

BoardGameFinder

CICLO FORMATIVO DE DESARROLLO DE
APLICACIONES MULTIPLATAFORMA

Autor: Enrique Candil Botello

Módulo: Proyecto Final de Ciclo
Ciclo: DAM

CONVOCATORIA:

1 - Introducción

- ¿En qué consiste el proyecto?

BoardGameFinder es una aplicación web y móvil diseñada para facilitar la búsqueda, consulta y gestión de juegos de mesa.

El objetivo principal de la aplicación es ofrecer una experiencia intuitiva y accesible a cualquier tipo de usuario, permitiendo guardar tus juegos favoritos y realizar búsquedas avanzadas para encontrar tu juego ideal según distintas necesidades o preferencias.

El proyecto ha sido desarrollado como una aplicación multiplataforma, integrando tanto un backend basado en Java con Spring Boot como un frontend construido con tecnologías web de HTML, JavaScript y CSS, estructurado para ser consumido desde aplicaciones móviles. Esta arquitectura permite una escalabilidad sencilla, además de garantizar una experiencia fluida y coherente en múltiples plataformas.

- Módulos implicados

La aplicación se puede dividir en varios módulos funcionales, cada uno de ellos con un número de responsabilidades para el funcionamiento de la aplicación:

1. Autenticación y gestión de usuarios:

Incluye el registro de nuevos usuarios, inicio de sesión seguro mediante JWT (Json Web Token), y funcionalidades de gestión como cambio de contraseña y eliminación de cuenta.

2. Sistema de seguridad y control de roles:

Implementación de roles escalables que determinan los permisos y accesos disponibles en la plataforma.

3. Gestión de juegos favoritos:

Permite a los usuarios añadir y eliminar juegos de su lista personal de favoritos, facilitando el acceso rápido a los títulos que más les interesan.

4. Buscador avanzado de juegos:

Herramienta de búsqueda con múltiples filtros configurables (número de jugadores, edad recomendada, tiempo estimado, editorial, categorías, etc.) que permite encontrar juegos según distintos criterios.

5. Panel de administración:

Módulo exclusivo para administradores que permite la creación, edición y eliminación de juegos de mesa desde una interfaz visual.

6. Consumo de API:

Toda la lógica de negocio está expuesta a través de una API REST, consumida desde el frontend con JavaScript. Esta arquitectura también permite el acceso desde aplicaciones móviles que se comuniquen con la API.

2 - Justificación de proyecto y objetivos

- Por qué has elegido este proyecto

La elección de **BoardGameFinder** como proyecto surge de un interés personal en el mundo de los juegos de mesa. Al estar familiarizado con los juegos de mesa tengo más ideas para el desarrollo de una aplicación práctica y funcional. Al trabajar en un tema que resulta motivador, la organización y desarrollo del proyecto se vuelve más fluida.

Además, este proyecto representa una excelente oportunidad para afianzar conocimientos sobre el diseño y la arquitectura de aplicaciones multiplataforma, aplicando tecnologías tanto nuevas como previamente aprendidas en el ciclo formativo. Se buscó también salir de la zona de confort, utilizando herramientas con las que no se había trabajado en profundidad anteriormente.

- Estado del arte

Actualmente existen plataformas de referencia en el ámbito de los juegos de mesa, siendo *BoardGameGeek* una de las más reconocidas y utilizadas a nivel mundial. Esta plataforma es extremadamente completa: ofrece bases de datos extensas, foros de opinión, reseñas y listas de popularidad.

Sin embargo, su complejidad puede suponer una barrera para usuarios que simplemente desean buscar juegos con criterios específicos y acceder a información básica de forma rápida. **BoardGameFinder** se plantea como una alternativa más sencilla y directa, que extrae una de las funcionalidades más útiles de *BoardGameGeek* —la búsqueda de juegos— y la presenta de forma simplificada, con un sistema de filtros intuitivo y una experiencia de usuario más accesible.

- Objetivos del proyecto

Este proyecto consta de varios objetivos tanto técnicos como profesionales:

1. Mejorar mis capacidades de desarrollo backend utilizando Spring Boot.
2. Diseñar y construir una API REST segura y escalable.
3. Aprender e integrar Spring Security y JWT para la autenticación y autorización.
4. Establecer una mejora en mis conocimientos en el diseño de estructuras HTML, CSS y JavaScript, tecnologías que no son el foco principal del ciclo de DAM pero que resultan fundamentales en el desarrollo web.
5. Comprender la estructura de una aplicación full-stack orientada a consumo desde frontend web y móvil.
6. Aplicar buenas prácticas de organización del código, diseño modular y gestión de roles.

- Público objetivo

La aplicación está dirigida a un público general, sin distinción de edad o experiencia técnica. El objetivo es que cualquier persona, desde jugadores ocasionales hasta familias o personas que buscan un regalo, puedan consultar información sobre juegos de mesa de manera rápida, intuitiva y sin complicaciones.

Orientada hacia la simplicidad y la eficiencia busca cubrir una necesidad concreta: obtener información útil sin tener que navegar por menús complejos o interfaces recargadas.

3 - Planificación

- Informe de horas totales que se van a necesitar

Tarea	Descripción	Horas estimadas
Análisis	Definición de requisitos, diseño funcional	2h
Diseño de la base de datos	Creación del modelo E/R y estructura de tablas	2h
Desarrollo del backend	Creación de la API REST, servicios, controladores, etc	10h
Seguridad y autenticación	Implementación de roles, JWT y Spring Security	5h
Desarrollo del frontend	HTML, JS y CSS, integración con la API	10h
Panel de administrador	Interfaz de operaciones CRUD, solo para administradores	2h
Pruebas y correcciones	Pruebas funcionales, corrección de bugs	2h
Documentación	Elaboración de la memoria y manual de usuario	5h
Implementación de filtros y búsquedas	Desarrollo de lógica de filtrado y consultas personalizables	2h
Total		40h

4 - Análisis

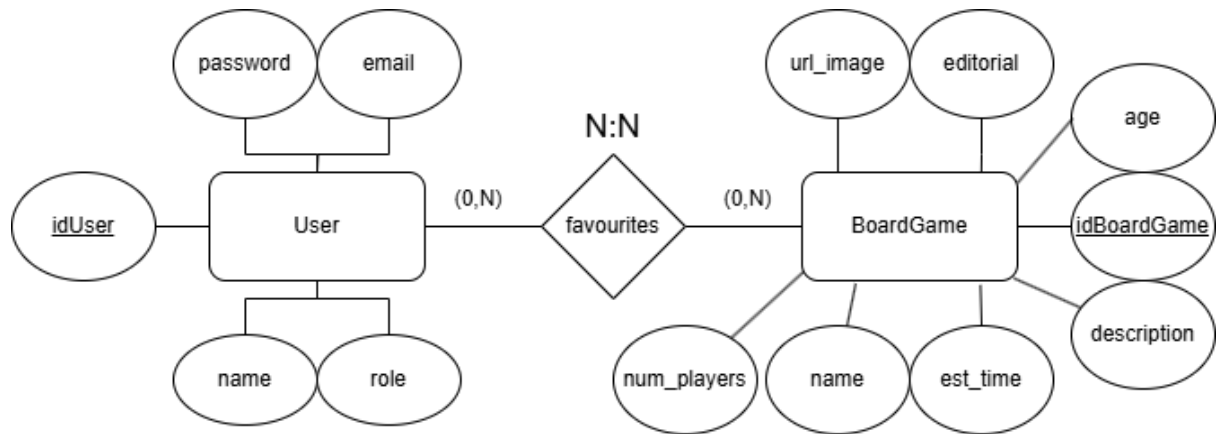
La aplicación **BoardGameFinder** permitirá a los usuarios gestionar su cuenta, buscar juegos de mesa según múltiples filtros, marcarlos como favoritos y, en el caso de los administradores, gestionar el catálogo de juegos de forma completa (crear, editar, eliminar).

Funcionalidades principales:

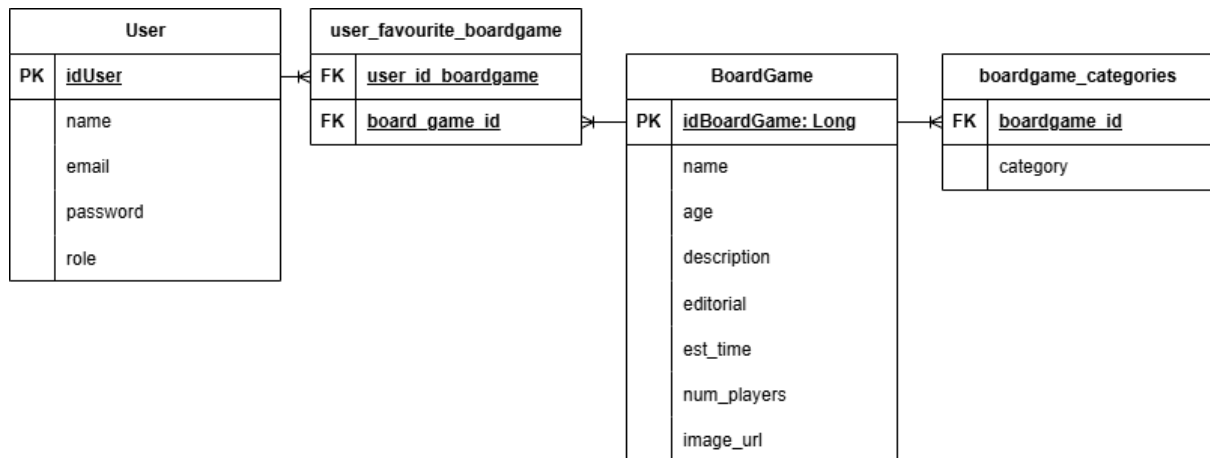
1. Registro, inicio de sesión y gestión de usuarios
 - Creación de cuenta
 - Autenticación mediante JWT
 - Cambio de contraseña
 - Eliminación de cuenta
2. Búsqueda de juegos de mesa
 - Filtros por: nombre, número de jugadores, duración, edad recomendada, editorial, categorías
 - Resultados dinámicos filtrados por API REST
3. Gestión de favoritos
 - Agregar o quitar juegos de la lista de favoritos del usuario autenticado
 - Consulta de juegos favoritos guardados
4. Panel de administración (solo admins)
 - Crear nuevos juegos de mesa
 - Editar juegos existentes
 - Eliminar juegos del catálogo
5. Consumo multiplataforma
 - Aplicación web con JS y Bootstrap
 - API preparada para consumo desde una app móvil (APK)

- Diagramas

Diagrama E/R:



Modelo E/R:



5 - Diseño

- Estructura de la base de datos

La base de datos está diseñada en torno a dos entidades principales:

- **User**: representa a los usuarios registrados. Cada usuario tiene un identificador único, un email, nombre, contraseña y un rol que determina su nivel de acceso.
- **BoardGame**: representa los juegos de mesa. Incluye información como el nombre, descripción, imagen, edad recomendada, editorial, tiempo estimado de juego y número de jugadores.

En el caso de **BoardGame**, todos los campos, menos name y description, son enumerados creados en la aplicación y almacenados como cadenas de texto en la base de datos.

Existe otra tabla llamada *boardgame_categories* que relaciona la tabla **BoardGame** con el enumerado Category, ya que queremos una lista de esta.

Existe una relación **muchos a muchos (N:M)** entre usuarios y juegos de mesa, ya que cada usuario puede guardar múltiples juegos como favoritos, y cada juego puede ser favorito de múltiples usuarios. Esta relación se gestiona mediante una tabla intermedia llamada *user_favourite_boardgame*.

- Arquitectura del proyecto

La aplicación sigue una arquitectura cliente-servidor basada en una API REST, dividiéndose en los siguientes módulos principales:

Backend(Java + Spring Boot)

- Spring Web: Expone endpoints REST que permiten interactuar con los recursos **User** y **BoardGame**.

- Spring Data JPA: Gestiona las entidades y el acceso a la base de datos
- Spring Security + JWT: Autentifica a los usuarios y asegura los endpoints a través del rol de un usuario.
- Lombok: Disminuye la cantidad de boilerplate en el desarrollo de la aplicación.
- Mysql-connector: Establece los campos para la conexión con nuestra base de datos MySQL en el archivo *application.properties*.

Frontend(HTML,CSS,JavaScript)

- Implementado con HTML estático, CSS personalizado y JavaScript vanilla para consumir la API mediante *fetch*.
- Uso de las librerías de BootStrap para mejorar el aspecto visual de la página web.

API y comunicación

- La API expone endpoint REST bajo rutas como *api/users*, *api/boardgames* y *api/auth*.
- La autenticación se realiza mediante **JWT**, que se incluye en las cabeceras de las peticiones
- Está preparada para ser consumida tanto por el cliente web como por una futura aplicación móvil (APK).

6 - Implementación y pruebas

- Desarrollo de la aplicación con los lenguajes y plataformas elegidas.

Backend

A continuación se explica la lógica tras las partes del código más importantes. Las imágenes del código estarán en el **Anexo II**.

- 1) Entidades principales: a la hora de construir nuestras entidades, usamos varias anotaciones en el código correspondientes a las librerías de JPA y Lombok, estas anotaciones se usan para construir campos como los getter y los setter del usuario, y para especificar que este objeto es una Entidad que será convertida a una tabla en la base de datos. Al igual que los campos también están dotados de anotaciones que especificara sus datos como columnas en la base de datos.
 - a) En **User** tenemos una relación manyToMany con **BoardGame** especificando el tipo de relación a *Eager*.
 - b) En **BoardGame** usamos anotaciones para marcar que los campos son enumerados, y especialmente la anotación *Lazy* en la relación con **User** para no mandar los datos de los usuarios.
- 2) Enumerados: los enumerados de nuestra aplicación poseen anotaciones de parseo a Json para que sus valores sean convertidos a una cadena de caracteres en la conversión a Json.
- 3) Seguridad: para la configuración de la seguridad en la aplicación, usamos Spring Security. *SecurityConfig* se encarga de configurar los premisas para el acceso a las diferentes direcciones URL de nuestra página web. También es donde configuramos el sistema de autenticación, el cual se encarga del manejo de roles en la aplicación, y por último, el filtro de JWT para la creación y extracción de datos de nuestro token web.

- 4) Repositorios: la capa de Repository es la encargada de hacer las llamadas a la base de datos desde nuestro backend, sus funciones están heredadas de la interfaz *JpaRepository*, la cual nos ofrece una serie de funciones sencillas que usamos en otras partes del código, como los servicios.
 - a) Dentro del repositorio de **BoardGame**, heredamos otra interfaz llamada *JpaSpecificationExecutor*, la cual nos permite, mediante la interfaz *Specification*, poder hacer consultas complejas a la base de datos.
- 5) Servicios: la capa de Service es la encargada de la mayoría de la validación de datos, como norma de negocio, en la aplicación. En esta capa recibimos los datos desde los controller y usamos el Repository para llamar a la base de datos.
 - a) Dentro de la capa de servicios, hay un servicio llamado *AuthService*, que es el encargado de usar la lógica de los tokens web, este servicio está creado a parte para evitar dependencias circulares en la aplicación.
- 6) Controladores: la capa de Controller es la que recibe las llamadas externas. En Java, se usa típicamente un objeto llamado *ResponseEntity* para enviar respuestas Https, para poder manejar elementos como el Status Http del servidor.
 - a) Todos los controladores usan anotaciones *Spring web* para establecer la URL de cada función (RequestMapping, GetMapping, etc)
 - b) La función *searchBoardGames()* en *BoardGameController*, hace uso de la lógica implementada en los enums para convertir los datos recibidos (String) a sus enums correspondientes, y así poder usarlos en la capa de servicio donde son asignados a la interfaz *Specification* para hacer la llamada en la base de datos.

FrontEnd

A continuación se explica la función de los archivos JS más relevantes. Las imágenes del código estarán en el **Anexo II**.

- 1) navbar-loader y footer-loader: estos scripts se hallan en todos los archivos HTML de la página web. Se usa para inyectar el componente parcial *navbar.html* y *footer.html* en las vistas de la página.
 - a) El navbar posee lógica para detectar si hay un usuario logueado o un admin. Dependiendo del tipo de usuario, algunas partes del navbar se hacen inaccesibles.
- 2) login y registro: los scripts se encargan de procesar la lógica de inicio de sesión y registro. Se hacen llamadas al endpoint correspondiente de la API para iniciar estos procesos, al igual que se usa una validación de datos en los campos de password y email.
- 3) home: el script de home es el más largo y se puede dividir en tres partes; la carga inicial de datos de todos los juegos de mesa, la carga de filtros y la llamada a la búsqueda por filtros.
- 4) adminDashboard: parecido a home, pero no usa filtros de búsqueda. La diferencia es la lógica para borrar, editar y crear juegos de mesa que solo un usuario administrador puede lograr.
- 5) boardGameView/FormCreate/FormEdit: estos scripts son muy parecidos, y lo que hacen es recoger información de un juego de mesa para mostrar los datos (View y Edit), al igual que mandar el propio objeto BoardGame para su edición o eliminación.

- Pruebas realizadas

Pruebas de autenticación

A continuación se explicaran las pruebas realizadas, las capturas de pantalla de las pruebas estarán en la parte de **Anexo I**.

- Registro de nuevo usuario.
- Inicio de sesión con credenciales válidas e inválidas.

Pruebas de gestión de usuarios

- Visualización de datos personales.
- Cambio de contraseña.
- Eliminación de cuenta.

Pruebas sobre juegos de mesa

- Creación, edición y eliminación de juegos por parte de un usuario con rol ADMIN.
- Añadir y eliminar juegos de la lista de favoritos del usuario.
- Visualización de los juegos favoritos.

Pruebas del buscador

- Búsqueda por nombre.
- Combinación de filtros de categoría, número de jugadores, tiempo estimado, etc.

Pruebas de seguridad

- Bloqueo de acceso a rutas protegidas sin token o con token inválido.

7 - Implantación y documentación

- Manual de instalación

Requisitos previos

1) Java 17 o superior

- a) Para poder descargar una versión moderna de Java visita:
<https://www.java.com/en/download/manual.jsp>

2) Maven

- a) Para descargar manualmente Apache Maven visita:
<https://maven.apache.org/install.html>

3) Lombok

- a) Si alguna librería falla al instalarse por medio de maven, se recomienda que se instale manualmente, para instalar lombok visita: <https://projectlombok.org/download>

4) MySql workbench

- a) Para descargar manualmente Mysql workbench visita:
<https://dev.mysql.com/downloads/installer/>, asegurarse de instalar justo con Workbench el servidor de Mysql para poder iniciar un un servidor local Mysql donde la app se conectará.

Backend

1) Clonar el repositorio del proyecto

- a) Usa este comando para clonarlo: `git clone <URL repository>`

2) Accede al directorio del backend y compila el proyecto

a) Usa este comando en el directorio correcto: mvn clean install

3) Configura los datos del archivo application.properties

```
spring.application.name=BoardGameFinder

spring.datasource.url=jdbc:mysql://localhost:3306/BoardGameFinder
spring.datasource.username=root
spring.datasource.password=#
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
server.port=8088

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

a) spring.datasource.password deberá ser la misma que la de tu servidor MySql local. Y usa el puerto deseado.

4) Ejecuta la aplicación manualmente o con un IDE

a) Para ejecutar la aplicación manualmente se usa este comando: mvn spring-boot:run

Frontend

Simplemente abre el archivo index.html en un buscador moderno, las páginas HTML están organizadas como vistas independientes.

- Manual de usuario

El manual de usuario se podrá ver en el **Anexo I** de pruebas realizadas, donde se exponen todas las vistas de la página web.

8 - Resultados y discusión

- Comentarios sobre el desarrollo y temporalización respecto a la planificada

Durante el desarrollo del proyecto se han seguido las fases planteadas inicialmente en la planificación, aunque ha habido algunos ajustes de tiempo según las necesidades reales de cada etapa.

Las fases del desarrollo del backend junto con la configuración de la seguridad y autenticación han tomado más tiempo del esperado dada a la poca familiaridad que tenía con las librerías de Spring Security y JWT.

Por otro lado, el desarrollo del frontend de la aplicación fue algo más ameno de lo que me imaginaba, gracias a la gran cantidad de guías y referencias online sobre las tecnologías HTML, JS y CSS.

Y por último, la fase de pruebas y corrección de errores también fue más larga de lo esperado, debido en parte a mi poca experiencia haciendo pruebas para aplicaciones de larga escala. La mayoría de errores que experimenté en esta fase no fueron muy grandes pero definitivamente se alargaron más allá de mis expectativas.

Cambios en relación a la tabla del punto 3:

Tarea	Descripción	Horas estimadas
Desarrollo del backend	Creación de la API REST, servicios, controladores, etc	10h → 15h
Seguridad y autenticación	Implementación de roles, JWT y Spring Security	5h → 10h
Desarrollo del frontend	HTML, JS y CSS, integración con la API	10h → 8hr
Pruebas y correcciones	Pruebas funcionales, corrección de bugs	2h → 5h
Total		40h → 55h

- Dificultades más importantes encontradas

La implementación de las librerías Spring Security y JWT han sido con diferencia los puntos más complicados del desarrollo de la aplicación. Varios problemas a la hora de desarrollar la aplicación se pueden resumir en los siguientes:

- **Número de archivos necesarios para Spring Security:** archivos como SecurityConfig, JwtUtil y JWTAuthenticationFilter son algunos de los archivos necesarios para que la validación rol y token funcionen en la página.
- **Dependencias circulares al iniciar la aplicación:** problemas a la hora de asignar los *Beans* en la inicialización de la aplicación.
- **Uso del token JWT para hacer llamadas a la API:** constante necesidad de validación del usuario, incluyendo la lógica para guardar el token y sacar información a partir de este mismo.
- **Diseño de filtros y búsquedas escalables:** uso de la interfaz *Specification*, la cual requiere la creación de más archivos y ajustes de lógica.
- **Consumo de la API mediante JS puro:** sin el uso de frameworks como Angular o React, las peticiones y manejo de funciones asíncronas es más complejo.

- Posibles mejoras y ampliaciones del proyecto

El proyecto es muy escalable y tiene hueco para implementar nuevas ideas y mejoras para la aplicación ya existente, algunas de estas mejoras puede ser:

- **Desarrollo de una aplicación móvil completa:** que consuma la API creada, aprovechando el diseño multiplataforma que tiene el proyecto
- **Sistema de valoraciones:** una funcionalidad sencilla y fácil de implementar, que permita a los usuarios dejar su opinión sobre los juegos de mesa
- **Mejora y ampliación de un sistema de excepciones:** a la aplicación le ayudaría tener un sistema más claro de errores para poder manejar los fallos más profesionalmente.
- **Uso de React o Angular para la mejora UX:** la aplicación actualmente no usa un framework de frontend y depende de JS puro, la implementación de un motor frontend podría mejorar la experiencia de usuario.
- **Uso de modales para crear o editar los juegos de mesa:** en vez de usar vistas adicionales para la creación y edición de los juegos, se puede hacer que el formulario se complete en la página del administrador
- **Implementación de un servidor:** el proyecto usa un servidor local para alojar la aplicación, el uso de un servidor online le daría un nivel de profesionalidad mayor.

9 - Conclusiones

- ¿Qué ha supuesto para tu formación la realización de este proyecto?

La realización de este proyecto me ha ayudado a asentar mis conocimientos sobre la arquitectura de una página web, la organización a la hora de crearla y sobre todo, el entendimiento de la estructura completa de una aplicación multiplataforma moderna, desde la base de datos y el backend, hasta el diseño y consumo del frontend.

- ¿En qué medida te ha servido para ampliar tus conocimientos?

He podido trabajar con Spring Boot y Spring Security, tecnologías que, aunque complejas en un inicio, me han permitido comprender mejor la gestión de usuarios, roles y autenticación en entornos reales. También he implementado por primera vez JSON Web Tokens (JWT), una tecnología clave para sistemas modernos de autenticación.

- ¿Qué destrezas has conseguido con la realización del proyecto que no hubieras conseguido a lo largo del curso?

Gracias a este proyecto he podido aprender y mejorar varias cosas en cuanto a mis conocimientos de arquitectura web. Entre las cosas que nunca había llegado a hacer o mejorar son:

- Diseño de APIs REST seguras y estructuradas.
- Organización del código y división por responsabilidades.
- Uso de JavaScript puro para el consumo de servicios web.
- Diseño básico de interfaces usando HTML, CSS y Bootstrap.

Para terminar, este proyecto ha sido una gran oportunidad a la hora de prepararme mejor para afrontar los retos profesionales en entornos reales de desarrollo web.

10 - Bibliografía y referencias

- **Documentación oficial de Spring Boot**
<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/>
- **Documentación oficial de Spring Security**
<https://docs.spring.io/spring-security/reference/>
- **Documentación oficial de JSON Web Token (JWT)**
<https://jwt.io/introduction>
- **Documentación de JPA y Hibernate**
<https://docs.oracle.com/javaee/7/tutorial/persistence-intro.htm>
- **Documentación de Bootstrap**
<https://getbootstrap.com/>
- **Documentación de JavaScript (MDN Web Docs)**
<https://developer.mozilla.org/es/docs/Web/JavaScript>
- **BoardGameGeek – Para análisis del estado del arte y funcionalidades similares**
<https://boardgamegeek.com/>
- **W3Schools – Referencia rápida para HTML, CSS y JavaScript**
<https://www.w3schools.com/>
- **Tutoriales y foros:**
Stack Overflow (<https://stackoverflow.com/>)
Baeldung (<https://www.baeldung.com/>) – Artículos sobre Spring Security y JWT.

11 - Anexos

A continuación se incluyen contenidos complementarios al cuerpo principal del proyecto.

Anexo I - Pruebas realizadas

Registro de un nuevo usuario:

BoardGameFinder

Sobre nosotrosInformaciónIniciar sesiónRegistrar

Registrarse

Nombre

Correo electrónico

Contraseña

REGISTRARSE

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto | Desarrollado con ❤️ y dados.

LOCALHOST:8088 DICE

Usuario creado exitosamente

Aceptar

Registrarse

Nombre

Correo electrónico

Contraseña

Inicio de sesión con credenciales validas e invalidas:

BoardGameFinder

Sobre nosotrosInformaciónIniciar sesiónRegistrar

Iniciar sesión

Correo electrónico

Contraseña

Credenciales incorrectas

ENTRAR

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto | Desarrollado con ♥ y dados.

BoardGameFinder

Inicio de sesión exitoso

Aceptar

Sobre nosotrosInformaciónIniciar sesiónRegistrar

Iniciar sesión

Correo electrónico

Contraseña

ENTRAR

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto | Desarrollado con ♥ y dados.

Visualización de datos personales:

BoardGameFinder

Sobre nosotrosInformaciónPerfilCerrar sesión

Mi perfil

Información personal

Nombre: PruebaAuth

Email: pruebaAuth@test.com

Juegos favoritos

No hay juegos favoritos aún

CAMBIAR CONTRASEÑA

ELIMINAR CUENTA

Cambio de contraseña:

BoardGameFinder

Sobre nosotrosInformaciónPerfilCerrar sesión

Información personal

Nombre: PruebaAuth

Email: pruebaAuth@test.com

Juegos favoritos

No hay juegos favoritos aún

CAMBIAR CONTRASEÑA

ELIMINAR CUENTA

Cambiar contraseña

Nueva contraseña

GUARDAR

CANCELAR

Eliminación de cuenta:

BoardGameFinder

Eliminar cuenta?

Aceptar

Cancelar

Sobre nosotros

Información

Perfil

Cerrar sesión

Mi perfil

Información personal

Nombre: PruebaAuth

Email: pruebaAuth@test.com

Juegos favoritos

No hay juegos favoritos aún

CAMBIAR CONTRASEÑA

ELIMINAR CUENTA

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto | Desarrollado con ❤️ y dados.

Iniciar sesión

Correo electrónico

pruebaAuth@test.com

Contraseña

.....

Credenciales incorrectas

ENTRAR

Creación, edición y eliminación de juegos por parte de un usuario con rol ADMIN:

LOCALHOST:8088 DICE

Inicio de sesión exitoso

Aceptar

Iniciar sesión

Correo electrónico

adminTest@example.com

Contraseña

ENTRAR

BoardGameFinder

Cerrar sesión



Panel de Administración

[Nuevo Juego](#)

Nombre	Número de Jugadores	Tiempo estimado	Acciones	
Ark nova	1 to 4 players	More than 2 hours	Editar	Eliminar
Flip 7	3 or more players	15 to 30 minutes	Editar	Eliminar
7 Wonders	4 or more players	30 to 45 minutes	Editar	Eliminar
Feed the Kraken	1 player	60 to 90 minutes	Editar	Eliminar
Catan	4 players	60 to 90 minutes	Editar	Eliminar

© 2025 BoardGameFinder. Todos los derechos reservados.

[GitHub del Proyecto](#) | Desarrollado con  y dados.

BoardGameFinder

Cerrar sesión

Crear nuevo juego de mesa

Nombre

Prueba

Descripción

Prueba desc

URL de la imagen

Número de jugadores

1 player

Tiempo estimado

Less than 15 minutes

Editorial

Devir

Edad recomendada

3+

Categorías

Seleccionar categorías

☒ Adventure

☐ Cards

☐ Dice

☐ Educational

☐ Exploration

☐ Fantasy

☐ Memory

☐ Casual

☐ Racing

☐ Trivia

☐ Deduccion

☐ Economy

☐ Strategy

☐ Humor

☐ Puzzle


☐ Zombie

Por favor, complete todos los campos.

Crear

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto

Desarrollado con  y dados.

BoardGameFinder

Juego creado exitosamente

Cerrar sesión

Aceptar

+ Crear nuevo juego de mesa

Nombre

Prueba

Descripción

Prueba desc
cccccc

URL de la imagen

https://as2.ftcdn.net/v2/jpg/02/45/70/11/1000_F_245701100_MAEWqeKIPiYlrYR8DTH67eBZ5Ku4TPAf.jpg

Número de jugadores

1 player

Tiempo estimado

Less than 15 minutes

Editorial

Devir

Edad recomendada

3+

Categorías

Seleccionar categorías

☒ Adventure

☐ Cards

☐ Dice

☐ Educational

☐ Exploration

☐ Fantasy

☐ Memory

☐ Casual

☐ Racing

☐ Trivia

☐ Deduccion

☐ Economy

☐ Strategy

☐ Humor

☐ Puzzle

☐ Zombie

Crear

© 2025 BoardGameFinder. Todos los derechos reservados.

GitHub del Proyecto

Desarrollado con y dados.

Módulo: Proyecto Final de Ciclo
Ciclo: DAM

Nombre	Número de Jugadores	Tiempo estimado	Acciones	
Ark nova	1 to 4 players	More than 2 hours	Editar	Eliminar
Flip 7	3 or more players	15 to 30 minutes	Editar	Eliminar
7 Wonders	4 or more players	30 to 45 minutes	Editar	Eliminar
Feed the Kraken	1 player	60 to 90 minutes	Editar	Eliminar
Catan	4 players	60 to 90 minutes	Editar	Eliminar
Prueba	1 player	Less than 15 minutes	Editar	Eliminar

BoardGameFinder

Juego actualizado exitosamente

Cerrar sesión

Aceptar

Editar juego de mesa

Nombre

Prueba editada

Descripción

Prueba desc

URL de la imagen

https://as2.ftcdn.net/v2/jpg/02/45/70/11/1000_F_245701100_MAEWqeKIPIYlrYR8DTH67eBZ5Ku4TPAf.jpg

Número de jugadores

1 player

Tiempo estimado

Less than 15 minutes

Editorial

Devir

Edad recomendada

3+

Categorías

Seleccionar categorías

Actualizar

© 2025 BoardGameFinder. Todos los derechos reservados.


GitHub del Proyecto | Desarrollado con y dados.

Nombre	Número de Jugadores	Tiempo estimado	Acciones	
Ark nova	1 to 4 players	More than 2 hours	Editar	Eliminar
Flip 7	3 or more players	15 to 30 minutes	Editar	Eliminar
7 Wonders	4 or more players	30 to 45 minutes	Editar	Eliminar
Feed the Kraken	1 player	60 to 90 minutes	Editar	Eliminar
Catan	4 players	60 to 90 minutes	Editar	Eliminar
Prueba editada	1 player	Less than 15 minutes	Editar	Eliminar

Finder

¿Eliminar "Prueba editada"?


Aceptar Cancelar

 **Panel de Administración**

Nuevo Juego

Nombre	Número de Jugadores	Tiempo estimado	Acciones
Ark nova	1 to 4 players	More than 2 hours	Editar Eliminar
Flip 7	3 or more players	15 to 30 minutes	Editar Eliminar
7 Wonders	4 or more players	30 to 45 minutes	Editar Eliminar
Feed the Kraken	1 player	60 to 90 minutes	Editar Eliminar
Catan	4 players	60 to 90 minutes	Editar Eliminar
Prueba editada	1 player	Less than 15 minutes	Editar Eliminar

Añadir y eliminar juegos de la lista de favoritos del usuario:

 **Juegos de Mesa**

Mostrar Filtros

Nombre	Número de Jugadores	Tiempo estimado	
Ark nova	1 to 4 players	More than 2 hours	
Flip 7	3 or more players	15 to 30 minutes	
7 Wonders	4 or more players	30 to 45 minutes	Añadir a Favoritos
Feed the Kraken	1 player	60 to 90 minutes	Añadir a Favoritos
Catan	4 players	60 to 90 minutes	Añadir a Favoritos

BoardGameFinder

Sobre nosotros Información Perfil Cerrar sesión

Mi perfil

Información personal
Nombre: Test
Email: asd@asd.com

Juegos favoritos

[Ark nova](#)[QUITAR](#)

[Flip 7](#)[QUITAR](#)

[CAMBIAR CONTRASEÑA](#) [ELIMINAR CUENTA](#)

Finder

¿Quitar "Flip 7" de favoritos?

Aceptar Cancelar

Sobre nosotros Información Perfil Cer

Mi perfil

Información personal
Nombre: Test
Email: asd@asd.com


Juegos favoritos

Ark nova

QUITAR

Flip 7


QUITAR

 **Juegos de Mesa**

Mostrar Filtros

Nombre	Número de Jugadores	Tiempo estimado	
Ark nova	1 to 4 players	More than 2 hours	
Flip 7	3 or more players	15 to 30 minutes	Añadir a Favoritos
7 Wonders	4 or more players	30 to 45 minutes	Añadir a Favoritos
Feed the Kraken	1 player	60 to 90 minutes	Añadir a Favoritos
Catan	4 players	60 to 90 minutes	Añadir a Favoritos

Buscar por nombre:

 **Juegos de Mesa**

Mostrar Filtros

Filtrar Juegos

Nombre:

Jugadores: -- Selecciona una op ▾

Tiempo: -- Selecciona una op ▾


Editorial: -- Selecciona una op ▾

Edad: -- Selecciona una op ▾

🔍

Categorías:

☐ Adventure ☐ Cards ☐ Dice ☐ Educational ☐ Exploration ☐ Fantasy ☐ Memory ☐ Casual ☐ Racing ☐ Trivia ☐ Deduccion ☐ Economy ☐ Strategy ☐ Humor ☐ Puzzle ☐ Zombie

 **Juegos de Mesa**

Mostrar Filtros

Filtrar Juegos

Nombre:

Jugadores:

-- Selecciona una op

Tiempo:

-- Selecciona una op

Editorial:

-- Selecciona una op

Edad:

-- Selecciona una op

Q

Categorías:

☐ Adventure

☐ Cards

☐ Dice

☐ Educational

☐ Exploration

☐ Fantasy

☐ Memory

☐ Casual

☐ Racing

☐ Trivia

☐ Deduccion

☐ Economy

☐ Strategy


☐ Humor

☐ Puzzle

☐ Zombie

Nombre	Número de Jugadores	Tiempo estimado	
Catan	4 players	60 to 90 minutes	<div>Añadir a Favoritos</div>

Combinación de filtros de categoría, número de jugadores, tiempo estimado, etc:

 **Juegos de Mesa**

Mostrar Filtros

Filtrar Juegos

Nombre:

Jugadores:

1 player

Tiempo:

-- Selecciona una op

Editorial:

-- Selecciona una op

Edad:

-- Selecciona una op

Q

Categorías:

☐ Adventure

☐ Cards

☐ Dice

☐ Educational

☐ Exploration

☐ Fantasy

☐ Memory

☐ Casual

☐ Racing

☐ Trivia

☐ Deduccion

☐ Economy


☐ Strategy

☐ Humor

☐ Puzzle

☐ Zombie

Nombre	Número de Jugadores	Tiempo estimado	
Feed the Kraken	1 player	60 to 90 minutes	<div>Añadir a Favoritos</div>
Prueba Filtros 1	1 player	30 to 45 minutes	<div>Añadir a Favoritos</div>

 **Juegos de Mesa**

Mostrar Filtros

Filtrar Juegos

Nombre: Jugadores: Tiempo: Editorial: Edad:

Categorías:

☒ Adventure ☐ Cards ☐ Dice ☐ Educational ☐ Exploration ☐ Fantasy ☐ Memory ☐ Casual ☐ Racing ☐ Trivia ☐ Deduccion ☐ Economy ☐ Strategy ☐ Humor ☐ Puzzle ☐ Zombie

Nombre	Número de Jugadores	Tiempo estimado	
Prueba Filtros 1	1 player	30 to 45 minutes	<input type="button" value="Añadir a Favoritos"/>
Prueba filtros 2	2 players	45 to 60 minutes	<input type="button" value="Añadir a Favoritos"/>

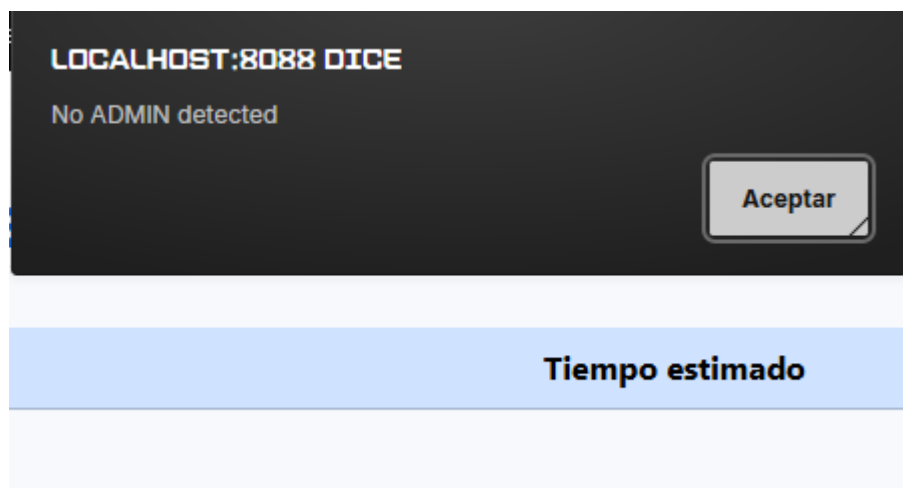
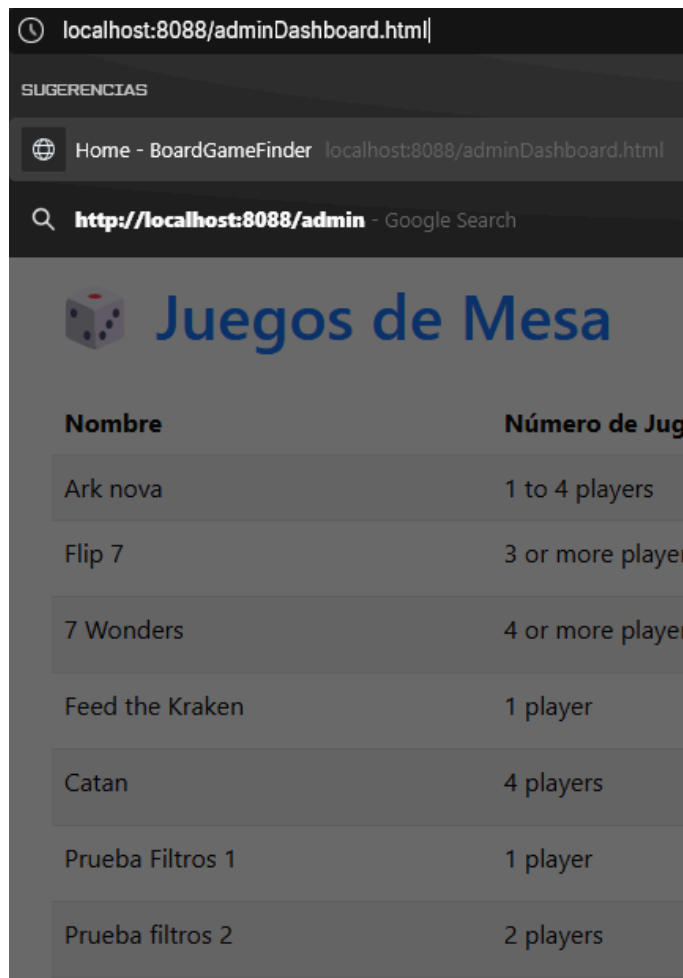
Bloqueo de acceso a rutas protegidas sin token o con token inválido:

LOCALHOST:8088 DICE
Inicio de sesión exitoso

Iniciar sesión

Correo electrónico

Contraseña



Anexo II - Fragmentos del código relevantes

Entidades principales: User y BoardGame

```
package com.BoardGameFinder.BoardGameFinder.Model;

import java.util.List;

import jakarta.persistence.*;
import lombok.*;

@Data
@Builder
@Entity
@Table(name = "User")
@NoArgsConstructor
@AllArgsConstructor
public class User{

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_user")
    private Long idUser;

    @Column(name = "name", nullable = false, length = 50)
    private String name;

    @Column(name = "email", nullable = false, length = 50, unique = true)
    private String email;

    @Column(name = "password", nullable = false, length = 200)
    private String password;

    @Enumerated(EnumType.STRING)
    @Column(name = "role" , nullable = false)
    private Role role;

    // Se crea una relacion N-N con BoardGame, ya que cada usuario va a t
    // Especificamos que el tipo de relacion es EAGER ya que vamos a quer
    // Si la app escala mucho tendria que ser una relacion LAZY y usar un

    @ManyToMany(fetch = FetchType.EAGER)
    @JoinTable({
        name = "user_favorite_boardgames",
        joinColumns = @JoinColumn(name = "user_id"),
        inverseJoinColumns = @JoinColumn(name = "board_game_id")
    })
    private List<BoardGame> favouriteBoardGames;
}
```

```
package com.BoardGameFinder.BoardGameFinder.Model;

import java.util.List;

import com.fasterxml.jackson.annotation.JsonIgnore;

import jakarta.persistence.*;
import lombok.*;

@Entity
@Data
@Builder
@AllArgsConstructor
@NoArgsConstructor
@Table(name = "BoardGame")
public class BoardGame {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id_board_game")
    private Long idBoardGame;

    @Column(nullable = false, length = 100)
    private String name;

    @Enumerated(EnumType.STRING)
    @Column(name = "num_players", length = 50)
    private NumPlayers numPlayers;

    @Enumerated(EnumType.STRING)
    @Column(name = "estimated_time", length = 50)
    private EstTime estTime;

    @Enumerated(EnumType.STRING)
    @Column(length = 100)
    private Editorial editorial;

    @Enumerated(EnumType.STRING)
    @Column(length = 50)
    private Age age;

    @Column(length = 500)
    private String description;

    @Column(name = "image_url", length = 255, nullable = true)
    private String imageUrl;

    @ElementCollection(targetClass = Category.class)
    @Enumerated(EnumType.STRING)
    @CollectionTable(name = "boardgame_categories", joinColumns = @JoinColumn(name = "boardgame_id"))
    @Column(name = "category")
    private List<Category> categories;

    // Se define la relacion N-N con usuario, usando "mappedBy" para indicar que no es unidireccional
    @JsonIgnore
    @ManyToMany(mappedBy = "favouriteBoardGames")
    private List<User> users;
}
```

Enumerados: Category

```
package com.BoardGameFinder.BoardGameFinder.Model;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonValue;

public enum Category {

    ADVENTURE("Adventure"),
    CARDS("Cards"),
    DICE("Dice"),
    EDUCATIONAL("Educational"),
    EXPLORATION("Exploration"),
    FANTASY("Fantasy"),
    MEMORY("Memory"),
    CASUAL("Casual"),
    RACING("Racing"),
    TRIVIA("Trivia"),
    DEDUCCION("Deduccion"),
    ECONOMY("Economy"),
    STRATEGY("Strategy"),
    HUMOR("Humor"),
    PUZZLE("Puzzle"),
    ZOMBIE("Zombie");

    private final String label;

    Category(String label) {
        this.label = label;
    }

    @JsonValue
    public String getLabel() {
        return label;
    }

    @JsonCreator
    public static Category fromLabel(String label) {
        for(Category e: values()) {
            if(e.label.equalsIgnoreCase(label)) {
                return e;
            }
        }
        throw new IllegalArgumentException("Unknown category" + label);
    }
}
```

Seguridad: SecurityConfig

```
package com.BoardGamerinder.BoardGamerinder.Security;

import java.util.List;

@Configuration
@EnableWebSecurity
@EnableMethodSecurity
public class SecurityConfig {

    private final CustomUserDetailsService customUserDetailsService;

    private final JwtUtil jwtUtil;

    private final UserRepository userRepository;

    public SecurityConfig(CustomUserDetailsService customUserDetailsService,
        JwtUtil jwtUtil, UserRepository userRepository) {
        this.customUserDetailsService = customUserDetailsService;
        this.jwtUtil = jwtUtil;
        this.userRepository = userRepository;
    }

    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -> csrf.disable())
            .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))
            .authorizeHttpRequests(auth -> auth
                .requestMatchers("/api/users/register", "/api/auth/**", "/static/**", "/js/**", "/partials/**",
                    "/css/**", "/index.html", "/", "/favicon.ico", "/images/**", "/*.html").permitAll()
                .requestMatchers("/api/users/**").hasAnyRole("USER")
                .anyRequest().authenticated())
            .addFilterBefore(jwtAuthenticationFilter(), UsernamePasswordAuthenticationFilter.class);
        return http.build();
    }

    @Bean
    public AuthenticationManager authenticationManager() {
        DaoAuthenticationProvider provider = new DaoAuthenticationProvider();
        provider.setUserDetailsService(customUserDetailsService);
        provider.setPasswordEncoder(passwordEncoder());
        return new ProviderManager(provider);
    }

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

    @Bean
    public JWTAuthenticationFilter jwtAuthenticationFilter() {
        JWTAuthenticationFilter filter = new JWTAuthenticationFilter();
        filter.setJwtUtil(jwtUtil);
        filter.setUserRepository(userRepository);
        return filter;
    }
}
```

```
@Bean
public CorsConfigurationSource corsConfigurationSource() {
    CorsConfiguration configuration = new CorsConfiguration();
    configuration.setAllowedOrigins(List.of("http://localhost:8080"));
    configuration.setAllowedMethods(List.of("GET", "POST", "PUT", "DELETE", "OPTIONS"));
    configuration.setAllowedHeaders(List.of("*"));
    configuration.setAllowCredentials(true);

    UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
    source.registerCorsConfiguration("/**", configuration);
    return source;
}
```

Repositorios: BoardGameRepository

```
package com.BoardGameFinder.BoardGameFinder.Repository;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.JpaSpecificationExecutor;
import org.springframework.stereotype.Repository;

import com.BoardGameFinder.BoardGameFinder.Model.BoardGame;

@Repository
public interface BoardGameRepository extends JpaRepository<BoardGame, Long>, JpaSpecificationExecutor<BoardGame> {
}
```

Servicios: BoardGameService

```
// Crea un nuevo juego de mesa
public BoardGame saveBoardGame(BoardGame boardGame) {

    if(boardGame.getIdBoardGame() == null) {
        return boardGameRepository.save(boardGame);
    }else {
        throw new BadRequestException("Board game already exists");
    }
}
```

```
// Busca todos los juegos de mesa
public List<BoardGame> getAllBoardGames() {

    if(boardGameRepository.findAll().isEmpty()) {
        throw new BadRequestException("Boad game list is empty");
    }
    return boardGameRepository.findAll();
}
```

```
// Busca un juego de mesa por su identificador
public BoardGame getBoardGameById(Long id){

    return boardGameRepository.findById(id).orElseThrow(() -> new BadRequestException("Board game does not exist"));
}
```

```
// Elimina un juego de mesa por su identificador
public void deleteBoardGame(Long id) {
    if(!boardGameRepository.existsById(id)) {
        throw new BadRequestException("Board game does not exist");
    }
    BoardGame boardGame = boardGameRepository.findById(id).get();
    List<User> users = boardGame.getUsers();
    for(User user:users) {
        user.getFavouriteBoardGames().remove(boardGame);
    }
    userRepository.saveAll(users);
    boardGameRepository.deleteById(id);
}
```

```
public BoardGame updateBoardGame(BoardGame boardGame) {  
  
    if(boardGameRepository.existsById(boardGame.getIdBoardGame())) {  
  
        BoardGame currentBoardGame = boardGameRepository.findById(boardGame.getIdBoardGame()).get();  
  
        // Comprobamos que los campos que nos llegan no están vacíos, no quiero actualizar datos vacíos  
        // La única excepción es la url de la imagen  
  
        if(boardGame.getAge() != null) {  
            currentBoardGame.setAge(boardGame.getAge());  
        }  
  
        if(boardGame.getEstTime() != null) {  
            currentBoardGame.setEstTime(boardGame.getEstTime());  
        }  
  
        if(boardGame.getDescription() != null && !boardGame.getDescription().isBlank()) {  
            currentBoardGame.setDescription(boardGame.getDescription());  
        }  
  
        if(boardGame.getEditorial() != null) {  
            currentBoardGame.setEditorial(boardGame.getEditorial());  
        }  
  
        if(boardGame.getName() != null && !boardGame.getName().isBlank()) {  
            currentBoardGame.setName(boardGame.getName());  
        }  
  
        if(boardGame.getNumPlayers() != null) {  
            currentBoardGame.setNumPlayers(boardGame.getNumPlayers());  
        }  
  
        if(!boardGame.getCategories().isEmpty()) {  
            currentBoardGame.setCategories(boardGame.getCategories());  
        }  
        return boardGameRepository.save(currentBoardGame);  
    }else {  
        throw new BadRequestException("Board game does not exist");  
    }  
}
```

```
public List<BoardGame> searchBoardGames(  
    String name,  
    NumPlayers numPlayers,  
    EstTime estTime,  
    Editorial editorial,  
    Age age,  
    List<Category> categories  
) {  
  
    // Para poder hacer una vista compleja en nuestra base de datos usamos JPA  
    // Ignora los campos nulos de nuestra query automáticamente lo que hace es  
  
    Specification<BoardGame> spec = Specification  
        .where(BoardGameSpecification.hasName(name))  
        .and(BoardGameSpecification.hasNumPlayers(numPlayers))  
        .and(BoardGameSpecification.hasEstTime(estTime))  
        .and(BoardGameSpecification.hasEditorial(editorial))  
        .and(BoardGameSpecification.hasAge(age))  
        .and(BoardGameSpecification.hasAllCategories(categories));  
  
    return boardGameRepository.findAll(spec);  
}
```


Controladores: BoardGameController

```
@GetMapping("/{id}")
public ResponseEntity<BoardGame> getBoardGame (@PathVariable Long id) {
    return ResponseEntity.ok(boardGameService.getBoardGameById(id));
}
```

```
@GetMapping("getall")
public ResponseEntity<List<BoardGame>> getAllBoardGames () {
    return ResponseEntity.ok(boardGameService.getAllBoardGames());
}
```

```
@PostMapping("/create")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<BoardGame> createBoardGame (@RequestBody BoardGame boardGame) {

    System.out.println(boardGame.toString());
    boardGameService.saveBoardGame(boardGame);

    return ResponseEntity.status(HttpStatus.CREATED).body(boardGame);
}
```

```
@DeleteMapping("delete/{id}")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<String> deleteBoardGame (@PathVariable Long id) {
    boardGameService.deleteBoardGame(id);
    return ResponseEntity.ok("Board game deleted");
}
```

```
@PutMapping("/update")
@PreAuthorize("hasRole('ADMIN')")
public ResponseEntity<BoardGame> updateBoardGame (@RequestBody BoardGame boardGame) {
    boardGameService.updateBoardGame(boardGame);
    return ResponseEntity.ok(boardGame);
}
```

```
@GetMapping("/searchboardgames")
public ResponseEntity<List<BoardGame>> searchBoardGames (
    @RequestParam(required = false) String name,
    @RequestParam(required = false) String numPlayers,
    @RequestParam(required = false) String estTime,
    @RequestParam(required = false) String editorial,
    @RequestParam(required = false) String age,
    @RequestParam(required = false) List<String> categories
) {

    NumPlayers numPlayersEnum = numPlayers != null ? NumPlayers.fromLabel(numPlayers) : null;
    EstTime estTimeEnum = estTime != null ? EstTime.fromLabel(estTime) : null;
    Editorial editorialEnum = editorial != null ? Editorial.fromLabel(editorial) : null;
    Age ageEnum = age != null ? Age.fromLabel(age) : null;
    List<Category> categoryEnums = categories != null ? categories.stream()
        .map(Category::fromLabel)
        .collect(Collectors.toList()) : null;

    List<BoardGame> query = boardGameService.searchBoardGames(name, numPlayersEnum,
        estTimeEnum, editorialEnum, ageEnum, categoryEnums);
    return ResponseEntity.ok(query);
}
```

navbar-loader:

```
document.addEventListener("DOMContentLoaded", () => {
  const navbarContainer = document.getElementById("navbar-container");

  fetch("/partials/navbar.html")
    .then(res => res.text())
    .then(html => {
      navbarContainer.innerHTML = html;
      applyNavbarLogic();
    })
    .catch(err => console.error("Error cargando el navbar:", err));
});
```

```
function applyNavbarLogic () {
  const token = localStorage.getItem("token");

  const loginBtn = document.getElementById("loginBtn");
  const registerBtn = document.getElementById("registerBtn");
  const profileBtn = document.getElementById("profileBtn");
  const logoutBtn = document.getElementById("logoutBtn");
  const homeLink = document.getElementById("homeRedirect");
  const aboutBtn = document.getElementById("aboutBtn");
  const infoBtn = document.getElementById("infoBtn");

  if(token) {
    if(loginBtn) loginBtn.classList.add("d-none");
    if(registerBtn) registerBtn.classList.add("d-none");
    if(profileBtn) profileBtn.classList.remove("d-none");
    if(logoutBtn) logoutBtn.classList.remove("d-none");

    const payload = JSON.parse(atob(token.split('.')[1]));
    const role = payload.role || null;

    if (homeLink) {
      homeLink.addEventListener("click", (e) => {
        e.preventDefault();
        if (role === "ADMIN") {
          window.location.href = "/adminDashboard.html";
        } else {
          window.location.href = "/home.html";
        }
      });
    }

    if (role === "ADMIN") {
      if (aboutBtn) aboutBtn.classList.add("d-none");
      if (infoBtn) infoBtn.classList.add("d-none");
      if (profileBtn) profileBtn.classList.add("d-none");
    }
  } else {
    if(loginBtn) loginBtn.classList.remove("d-none");
    if(registerBtn) registerBtn.classList.remove("d-none");
    if(profileBtn) profileBtn.classList.add("d-none");
    if(logoutBtn) logoutBtn.classList.add("d-none");

    if(homeLink) {
      homeLink.addEventListener("click", (e) =>{
        e.preventDefault();
        window.location.href = "/home.html";
      });
    }
  }
}
```

```
function logout() {  
    localStorage.removeItem("token");  
    window.location.href = "/index.html";  
}
```

login:

```
document.addEventListener("DOMContentLoaded", function() {  
  
    const loginForm = document.getElementById("loginForm");  
    const emailInput = document.getElementById("email");  
    const passwordInput = document.getElementById("password");  
    const loginError = document.getElementById("loginError");  
  
    function isValidEmail(email) {  
        const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;  
        return regex.test(email);  
    }  
  
    function parseJwt(token) {  
        const base64Url = token.split('.')[1];  
        const base64 = base64Url.replace(/-/g, '+').replace(/_/g, '/');  
        const jsonPayload = decodeURIComponent(  
            atob(base64)  
                .split('')  
                .map(c => '%' + ('00' + c.charCodeAt(0).toString(16)).slice(-2))  
                .join('')  
        );  
        return JSON.parse(jsonPayload);  
    }  
}
```

```
function mostrarError(mensaje) {  
    const errorDiv = document.getElementById("loginError");  
    errorDiv.textContent = mensaje;  
    errorDiv.classList.remove("d-none");  
    errorDiv.scrollIntoView({ behavior: "smooth" });  
}  
  
);
```

```
loginForm.addEventListener("submit", async (e) => {
    e.preventDefault();

    loginError.textContent = "";

    const email = emailInput.value.trim();
    const password = passwordInput.value;

    if (!email || !password) {
        mostrarError("Por favor, complete todos los campos correctamente.");
        return;
    }

    if (!isValidEmail(email)) {
        mostrarError("Formato de email invalido");
        return;
    }

    if (password.length < 6) {
        mostrarError("La contraseña necesita tener 6 caracteres como minimo");
        return;
    }

    try {
        document.getElementById("loginError").classList.add("d-none");

        const response = await fetch("http://localhost:8088/api/auth/login", {
            method: "POST",
            headers: { "Content-Type": "application/json" },
            body: JSON.stringify({ email, password })
        });

        if (!response.ok) {
            const message = await response.text();
            throw new Error(message);
        }

        const data = await response.json();
        localStorage.setItem("token", data.token);

        const decoded = parseJwt(data.token);
        const role = decoded.role || decoded.roles || decoded.authorities || null;

        alert("Inicio de sesión exitoso");

        if(role == "ADMIN"){
            window.location.href= "/adminDashboard.html";
        }else{
            window.location.href = "/home.html";
        }
    } catch (err) {
        mostrarError("Credenciales incorrectas");
    }
});
```

home:

```
document.addEventListener("DOMContentLoaded", () => {  
    const token = localStorage.getItem("token");  
  
    if (!token) {  
        alert("Sesión no iniciada");  
        window.location.href = "/login.html";  
        return;  
    }  
  
    const tableBody = document.querySelector("#boardGamesTable tbody");  
    let favouriteIds = new Set();  
  
    function clearTable() {  
        tableBody.innerHTML = "";  
    }  
  
    async function loadEnumOptions() {  
        try {  
            const response = await fetch("http://localhost:8088/api/enums/all", {  
                headers: {  
                    "Content-Type": "application/json",  
                    "Authorization": `Bearer ${token}`  
                }  
            });  
  
            if (!response.ok) {  
                const message = await response.text();  
                throw new Error(message);  
            }  
  
            const enums = await response.json();  
  
            fillSelect("filterPlayers", enums.numPlayers);  
            fillSelect("filterTime", enums.estTime);  
            fillSelect("filterEditorial", enums.editorial);  
            fillSelect("filterAge", enums.age);  
            populateCategoryCheckboxes(enums.categories);  
  
        } catch (error) {  
            console.error(error);  
            alert("Error al cargar los filtros");  
        }  
    }  
})
```

```
function fillSelect(selectId, options, multiple = false) {
  const select = document.getElementById(selectId);
  select.innerHTML = "";

  if (!multiple) {
    const defaultOption = document.createElement("option");
    defaultOption.value = "";
    defaultOption.textContent = "-- Selecciona una opción --";
    select.appendChild(defaultOption);
  }

  options.forEach(option => {
    const opt = document.createElement("option");
    opt.value = option;
    opt.textContent = option;
    select.appendChild(opt);
  });
}
```

```
function populateCategoryCheckboxes(categories) {
  const container = document.getElementById("filterCategories");
  container.innerHTML = "";

  categories.forEach(cat => {
    const div = document.createElement("div");
    div.classList.add("form-check");

    const input = document.createElement("input");
    input.type = "checkbox";
    input.classList.add("form-check-input");
    input.name = "filterCategory";
    input.value = cat;
    input.id = `cat-${cat}`;

    const label = document.createElement("label");
    label.classList.add("form-check-label");
    label.htmlFor = input.id;
    label.textContent = cat;

    div.appendChild(input);
    div.appendChild(label);
    container.appendChild(div);
  });
}
```

```
async function loadUserFavourites() {
  try {
    const response = await fetch("http://localhost:8088/api/users/me", {
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${token}`
      }
    });

    if (!response.ok) {
      const message = await response.text();
      throw new Error(message);
    }

    const user = await response.json();
    user.favouriteBoardGames.forEach(game => favouriteIds.add(game.idBoardGame));
  } catch (error) {
    console.error(error);
    alert("Error al obtener juegos favoritos del usuario");
  }
}
```

```
async function loadBoardGames() {
  clearTable();
  try {
    const response = await fetch("http://localhost:8088/api/boardgames/getall", {
      method: "GET",
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${token}`
      }
    });

    if (!response.ok) {
      const message = await response.text();
      throw new Error(message);
    }

    const boardGames = await response.json();
    renderBoardGames(boardGames);
  } catch (error) {
    console.error(error);
    alert("Error al cargar los juegos");
  }
}
```

```
function renderBoardGames(boardGames) {  
  clearTable();  
  boardGames.forEach(game => {  
    const row = document.createElement("tr");  
  
    const nameCell = document.createElement("td");  
    nameCell.textContent = game.name;  
  
    const playersCell = document.createElement("td");  
    playersCell.textContent = game.numPlayers;  
  
    const timeCell = document.createElement("td");  
    timeCell.textContent = game.estTime;  
  
    const actionCell = document.createElement("td");  
  
    if (!favouriteIds.has(game.idBoardGame)) {  
      const favButton = document.createElement("button");  
      favButton.textContent = "Añadir a Favoritos";  
      favButton.classList.add("btn", "btn-outline-danger", "btn-sm");  
  
      favButton.addEventListener("click", async (e) => {  
        e.stopPropagation();  
        try {  
          const response = await fetch(`http://localhost:8088/api/users/me/fav/${game.idBoardGame}`, {  
            method: "POST",  
            headers: {  
              "Content-Type": "application/json",  
              "Authorization": `Bearer ${token}`  
            }  
          });  
  
          if (!response.ok) throw new Error(await response.text());  
  
          alert(`${game.name} añadido a favoritos`);  
          favButton.remove();  
        } catch (err) {  
          console.error(err);  
          alert("Error al añadir a favoritos");  
        }  
      });  
  
      actionCell.appendChild(favButton);  
    }  
  
    row.appendChild(nameCell);  
    row.appendChild(playersCell);  
    row.appendChild(timeCell);  
    row.appendChild(actionCell);  
  
    row.addEventListener("click", () => {  
      localStorage.setItem("selectedBoardGameId", game.idBoardGame);  
      window.location.href = "/boardGameView.html";  
    });  
  
    tableBody.appendChild(row);  
  });  
}
```


Módulo: Proyecto Final de Ciclo

Ciclo: DAM

```
async function applyFilters(e) {
  e.preventDefault();

  const params = new URLSearchParams();

  const name = document.getElementById("filterName").value;
  const numPlayers = document.getElementById("filterPlayers").value;
  const estTime = document.getElementById("filterTime").value;
  const editorial = document.getElementById("filterEditorial").value;
  const age = document.getElementById("filterAge").value;
  const categories = Array.from(document.querySelectorAll("input[name='filterCategory']:checked"))
    .map(cb => cb.value);

  if (name) params.append("name", name);
  if (numPlayers) params.append("numPlayers", numPlayers);
  if (estTime) params.append("estTime", estTime);
  if (editorial) params.append("editorial", editorial);
  if (age) params.append("age", age);
  categories.forEach(cat => params.append("categories", cat));

  try {
    const response = await fetch(`http://localhost:8088/api/boardgames/searchboardgames?${params.toString()}`, {
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${token}`
      }
    });

    if (!response.ok) {
      const message = await response.text();
      throw new Error(message);
    }

    const boardGames = await response.json();
    renderBoardGames(boardGames);
  } catch (error) {
    console.error(error);
    alert("Error al aplicar filtros");
  }
}

document.getElementById("filtersForm").addEventListener("submit", applyFilters);

document.getElementById("toggleFilters").addEventListener("click", () => {
  const container = document.getElementById("filtersContainer");
  container.style.display = container.style.display === "none" ? "block" : "none";
});

(async () => {
  await loadEnumOptions();
  await loadUserFavourites();
  await loadBoardGames();
})();
```

adminDashboard:

```
document.addEventListener("DOMContentLoaded", () => {
  const token = localStorage.getItem("token");

  if (!token) {
    alert("Sesión no iniciada");
    window.location.href = "/login.html";
    return;
  }
});
```

Módulo: Proyecto Final de Ciclo

Ciclo: DAM

```
async function loadGames() {
  try {
    const response = await fetch("http://localhost:8088/api/boardgames/getall", {
      headers: {
        "Content-Type": "application/json",
        "Authorization": `Bearer ${token}`
      }
    });

    if(!response.ok){
      const errorText = await response.text();
      throw new Error(errorText || "Error al cargar la tabla");
    }

    const games = await response.json();
    tableBody.innerHTML = "";

    games.forEach(game => {
      const row = document.createElement("tr");

      const nameCell = document.createElement("td");
      nameCell.textContent = game.name;

      const playersCell = document.createElement("td");
      playersCell.textContent = game.numPlayers;

      const timeCell = document.createElement("td");
      timeCell.textContent = game.estTime;

      const actionCell = document.createElement("td");

      const editBtn = document.createElement("button");
      editBtn.textContent = "Editar";
      editBtn.classList.add("btn", "btn-sm", "btn-warning", "me-2");
      editBtn.addEventListener("click", (e) => {
        e.stopPropagation();
        localStorage.setItem("editBoardGameId", game.idBoardGame);
        window.location.href = "/boardGameFormEdit.html";
      });

      const deleteBtn = document.createElement("button");
      deleteBtn.textContent = "Eliminar";
      deleteBtn.classList.add("btn", "btn-sm", "btn-danger");
      deleteBtn.addEventListener("click", async (e) => {
        e.stopPropagation();
        if (!confirm(`¿Eliminar "${game.name}"?`)) return;
        try {
          const res = await fetch(`http://localhost:8088/api/boardgames/delete/${game.idBoardGame}`, {
            method: "DELETE",
            headers: { "Authorization": `Bearer ${token}` }
          });
          if (!res.ok) throw new Error(await res.text());
          loadGames();
        } catch (err) {
          console.error(err);
          alert("Error al eliminar juego");
        }
      });

      actionCell.appendChild(editBtn);
      actionCell.appendChild(deleteBtn);

      row.appendChild(nameCell);
      row.appendChild(playersCell);
      row.appendChild(timeCell);
      row.appendChild(actionCell);

      row.addEventListener("click", () => {
        localStorage.setItem("selectedBoardGameId", game.idBoardGame);
        window.location.href = "/boardGameView.html";
      });

      tableBody.appendChild(row);
    });
  } catch (error) {
    console.error(error);
    alert("Error al cargar juegos");
  }
}

loadGames();
```

```
actionCell.appendChild(editBtn);
actionCell.appendChild(deleteBtn);

row.appendChild(nameCell);
row.appendChild(playersCell);
row.appendChild(timeCell);
row.appendChild(actionCell);

row.addEventListener("click", () => {
  localStorage.setItem("selectedBoardGameId", game.idBoardGame);
  window.location.href = "/boardGameView.html";
});

tableBody.appendChild(row);
});
} catch (error) {
  console.error(error);
  alert("Error al cargar juegos");
}
}

loadGames();
```

boardGameFormCreate:

```
document.addEventListener("DOMContentLoaded", async () => {
  const token = localStorage.getItem("token");
  if (!token) {
    alert("Sesión no iniciada");
    window.location.href = "/login.html";
    return;
  }
});
```

```
function mostrarError(mensaje) {
  boardGameError.textContent = mensaje;
  boardGameError.classList.remove("d-none");
  boardGameError.scrollIntoView({ behavior: "smooth" });
}

};
```

```
const form = document.getElementById("boardGameForm");
form.addEventListener("submit", async (e) => {
  e.preventDefault();

  boardGameError.textContent = "";

  const name = document.getElementById("name").value.trim();
  const description = document.getElementById("description").value.trim();
  const imageUrl = document.getElementById("imageUrl").value.trim();
  const numPlayers = document.getElementById("numPlayers").value;
  const estTime = document.getElementById("estTime").value;
  const editorial = document.getElementById("editorial").value;
  const age = document.getElementById("age").value;
  const selectedCategories = Array.from(document.querySelectorAll("#categoriesContainer input:checked"))
    .map(cb => cb.value);

  if (!name || !description || !imageUrl || !numPlayers || !estTime || !editorial || !age) {
    mostrarError("Por favor, complete todos los campos.");
    return;
  }

  if (selectedCategories.length == 0) {
    mostrarError("Debe seleccionar al menos una categoría.");
    return;
  }

  const gameData = {
    name,
    description,
    imageUrl,
    numPlayers,
    estTime,
    editorial,
    age,
    categories: selectedCategories
  };
});
```

```
const boardGameError = document.getElementById("boardGameError");

try {
  const response = await fetch("http://localhost:8088/api/enums/all", {
    headers: {
      "Content-Type": "application/json",
      "Authorization": `Bearer ${token}`
    }
  });

  if (!response.ok) {
    const message = await response.text();
    throw new Error(message);
  }

  const enums = await response.json();

  for (const [key, values] of Object.entries(enums)) {
    if (key === "categories") {
      const container = document.getElementById("categoriesContainer");

      container.classList.add("d-flex", "flex-wrap", "gap-3");

      values.forEach(value => {
        const div = document.createElement("div");
        div.classList.add("form-check");

        const checkbox = document.createElement("input");
        checkbox.type = "checkbox";
        checkbox.className = "form-check-input";
        checkbox.name = "categories";
        checkbox.value = value.toUpperCase();
        checkbox.id = `cat-${value}`;

        const label = document.createElement("label");
        label.className = "form-check-label";
        label.setAttribute("for", checkbox.id);
        label.textContent = value;

        div.appendChild(checkbox);
        div.appendChild(label);
        container.appendChild(div);
      });
    } else {
      const select = document.getElementById(key);
      if (select) {
        values.forEach(value => {
          const option = document.createElement("option");
          option.value = value;
          option.textContent = value;
          select.appendChild(option);
        });
      }
    }
  }
}
```

