
        itemRepository.deleteById(id);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping("/productos")
    public ResponseEntity<Object> itemList(@RequestParam("page") int page, @Nullable @Re
                                       @Nullable @RequestParam("brand") String brand
        name = name == null ? "" : name;
        brand = brand == null ? "" : brand;
        return new ResponseEntity<>(itemRepository.getItemsInRangeBy(PageRequest.of(page
        HttpStatus.OK);
    }
}

```

Fig26. Controller de productos

Repositorios



Los repositorios contienen interfaces con las consultas SQL necesarias que serán ejecutadas a través de los controladores y que servirán, por ejemplo, para crear un buscador de productos cuyos nombres contengan X palabras y pertenezcan a X marcas.

Fig27. Repositorios

```
public interface ItemRepository extends CrudRepository<Item, Long> {  
    @Query("select i from Item i")  
    List<Item> getItemsInRange(Pageable pageable);  
  
    @Query("select i from Item i where i.name like %:name% and i.brand like %:brand%")  
    List<Item> getItemsInRangeBy(Pageable pageable, @Param("name") String name, @Param("brand") String brand);  
  
    @Query("select distinct i.brand from Item i")  
    List<String> getBrands();  
  
    @Query(value="delete from shops_items where item_id = ?1", nativeQuery = true)  
    Integer deleteItemRelationShopById(long id);  
}
```

Fig28. Repositorio de Productos

Seeder

Para comprobar el correcto funcionamiento de la aplicación, se ha utilizado un seeder, o lo que es lo mismo, se han creado datos para introducirlos en la base de datos cada vez que arranca la aplicación, estos datos se utilizan para poder probar que nuestra aplicación funciona correctamente.

```
@Override
public void run(String... args) {
    if (false) {
        //Create a new Category
        Category category1 = categoryRepository.save(new Category( name: "categoria1"));
        Category category2 = categoryRepository.save(new Category( name: "categoria2"));
        Category category3 = categoryRepository.save(new Category( name: "categoria3"));

        //Create a new Item
        Item item1 = itemRepository.save(new Item( brand: "Adidas", name: "item1", color: "rojo",
        Item item2 = itemRepository.save(new Item( brand: "Nike", name: "item2", color: "azul", st
        Item item3 = itemRepository.save(new Item( brand: "Reebok", name: "item2", color: "azul",
        Item item4 = itemRepository.save(new Item( brand: "Nike", name: "item3", color: "azul", st
        Item item5 = itemRepository.save(new Item( brand: "Reebok", name: "item3", color: "rojo",
        Item item6 = itemRepository.save(new Item( brand: "Nike", name: "item3", color: "verde", s
        Item item7 = itemRepository.save(new Item( brand: "Nike", name: "item3", color: "verde", s
        Item item8 = itemRepository.save(new Item( brand: "Nike", name: "item4", color: "azul", st
        Item item9 = itemRepository.save(new Item( brand: "Nike", name: "item2", color: "azul", st
```

Fig29. Seeder

Seguridad

Para la seguridad de la aplicación se ha utilizado un sistema que se encarga de las peticiones de autenticación y del guardado de un token de seguridad JWT (JSON Web Token) en una cookie, que expirará cuando el usuario cierre la sesión o un mes después del inicio de sesión.

Testeo

Se ha utilizado el programa Insomnia para hacer peticiones al servidor y comprobar que la aplicación funcione correctamente.

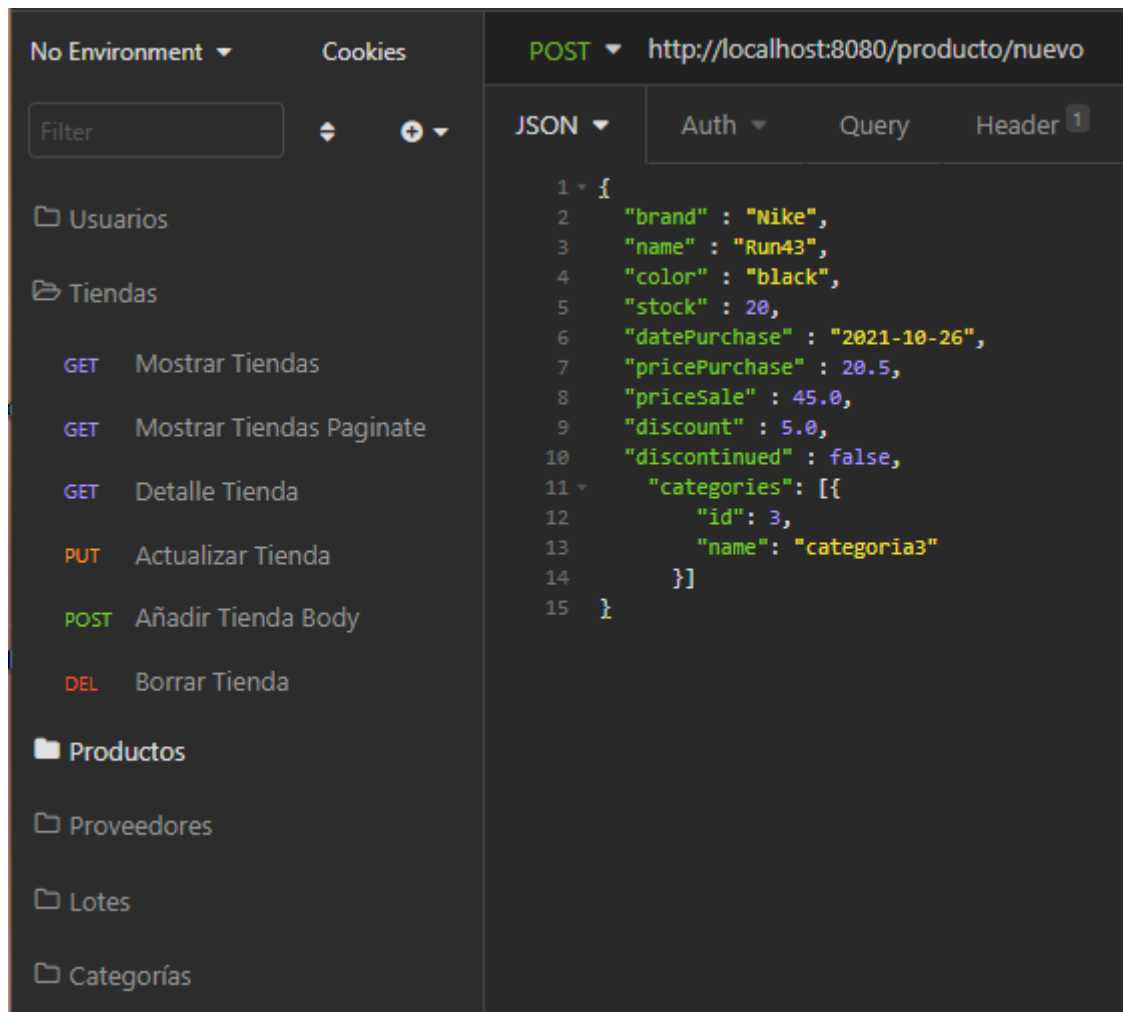


Fig30. Organización y petición Insomnia

Una vez realizadas las pruebas, se ha generado un OpenAPI con las rutas y códigos HTTP para cada una de las peticiones.

Entorno cliente

Como ya hemos visto, para el entorno cliente estamos usando React, que está desglosado en componentes, existiendo un componente principal que tiene hijos, que a su vez pueden tener otros hijos, de esta forma, utilizamos una SPA con llamadas asíncronas a los componentes.

```
function App() {
  return (
    <div className="App">
      <div id="background">
        
      </div>
      <Router>
        <Navbar />
        <Switch>
          <Route path="/" exact>
            <Login />
          </Route>
          <Route path="/register" exact>
            <Register />
          </Route>
          <Route path="/logout">
            <Logout />
          </Route>
          <Route path="/usuarios/:pagination">
            <Users />
          </Route>
        </Switch>
      </Router>
    </div>
  );
}
```

Fig31. App.jsx React

Desde el archivo App.jsx es desde donde llamamos al resto de componentes para crear nuestra SPA.



```
const Products = (props) => {
  const auth = Auth();
  const [products, setProducts] = useState(null);
  const [brands, setBrands] = useState(null);
  // Paginación
  const [name, setName] = useState("");
  const [brand, setBrand] = useState("");
  const [page, setPage] = useState(0);
  const {pagination} = useParams();

  useEffect(() => {
    // TODO: reutilizar funcion :D
    Complexity is 5 Everything is cool!
    const changePage = () => {
      if (pagination === undefined ||
        isNaN(parseInt(pagination)) ||
        pagination < 0) {
        props.history.push("/productos/0");
        showProducts(auth.authToken, 0, name, brand);
      }
    };
  }, [pagination]);
}
```

Fig32. Ejemplo de componente

Para cada uno de los componentes, contamos con una función llamada `useEffect`, que se ejecuta al iniciar el componente y otra serie de funciones que serán llamadas desde el código, que en este caso, en vez de ser HTML, será JSX.

Para hacer las peticiones asíncronas utilizamos la API Fetch, que es la encargada de acceder y manipular las peticiones y respuestas.

```
const getBrands = async (token) => {  
  await fetch(`${fetchBase}/productos/brand`, {  
    method: 'GET',  
    headers: {  
      'Authorization': `Bearer ${token}`  
    },  
    mode: "cors"  
  })  
  .then(res => res.json())  
  .then(data => setBrands([...data]));  
}
```

Fig33. Petición fetch con método GET

Manual de uso

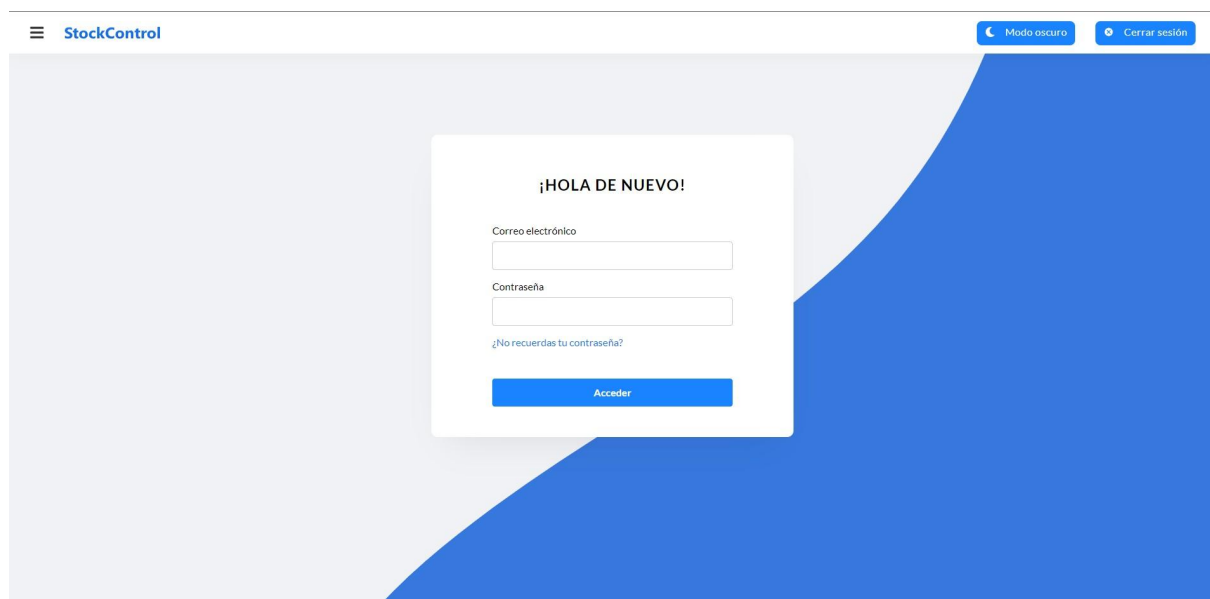


Fig34. Inicio de sesión

El procedimiento para usar la aplicación es sencillo, lo primero que vemos es una ventana de login, una vez introducidos los datos accedemos directamente a la ventana de la vista de tablas.

En esta vista vemos los productos, con todas sus características, además de un botón de añadir y otro de eliminar producto, un buscador por nombre o marca,

un botón de añadir producto nuevo y unos botones de paginación para mostrar los siguientes productos.

PRODUCTOS

Busca un producto Marca: Todas Buscar

Id	Marca	Nombre	Color	Stock	Fecha De Compra	Precio De Compra	Precio De Venta	Descuento	Descatalogado	Categorías	Lotes	Editar	Borrar
1	Adidas	Item1	rojo	10	2021-12-08	15	25	10	No	categoria1, categoria2	2021-12-18, 2021-12-18		
2	Nike	Item2	azul	5	2021-12-08	10	20	0	No	categoria1, categoria3	2021-12-18		
3	Reebok	Item2	azul	5	2021-12-08	5	30	0	Si	categoria1, categoria3	2021-12-18, 2021-12-18		
4	Nike	Item3	azul	3	2021-12-08	10	20	25	No	categoria1, categoria3	2021-12-18		
5	Reebok	Item3	rojo	7	2021-12-08	20	40	8	No	categoria1, categoria3	2021-12-18, 2021-12-18		
6	Nike	Item3	verde	5	2021-12-08	15	50	0	Si	categoria1, categoria3	2021-12-18		

Añadir producto Anterior Siguiete

Fig35. Vista de productos

Si pulsamos el botón de editar, encontramos un formulario y si pulsamos el de añadir encontramos otro formulario similar:

ACTUALIZAR PROVEEDOR

Nombre
Proveedor1

Localización
Jerez

Teléfono
666001122

Email
proveedor1@email.com

Código Postal
11130

Actualizar

AÑADIR TIENDA

Nombre

Localización

Teléfono

Email

Código Postal

Añadir

Fig36. Actualizar proveedor

Fig37. Añadir nueva tienda

Para movernos entre las diferentes secciones encontramos un menú lateral desplegable.

También podemos cerrar sesión desde el botón del menú superior.

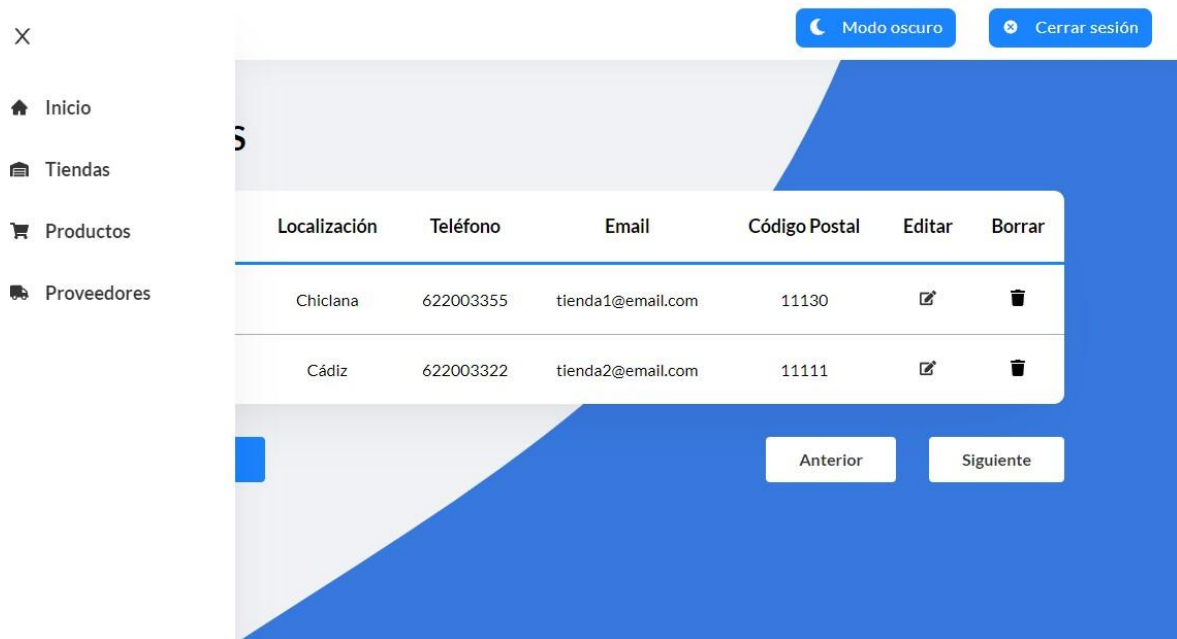


Fig38. Menú lateral desplegable

Conclusiones

Respecto a las conclusiones, comparando la versión 1.0 del proyecto con la versión original, es posible decir que la mayoría de las funcionalidades están implementadas y la aplicación es funcional, sin embargo, hay una serie de características que están presentes como mejoras futuras y que sería interesante implementar para que la aplicación fuese más completa.

Mejoras futuras

Respecto a las mejoras que podrían hacerse en la aplicación, son varias:

- Implementar el modo oscuro, para lo que ya se ha habilitado un botón en el menú superior.
- Añadir un panel de usuario, que está ya preparado en servidor.
- Añadir roles a los usuarios, que también está preparado en servidor.
- Añadir un registro solo para usuarios con rol de administrador, el registro está preparado tanto en servidor como en cliente.

Índice de imágenes

Figura 1	4
Figura 2	4
Figura 3	4
Figura 4	5
Figura 5	5
Figura 6	6
Figura 7	6
Figura 8	7
Figura 9	7
Figura 10	8
Figura 11	8
Figura 12	9
Figura 13	9
Figura 14	9
Figura 15	10
Figura 16	10
Figura 17	11
Figura 18	11
Figura 19	12
Figura 20	13
Figura 21	14
Figura 22	14
Figura 23	15
Figura 24	15
Figura 25	15
Figura 26	16
Figura 27	17
Figura 28	17
Figura 29	18
Figura 30	19
Figura 31	20
Figura 32	20
Figura 33	21
Figura 34	21
Figura 35	22
Figura 36	22
Figura 37	22
Figura 38	23

Bibliografía y referencias

1. <https://es.reactjs.org/docs/getting-started.html>
2. <https://es.reactjs.org/docs/handling-events.html>
3. <https://stackoverflow.com/>
4. <https://www.youtube.com/watch?v=CXa0f4-dWi4>
5. <https://codepen.io/>
6. <https://www.baeldung.com/rest-with-spring-series>

7. https://www.tutorialspoint.com/spring_boot/spring_boot_building_restful_web_service_s.htm
8. <https://spring.io/guides/gs/accessing-data-mysql/>
9. <https://spring.io/guides/tutorials/rest/>
10. https://www.jetbrains.com/help/idea/manage-projects-hosted-on-github.html#jump_to_github_version