IES AZARQUIEL

Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web (DAW)

Curso 2024/2025

Proyecto Final

EuroExplorer: Plataforma de rutas turísticas por Europa

Autora: Elena Díaz Pérez

Grupo: 2º DAW

Tutor del proyecto: José Javier Martínez Cañadas

Fecha de entrega: 02/06/2025

Índice

Capítulo 1 – Introducción y objetivos

En la actualidad, viajar se ha convertido en una parte fundamental de la vida de muchas personas. Ya sea para desconectar, conocer nuevas culturas o simplemente disfrutar del tiempo libre, cada vez hay más interés en explorar nuevos destinos. Sin embargo, con tantas opciones disponibles, elegir un lugar puede ser complicado.

A partir de esto nace Euro Explorer, una aplicación web desarrollada como proyecto final del grado de Desarrollo de Aplicaciones Web (DAW). Su objetivo es ayudar a los usuarios a encontrar su destino ideal en Europa, mostrándoles rutas turísticas clasificadas por tipo (naturaleza, costa, cultural), país y región.

Cada ruta incluye información útil como itinerario, ubicación, otros datos, valoración media y comentarios de otros usuarios y pueden descargar la información en PDF para poder consultarla sin conexión. Además, los usuarios pueden registrarse para guardar rutas en favoritos, dejar sus propias valoraciones y comentarios.

Una de las funciones destacadas es la sección de “Aleatorio”, que muestra tres rutas de cada tipo al azar, ofreciendo ideas para quienes no saben a dónde viajar. También hay una sección de “Las rutas mejor valoradas”, donde se muestran las tres más populares según las puntuaciones de los usuarios.

La aplicación permite navegar sin estar registrado, pero para poder comentar, puntuar o añadir a favoritos una ruta, sí es necesario iniciar sesión. Las contraseñas están cifradas para proteger los datos personales de los usuarios.

Todo el desarrollo del proyecto lo he realizado de forma individual, aplicando lo aprendido a lo largo del grado. Supone un reto completo que combina diseño web, programación en frontend y backend, gestión de base de datos y despliegue. Además, he intentado que el resultado final no solo cumpla con los requisitos académicos, sino que sea una herramienta útil y atractiva para cualquier persona interesada en viajar.

Objetivos del proyecto

Objetivo general:

El principal objetivo es desarrollar una aplicación web llamada EuroExplorer que permita al usuario descubrir rutas turísticas por Europa, organizado por tipo, región y país, incluyendo

funcionalidades como puntuaciones, comentarios, favoritos y descarga de información en PDF para dar más agilidad al usuario.

Objetivos específicos:

- Aplicar los conocimientos aprendidos durante el curso para hacer una aplicación web completa y funcional.
- Diseñar una interfaz clara, intuitiva y adaptada a dispositivos móviles.
- Permitir a los usuarios consultar rutas sin necesidad de registrarse.
- Implementar una sección de las rutas mejor valoradas.
- Permite la descarga de información de las rutas en formato PDF.
- Asegurar una arquitectura del código clara y bien estructurada.

Capítulo 2 – Especificación de Requisitos

En este capítulo explico los requisitos funcionales y no funcionales de EuroExplorer, así como los aspectos técnicos y los distintos tipos de usuarios que pueden interactuar con la web. Todo esto sirve como guía para desarrollar el proyecto de forma clara y organizada.

2.1 Requisitos funcionales

Son las funcionalidades que debe tener la aplicación para qué funcione perfectamente.

En este caso, EuroExplorer debe permitir lo siguiente:

- Explorar rutas turísticas por Europa.
- Filtrar rutas por tipo: costa, cultural o naturaleza y a su vez por región.
- Mostrar los detalles de cada ruta (nombre, país, región, itinerario, ubicación, comentarios, puntuación, etc.).
- Permitir registrarse como usuario.
- Iniciar sesión y cerrar sesión.
- Cifrar las contraseñas del usuario para garantizar la seguridad.
- Puntuar rutas.
- Comentar rutas.
- Añadir o quitar rutas de una lista de favoritos.
- Generar un PDF con los detalles de la ruta.
- Mostrar rutas aleatorias por tipo (costa, naturaleza, cultural).
- Recargar rutas aleatorias para obtener nuevas sugerencias.
- Mostrar el top 3 de rutas mejores valoradas.
- Permitir que cualquier usuario pueda navegar por la web sin registrarse.
- Permitir funciones como comentar, puntuar o guardar en favoritos solo para usuarios registrados.

2.2 Requisitos no funcionales

Estos requisitos definen cómo debe comportarse la aplicación a nivel de rendimiento, usabilidad, seguridad...

- La interfaz debe ser responsiva y adaptarse a distintos dispositivos (móviles, tablets, ordenadores).

- La navegación debe ser clara, sencilla e intuitiva.
- Las contraseñas deben cifrarse para proteger los datos del usuario.
- El proyecto debe seguir una arquitectura por capas MVC (Modelo-Vista-Controlador).
- El código debe estar ordenado y comentado para facilitar el mantenimiento y futuras mejoras.

2.3 Usuarios del sistema

Existen dos perfiles principales en la aplicación:

- **Usuario no registrado**: Puede navegar libremente por toda la web, ver rutas, acceder a la sección aleatorio, también ver los países, y detalles de las rutas y descargar la información en PDF, pero no puede comentar, puntuar ni guardar o quitar rutas de favoritos.
- **Usuario registrado**: Puede acceder a todas las funcionalidades como comentar, puntuar, marcar favoritos o quitar de favoritos. Además, puede ver la página de preguntas frecuentes donde hay información útil para el usuario y su listado de rutas favoritas, en la cual verá todas las rutas a las que le ha dado me gusta y tendrá opción de quitarlas de esta lista.

Capítulo 3 – Planificación Temporal y Evaluación de Costes

3.1 Planificación temporal

El desarrollo de EuroExplorer lo he llevado a cabo durante aproximadamente dos meses, trabajando casi a diario, dedicando entre 1 y 4 horas al día. Aunque la planificación inicial dice unas 30-40 horas, el tiempo real ha sido mayor debido a la complejidad de algunas funcionalidades, por ejemplo, la creación de datos para la base, mejorar la interfaz...

Aunque la duración supera lo previsto, este esfuerzo extra me ha permitido pulir detalles, resolver errores y mejorar la calidad general de toda la web.

3.2 Evaluación de costes

Como se trata de un proyecto del curso, no hay costes económicos. Sin embargo, si fuese un proyecto real, si habría coste.

En primer lugar, hay que tener en cuenta la mano de obra, que sería el coste más caro. En cuanto a las herramientas de desarrollo he usado InteliJ, Visual Studio Code y Oracle Cloud. En mi caso, no he tenido que pagar nada por la base de datos en Oracle ya que el centro nos ofrece una base de datos gratuita, pero si fuese para un proyecto real tendría costes.

Las tecnologías que he utilizado son gratuitas (Spring Boot, Thymleaf, Bootstrap, Select2, Flying Saucer) por lo que no genera costes.

Respecto al alojamiento web, aunque la aplicación este pensada para ejecutarse principalmente en local, la he desplegado en Render con su plan gratuito. Si fuese para uso profesional habría que pagar un plan superior ya que este plan gratuito tiene muchas limitaciones.

3.3 Conclusión de la planificación y costes

La planificación del proyecto ha sido flexible adaptándome a los cambios que iban surgiendo. Durante los dos meses que ha durado el proyecto, he trabajado de forma regular en las distintas partes: backend, frontend, base de datos, diseño y pruebas.

Aunque he invertido más tiempo del previsto inicialmente, este esfuerzo me ha permitido mejorar muchos detalles y conseguir un resultado más completo y profesional.

Si EuroExplorer se publicara como un proyecto profesional: En caso de desplegar la aplicación, también habría que contratar un dominio web, lo que supone otro gasto más. Actualmente, se accede mediante la URL que proporciona Render, pero para un uso profesional sería mejor tener un dominio propio. Por lo que habría que añadir costes como el registro de un dominio personalizado (entre 10 y 20€ anuales), un plan de hosting (desde 7€ al mes), e incluso el pago por una base de datos con más capacidad en Oracle Cloud si se supera el plan gratuito. A todo esto, habría que sumar el coste del tiempo de desarrollo.

Como ya he dicho mi página web está desplegada, y esta es la URL por la cual se puede acceder: https://euroexplorer.onrender.com

Capítulo 4 – Tecnologías utilizadas

A lo largo del proyecto he utilizado varias tecnologías modernas con las que he podido construir una aplicación web completa y funcional. A continuación, explico las principales herramientas que he usado, agrupadas por partes del proyecto:

4.1 Backend

El backend de EuroExplorer lo he desarrollado con Java y Spring Boot, un framework moderno y muy usado en entornos profesionales. He elegido Spring Boot porque permite crear aplicaciones web completas de forma rápida y estructurada y por su integración con herramientas como JPA, Thymeleaf o Spring Security.

He podido organizar el proyecto de forma limpia, siguiendo el patrón MVC, y añadir funcionalidades como seguridad, controladores, servicios, acceso a base de datos…

Tecnologías principales empleadas:

- **Spring Data JPA**: Lo he usado para gestionar las operaciones con la base de datos de forma sencilla, mediante interfaces que extienden JpaRepository.
- **Spring Security**: para gestionar la seguridad de la aplicación. He cifrado las contraseñas y he controlado el acceso a funcionalidades solo para usuarios registrados.
- **Thymeleaf:** Aunque también forma parte del frontend, va muy ligado a Spring Boot. Lo he usado para generar las vistas HTML con los datos dinámicos que llegan desde el backend.

Dependencias destacadas:

- **Spring-boot-started-web**: para construir los controladores y gestionar las rutas HTTP.
- **Spring-boot-starter-data-jpa**: para gestionar la persistencia de datos con JPA o Hibernate.
- **Spring-boot-starter-thymeleaf**: para la integración de las vistas HTML.
- **Spring-security-crypto**: para cifrar contraseñas de forma segura.
- **Ojdbc11**: driver para conectar con bases de datos Oracle.
- **Flying-saucer-pdf**: generación de documentos PDF desde HTML.

4.2 Frontend

Para el desarrollo del frontend del proyecto he utilizado una combinación de tecnologías que me han permitido crear una interfaz de usuario moderna, funcional y atractiva, manteniendo siempre la facilidad de mantenimiento y el enfoque en la experiencia del usuario.

- **Thymeleaf:** Como motor de plantillas de Spring, me ha permitido incrustar de forma sencilla datos del backend directamente en las páginas HTML. Gracias a su sintaxis natural y su integración con el modelo MVC de Spring, ha sido posible generar contenido dinámico sin complicaciones.
- **Bootstrap**: He optado por utilizar Bootstrap como base para el diseño de la interfaz. La razón principal ha sido su facilidad de uso y su amplia colección de componentes prediseñados, que permiten construir interfaces responsivas y estéticamente agradables con poco esfuerzo. Inicialmente valoré la opción de utilizar Tailwind CSS, que ofrece un enfoque más flexible, pero finalmente me decidí por Bootstrap debido a lo mucho que lo he utilizado en proyectos similares y a la gran cantidad de documentación y ejemplos disponibles, lo cual ha acelerado el desarrollo.
- **CSS personalizado**: Aunque Bootstrap cubre una gran parte de las necesidades visuales, he añadido estilos personalizados con CSS, para darle al proyecto una identidad visual única. Esto ha permitido ajustar ciertos detalles de diseño que no estaban completamente cubiertos por Bootstrap, como la paleta de colores, márgenes específicos o animaciones suaves. Gracias a esta combinación, la interfaz mantiene un equilibrio entre coherencia visual y personalización.
- **JavaScript:** He utilizado JavaScript en aquellas partes del proyecto donde era necesario aportar interactividad adicional. Algunos ejemplos son la validación de formularios, la implementación de confirmaciones antes de eliminar elementos (como comentarios). Aunque no ha sido el lenguaje principal en el frontend, ha resultado muy útil para completar la funcionalidad y hacer que la aplicación sea más dinámica y usable.

4.3 Base de datos

- **Oracle Cloud (Autonomous Database Free)**: Es el sistema de base de datos que he utilizado. El centro nos proporciona acceso gratuito y desde ahí he creado las tablas y añadido los datos.
- **Oracle SQL Developer**: Herramienta que me ha servido para gestionar y visualizar la base de datos desde el ordenador.
- **Conexión mediante Wallet**: He incluido el wallet dentro del proyecto para poder conectarme a la base de datos desde el backend, incluso después de subir el proyecto a GitHub y desplegarlo en Render.
- **Oracle SQL Data Modeler**: Herramienta para hacer un modelo de entidad-relación.

He decidido utilizar una base de datos en la nube de Oracle Cloud en lugar de otras opciones más comunes como MySQL, por varias razones.

En primer lugar, Oracle Cloud ofrece una solución de base de datos gestionada, lo que significa que muchas tareas como el mantenimiento, las copias de seguridad o las actualizaciones se realizan de forma automática. Esto me ha permitido centrarme más en el desarrollo de la aplicación y menos en la administración del sistema.

Además, Oracle incluye funcionalidades avanzadas como el ajuste automático del rendimiento, el cifrado de datos, y herramientas para la recuperación ante errores, que no están disponibles de forma nativa en MySQL sin configuraciones adicionales.

Por todo ello, he considerado que Oracle Cloud era la opción más completa y adecuada para este proyecto, tanto a nivel técnico como formativo.

4.4 Librerías externas

Select2: Mejora los selectores en los filtros para que sean más fáciles de usar y visuales.

Flying Saucer: Librería que me permite generar documentos PDF con el contenido de una ruta.

4.5 Despliegue y herramientas externas

Render: es la plataforma donde he desplegado la aplicación de forma gratuita. He usado un plan básico para que esté accesible desde cualquier sitio.

- **Dockerfile**: He creado un Dockerfile para poder empaquetar todo el backend con las dependencias necesarias y subirlo correctamente a Render.
    - **GitHub**: Todo el proyecto publicado en GitHub, para poder trabajar de forma segura y subir el código al despliegue. Este es el enlace al repositorio con todo el proyecto: https://github.com/elenadiazp/EuroExplorer

Capítulo 5 – Desarrollo e Implementación

5.1 Arquitectura general del proyecto

Mi proyecto está organizado usando una arquitectura en capas siguiendo el patrón MVC (Modelo-Vista-Controlador). Esto me ha ayudado a separar bien las distintas partes del programa, por lo que está más ordenado, es más fácil de entender y también más fácil de modificar o ampliar.

A continuación, explico cada parte:

Modelo

En la carpeta **entidades** es donde tengo definidas las entidades que representas las tablas de la base de datos, como, por ejemplo:

- Usuario
- Ruta
- País
- Región

```java
@Entity  elenadiazp
@Table(name = "RUTA")
public class Ruta {
    @Id
    @Column(name = "ID", nullable = false)
    private Long id;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @OnDelete(action = OnDeleteAction.RESTRICT)
    @JoinColumn(name = "IDTIPORUTA", nullable = false)
    private TipoRuta idtiporuta;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @OnDelete(action = OnDeleteAction.RESTRICT)
    @JoinColumn(name = "IDREGION", nullable = false)
    private Region idregion;
```

Además. He creado algunos DTOs (objetos para trasferir datos) para no trabajar directamente con las entidades en ciertas partes del proyecto. Esto hace que el código sea más claro y más seguro a la hora de manejar la información.

Repositorio

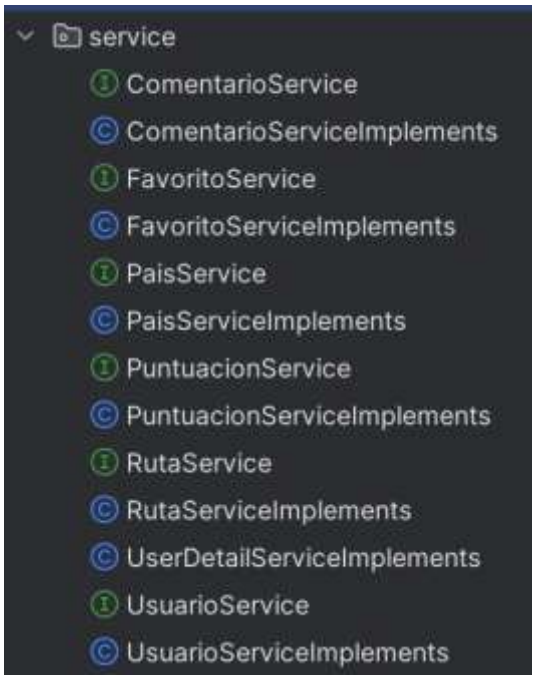
En la carpeta **daos** están las interfaces que extienden de JpaRepository. Estas interfaces me permiten acceder fácilmente a la base de datos y hacer operaciones como guardar, buscar o eliminar sin tener que escribir consultas SQL directamente.

```
public interface ComentarioRepository extends JpaRepository<Comentario, Long> {  3 usages  ▲ elenadiazp
    Optional<Comentario> findByIdUsuarioAndIdRuta(Usuario usuario, Ruta ruta);  4 usages  ▲ elenadiazp
    List<Comentario> findAllByIdRuta(Ruta ruta);  1 usage  ▲ elenadiazp
    Page<Comentario>findAllByIdRuta_IdOrderByFechaDesc(Long rutaId, Pageable pageable);  1 usage  ▲ elenadiazp
}
```

Servicio e implementación

Aquí es donde hago la lógica del proyecto.

- He separado los servicios en dos partes: la interfaz (donde se definen las funciones) y la implementación (donde se escribe el código real de esas funciones) para que se vea más limpio.
- En esta parte gestiono lo que pasa entre el controlador y el repositorio, hago comprobaciones, valido datos y organizo la información antes de enviarla a la siguiente capa.



Controladores

Los controladores son los que reciben las peticiones que hace el usuario cuando navega por la web.

- Tengo controladores para cada entidad, como RutaController, UsuarioController…
- También tengo un BienvenidoController que simplemente muestra la página de bienvenida cuando alguien inicia sesión.
- Además, tengo un PdfController que se encarga de recoger los datos necesarios y generar un documento PDF con la información de la ruta seleccionada.

Vista

Las vistas son los archivos .html que se muestran al usuario. Uso Thymeleaf para que el contenido sea dinámico, es decir, que se muestre diferente según los datos.

Tengo un total de 12 páginas, algunas son más estáticas como:

- Peguntasfrecuente.html, que simplemente da información útil.
- Ruta-pdf.html, que es la plantilla usada para generar los PDFs.
- El resto son páginas dinámicas como la de rutas, detallerutas, favoritos….

Recursos

En la carpeta de recursos tengo:

- Archivos CSS para personalizar el estilo de las páginas.
- Imágenes que se usan en las distintas secciones de la web.

Seguridad

Además de todo esto, tengo una carpeta llamada **config** que tiene dentro el fichero de configuración de la seguridad de la web.

Conexiones a la base de datos

Para la conexión a la base de datos he utilizado Oracle Cloud Autonomous Database, en su versión gratuita, ya que el centro nos proporciona acceso a este servicio. Como es una base de datos en la nube, es necesario descargar el wallet de Oracle, que contiene la configuración y certificados necesarios para poder conectarse de forma segura desde una aplicación externa.

Este wallet lo he incluido dentro del propio proyecto (en una carpeta llamada /wallet) para que, al subir el proyecto a GitHub y desplegarlo con Render, la conexión siga funcionando correctamente.

La configuración de la conexión está definida en el archivo application.properties, y contiene las siguientes propiedades:

```
spring.datasource.url=jdbc:oracle:thin:@euroexplorer_high
spring.datasource.username=ADMIN
spring.datasource.password=Azarquiel2025
spring.datasource.driver-class-name=oracle.jdbc.OracleDriver

spring.datasource.hikari.data-source-
properties.oracle.net.tns_admin=./wallet/Wallet_EuroExplorer
```

A continuación, explico qué significa cada línea:

- **Spring.datasourcce.url**: Es la URL de conexión JDBC. En este caso, se conecta al alias euroexplorer_high, que está definido en los archivos del wallet descargado.
- **Spring.datasource.username**: Usuario de la base de datos. En este caso, uso el usuario ADMIN, que es el que viene por defecto en Oracle Cloud cuando se crea la base de datos.
- **Spring.datasource.password**: Contraseña del usuario. No la muestro por motivos de seguridad, pero se escribe aquí directamente para poder hacer la conexión.

- **Spring.datasource.driver-class-name**: Indica qué driver JDBC se usa para conectar con la base de datos Oracle. En este caso es Oracle.jdbc.OracleDriver.
- **Spring.datasource.hikari.data-spurce-properties.oracle.net.tns_admin**: Esta línea es clave para que funcione con Oracle Cloud. Le dice a la aplicación dónde se encuentra el wallet que necesita para conectar con la base de datos. En este caso, está en ./wallet/Walllet_EuroExplorer, una ruta dentro del proyecto.

Estructura base de datos

Para el diseño de la base de datos de EuroExplorer, he realizado un modelo entidad-relación usando la herramienta Oracle SQL Data Modeler, en el que se representan las principales entidades del sistema y las relaciones entre ellas. Este modelo ha sido fundamental para estructurar la información y facilitar el desarrollo de la aplicación.

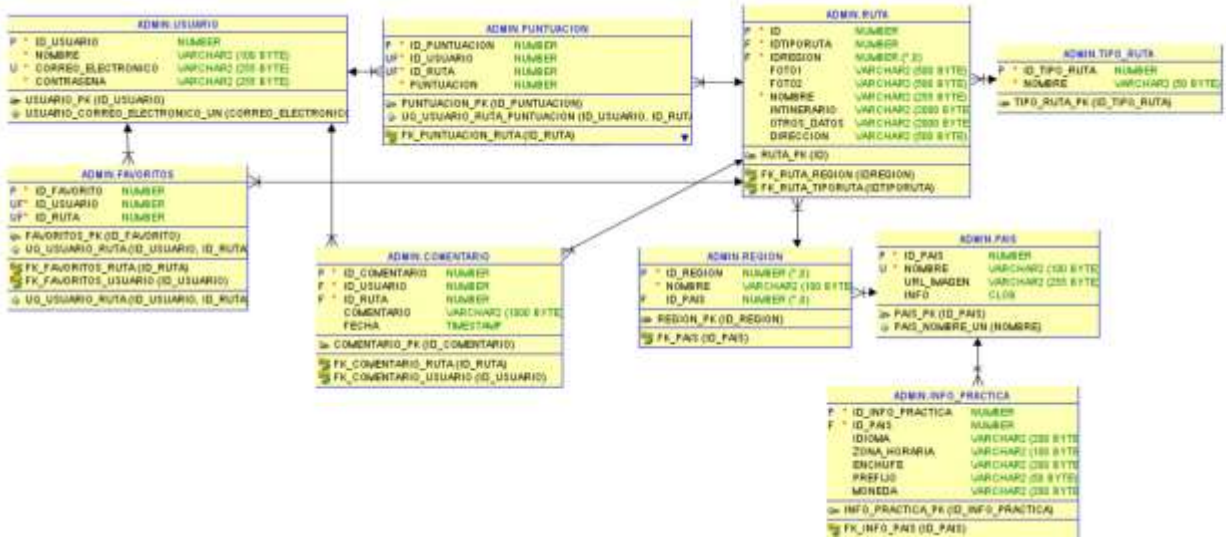
En la siguiente imagen se muestra el modelo, la tabla ruta con todos sus campos, en la que la clave primaria es el id. La entidad tiene relación con:

- La entidad **Puntuación**: una ruta puede tener muchas puntuaciones de distintos usuarios.
- La entidad **Tipo_ruta**: una ruta pertenece a un tipo de ruta, pero un tipo puede agrupar varias rutas.
- La entidad **Región**: una ruta puede pertenecer solo a una región, pero una región puede tener varias rutas.

También se observa la tabla Usuario que está relacionada con la tabla Puntuación, ya que un usuario puede puntuar muchas rutas. La tabla Puntuación actúa como tabla intermedia que conecta usuarios con rutas, incluyendo como claves los ids de ambas entidades, lo que permite saber quién ha valorado qué ruta. Lo mismo pasa con la tabla Favoritos y Comentarios.

La entidad Ruta está relacionada con el país: una región solo puede pertenecer a un país, pero un país puede tener varias regiones.

Además, la entidad País está relacionada con la entidad Info_practica.



Backend(Spring Boot)

El backend de la aplicación está desarrollado con Spring Boot y se encarga de gestionar toda la lógica de negocio de la aplicación, así como la conexión con la base de datos. Está dividido en diferentes capas: entidades (modelo), repositorios, servicios, controladores y vistas.

Gestión de rutas, usuarios, puntuaciones, favoritos, comentarios

Cada entidad (Ruta, Usuario, Favorito, Puntuación…) tiene su propio controlador, servicio y repositorio.

Por ejemplo, en la gestión de rutas:

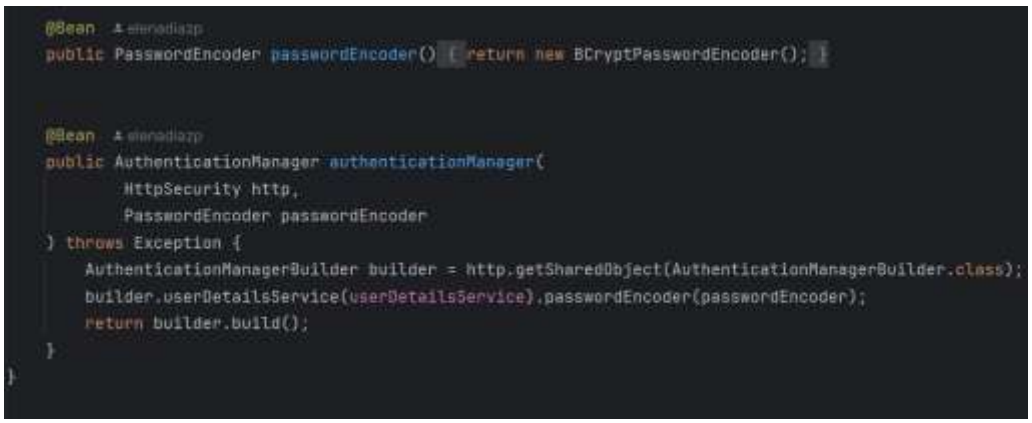
- He creado métodos para listar todas las rutas, buscar por país o tipo, ver los detalles de una ruta concreta, y también mostrar rutas aleatorias o las mejor valoradas.
- Estos métodos están en el RutaService/RutaServiceImplements, que se comunica con el RutaRepository (acceso a base de datos) y se exponen a través del RutaController.

Lo mismo ocurre con usuarios, comentarios, favoritos… Hay servicios para gestionar sus operaciones (crear, leer, actualizar, eliminar), además de funciones específicas como añadir o quitar de favoritos, puntuar, etc.

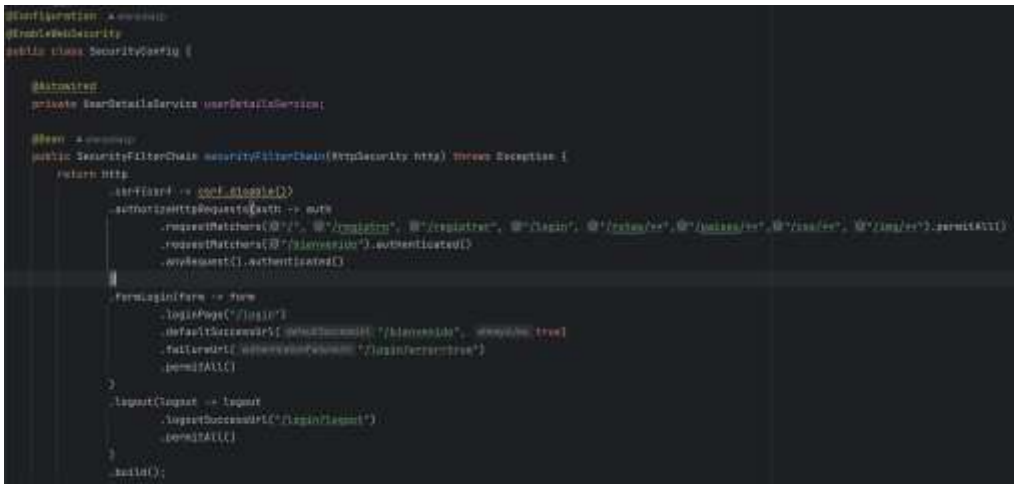
Seguridad: cifrado de contraseñas y control de acceso

Para la seguridad de la aplicación:

Las contraseñas de los usuarios están cifradas con BCrypt, lo que garantiza que no se almacenan en texto plano en la base de datos.



He aplicado un control de acceso básico: cualquier persona puede ver las rutas, pero solo los usuarios registrados pueden comentar, puntuar o guardar en favoritos.



Servicio REST y controladores

Aunque mi aplicación es MVC tradicional y no utiliza una API REST pública, internamente los controladores funcionan como intermediarios entre la lógica y las vistas:

- Reciben peticiones del usuario
- Llaman al servicio correspondiente.
- Devuelven una vista con los datos necesarios.

Ejemplo:

El RutaController tiene un método que recibe una petición para ver una ruta, llama al RutaService y luego devuelve la vista detalleruta.html con los datos de esa ruta.

Validaciones y lógica de negocio

Dentro de los servicios aplico validaciones personalizadas, por ejemplo:

- No dejar que un usuario puntúe dos veces a la misma ruta.
- No permitir guardar en favoritos rutas duplicadas.
- Validar que los campos no estén vacíos antes de guardar un comentario.

También uso validaciones en los formularios del frontend y en algunos casos desde el backend para asegurarme de que los datos son correctos.

```
<form th:action="@{/rutas/{id}/comentar(id=${ruta.id})}" method="post">
    <textarea class="p-2" name="comentarioTexto" rows="4" cols="50"
              required minlength="5" placeholder="Escribe tu comentario aqui..."></textarea>
    <br/>
    <button class=" btn btn-1" type="submit" >
       Añadir comentario
    </button>
</form>
```

Algunos ejemplos de métodos utilizados:

Funcionalidades	Entidad principal	Métodos principales en el servicio
Ver rutas	Ruta	findById (), buscarRutas(), RutasAleatoriasPorTipo(), findByTipo()
Ver países	Países	FindAll(), findById(), buscarPorNombre()
Puntuar	Puntuación	GuardarOPuntuar(),buscarPor Usuario()
Registrar/Login	Usuario	Registrar(), buscarPorCorreo()

5.2 Frontend (Thymeleaf + Bootstrap)

Para el frontend he utilizado Thymeleaf como motor de plantillas, que permite mostrar datos del backend de forma dinámica dentro de archivos HTML. Además, he usado Bootstrap y algo de CSS personalizado para darle un diseño más atractivo y adaptado a todos los dispositivos.

Estructura de las páginas

La aplicación tiene varias páginas principales:

- **Inicio**: página home de bienvenida con accesos rápidos a las secciones destacadas como rutas mejor valoradas, rutas por tipo…
- **Países**: se muestra en formato de cards con una imagen de fondo y el nombre del país, al pinchar en ellos nos llevará a información del país y sus rutas.
- **Rutas**: contiene la información del país con datos de interés de ese país como enchufe, idioma… Y un listado de rutas que se muestran en formato card con información básica: nombre, puntuación media, etiqueta de a qué región pertenece, y un botón para ver más.
- **Detalle de ruta**: al hacer clic en una ruta, se accede a una página con más detalles, como el itinerario, otros datos, ubicación, puntuación media, comentarios y un botón para descargar en PDF.
- **Login y Registro**: formularios para iniciar sesión o registrarse como usuario.
- **Favoritos, bienvenido y preguntas frecuentes:** páginas accesibles solo cuando el usuario ha iniciado sesión.
- **Aleatorio:** aparecen 3 rutas en formato card por cada tipo de ruta, y hay un botón para recargar la página.
- **Ruta-pdf:** página donde se define el formato del PDF.
- **Rutas por tipo**: es una página común para los tres tipos, solo que según qué tipo de ruta te muestra una información u otra. Tiene una pequeña introducción y un listado de rutas de ese tipo en formato card, y además esta paginado.

Cada una de estas páginas se encuentra en la carpeta /templates y están conectadas al backend mediante Thymeleaf.

Uso de componentes

- **Cards**: se usan para mostrar rutas o países en el listado de forma ordenada y visual. Los cards de rutas tienen el nombre de la ruta, una imagen, una etiqueta con el país/region al que pertenecen puntuación media y un corazón para poder añadirla a favoritos.

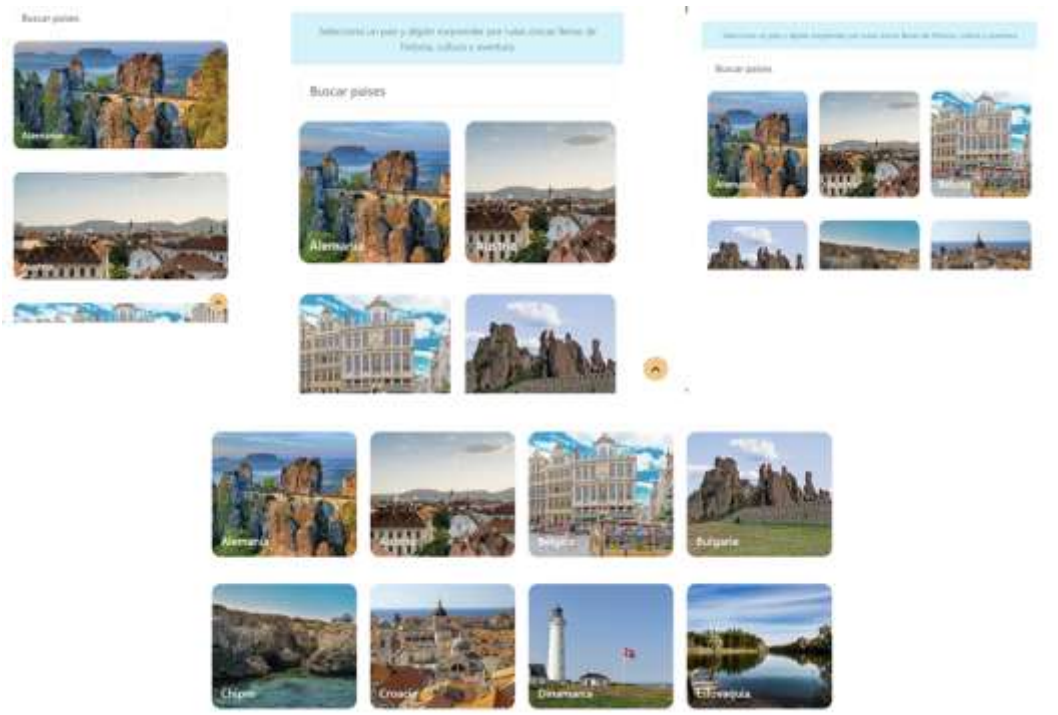
- **Header y Footer**: las barras de navegación son visibles en todas las páginas, y muestra opciones distintas dependiendo de si el usuario ha iniciado sesión o no (por ejemplo: “Iniciar sesión” o “Usuario” con un desplegable a más páginas).

Diseño responsivo

El diseño es completamente responsivo gracias a Bootstrap, lo que significa que se adapta bien a móviles, tablets y ordenadores. He probado diferentes resoluciones para asegurarme de que todo se ve bien.

Por ejemplo:

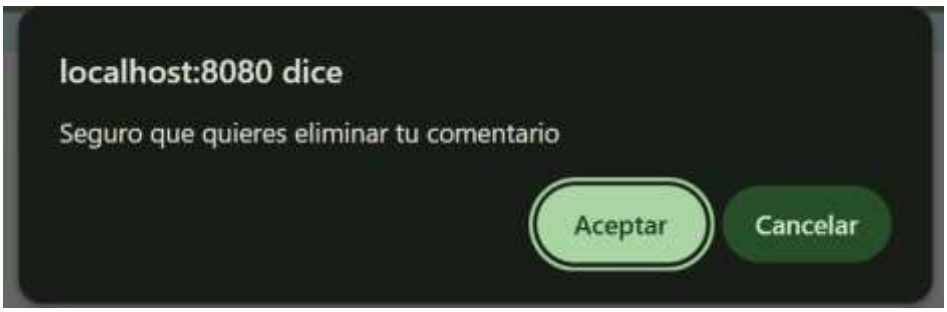
- En pantallas pequeñas, el menú de navegación se convierte en un botón desplegable.
- Las cards se reorganizan automáticamente en filas de una sola columna.
- Aquí se muestra un ejemplo con los cards de los países. Primero el tamaño más pequeño 1 columna, en el tamaño sm se muestran 2 columnas, en el tamaño mediano se muestran 3 columnas y en el tamaño lg o más grande se muestran 4.



Comentarios

Una vez que ha iniciado sesión el usuario y comenta se muestra el mensaje de “comentario guardado correctamente”. Debajo se muestra la fecha y el texto que hay en el comentario para saber que hay puesto en el comentario mientras que lo edita el usuario. Ya que un usuario solo puede comentar una vez la misma ruta.

Además, hay un botón de eliminar comentario por si el usuario cambia de opinión y no quiere comentar la ruta. Al pulsar en él, te muestra un mensaje de si estás seguro de que quieres borrar el comentario. Lo mismo pasa en el botón de eliminar ruta de la lista de favoritos.



En el listado de comentarios de todos los usuarios, para hacerlo más responsivo los he paginado, muestra dos comentarios como máximo por página.

Filtros, etiquetas y usabilidad

- En el listado de rutas, cada ruta muestra su país y región con una etiqueta de color.
- Cuando entras en un país, las rutas ya no muestran el país sino la región, porque ya estás filtrando por país.
- El usuario puede filtrar por país antes de acceder a las rutas, y hay un buscador para facilitar la búsqueda. Una vez dentro de un país puede filtrar por tipo o por región, que

permite buscar regiones fácilmente con autocompletado gracias a la librería Select2. Ambos filtros pueden usarse a la vez.

- En general he intentado que la navegación sea lo más intuitiva posible, con botones claros, mensajes visibles cundo una acción se completa y un diseño limpio.

```
<div th:if="${mensaje != null}" class="alert alert-success">
  <p th:text="${mensaje}"></p>
</div>
```

- En las páginas que lo necesitaban he añadido un botón abajo a la derecha en forma de flecha, para que al pulsar en él suba hasta el inicio de la página para facilitar al usuario subir.

Personalización con CSS

Aunque Bootstrap me ha dado una base sólida para el diseño y para hacerlo responsivo, he añadido estilos CSS personalizados para darle un toque propio y mejorar detalles específicos que no venían por defecto.

Algunos ejemplos de personalización:

- **Colores y tipografía**: Ajusté los colores de toda la página para que combinen mejor con la imagen general de EuroExplorer, usando tonos de verdes, naranjas y azules. También cambié el tipo y tamaño de la letra en algunas secciones para mejorar la legibilidad.
- **Espacios y márgenes**: Modifiqué márgenes y paddings en los cards y formularios para que los elementos no estuvieran demasiado juntos y el diseño se viese más limpio.
- **Efectos de Hover**: Añadí efectos al pasar el ratón sobre los cards, botones y estrellas para que al usuario se le haga más interactivo, agradable e intuitivo.

Estos cambios CSS están guardados en un archivo aparte dentro de la carpeta de recursos (/static/css/styles.css) y se usa en todas las páginas para mantener una apariencia uniforme.

De esta forma, consigo un diseño moderno, agradable y funcional, sin perder la facilidad de uso.

5.3 Funcionalidades clave implementadas

A continuación, explico las funcionalidades principales que se han implementado en la aplicación EuroExplorer:

Sistema de login y registro:

He desarrollado un sistema completo de autenticación. Los formularios de login y registro están en páginas propias (login.html y registro.html) en lugar de modales, ya que al gestionar errores como “usuario no válido” o “campos vacíos”, mantener un modal abierto y mostrar los errores correctamente daba problemas.

El sistema utiliza cifrado de contraseñas para garantizar la seguridad, y permite que solo los usuarios registrados puedan comentar, puntuar o guardarlas como favoritas.

```
@Override  1 usage  elenadiazp *
public Usuario registrar(Usuario usuario) {
    if(usuarioRespository.findByCorreoElectronico(usuario.getCorreoElectronico()).isPresent()){
        throw new RuntimeException("El correo electronico ya existe");
    }
    System.out.println("antes cifrar" + usuario.getContrasena());

String encode = passwordEncoder.encode(usuario.getContrasena());

    System.out.println("cifrado" + encode);
usuario.setContrasena(encode);
    return usuarioRespository.save(usuario);
}
```

Aquí podemos ver el UsuarioServiceImplements, el método que utilizo para registrar. Primero buscamos por el correo electrónico si existe, y una vez comprobado eso cogemos la contraseña y la ciframos para luego añadirla al usuario de nuevo. Finalmente guardamos el usuario con la contraseña cifrada.

Como he comentado antes tengo la configuración de la seguridad y el encriptado en un archivo, llamado SecurityConfig. En este apartado del archivo podemos ver el @Bean de BCryptPasswordEncoder.

```
@Bean  elenadiazp
public PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
```

Aunque Spring Boot configura Spring Security por defecto, tuve que indicarle como encontrar a los usuarios en mi base de datos y como comprobar sus contraseñas cifradas. Para ello, creé una clase personalizada donde defino como buscar a un usuario por su correo y un método de autenticación (UserDetailServiceImplements | AuthenticationManager |. Gracias a eso puedo proteger ciertas páginas con sec:authorize y asegurar funcionalidades solo para usuarios autenticados.

Aquí explico el método AuthenticationManager delcarado como @Bean y creado en la carpeta config:

```
@Bean  elenadiazp
public AuthenticationManager authenticationManager(
        HttpSecurity http,
        PasswordEncoder passwordEncoder
) throws Exception {
    AuthenticationManagerBuilder builder = http.getSharedObject(AuthenticationManagerBuilder.class);
    builder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
    return builder.build();
}
```

Este método recibe los parámetros para poder comparar contraseñas cifradas. Después utilizo esa herramienta para poder indicarle a Spring que métodos utilizar. Finalmente, construyo el AuthenticationManager, que se encargara de la autenticación de verificar usuario y contraseña.

Comentarios y valoraciones

Cada ruta tiene una sección donde los usuarios autenticados pueden dejar comentarios y puntuación. Las valoraciones se muestran en forma de estrellas y se calcula la media de forma automática, que se refleja en la vista de detalle.

Además, el usuario solo puede comentar y puntuar la misma ruta solo una vez, aunque luego puede editarlos.

```java
@Override 1 usage  elenadiazp
public void guardarOPuntuar(Puntuacion nuevaPuntuacion) {
    Long idUsuario = nuevaPuntuacion.getIdUsuario().getId();
    Long idRuta= nuevaPuntuacion.getIdRuta().getId();
    Optional<Puntuacion> existente= puntuacionRepository.findByIdUsuario_IdAndIdRuta_Id(idUsuario, idRuta);
    if(existente.isPresent()){
        Puntuacion puntuacionExistente= existente.get();
        puntuacionExistente.setPuntuacion(nuevaPuntuacion.getPuntuacion());
        puntuacionRepository.save(puntuacionExistente);
    }else{
        puntuacionRepository.save(nuevaPuntuacion);
    }

}
```

Aquí muestro una sección del fichero PuntuacionServiceImplements, donde si existe un usuario con esa ruta que haya puntuado, se edita la puntuación. Sin embargo, si no existe todavía guardamos esa nueva puntuación.

```java
@Override 6 usages  elenadiazp
public Double calcularPuntuacionMedia(Long idRuta) {
    List<Puntuacion> puntuaciones = puntuacionRepository.findByIdRuta_Id(idRuta);
    if(puntuaciones.isEmpty()){
        return 0.0;
    }
    return puntuaciones.stream().mapToDouble(Puntuacion::getPuntuacion).average().orElse( other: 0.0);
}
```

Y aquí podemos ver como calculo la media, si aún está vacío pone un 0.0 y sino cojo todas las puntuaciones de esa ruta y hago averague (la media).

Favoritos

Cada tarjeta de ruta tiene un icono de corazón que actúa como botón para marcar la ruta como favorita. Se recarga toda la página al darle me gusta o quitarlo.

Las rutas marcadas como favoritas se pueden consultar desde la sección de usuario (“Mis favoritos”), donde se muestran todas en formato card, con un botón para eliminarlas de la lista. Además, dentro del detalle de la ruta también se puede añadir o quitar de favoritos directamente.

```
<div class="col-6 text-end botton-0">
    <form th:action="@{/favoritos/eliminar}" method="post">
        <input type="hidden" name="idRuta" th:value="${ruta.id}" />
        <input type="hidden" name="redirectUrl" th:value="${currentUrl}">
        <button type="submit" style="...">
    <span th:if="${idsFavoritos != null and idsFavoritos.contains(ruta.id)}">
      <i class="fas fa-heart text-danger fa-2x"></i>
    </span>
          <span th:unless="${idsFavoritos != null and idsFavoritos.contains(ruta.id)}"
                class="color-black">
       <i class="far fa-heart fa-2x "></i>
    </span>
        </button>
    </form>

</div>
```

Aquí se ve como controlo en los cards de las rutas para añadirlas o quitarlas, y el controlador devuelve la misma página donde se encuentre, pero con los datos actualizados.

Generación de PDF

Desde la vista de detalle de cada ruta, lo usuarios pueden generar un archivo PDF con toda la información de esa ruta (descripción, localización, etc.). Esta funcionalidad se ha implementado usando la librería Flying Saucer PDF y permite descargar la información para tenerla a mano sin conexión a internet.

```
@GetMapping("/rutas/{id}/pdf")  elenadiazp
public void exportarPdf(@PathVariable Long id, HttpServletResponse response) throws Exception {
    RutaDto ruta=rutaService.findById(id);
    Long idPais= ruta.getIdregion().getIdPais().getId();
    PaisDto pais= paisService.findById(idPais);
    List<Comentario> comentarios = comentarioService.findAllByRuta(ruta);

    String logoRuta = new File( pathname: "src/main/resources/static/img/logo.jpg").getAbsolutePath();
    String logoUrl = "file:///"+logoRuta.replace( target: "\\", replacement: "/");

    Context context = new Context();
    context.setVariable( name: "pais", pais);
    context.setVariable( name: "comentarios", comentarios);
    context.setVariable( name: "ruta", ruta);
    context.setVariable( name: "logoUrl", logoUrl);
```

En esta primera parte del PdfController, recupero los datos necesarios de la ruta, después también recupera la imagen del Logo de EuroExplorer. Finalmente, preparto las variables para generar el PDF.

```
String html = templateEngine.process( template: "ruta-pdf", context);
ITextRenderer renderer = new ITextRenderer();
renderer.setDocumentFromString(html);
renderer.layout();

response.setContentType("application/pdf");
response.setHeader( s: "Content-Disposition", s1: "attachment; filename=ruta-" + id +".pdf");
try (OutputStream outputStream = response.getOutputStream()){
    renderer.createPDF(outputStream);
}
```

Y en esta segunda parte del mismo código cojo todos los datos que he preparado y uso Flying Saurcer para convertir el HTML en PDF. Al final del código indico al navegador que el contenido es un PDF y le pongo un nombre al PDF y me encargo de cerrar el proceso.

En la siguiente imagen muestro el aspecto que tiene el PDF:

Sección de rutas aleatorias

En la sección “Aleatorio”, se muestran tres rutas aleatorias por cada tipo. Es una forma divertida de descubrir nuevos destinos si el usuario no tiene claro que buscar. Al pulsar el botón “Recargar rutas aleatorias”, se generan nuevas rutas.

```java
@Override  3 usages  ▲ elenadiazp *
public List<RutaDto> rutasAleatoriasPorTipo(String tipo) {
    List<Ruta> rutas= rutaRepository.findByIdtiporutaNombreIgnoreCase(tipo);
    Collections.shuffle(rutas);//para obtener orden aleatorio
    return rutas.stream().limit( maxSize: 3).map(ruta -> new RutaDto(ruta.getId(),
            ruta.getIdtiporuta(), ruta.getIdregion(), ruta.getFoto1(),ruta.getNombre(),
            ruta.getIdregion().getIdPais().getNombre())).toList();
}
```

Como se muestra en la imagen, simplemente uso shuffle para mezclar las rutas y obtenerlas de forma aleatoria y luego al devolverlas pongo un límite para que solo me devuelva 3 rutas.

Rutas mejores valoradas

En la sección de “Top valoradas”, se muestran las rutas con mayor puntuación global. Esta funcionalidad destaca las rutas más populares según las valoraciones del usuario, ayudando a otro a inspirarse.

```java
@Override  1 usage  ▲ elenadiazp *
public List<RutaDto> obtenerTop3MejorValoradas() {
    Pageable top3 = PageRequest.of( pageNumber: 0, pageSize: 3);
    Page<Ruta> rutasPage = rutaRepository.findTopRutasByPuntuacionMedia(top3);

    return rutasPage.stream().map(ruta -> new RutaDto(ruta.getId(),ruta.getIdtiporuta(),
            ruta.getIdregion(),ruta.getFoto1(), ruta.getNombre(),
            ruta.getIdregion().getIdPais().getNombre())).toList();
}
```

En la imagen se muestra el código de RutaServiceImplements, en concreto la sección de obtener las 3 mejores rutas puntuadas. Uso pageable para limitar que solo me de 3 rutas. Y hago una llamada al repository para usar este método.

```java
@Query("SELECT r FROM Ruta r JOIN r.puntuacions p " +  1 usage  ▲ elenadiazp
        "GROUP BY r.id, r.direccion, r.foto1, r.foto2, r.idregion, r.idtiporuta, r.intinerario, r.nombre, r.otrosDatos " +
        "ORDER BY AVG(p.puntuacion) DESC")
Page<Ruta> findTopRutasByPuntuacionMedia(Pageable pageable);
```

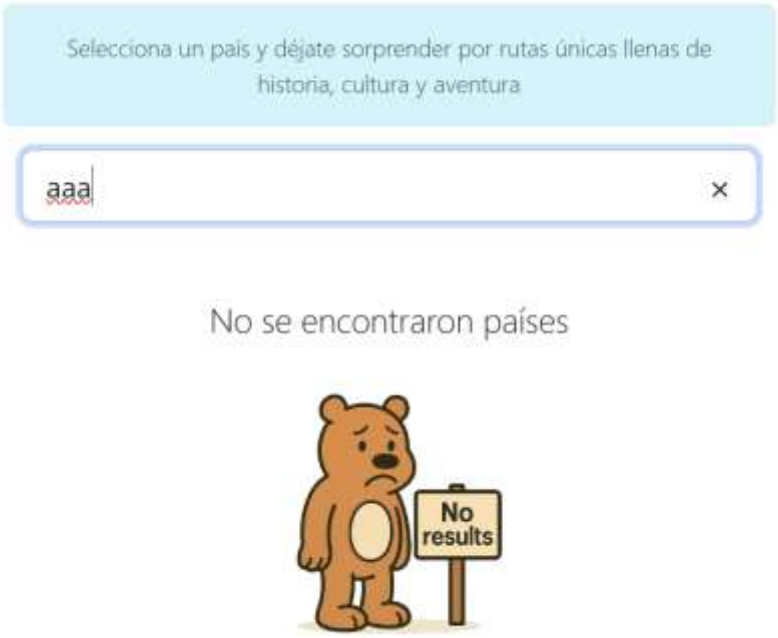
Aquí hago una @Query para poder obtener las rutas con la puntuación media más alta.

Buscador con JavaScript

La que incluye un buscador que permite buscar países por nombre. Está implementado con Java Script para ofrecer resultado en tiempo real.

Es sensible a tildes, por lo que si escribes Hungría sin tilde no se encontrara, algo que sería una posible mejora a futuro.

Si no se encuentran coincidencias, se muestra una imagen de aviso junto con un mensaje:



Como se ve en la siguiente imagen, es una parte del código de países.html, del buscador. La función carga y muestra países desde el backend, dependiendo de la búsqueda y además muestra una animación mientras se realiza la búsqueda.

Se hace un fecht enviando el texto de la búsqueda y cuando llegan los resultados vacía el contenedor.

```
<script>
    const searchInput = document.getElementById('buscador');
    const countryCards = document.getElementById('contenedor-paises');
    const clearBtn = document.getElementById('clear-btn');
    //cargar paises
    function loadCountries(query = '') {  Show usages   elenadiazp
        const spinner = document.getElementById('spinner');
        spinner.classList.remove('d-none');


        fetch(`/paises/buscar?query=${encodeURIComponent(query)}`)
            .then(response => response.json())
            .then(data => {
                countryCards.innerHTML = ''; //limpiar el contenido actual
```

Si no hay resultados muestra un mensaje de no encontrado. Sin embargo, si hay resultados crea dinámicamente los cards de los países recibidos.

```
if (data.length === 0) {
    countryCards.innerHTML =
        <div class="col-12 text-center py-5 mb-5 mt-4">
            <p class="fs-4">No se encontraron paises</p>
            <img src="/img/pasonoresults.png" class="img-fluid" width="200px">
        </div>';
    return;
}
data.forEach(pais => {
    const card = document.createElement('div');
    card.classList.add('col-12', 'col-sm-6', 'col-md-4', 'col-lg-3', 'mb-5');
    card.innerHTML = `
        <a href="/rutas/pais/${pais.id}">
        <div class="card border-0 shadow-sm  rounded-4 overflow-hidden h-100  ">
            <div class="position-relative">
                <img class=" img-pais " src="${pais.urlImagen}" alt="Imagen de ${pais.nombre}" />
                <div class="position-absolute bottom-0 w-100 p-3 text-start ">

                    <h5 class=" text-white fw-semibold mb-0" >${pais.nombre}</h5>

                </div>
            </div>
```

Funcionalidad con JS

He implementado un script en algunas páginas, para asegurar que, al navegar hacia atrás en el navegador, la información que se muestra esté actualizada, por ejemplo: en la página detalle ruta si el usuario puntúa y da me gusta a la ruta, al volver hacia atrás recarga la página para que se muestre la información actualizada.

```
<script>
    window.addEventListener('pageshow', function (event) {
        if (event.persisted) {
            window.location.reload();
        }
    });
</script>
```

Despliegue y hosting

Para desplegar el proyecto y que se pueda acceder desde cualquier lugar, como ya he dicho, he utilizado Render, una plataforma de hosting gratuita. Para ello, he tenido que adaptar el proyecto y crear algunos archivos necesarios:

Dockerfile: para poder desplegar el proyecto, he usado un Dockerfile con multi-stage build. Es una técnica que permite compilar primero en una imagen más pesada y completa y luego copiar solo lo necesario en una segunda imagen más ligera. Como se muestra en la imagen, en la primera etapa se usa una imagen de Maven con JDK 21 para compilar y generar el archivo .jar. En la segunda etapa, únicamente se ha copiado el .jar y el wallet para la conexión con Oracle.

```
# Etapa 1: utiliza imagen maven con JDK para compilar el proyecto
FROM maven:3.9.4-eclipse-temurin-21 as build
#indicar el directorio de tarbajo  con los archivos necesarios
WORKDIR /app
#copiar los archivos necesarios
COPY pom.xml .
COPY src ./src
#complia y genera .jar, y se omite los test para que sea más rápido
RUN mvn clean package -DskipTests

# Etapa 2: imagen ligera solo para ejecutar
FROM eclipse-temurin:21-jdk-alpine
#define el directorio de trabajo
WORKDIR /app
#copia el .jar y el wallet
COPY --from=build /app/target/proyecto-0.0.1-SNAPSHOT.jar app.jar
COPY wallet ./wallet
#puerto donde se va a ejecutar la aplicación
EXPOSE 8080
# inicia la aplicacion
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Oracle Cloud + Wallet: La base de datos está alojada en Oracle Cloud. Para que la aplicación pueda conectarse desde Render, he incluido el archivo wallet dentro del proyecto y como ya comentado antes, la configuración en el aplication.propeties.

Dificultades y soluciones:

- Uno de los principales problemas fue que al subir a Render, la conexión a la base de datos no funcionaba. Esto lo solucione asegurándome de que el wallet estuviera correctamente incluido en la ruta del proyecto.
- También tuve que ajustar el aplication.propeties para que la app funcionara tanto localmente como en el entorno de producción.
- Al principio también tuve que investigar cómo Render interpretaba el Dockerfile y hacer pruebas hasta que se desplegó correctamente.
- Gracias a esto, ahora la aplicación está funcionando online y se puede acceder a ella sin necesidad de instalar nada localmente. El único problema que hay es, que al ser el plan gratuito de Render, si no se está usando la web desplegada durante unos minutos se para, y para cuando quieres acceder otra vez a ella tiene que volver a desplegarse automáticamente lo que tarda unos minutos. Además, con el plan gratuito va un poco lenta, pero si pagase, iría perfectamente.

5.4 Pruebas y ajustes

Durante el desarrollo del proyecto, he realizado distintas pruebas para asegurarme de que todo funcionara correctamente y también para mejorar la experiencia del usuario.

- **Pruebas visuales**: como ya he comentado antes, he revisados cómo se ve la web en diferentes dispositivos (ordenador, móvil, tablet) para asegurarme de que el diseño sea responsivo. También comprobé que los colores, tamaños de letras, imágenes y botones fueran adecuados y legibles.
- **Pruebas funcionales**: Antes de subir el proyecto, he probado todas las funcionalidades registro e inicio sesión, comentar, buscador…).
- **Pruebas de seguridad**: comprobé que no pudieran hacer acciones como comentar o puntuar si no has iniciado sesión.

Ajustes tras detectar errores:

- Al principio tenía el login como un modal, pero daba problemas cuando había errores de validación (no se quedaba abierto en la misma página), así que lo cambié por una vista con formulario normal. También ajuste en rutas cuando filtras por región y tipo, por si la combinación que hagas no existe, salga una imagen y un mensaje de no encontrado. Y mejoré el diseño de las rutas para que quedase más limpio y claro.

Capítulo 6 - Conclusiones y líneas futuras

Desarrollar EuroExplorer ha sido un reto completo en todos los sentidos. A lo largo del proyecto he podido aplicar casi todos los conocimientos adquiridos durante el curso: desde la parte de backend con Spring Boot y la base de datos, hasta la parte visual y de diseño con HTML, CSS, Bootstrap y Thymeleaf.

He aprendido a planificar un proyecto web desde cero, organizar las funcionalidades por fases y a enfrentarme a errores que al principio parecían imposibles de resolver. Además, me ha ayudado a mejorar mucho mi lógica de programación, a manejar mejor los tiempos, y a escribir un código más limpio y estructurado.

También me he dado cuenta de que hay muchas decisiones que no son solo técnicas, sino también de usabilidad: cómo mostrar los datos, qué etiquetas poner, qué filtros tener, o cómo organizar el contenido para que sea cómodo de usar.

Aunque he invertido muchas más horas de las que se estimaban, ha merecido la pena. Estoy orgullosa del resultado final porque no es solo un proyecto académico, sino una aplicación que realmente podría ser útil para quien quiera viajar por Europa.

6.1 Líneas futuras

Aunque la aplicación funciona correctamente y cumple con los objetivos planteados, hay algunas mejoras que me gustaría añadir en un futuro:

- **Sistema de administrador**: para poder gestionar las rutas, usuarios y comentarios directamente desde un panel.
- **Mapa interactivo**: mostrar rutas directamente en un mapa como Google Maps o Leaflet.
- Más países y rutas: ahora hay un conjunto limitado, pero se podría ampliar fácilmente.
- **Sección de usuario:** hacer de que la sección del usuario pueda cambiar su contraseña y sus datos.
- **Mejorar la seguridad**: que tenga mayor seguridad en cuanto a las contraseñas.
- **Tema**: que haya una opción del tema modo claro-oscuro-sistema

Este proyecto me ha servido como una experiencia real de desarrollo, muy parecida a lo que podría encontrarme en un entorno profesional. Ha reforzado mi interés por el desarrollo web y por seguir aprendiendo tecnologías nuevas para seguir mejorando.

Capítulo 7 – Bibliografía

7.1 Apuntes de clase:

- Profesor de DWES.(2025). Apuntes de clase de Desarrollo Web en Entorno Servidor.
  Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web, IES Azarquiel.
- Profesor de LMSG.(2023/2024). Apuntes de clase de Lenguaje de Marcas y Sistemas de Gestión de Información.
  Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web, IES Azarquiel.
- Profesor de BBDD.(2023/2024). Apuntes de clase de Bases de Datos.
  Ciclo Formativo de Grado Superior en Desarrollo de Aplicaciones Web, IES Azarquiel.

7.2 Páginas oficiales y documentación técnica:

- Select2. (s.f.):Librería. https://select2.org/
- Thymeleaf.(s.f): https://www.thymeleaf.org/
- Flying Saucer Project. (s.f.): Librería para generar PDF de HTML. https://code.google.com/archive/p/flying-saucer/
- Bootstrap.(s.f): https://getbootstrap.com/
- Render.(s.f): Herramienta para desplegar. https://render.com/

7.3 Imágenes:

- Pexels. (s.f.). Banco de imágenes gratuitas. https://www.pexels.com/es-es/
- Pixabay. (s.f) Banco de imágenes gratuitas. https://pixabay.com/es/
- Wikimedia. (s.f) Banco de imágenes gratutitas. https://commons.wikimedia.org/wiki/Portada
- Google Imágenes. (s.f.). Imágenes buscadas para las rutas. https://www.google.com/

7.4 Generación de contenido:

OpenAI.(2025). ChapGPT: generación de información sobre rutas. https://chat.openai.com