17 DE MAYO DE 2023

# DOCUMENTACIÓN FINAL


GONZALO ALONSO LIDÓN
JUAN C. VECINO DE HARO

# Contenido

# Introducción

El proyecto que se ha desarrollado consiste en la creación de una página web que busca brindar una experiencia similar a la reconocida plataforma de comercio electrónico, Amazon. Esta página web permitirá a los usuarios explorar una amplia variedad de productos y realizar compras de manera segura y conveniente.

El objetivo principal del proyecto es proporcionar una plataforma intuitiva y funcional donde los usuarios puedan registrarse, navegar por diferentes productos, agregar artículos a su carrito de compras y finalizar sus compras mediante un proceso de pago seguro.

La página web se ha desarrollado utilizando tecnologías de vanguardia, aprovechando el framework de desarrollo Spring y la tecnología JPA (Java Persistence API) para interactuar con la base de datos subyacente. Además, se han implementado diversos servicios y funciones que garantizan un flujo de trabajo eficiente y una experiencia de usuario fluida.

Además de la funcionalidad de compra, se ha implementado un sistema de estadísticas para recopilar información sobre las páginas más visitadas y los productos más populares. Esta información será de gran utilidad para mejorar la navegación del sitio y brindar recomendaciones personalizadas a los usuarios.

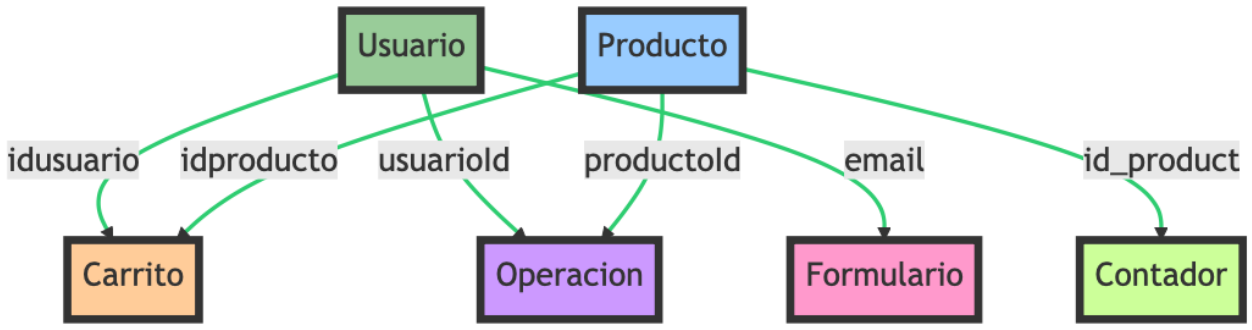
En resumen, el proyecto de desarrollo de esta página web de comercio electrónico tiene como objetivo proporcionar una experiencia similar a Amazon, permitiendo a los usuarios explorar, agregar productos al carrito y realizar compras de manera segura. La implementación se basa en tecnologías modernas y se ha prestado atención a la funcionalidad y la seguridad del sistema.

# Entidades

1. **Carrito**: Esta entidad representa el carrito de compras de un usuario en tu aplicación. Cada registro en esta tabla representa un producto en el carrito de un usuario específico. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **idusuario**: Identifica al usuario que posee el carrito.
    - **idproducto**: Identifica el producto que se ha añadido al carrito.
    - **unidades**: Representa la cantidad de ese producto en el carrito.
2. **Contador**: Esta entidad se utiliza para llevar un registro de las estadísticas de la página, como qué páginas son las más visitadas. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **nombre**: El nombre de la página o vista que se está contando.
    - **valor**: El número de veces que se ha visitado la página o vista.

3. **Formulario**: Esta entidad almacena los formularios enviados por los usuarios. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **nombre**, **email**, **telefono**, **mensaje**: Estos campos representan la información proporcionada por el usuario en el formulario.
4. **Operacion**: Esta entidad registra las compras realizadas por los usuarios. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **usuarioId**: Identifica al usuario que realizó la compra.
    - **productoId**: Identifica el producto que se compró.
    - **tipo**: El tipo de operación realizada.
    - **fecha**: La fecha y hora en que se realizó la operación.
5. **Producto**: Esta entidad representa los productos disponibles para la compra en tu aplicación. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **id_product**: Un identificador único para cada producto.
    - **nombre**, **descripcion**, **price**, **url**, **valoracion**: Estos campos representan la información del producto, incluyendo su nombre, descripción, precio, URL de la imagen y valoración.
6. **Usuario**: Esta entidad representa a los usuarios registrados en tu aplicación. Las columnas son las siguientes:
    - **id**: Es la clave primaria de la tabla, un identificador único para cada registro.
    - **email**, **usuario**, **credenciales**, **rol**, **domicilio**: Estos campos representan la información del usuario, incluyendo su email, nombre de usuario, credenciales, rol y dirección.

Diagrama de la Base de Datos



- La entidad "Usuario" se relaciona con "Carrito" y "Operacion" a través del campo **idusuario** y **usuarioId** respectivamente.
- La entidad "Producto" se relaciona con "Carrito", "Operacion" y "Contador" a través de los campos **idproducto**, **productoId** y **id_product** respectivamente.
- La entidad "Usuario" también se relaciona con "Formulario" a través del campo **email**.

## Controladores

**ControladorCarrito**

Este controlador gestiona las operaciones CRUD (Crear, Leer, Actualizar, Borrar) para la entidad **Carrito**. Utiliza **RepoCarrito** para interactuar con la base de datos.

Métodos

- **crea(@RequestBody Carrito carrito)**: Este método se encarga de crear un nuevo carrito en la base de datos. Se accede a él a través de una petición POST a "/api/carrito". Recibe como parámetro un objeto **Carrito** en el cuerpo de la petición y devuelve el carrito creado. El estado HTTP de la respuesta es 201 (CREATED).
- **lee(@PathVariable("id_usuario") String id_usuario)**: Este método se encarga de leer el carrito de un usuario específico en la base de datos. Se accede a él a través de una petición GET a "/api/carrito/{id_usuario}", donde "{id_usuario}" es el ID del usuario cuyo carrito se quiere leer. Devuelve una lista de objetos **Carrito** que pertenecen al usuario especificado. El estado HTTP de la respuesta es 200 (OK).
- **borra(@PathVariable("id_carrito") Long id_carrito)**: Este método se encarga de borrar un carrito específico de la base de datos. Se accede a él a través de una petición DELETE a "/api/carrito/del/{id_carrito}", donde "{id_carrito}" es el ID del carrito que se quiere borrar. Eta request, como es lógico no devuelve nada dado a que solamente borra. El estado HTTP de la respuesta es 200 (OK).

**ControladorFormulario**

Este controlador gestiona las operaciones para la entidad **Formulario**. Utiliza **RepoFormulario** para interactuar con la base de datos.

Métodos

- **registrar_mensaje(@RequestBody Formulario formulario)**: Este método se encarga de registrar un nuevo mensaje en la base de datos. Se accede a él a través de una petición POST a "/api/formulario". Recibe como parámetro un objeto **Formulario** en el cuerpo de la petición. Antes de guardar el formulario, verifica si ya existe un formulario con el mismo correo electrónico. Si existe, lanza una excepción con el estado HTTP 400 (BAD REQUEST) y un

mensaje de error. Si no existe, guarda el formulario y lo devuelve. El estado HTTP de la respuesta, si la operación es exitosa, es 201 (CREATED).

**ControladorOperacion**

Este controlador gestiona las operaciones para la entidad **Operacion**. Utiliza **ServicioOperaciones** para interactuar con la base de datos.

Métodos

- **insertarOperacion(@PathVariable("id_usuario") String id_usuario)**: Este método se encarga de insertar una nueva operación en la base de datos. Se accede a él a través de una petición POST a "/operacion/{id_usuario}", donde "{id_usuario}" es el ID del usuario que realiza la operación. No devuelve nada. El estado HTTP de la respuesta es 200 (OK).

**ControladorProductos**

Este controlador gestiona las operaciones para la entidad **Producto**. Utiliza **RepoProducto** para interactuar con la base de datos.

Métodos

- **getProductos()**: Este método se encarga de obtener todos los productos de la base de datos. Se accede a él a través de una petición GET a "/api/productos". Devuelve una lista de objetos **Producto**. El estado HTTP de la respuesta es 200 (OK).
- **getProducto(@PathVariable("id") Long id)**: Este método se encarga de obtener un producto específico de la base de datos. Se accede a él a través de una petición GET a "/api/productos/{id}", donde "{id}" es el ID del producto que se quiere obtener. Devuelve un objeto **Producto** si el producto existe, de lo contrario lanza una excepción. El estado HTTP de la respuesta, si la operación es exitosa, es 200 (OK). Este método es el que nos va a servir en la página de producto para hacerla personalizada a cada uno de ellos en función de cual se haya hecho click.

**ControladorUsuario**

Este controlador gestiona las operaciones para la entidad **Usuario**. Utiliza **RepoUsuario** para interactuar con la base de datos.

Métodos

- **login(@RequestHeader("Authorization") String credenciales)**: Este método se encarga de autenticar a un usuario. Se accede a él a través de una petición GET a "/api/usuario/login". Recibe las credenciales del usuario en el encabezado de la petición. Si las credenciales son válidas, devuelve el objeto **Usuario** correspondiente, de lo contrario lanza una excepción con el estado HTTP 401 (UNAUTHORIZED).
- **registrar(@RequestBody Usuario usuario)**: Este método se encarga de registrar un nuevo usuario. Se accede a él a través de una petición POST a "/api/usuario". Recibe como parámetro

un objeto **Usuario** en el cuerpo de la petición. Antes de guardar el usuario, verifica si ya existe un usuario con el mismo correo electrónico. Si existe, lanza una excepción con el estado HTTP 400 (BAD REQUEST). Si no existe, guarda el usuario y lo devuelve. El estado HTTP de la respuesta, si la operación es exitosa, es 201 (CREATED).

- **getuser(@PathVariable String email)**: Este método se encarga de obtener un usuario específico de la base de datos. Se accede a él a través de una petición GET a "/api/{email}", donde "{email}" es el correo electrónico del usuario que se quiere obtener. Devuelve un objeto **Usuario** si el usuario existe. El estado HTTP de la respuesta es 200 (OK).

**ServicioOperacion**

El servicio "ServicioOperacionesImpl" es una implementación de la interfaz "ServicioOperaciones". Este servicio se encarga de realizar operaciones relacionadas con las transacciones y manipulación de datos en la base de datos.

El servicio dispone de una única función. Esta se encarga de insertar una nueva operación en la base de datos. Recibe como parámetro el ID del usuario para el cual se realizará la operación. Cabe recalcar, que se ha decidido hacer uso de un servicio porque no se podría hacer esto con una sola *query*.

En primer lugar, el servicio obtiene una lista de carritos asociados al usuario mediante el repositorio "RepoCarrito". Luego, itera sobre cada carrito de la lista y crea una nueva instancia de la entidad "Operacion". Asigna el ID del usuario y el ID del producto asociados al carrito a la operación. También establece el tipo de operación como "COMPRA" y la fecha actual.

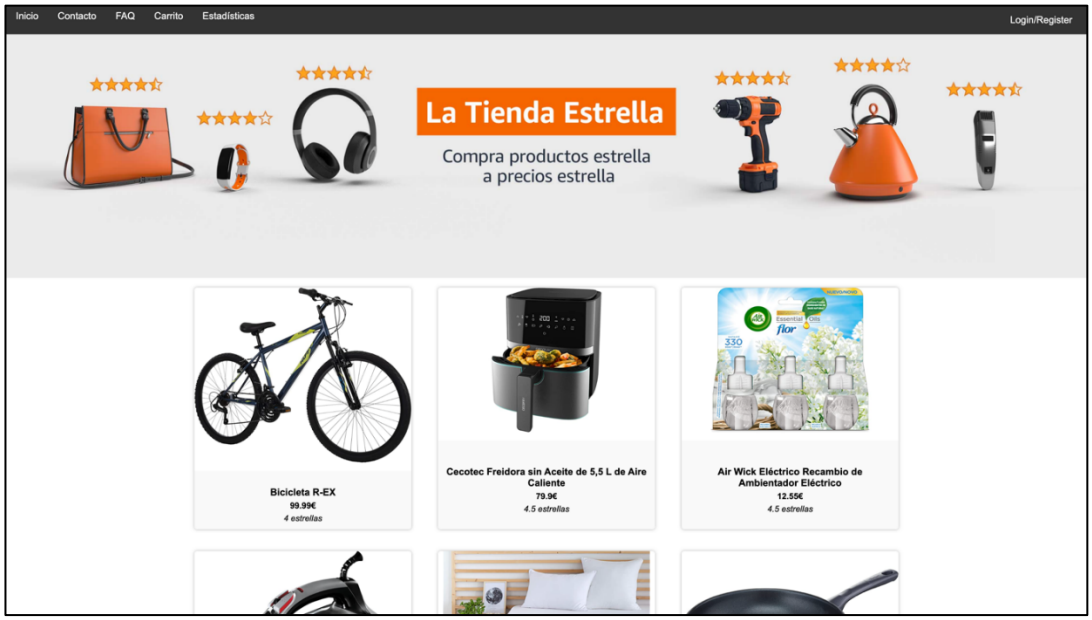
A continuación, se guarda la operación en la base de datos mediante el repositorio "RepoOperacion" utilizando el método "save". Luego, se elimina el carrito correspondiente utilizando el método "deleteById" del repositorio "RepoCarrito".

Si no se encuentran carritos para el usuario especificado, se lanza una excepción del tipo "IllegalStateException" con un mensaje indicando que no se encontraron carritos para ese usuario.

En resumen, el servicio "ServicioOperacionesImpl" proporciona una funcionalidad para insertar operaciones en la base de datos, asociadas a usuarios y productos específicos. Además, realiza la eliminación de los carritos correspondientes después de realizar la operación. Este servicio es parte de una aplicación Spring con JPA y está diseñado para interactuar con los repositorios "RepoCarrito" y "RepoOperacion" para persistir los datos en la base de datos.

# Web App

## Página principal




Como vista principal de la página hemos optado por un diseño muy atractivo por medio de la introducción de un “carrusel” de anuncios en la parte superior. De la misma manera, hacemos uso de tonos anaranjados a lo largo de toda la web app para que intentar mantener un hilo conductor a lo largo de todo el proyecto.

A continuación del carrusel disponemos de la vista de los productos disponibles para la compra junto con la información más esencial como el precio o valoración. Esta sección es completamente dependiente del JavaScript y está directamente relacionada con la información disponible en la base de datos acerca de los productos.

Por último, tenemos que considerar que las pestañas superiores nos permiten navegar de forma dinámica por toda la aplicación y el botón “Login/Register” hacer uso de nuestras credenciales para navegar a través de la pagina web. Este cambia si estamos logados como un usuario.

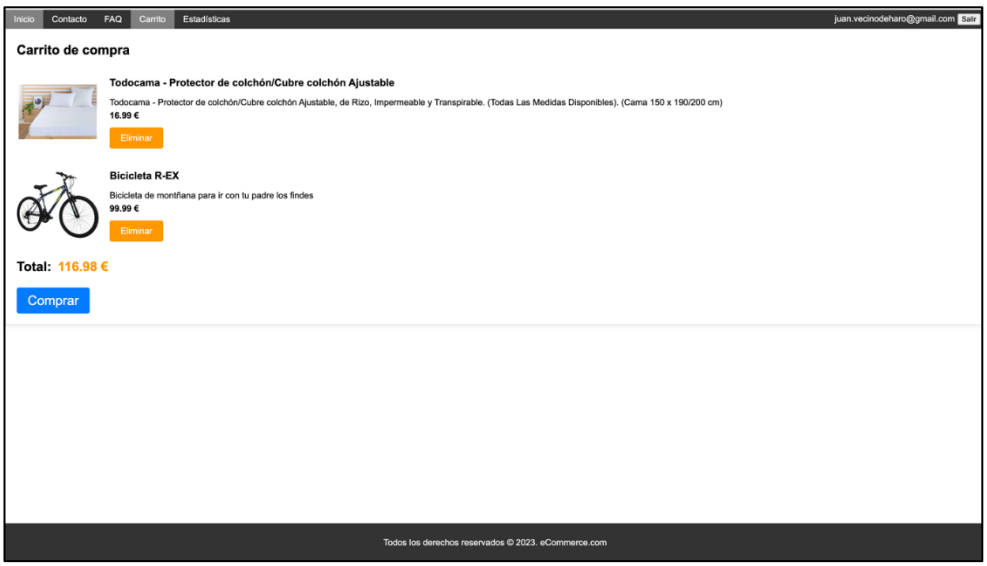
Cabe recalcar, que esta información cuando nos registramos se va a almacenar en el LocalStorage, lo cual nos va a permitir refrescar la página y manteniendo la sesión. Esto va a dar una buena experiencia al usuario en nuestra página.

## Página de un producto


La siguiente vista corresponde con la del producto seleccionado. En esta, como podemos ver, tenemos una vista principal de la imagen del producto, acompañado de una breve descripción de este, su precio, valoración y un botón “Comprar” que añade el producto seleccionado al carrito del usuario logado en el momento.

Tenemos que considerar que la pagina de HTML es común para todo producto y este cambia con ayuda del archivo JavaScript correspondiente dependiendo del producto que hayamos seleccionado. Cabe destacar que toda la información del producto se encuentra en su tabla en la base de datos, haciendo así mucho más dinámico y configurable el añadido o retirada de productos, fomentando la escalabilidad de la web app.

# Página del Carrito



A continuación, tenemos que considerar la “cesta”. En esta vista podemos ver que a medida que vayamos añadiendo productos a ella esta incrementara de manera que podamos tramitar el pago una vez tengamos todos los que deseemos dentro de ella a través del botón comprar.

Cabe destacar que se pueden tanto añadir como suprimir productos y que el precio de estos afectara directamente al total que figura en la página. Cada usuario dispone de un “Carrito” personal, y cuando estos salen de la pagina no se pierde para posibles futuras compras.

# Test

## Test ControladorUsuario (E2E)

Este es una prueba de extremo a extremo (E2E) para el **ControladorUsuario**. El objetivo de esta prueba es simular el flujo completo de un usuario que intenta registrarse y acceder al sistema.

La prueba sigue los siguientes pasos:

1. Inicialmente, el test intenta realizar un inicio de sesión con unas credenciales de usuario que aún no existen en el sistema. Como se espera, el sistema responde con un estado HTTP 401 (UNAUTHORIZED), indicando que las credenciales proporcionadas no son válidas.
2. A continuación, el test procede a registrar un nuevo usuario en el sistema. Para ello, crea un objeto **Usuario** con los datos del nuevo usuario y realiza una petición POST a "/api/usuario". El sistema debería responder con un estado HTTP 201 (CREATED), indicando que el usuario ha sido creado con éxito.
3. Para verificar que el usuario ha sido creado correctamente, el test comprueba que el cuerpo de la respuesta coincide con los datos del usuario que se acaba de registrar.
4. Después de registrar al usuario, el test intenta registrar nuevamente al mismo usuario. Como el correo electrónico del usuario ya está en uso, el sistema debería responder con un estado HTTP 400 (BAD REQUEST), indicando que no se puede registrar al usuario.
5. A continuación, el test realiza una petición GET a "/api/{email}", donde "{email}" es el correo electrónico del usuario que se acaba de registrar. El sistema debería responder con un estado HTTP 200 (OK), y el cuerpo de la respuesta debería coincidir con los datos del usuario registrado.
6. Finalmente, el test intenta nuevamente realizar un inicio de sesión con las credenciales del usuario que se acaba de registrar. Esta vez, como el usuario ya existe en el sistema, el inicio de sesión debería ser exitoso y el sistema debería responder con un estado HTTP 200 (OK).

Este test asegura que el sistema maneja correctamente el registro y el inicio de sesión de los usuarios, y que rechaza correctamente los intentos de registro cuando el correo electrónico ya está en uso. Además, verifica que los datos del usuario se guardan y se recuperan correctamente de la base de datos.

Posteriromente se han realizado unos test unitarios de la aplicación en donde se pruban cada uno de los módulos de forma separada.

# Conclusión

La funcionalidad de una web app a través del *framework* de Spring prueba ser muy efectiva y versátil a la hora de programar y configurarla. Considerando que hemos intentado emular el comportamiento de una web de *e-commerce* por medio de credenciales y usuarios logados, y el uso de bases de datos para ello, podemos probar que el funcionamiento de esta es correcto dentro de lo establecido.

El uso de una JPA y el funcionamiento dinámico de la interfaz es muy útil a la hora de llevar a cabo proyectos de mayor envergadura.