

Music Rockstar Shop

Proyecto de fin de ciclo

Nombre del alumno o de la alumna: Adrián Plaza González

Curso académico: Desarrollo de aplicaciones web

Tutora/Tutor del proyecto: Carmelo Escribano

ÍNDICE PAGINADO

1. RESUMEN.....	3
2. JUSTIFICACIÓN DEL PROYECTO.....	4
3. OBJETIVOS.....	5
4. DESARROLLO.....	6
4.1. FUNDAMENTACIÓN TEÓRICA.....	6
4.1.1 ANÁLISIS DE LAS HERRAMIENTAS NECESARIAS.....	7
4.1.2 PROCESO DE DESARROLLO.....	10
4.2 MATERIALES Y MÉTODOS.....	27
4.3 RESULTADOS Y ANÁLISIS.....	29
5. CONCLUSIONES.....	30
6. LÍNEAS DE INVESTIGACIÓN FUTURAS.....	31
7. BIBLIOGRAFÍA.....	32

1. RESUMEN

El objetivo de este proyecto de fin de grado es llevar a cabo el desarrollo de una aplicación web de una empresa de venta de instrumentos musicales, es decir, una aplicación *E-commerce*.

La aplicación deberá poseer varias características previamente ideadas:

- Un *feed* principal desde donde poder navegar a cualquier otra función.
- Un sistema de registro e inicio de sesión de usuarios eficaz e intuitivo además de incluir un sistema de cifrado para las contraseñas.
- Una página que representa la lista de productos disponibles desde la cual acceder a cada producto.
- Página de producto con la información de producto, además de un sistema de opiniones y valoraciones.
- Sistema de usuarios, en el cual cada usuario tiene su propia cesta de la compra, sus métodos de pago, sus direcciones y sus pedidos.
- Página de compra en la que se efectúe un pedido.

2. JUSTIFICACIÓN DEL PROYECTO

El desarrollo de esta aplicación cubre la necesidad creciente de usuarios que buscan comprar instrumentos de música online.

En los últimos años, los hábitos de los consumidores tienden a recurrir más a las compras en línea debido a su comodidad y facilidad de acceso.

Debido a ello la demanda de productos en línea ha aumentado de manera exponencial, incluyendo los productos musicales.

La industria musical cuenta con gran cantidad de consumidores, lo que permite llegar a una gran cantidad de posibles clientes.

3. OBJETIVOS

A. OBJETIVO GENERAL

Desarrollar la página web de un negocio de venta de instrumentos musicales sencilla, fácil de usar y con todos los aspectos necesarios para su funcionamiento.

B. OBJETIVOS ESPECÍFICOS

- Estudiar las herramientas necesarias para el desarrollo.
- Diseñar la interfaz de usuario.
- Crear base de datos que almacena usuarios, productos, pedidos etc...
- Crear las páginas previamente diseñadas en las que los usuarios naveguen.
- Desarrollar lógica e interfaz del inicio de sesión y el registro de usuarios.
- Desarrollar la lógica necesaria para crear usuarios, crear pedidos, añadir productos a la cesta y otras funciones.
- Desarrollar *scripts* de automatización para testear la aplicación una vez ya terminada.

4. DESARROLLO

4.1. Fundamentación teórica

La página web será una plataforma e-commerce en la que los usuarios podrán visualizar productos, añadirlos al carrito y crear un pedido.

El usuario debe primero registrarse en la aplicación o iniciar sesión en la aplicación, en caso de no iniciar sesión, el usuario solo podrá visualizar productos sin realizar ninguna acción.

Una vez registrados, los clientes podrán añadir productos a la cesta desde la página de productos además de dejar una opinión de cada uno.

Para realizar un pedido, primero deben tener agregada una dirección y una tarjeta desde sus respectivas páginas.

Cuando el pedido haya finalizado, aparecerá en la lista de pedidos del usuario.

Desde la página de inicio se podrá navegar para visualizar los pedidos, las direcciones, las tarjetas y el carrito de compras del cliente.

Desde el apartado de pedidos se podrá visualizar los distintos datos como los productos, la dirección, la fecha de realización y el método de pago utilizado.

Desde la página de direcciones y la página de tarjetas se podrán observar una lista de estas y borrar o crear nuevas.

Desde el carrito se podrá borrar productos y finalizar pedido, que llevará a la pantalla de pago.

Desde la pantalla de pago se visualizará un resumen del pedido y se deberá elegir una dirección y una tarjeta de crédito para continuar.

Una vez realizado el pedido, aparecerá una pantalla resumiendo el pedido y sus datos.

4.1.1 Análisis de las herramientas necesarias.

Para el desarrollo de la página web se plantearon dos posibilidades:

Desarrollarla en mediante un framework front-end, como por ejemplo, *Angular*, *React* o *Vue*.

O, desarrollarla mediante *Spring Boot*, una herramienta que facilita el desarrollo en *Spring framework* y que da soporte para el desarrollo de aplicaciones y páginas webs basadas en Java.

Finalmente la elección fue *Spring boot* debido a su escalabilidad y robustez en el desarrollo. Además se puede integrar fácilmente con otras tecnologías front-end.

También proporciona algunas características de seguridad, como puede ser la dependencia *Spring Security*.

Utiliza *Maven* o *Gradle*, que facilita el manejo de librerías.

Para utilizar *Spring Boot* es necesario un entorno de desarrollo (*IDE*). Entre las opciones, las más destacadas son *IntelliJ* y *Eclipse*. El *IDE* elegido fue *IntelliJ* por preferencia personal.

La versión de *Spring Boot* que se utilizará es la última disponible al comenzar el desarrollo.

Para la base de datos se eligió una relacional, *SQL*, para ello hay dos opciones, una base de datos en la nube o en local.

Tras investigar las opciones disponibles se descartó una base de datos en la nube debido a que las mejores opciones no son gratuitas.

Finalmente la base de datos se almacenará en *MySQL*. Para su uso serán necesarias varias herramientas:

- *Xampp*, para levantar el servicio de *SQL*.

- *MySQL Workbench*, un software para el diseño y administración de base de datos que proporciona una interfaz visual.
- *MySQL Connector*, una dependencia de *Spring Boot* para conectar la base de datos con la aplicación.

Esta base de datos se encargará de almacenar los productos, los usuarios y sus pedidos, direcciones y sus tarjetas, además de el carrito de la compra de estos.

Los productos estarán compuestos por su identificador, nombre, descripción, tipo (si son instrumentos de cuerda, viento...), imagen, días de entrega, precio, precio rebajado e inventario.

Los pedidos tendrán identificador, precio total, tarjeta utilizada, dirección, fecha de realización y la lista de productos de este.

Las direcciones poseerán el identificador, el nombre del comprador, su teléfono, la calle, número del portal, código postal, ciudad, provincia y observaciones.

Las tarjetas estarán compuestas por su id, el nombre completo de el poseedor, número y cvc.

Por último el usuario tendrá, email (funcionará como identificador), contraseña, lista de pedidos, direcciones ,tarjetas y carrito.

Para el registro de usuarios se utilizará *Spring Security*. Esta dependencia simplifica el desarrollo del registro e inicio de sesión de usuarios y ofrece una pantalla de login ya creada la cual será modificada según las necesidades del proyecto.

Además, para almacenar las contraseñas se necesitará un sistema de encriptado. *Spring Security* también ofrece una solución para ello.

La aplicación necesitará ser desplegada, tras investigar opciones, se llegó a la conclusión de que la mejor opción es desplegar en local, ya que la mayoría de herramientas para desplegar aplicaciones web son de pago.

Para desplegar la aplicación de manera local *IntelliJ* facilita un servicio *Apache Tomcat* para el despliegue de manera sencilla.

El diseño de la interfaz de usuario se realizará en *Figma* y el del logo en la página web *Canva*. Además se deberá elegir los colores que usará la aplicación.

También será necesario el uso de *Thymeleaf*, un motor de plantillas *Java* para aplicaciones.

Se utilizará para generar vistas *HTML* dinámicas que presentan los datos ofrecidos desde el *back-end* de la aplicación.

Además se hará uso de *bootstrap*, un framework *front-end* que facilita y agiliza el desarrollo de las interfaces de usuario.

Los productos, al necesitar imágenes, se necesitarán almacenarse en la base de datos. Para que la base de datos no acabe siendo demasiado pesada con el crecimiento de la página web, las imágenes se almacenarán en modo de url.

Para ello las imágenes se subirán a la página web *Imgur*, que la almacena y ofrece una *url*.

Por último, una vez terminado el desarrollo, se necesitará comprobar que su funcionalidad está pulida y es consistente. Para ello se desarrollarán scripts de automatización para testear la aplicación de forma automática.

Tras investigar distintas soluciones se eligió *Robot Framework*, un *framework* de automatización de tests de desarrollo dirigido, junto a la librería *Selenium*, utilizada para el desarrollo de pruebas de aplicaciones web.

4.1.2 Proceso de desarrollo.

El primer paso fue crear el logo y elegir los colores de la aplicación.

El logo se creó mediante la página web [canva.com](https://www.canva.com) y los colores principales de la aplicación son el naranja, el negro, el blanco, además de otra variante de naranja para algunos detalles.



(Paleta de colores)

El logo esta formado por el nombre de la aplicación con la fuente Virtual que ofrece Canva, con los colores negro y naranja de la aplicación en un fondo blanco.



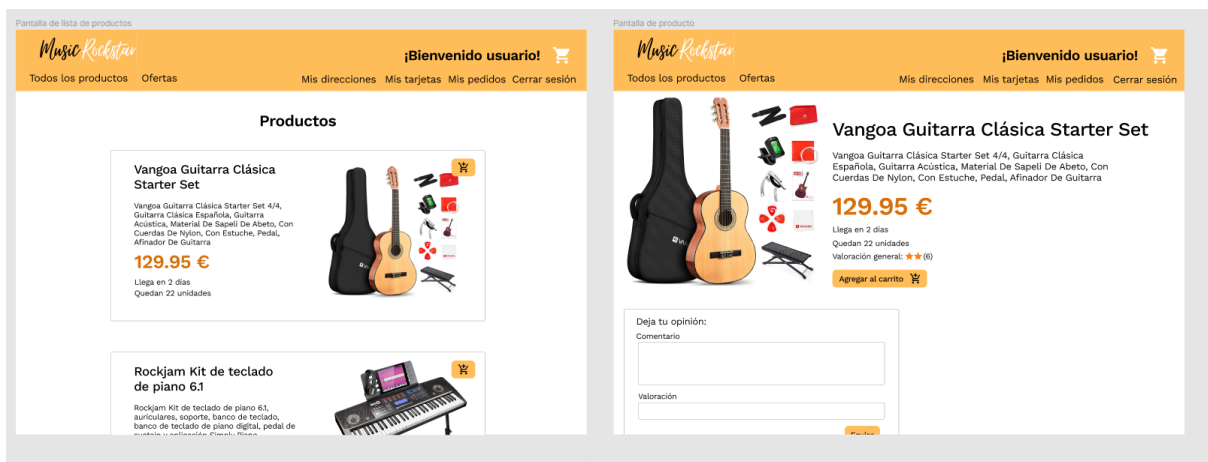
(Logo de *Music Rockstar*)

Además se crearon otras variantes con el fondo naranja para su uso en la aplicación.

The logo for Music Rockstar, featuring the brand name in a white, handwritten-style script font against a solid orange rectangular background.

(Variante del logo de *Music Rockstar*)

Después se creó un prototipo de la aplicación en *Figma*.



(Prototipo diseñado en *Figma*)

El primer paso para el desarrollo fue crear el proyecto en *IntelliJ*.

Para ello es necesario indicar el lenguaje que se va a utilizar, *Java* o *Kotlin* y el tipo de proyecto, *Gradle* ó *Maven*.

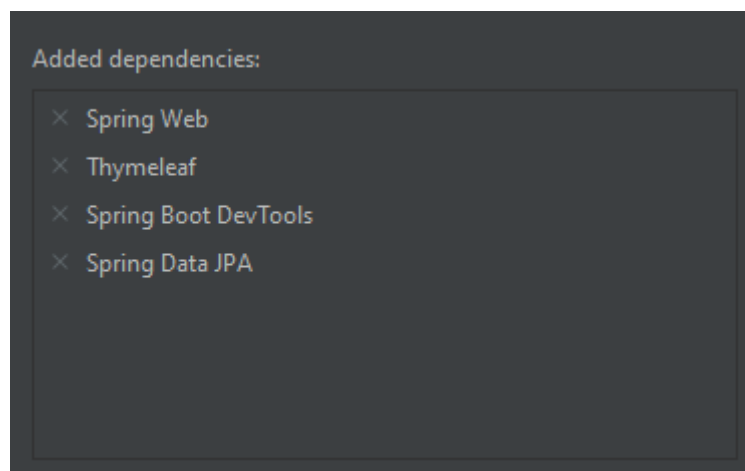
Se eligió *Java 17* y *Maven*.

La versión de *Spring Boot* elegida fue la 3.0.6, la última versión disponible en el momento.

Lo siguiente fue elegir las dependencias que utilizara en el proyecto (*Starters* de *Spring boot*).

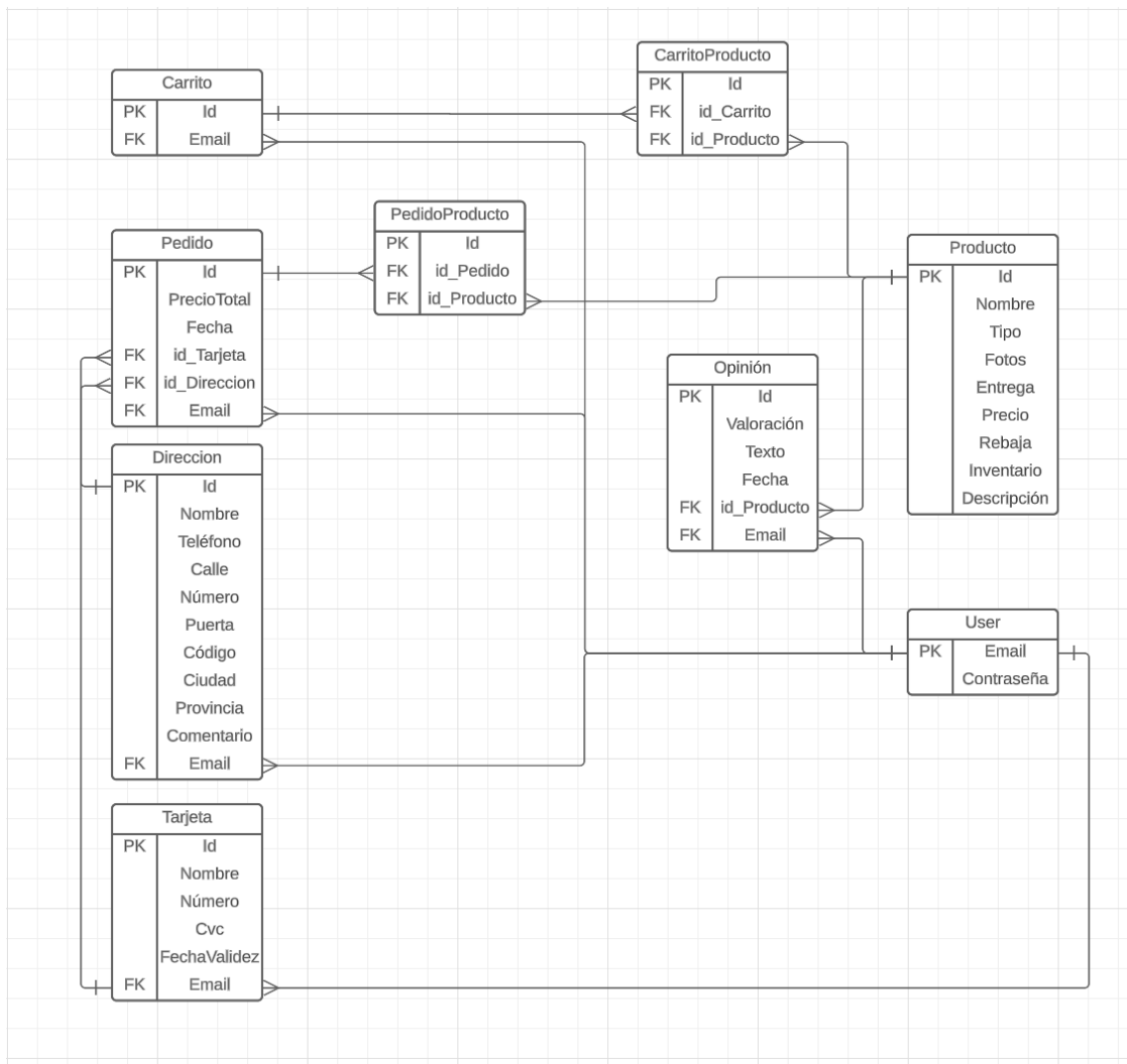
En un principio se añadieron las dependencias *Spring Web*, *Thymeleaf*, *Spring Boot DevTools* y *Spring Data JPA*.

En caso de necesitar otras dependencias, se añadirán desde el archivo *pom.xml* de *Maven*.



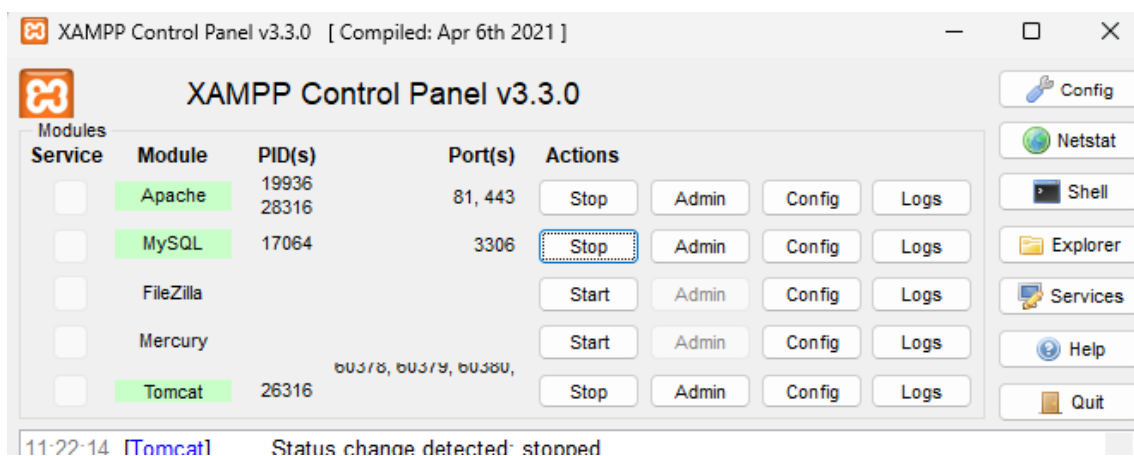
(*Spring Starters* seleccionados)

Lo siguiente fue diseñar la base de datos que almacenará productos y usuarios, junto a la información de estos, como las valoraciones de los productos o las direcciones y tarjetas guardadas de los usuarios, los pedidos y los carritos de compra.



(Diagrama de entidad relación de la base de datos)

Antes de crear la base de datos se necesita levantar el servicio de *MySQL*. Para ello se utiliza *Xampp* que levanta el servicio en el puerto 3306.



Para crear la base de datos se utilizó *MySql Workbench* como interfaz en la que ejecutar las sentencias *SQL*.

Para la conexión de acceso a datos con la aplicación se necesita una dependencia que no está añadida, *MySql Connector*.

Con *MySql Connector* y desde el archivo `application.properties` se configura la conexión creando el *datasource*.

Una vez creada la conexión, el *starter Spring Data JPA* que se eligió al crear el proyecto ofrece la dependencia *JPA* que proporciona una interfaz de programación de aplicaciones para el mapeo objeto-relacional (*ORM*).

JPA facilita la manipulación de los datos utilizando objetos *Java*.

Para su implementación se crea por cada tabla una entidad *JPA* a partir de una clase *Java*, un repositorio que facilita métodos para el manejo de los datos y una clase *Service* desde la que llamar a los métodos del repositorio y crear otros según las necesidades.

Por ejemplo, para la tabla dirección se necesita una clase de entidad generada desde la clase *java* dirección.

```

@Entity
@NamedQuery(name="Direccion.findAll", query="SELECT d FROM Direccion d")
public class Direccion implements Serializable {
    no usages
    private static final long serialVersionUID = 1L;

    2 usages
    @Id
    private int id;
    4 usages
    private String nombre;
    4 usages
    private String telefono;
    4 usages
    private String calle;
    4 usages
    private int numero;
    4 usages
    private String puerta;
    4 usages
    private String codigo;
    4 usages

```

(Clase *Dirección*)

Una vez creada es necesario una interfaz que extienda la clase *Jpa Repository* que proporciona los métodos necesarios para el manejo de la tabla en la base de datos.

```

2 usages  Programador
public interface DireccionRepositorio extends JpaRepository<Direccion, Integer> {
}

```

(Repositorio de la tabla *Dirección*)

Adicionalmente, se crea una clase *Service* que usando el repositorio crea los métodos necesarios para crear direcciones, borrarlas o devolver la lista de direcciones según el email.

```

public class DireccionService {

    4 usages
    @Autowired
    DireccionRepositorio direcciones;

    no usages  Programador
    public DireccionService(DireccionRepositorio direcciones) { this.direcciones = direcciones; }

    1 usage  Programador
    public void guardarDireccion(Direccion direccion) { direcciones.save(direccion); }

    6 usages  Programador
    public List<Direccion> listaDireccionesEmail(String email) {
        List<Direccion> direccionesAll=direcciones.findAll();
        List<Direccion> direccionesEmail = new ArrayList<>();
        for (int i = 0; i<direccionesAll.size();i++){
            if (direccionesAll.get(i).getEmail().equalsIgnoreCase(email)){
                direccionesEmail.add(direccionesAll.get(i));
            }
        }
        return direccionesEmail;
    }
}

```

(Servicio de la tabla *Dirección*)

Para el registro e inicio de sesión de usuarios se necesita la dependencia *Spring Security* añadida en el principio del desarrollo.

Para la configuración del registro de usuarios se crea la clase *DatabaseWebSecurity*. Se le proporciona un objeto *DataSource* que contiene la base de datos y se indica que la tabla que maneja el registro de usuarios es *users*.

El método *filterChain(HttpSecurity http)* configura las reglas de autenticación y autorización de la aplicación.

En el caso de *MusicRockstar*, todos los usuarios pueden entrar a cualquier página, entonces en cuanto a autorización cualquier petición estará permitida para cualquier usuario.

En cuanto a autenticación, *Spring Security* ya proporciona una página de login y un método de login totalmente funcional por lo que el único paso es modificar la vista *login.html*.

Para el registro si es necesario crear otra pantalla y otro método para guardar usuarios.

El proceso de autenticación en la aplicación se lleva a cabo a través de dos pantallas, la de *login* y la de *sign up*.

Además se crea el método *passwordEncoder* para el encriptado de las contraseñas.

```
no usages  Programador
@Autowired
public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception{
    auth.jdbcAuthentication().dataSource(origenDatos).usersByUsernameQuery("SELECT username, password, e
}

no usages  Programador *
@Bean
public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.authorizeHttpRequests()
        .requestMatchers(...patterns: "/css/**").permitAll()
        .requestMatchers(...patterns: "/", "/login", "/signup").permitAll()
        .anyRequest().permitAll()
        .and().formLogin().loginPage("/login").permitAll()
        .and().logout().permitAll()
        .and().exceptionHandling().accessDeniedPage(accessDeniedUrl: "/denegado");
    return http.build();
}

no usages  Programador
@Bean
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

(Clase *DatabaseWebSecurity*)



Iniciar sesión

Sign In

¿No tienes cuenta? [Regístrate aquí.](#)

(Página de inicio de sesión)

Para el encriptado de las contraseñas se utilizara un objeto *PasswordEncoder* de Spring Security. Basta con el método *encode* para encriptar una cadena de texto, en este caso la contraseña recibida del formulario.

```

@PostMapping("/registrarUser")
public ModelAndView crearUser(@ModelAttribute Users user, Model model) {
    ModelAndView mv = new ModelAndView();
    Optional<Users> userBuscado = users.bbuscarUser(user.getUsername());
    String passwordEncriptado = passwordEncoder.encode(user.getPassword());
    user.setPassword(passwordEncriptado);
    if (userBuscado.isPresent()){
        System.out.println("El correo "+user.getUsername()+" ya esta utilizado");
    }else{
        users.registrarUser(user);
    }
    mv.setViewName("login");
    return mv;
}

```

(Método de registro de usuarios en la clase *controlador*)

```

adrian@campusfp.es $2a$10$PKhGoSYeZj43esL9mC/JV.MFB/daUz/0EZwugQozMgbXjIH7KdJuC

```

(Registro de la tabla *user*)

Spring Boot utiliza el modelo *MVC* (Modelo Vista Controlador). Por lo que es necesario una clase controlador que recibe peticiones desde las vistas, procesa la información y envía la respuesta necesaria para el funcionamiento de la página.

Para cada pantalla se necesita un archivo *html* para la vista, y un método del controlador que le proporcione datos.

También es necesario la dependencia *Thymeleaf* que se añadió al principio del desarrollo. *Thymeleaf* permite generar vistas dinámicas accediendo a los datos que proporciona el controlador.

Un ejemplo sería la página que muestra los productos.

En la página de productos, se genera una vista con la lista de productos. Con la expresión `th:each` de *Thymeleaf*, se itera la lista de productos que recibida desde el controlador y por cada producto se muestra su información.

Con la expresión `th:href` se ejecuta una petición al controlador en el que se pasa el id del producto como parámetro.

```
<th:block th:each="producto : ${productos}">
  <div class="card-body" style="width: 60rem; margin-top: 2rem; margin-bottom: 2rem;">
    <div class="card flex-row justify-content-around">
      <div style="width: 50%; height: 50%; margin-top: 2rem">
        <div>
          <a th:href="@{/producto/{id}(id = ${producto.getId()})}"><h3 th:text="${producto.getNombre()}" class="card-title"></h3></a>
          <h5 th:text="${producto.getDescripcion()}" class="card-title"></h5>
          <h1 th:text="${producto.getPrecio()}" class="textoOrange"></h1>
          <div th:unless="${producto.getSinRebaja() == 0}">
            <s><h3 th:text="${producto.getSinRebaja()}" e' "></h3></s>
          </div>
          <div th:if="${producto.getEntrega() > 1}">
            <h6 th:text="Llega en '+${producto.getEntrega()}' dias" ></h6>
          </div>
          <div th:if="${producto.getEntrega() == 1}">
            <h6 th:text="Llega en '+${producto.getEntrega()}' dia" ></h6>
          </div>
          <div th:if="${producto.getInventario() > 5}">
            <h6 th:text="Quedan '+${producto.getInventario()}' unidades" ></h6>
          </div>
          <div th:if="${producto.getInventario() < 6 && producto.getInventario() > 1}">
            <h6 style="..." th:text="Quedan '+${producto.getInventario()}' unidades" ></h6>
          </div>
          <div th:if="${producto.getInventario() == 1}">
            <h6 style="..." th:text="Queda '+${producto.getInventario()}' unidad" ></h6>
          </div>
        </div>
      </div>
    </div>
  </div>
</th:block>
```

(Vista de la lista productos)

Desde el controlador se recoge la *id* del producto con con la anotación `@PathVariable`, con la clase Servicio de producto se recoge el valor de la base de datos con el método `bbuscarProducto()` que utiliza el `findById()` proporcionado por el repositorio.

Este método devuelve un objeto *Optional*, en caso de que se encuentre un producto con ese id se recuperará el producto en cuestión y se almacenará en un objeto *Producto*.

Con este objeto, se calculará la valoración media de sus opiniones y se enviará a la vista de producto junto al producto.

Además se enviará otro parámetro, el usuario, en caso de que este logueado, este se recupera con la clase *Authentication*.

En caso de no estarlo, no se enviará a la vista.

Por último retorna un objeto *ModelAndView* que redirecciona al usuario a la página de producto.

```
@RequestMapping("/producto/{id}")
public ModelAndView producto(@PathVariable int id, Authentication auth) {
    ModelAndView mv = new ModelAndView();

    Optional<Producto> producto = productoService.bbuscarProducto(id);
    Producto p;
    if (producto.isPresent()) p=producto.get(); else p=null;
    List<Opinion> opiniones = p.getOpiniones();
    int o = 0;
    for (int i = 0; i< opiniones.size(); i++){
        o = o + opiniones.get(i).getValoracion();
    }
    double mediaValoracion = (double) o / opiniones.size();
    long mediaRedondeada = Math.round(mediaValoracion);
    if (opiniones.size()!=0){
        mv.addObject( attributeName: "valoracionMedia", mediaRedondeada);
        mv.addObject( attributeName: "valoracionNumero", opiniones.size());
    }
    mv.addObject( attributeName: "producto",p);
    mv.setViewName("producto");
    mv.addObject( attributeName: "opinion", new Opinion());
    try{
        String correo = auth.getName();
        String[] correoSeparado = correo.split( regex: "@");
        String user = correoSeparado[0];
        mv.addObject( attributeName: "correo",user);
    }catch (Exception e){

    }
    return mv;
}
```

(Método del controlador de la página de producto)

En la página producto se recoge el producto y se muestran los datos con expresiones de *Thymeleaf*.

La expresión *th:if="{correo}"* comprueba si se ha enviado el correo a la vista. Si *correo* se ha enviado el botón de añadir al carrito estará habilitado y la sección de opinar será visible, en caso contrario con *th:unless="{correo}"* se mostrará el botón deshabilitado y no aparecerá el formulario para opinar.

```
<h1 class="nombre" th:text="{producto.getNombre()}"></h1>
<p class="descripcion" th:text="{producto.getDescripcion()}"></p>
<div class="d-flex flex-row">
<div th:if="{producto.getInventario()>5}">
  <h6 th:text="'Quedan '+{producto.getInventario()}+' unidades'" ></h6>
</div>
<div th:if="{producto.getInventario()<6 && producto.getInventario()>1}">
  <h6 style="..." th:text="'Quedan '+{producto.getInventario()}+' unidades'" ></h6>
</div>
<div th:if="{producto.getInventario()==1}">
  <h6 style="..." th:text="'Queda '+{producto.getInventario()}+' unidad'" ></h6>
</div>
<div th:if="{producto.getEntrega()>1}">
  <h6 th:text="'Llega en '+{producto.getEntrega()}+' dias'" ></h6>
</div>
<div th:if="{producto.getEntrega()==1}">
  <h6 th:text="'Llega en '+{producto.getEntrega()}+' dia'" ></h6>
</div>
<th:block th:if="{valoracionMedia}">
<div th:if="{correo}">
  <form method="post" th:action="@{/agregarCarrito}" th:object="{producto}">
</div>
```

(Líneas de código de la vista de producto)



[Todos los productos](#)
[Ofertas](#)

[Iniciar sesion](#)



Vangoa Guitarra Clásica Starter Set 4/4

Vangoa Guitarra Clásica Starter Set 4/4, Guitarra Clásica Española, Guitarra Acústica, Material De Sapeli De Abeto, Con Cuerdas De Nylon, Con Estuche, Pedal, Afinador De Guitarra

129.95 €

Quedan 62 unidades


Llega en 2 días

Valoración general: ★★ (6)

Agregar al carrito

Inicia sesión para dejar un comentario.


(Página de producto sin iniciar sesión)



[Todos los productos](#)
[Ofertas](#)

¡Bienvenido adrianplaza17!

[Mis direcciones](#)
[Mis tarjetas](#)
[Mis pedidos](#)
[Cerrar sesión](#)



Vangoa Guitarra Clásica Starter Set 4/4

Vangoa Guitarra Clásica Starter Set 4/4, Guitarra Clásica Española, Guitarra Acústica, Material De Sapeli De Abeto, Con Cuerdas De Nylon, Con Estuche, Pedal, Afinador De Guitarra

129.95 €

Quedan 62 unidades

Llega en 2 días

Valoración general: ★★ (6)

Agregar al carrito

Deja tu opinión:

Comentario


(Página de producto tras iniciar sesión)


Para el proceso de compra se crean la página de carrito, la de pago, y la de producto finalizado.

El carrito está compuesto por una lista de productos, al pulsar en tramitar pedido redirige a la pantalla de pago.

En la pantalla de pago se debe elegir una de las direcciones y una de las tarjetas que posee el usuario. También muestra un resumen de los productos que se están comprando.

Una vez se pulsa en el botón de finalizar pedido, se guarda la lista de productos en la sesión con el método `setAttribute()` del objeto `HttpSession` de la dependencia `jakarta.servlet` y se ejecuta el método que crea el pedido recogiendo los productos de la sesión.

[Todos los productos](#) [Ofertas](#)

¡Bienvenido **adrianplaza17!** 

[Mis direcciones](#) [Mis tarjetas](#) [Mis pedidos](#) [Cerrar sesión](#)

Vangoa Guitarra Clásica Starter Set 4/4

Vangoa Guitarra Clásica Starter Set 4/4, Guitarra Clásica Española, Guitarra Acústica, Material De Sapeli De Abeto, Con Cuerdas De Nylon, Con Estuche, Pedal, Afinador De Guitarra

129.95 €

Llega en 2 días
Quedan 62 unidades

[Eliminar](#) 





Total
(1 producto):
129.95 €



[Tramitar pedido](#)

(Página de carrito de la compra)

Dirección:

Calle Galdos 16, 3ºA. 28904  

Tarjeta:

Adrian Plaza Gonzalez - 5004291838219853  


Resumen

Vangoa Guitarra Clásica Starter Set 4/4

Vangoa Guitarra Clásica Starter Set 4/4, Guitarra Clásica Española,
Guitarra Acústica, Material De Sapeli De Abeto, Con Cuerdas De Nylon,
Con Estuche, Pedal, Afinador De Guitarra

129.95 €

Llega en 2 días



(Página de pago)

Una vez finalizado el proceso de desarrollo se van a realizar scripts para el testeo de la aplicación.

Para ello se va a utilizar *Robot Framework* con la librería *Selenium*, un marco de pruebas de software. Este proporciona una sintaxis simple que facilita el desarrollo de casos de prueba.

Para la creación de los scripts se necesitan tres archivos:

- Uno con las variables que almacenan los elementos web de la página mediante el *xpath* de estos, y valores que se utilizarán en los inputs.
- El archivo de keywords, los keywords son funciones que ejecutan las acciones que se indican en ellos.
- El archivo de test cases, que es desde donde se ejecutan las pruebas. Estos están compuestos por keywords o otras acciones según los casos.

Los casos de prueba desarrollados para la aplicación son:

- *MusicRockstar Purchase*, que inicia sesión, añade un producto al carrito y ejecuta el pedido.
- *Create and Delete Tarjeta*, que inicia sesión y crea una tarjeta y la borra.
- *Create and Delete Dirección*, que hace lo propio, creando una dirección y borrándola.

```
1  *** Settings ***
2  Library       Selenium2Library
3  Resource      keywords.robot
4
5  *** Test Cases ***
6  MusicRockstar Purchase
7      Go to MusicRockstar
8      Sign In with user adrianplaza17@gmail.com and password 1234
9      Agregar producto a carrito
10     Tramitar pedido
11     BuiltIn.Sleep    5
12
13  Create and delete Tarjeta
14      Go to MusicRockstar
15      Sign In with user adrianplaza17@gmail.com and password 1234
16      Crear tarjeta y borrar tarjeta
17
18  Create and delete Direccion
19      Go to MusicRockstar
20      Sign In with user adrianplaza17@gmail.com and password 1234
21      Crear direccion y borrar direccion
```

(Test Cases de la aplicación)

```
Sign In with user ${User} and password ${Password}
    Wait Until Element Is Visible    ${SignIn_a}
    Click Element    ${SignIn_a}
    Wait Until Element Is Visible    ${User_input}
    Input Text    ${User_input}    ${User}
    Wait Until Element Is Visible    ${Password_input}
    Input Text    ${Password_input}    ${Password}
    Wait Until Element Is Visible    ${SignIn_btn}
    Click Element    ${SignIn_btn}
    BuiltIn.Sleep    2

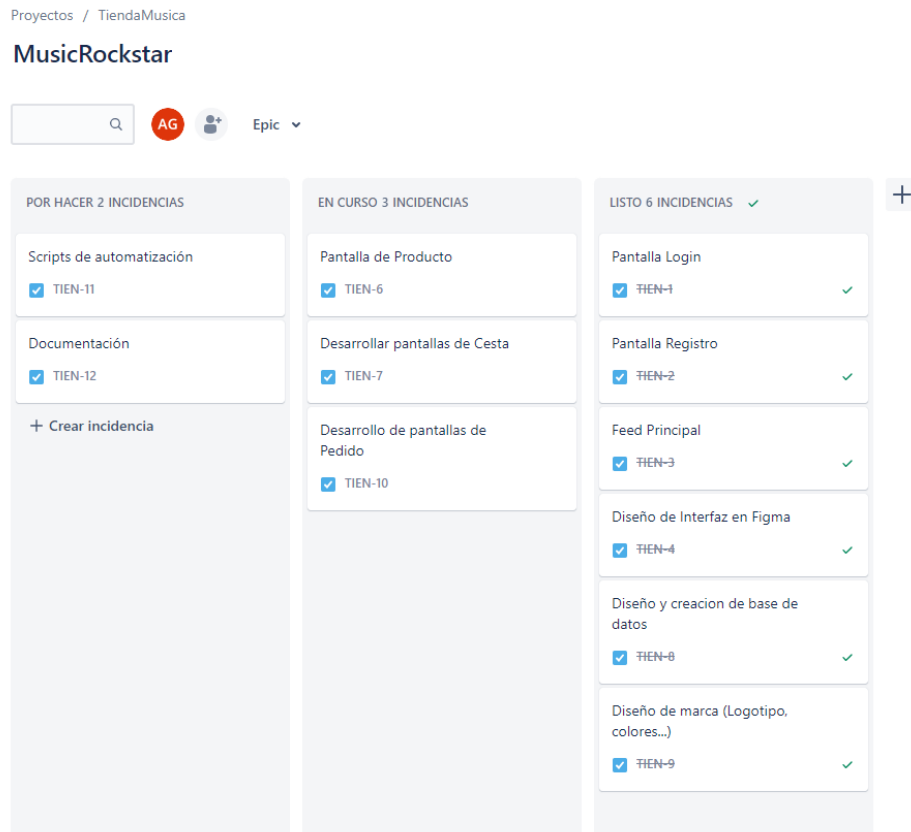
Agregar producto a carrito
    Wait Until Element Is Visible    ${TodosProductos_a}
    Click Element    ${TodosProductos_a}
    Wait Until Element Is Visible    ${ProductImage_a}
    Click Element    ${ProductImage_a}
    Wait Until Element Is Visible    ${AddCarrito}
    Click Element    ${AddCarrito}
```

(Keywords de login y agregar al carrito)

4.2 Materiales y métodos.

Para la organización en el proceso de desarrollo se sigue la forma de metodología ágil Kanban, que consiste en la elaboración de un cuadro en el que se reflejan tres columnas de tareas; pendientes, en proceso y terminadas.

Para la creación de la tabla se utilizó Jira.



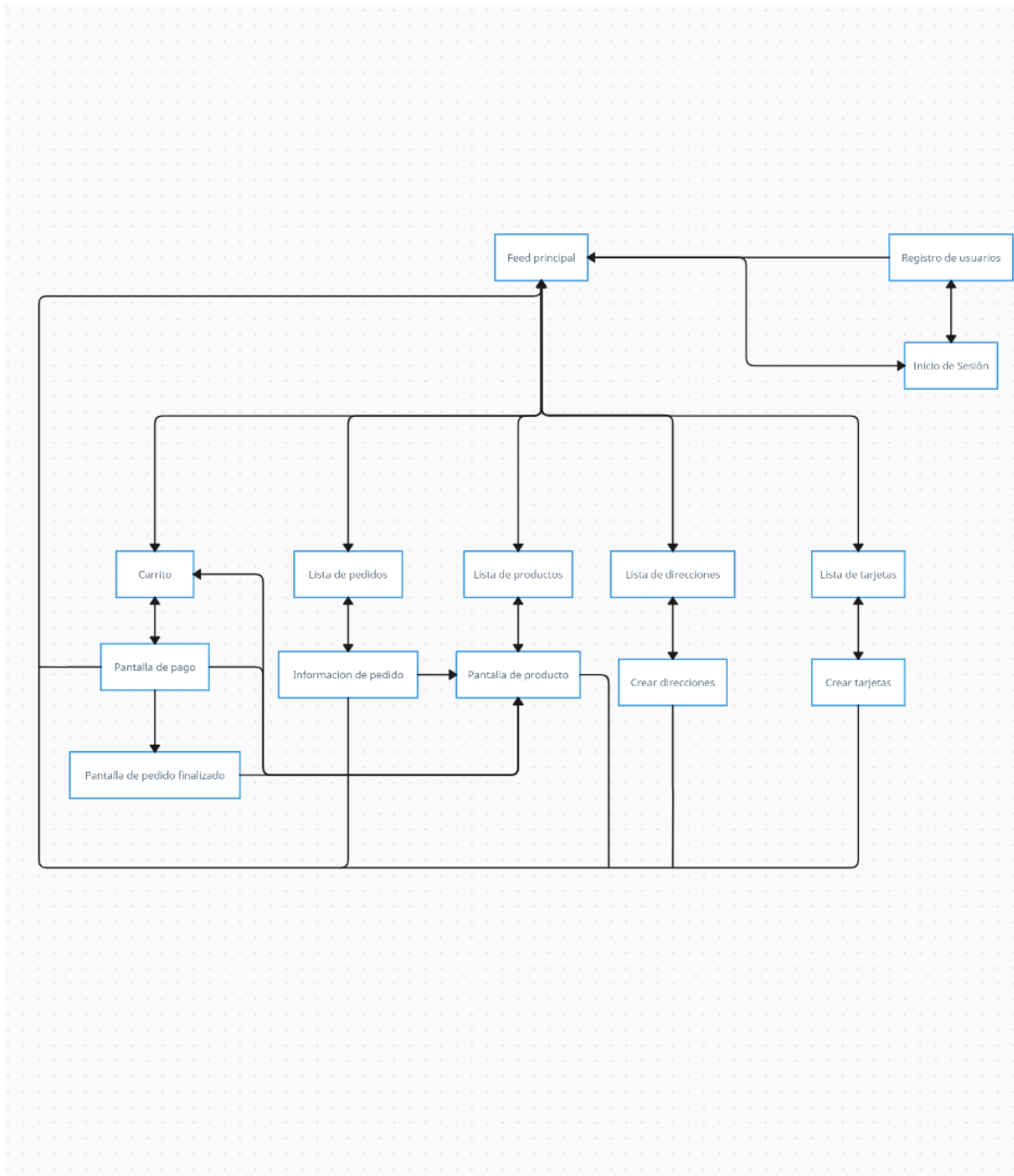
(Kanban en Jira)

También es necesario el uso de una herramienta de control de versiones y guardado en la nube.

GitHub fue la herramienta elegida, desde Git es posible guardar en la nube la aplicación en repositorios y controlar las versiones de esta. Además IntelliJ cuenta con la integración de GitHub para su uso dentro del IDE de forma sencilla.

En caso de surgimiento de dudas, el primer paso fue revisar la documentación oficial. Si el problema no es resuelto, se buscaría información en páginas de terceros como foros o páginas dedicadas.

Se seguirá un diagrama de casos de uso, que proporciona la información de las acciones posibles que ejecutará la aplicación.



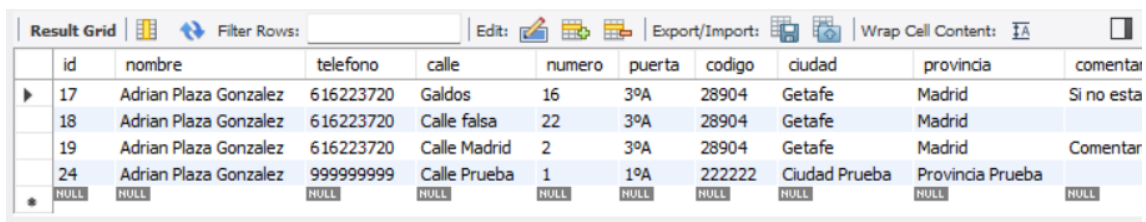
(Diagrama de casos de uso)

4.3 Resultados y análisis.

Para comprobar los resultados y cambios en la fase de desarrollo, se desplegaba la aplicación desde IntelliJ y se navegaba el puerto 8080 para comprobar la aplicación manualmente.

En las etapas finales del desarrollo se ejecutaron los scripts de pruebas de Robot Framework para comprobar el funcionamiento de la aplicación.

Para comprobar la conexión de la base de datos y que se agregan registros se comprueban a través de *MySQL Workbench*, que posee una interfaz de usuario para comprobar las tablas de manera visual.



	id	nombre	telefono	calle	numero	puerta	codigo	ciudad	provincia	comentario
▶	17	Adrian Plaza Gonzalez	616223720	Galdos	16	3ªA	28904	Getafe	Madrid	Si no esta
	18	Adrian Plaza Gonzalez	616223720	Calle falsa	22	3ªA	28904	Getafe	Madrid	
	19	Adrian Plaza Gonzalez	616223720	Calle Madrid	2	3ªA	28904	Getafe	Madrid	Comentar
	24	Adrian Plaza Gonzalez	999999999	Calle Prueba	1	1ªA	22222	Ciudad Prueba	Provincia Prueba	
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

(Tabla dirección en *MySQL Workbench*).

5. CONCLUSIONES

Tras el periodo de desarrollo, la aplicación terminó cumpliendo los objetivos y expectativas planteados para la página web.

También la aplicación permite mejoras posibles que pudieran hacer la página web aún más completa y añadir nuevas funcionalidades.

Las herramientas planteadas en un principio para el desarrollo fueron eficaces y no generaron inconvenientes.

Todas las funciones planteadas se añadieron correctamente, como son agregar productos al carrito, realizar compras, el sistema de inicio de sesión de usuarios, crear valoraciones de los productos o crear métodos de pago y direcciones.

En conclusión, el resultado del proyecto de fin de grado es el esperado, pues cumple con el desarrollo de una aplicación *E-commerce* como todos los apartados indispensables de una aplicación de este tipo. Aun así quedan pendientes algunas mejoras como mejoras en el apartado visual, añadir más opciones de pago, una base de datos remota o desplegar la aplicación en la nube.

Este es el repositorio de *GitHub* de la aplicación:

<https://github.com/AdrianPlaza17/MusicRockstarShop>

6. LÍNEAS DE INVESTIGACIÓN FUTURAS

Estas son algunas mejoras y nuevas funciones que se podrían implementar en la aplicación:

- Migrar la base de datos a una base de datos remota (algunas opciones son Azure o Amazon Web Services).
- Desplegar la aplicación en la nube (algunas opciones son Azure o Amazon Web Services).
- Añadir opción para registrarse desde google, facebook etc... (Se podría desarrollar con Google Firebase).
- Mejorar el apartado visual.
- Añadir opciones para que los usuarios puedan subir sus propios productos. (Agregar usuarios con distintas autorizaciones como vendedor, comprador o ambas).

7. BIBLIOGRAFÍA

Autor o editor	Fecha	Título	Fuente	
Eugen Paraschiv	Sin fecha	Spring Security Form Login	www.baeldung.com	https://www.baeldung.com/spring-security-login
Elivar Largo	14 de octubre de 2021	Proyecto eCommerce con Spring Boot: Crear el repository y clase servicio CRUD producto 07	Youtube	https://www.youtube.com/watch?v=KuRYI7_BNjc&list=PL3vXkSIW2FvU9z7Gz_Nn3E69HjEv55_G&index=7&ab_channel=ElivarLargo
Ramesh Fadatare	Sin fecha	Spring Boot User Registration and Login Example Tutorial	www.javaguides.net	https://www.javaguides.net/2018/10/user-registration-module-using-spring-boot-springmvc-springsecurity-hibernate5-thymeleaf-mysql.html
Balvinder Singh	16 de mayo de 2019	Fetch Image from URL in Spring Boot(Java) to Upload/save locally	medium.com	https://medium.com/tekraxe/fetch-image-from-url-in-spring-boot-java-to-upload-save-locally-6b90c2bdc2ba

Abhinav Chola	27 de enero de 2022	Spring Boot MySQL Integration: 6 Easy Steps	hevodata.com	https://hevodata.com/learn/spring-boot-mysql/
Noel Rodríguez Calle	6 de marzo de 2023	Uso de Controladores en Spring Boot	refactorizando.com	https://refactorizando.com/uso-controladores-spring-boot/
La Tecnología Avanza	19 de noviembre de 2021	Inicio de sesión y registro de usuarios con Spring Security + Thymeleaf + MySQL y Bootstrap	Youtube	https://www.youtube.com/watch?v=0wTsLRxS3gA&ab_channel=LaTecnolog%C3%ADAvanza
benzkoder	12 de mayo de 2023	JPA Many to Many example with Hibernate in Spring Boot	www.benzkoder.com	https://www.benzkoder.com/jpa-many-to-many/
neovasolutions	2 de agosto de 2022		www.neovasolutions.com	https://www.neovasolutions.com/2022/08/02/how-to-handle-keyboard-actions-in-robot-framework/#:~:text=Robot%20Framework%20provides%20the%20'Press,pressing%20keys%20on%20the%20element.