



**XUNTA  
DE GALICIA**

CONSELLERÍA DE CULTURA,  
EDUCACIÓN, FORMACIÓN  
PROFESIONAL E UNIVERSIDADES



**IES SAN MAMEDE**

📍 Rúa do Castelo N° 3. 32700. Maceda (Ourense)  
☎ Tlf: 988 788 561 ✉ ies.san.mamede@edu.xunta.gal  
🌐 <http://www.iessanmamede.com>

## Trivial

**Nome Alumno/a:**

***Iván Baños Piñeiro***

**Curso: 2º DAM**

**Materia: Proxecto Final Módulo Acceso a Datos**

**Docente: Javier Feijóo López**

**🔗 Proxecto: <https://github.com/IvanBanhosPinheiro/ProyectoAD.git>**

## Contido

1. Introducción	2
2. Deseño da Base de Datos – Modelo EER	2
3. Tecnoloxías Empregadas e Configuración da Contorna	2
4. Deseño do Backend – API REST con Spring Boot	2
5. Implementación de Autenticación e Seguridade	3
6. Desenvolvemento do Backend con Thymeleaf	3
7. Documentación da API e Probas en Postman	3
8. Conclusións e Valoración Persoal	3
9. Anexos (Opcional)	3

## 1. Introducción

El sistema es una API para un juego de trivial que permite gestionar categorías, preguntas, respuestas y partidas de los usuarios.

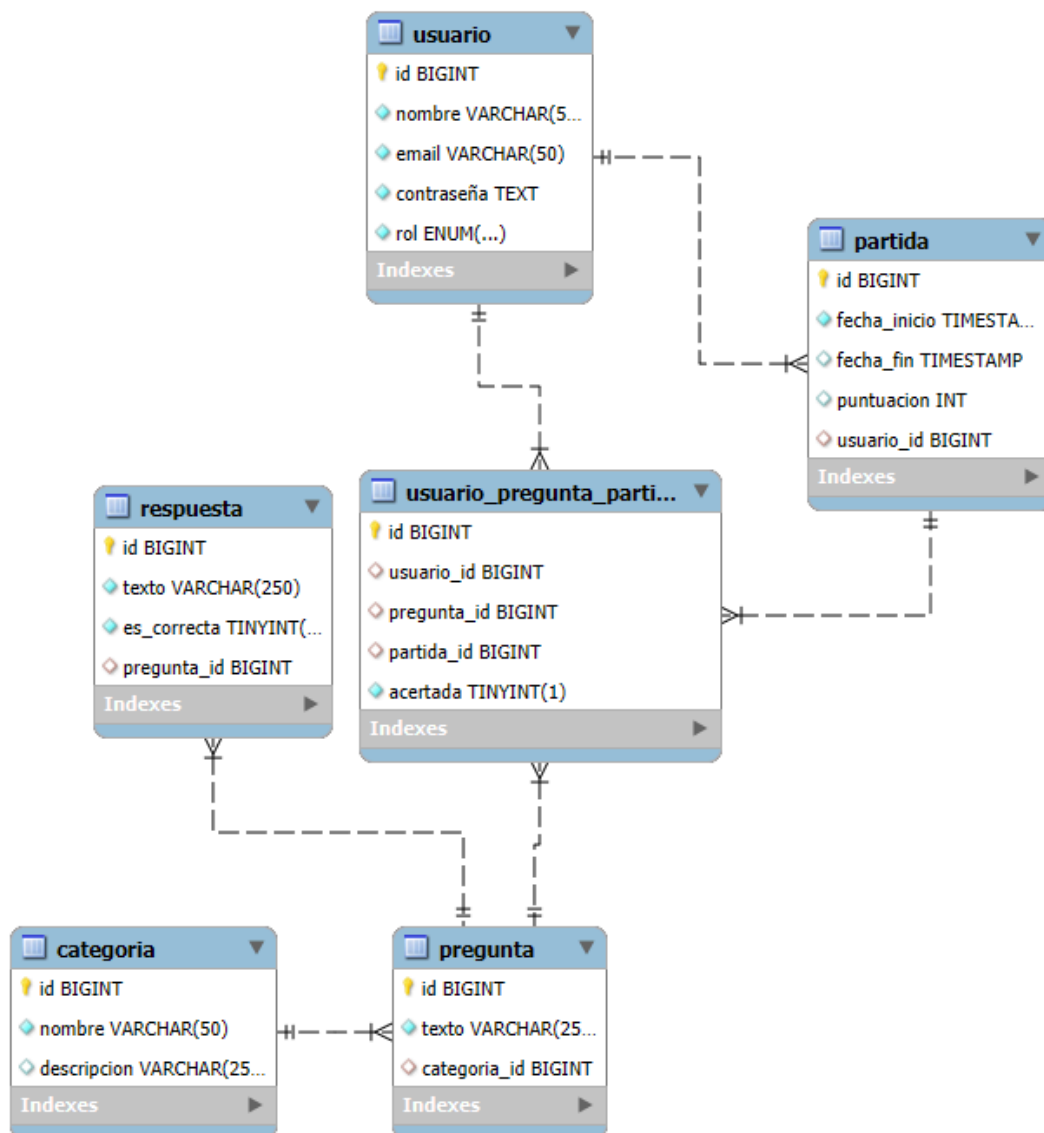
Finalidad: Ofrecer una plataforma para jugar al trivial de forma organizada y accesible.

Problema que resuelve: Facilita la gestión del juego, permitiendo almacenar preguntas, registrar partidas y llevar un control del rendimiento de los jugadores.

Usuarios principales: Personas con ganas de jugar y divertirse respondiendo preguntas.

## 2. Diseño da Base de Datos – Modelo EER

- Modelo Entidade-Relación.



- Explicación de cada entidad y de sus relaciones.

**Usuario:**

Almacena información sobre los usuarios del sistema.

Campos: id (identificador único), nombre, email (único y necesario para identificación), contraseña (almacenada de forma segura), rol (define si es administrador o usuario regular).

**Categoría:**

Representa las categorías a las que pueden pertenecer las preguntas.

Campos: id (identificador único), nombre (nombre de la categoría), descripcion (opcional, para describir la categoría).

**Pregunta:**

Contiene las preguntas del juego.

Campos: id (identificador único), texto (el enunciado de la pregunta), categoria\_id (clave foránea que referencia a la categoría a la que pertenece la pregunta).

**Respuesta:**

Almacena las posibles respuestas a cada pregunta, indicando si son correctas o no.

Campos: id (identificador único), texto (el texto de la respuesta), es\_correcta (booleano que indica si la respuesta es correcta), pregunta\_id (clave foránea que referencia a la pregunta a la que pertenece la respuesta).

**Partida:**

Registra las partidas jugadas por los usuarios, con su puntuación y fechas.

Campos: id (identificador único), fecha\_inicio (fecha y hora de inicio de la partida), fecha\_fin (opcional, fecha y hora de fin de la partida), puntuacion (puntuación obtenida en la partida), usuario\_id (clave foránea que referencia al usuario que jugó la partida).

**Usuario\_Pregunta\_Partida:**

Relaciona qué preguntas ha respondido cada usuario en cada partida, y si las ha acertado.

Campos: id (identificador único), usuario\_id (clave foránea que referencia al usuario), pregunta\_id (clave foránea que referencia a la pregunta respondida), partida\_id (clave foránea que referencia a la partida en la que se respondió la pregunta), acertada (booleano que indica si la pregunta fue acertada por el usuario).

- Descripción das claves primarias e foráneas.

**Claves Primarias (PK - Primary Key)**

Cada tabla tiene una clave primaria que identifica de forma única cada registro.

usuario(id) → Identifica de manera única a cada usuario.

categoria(id) → Identifica de manera única a cada categoría.

pregunta(id) → Identifica de manera única a cada pregunta.

respuesta(id) → Identifica de manera única a cada respuesta.

partida(id) → Identifica de manera única a cada partida.

usuario\_pregunta\_partida(id) → Identifica de manera única cada relación entre usuario, pregunta y partida.

**Claves Foráneas (FK - Foreign Key)**

Las claves foráneas establecen relaciones entre las tablas, asegurando la integridad referencial.

pregunta(categoria\_id) REFERENCES categoria(id)

Relaciona cada pregunta con una categoría específica.

respuesta(pregunta\_id) REFERENCES pregunta(id)

Relaciona cada respuesta con una pregunta específica.

partida(usuario\_id) REFERENCES usuario(id)

Indica qué usuario jugó cada partida.

usuario\_pregunta\_partida(usuario\_id) REFERENCES usuario(id)

Relaciona un usuario con la pregunta que respondió en una partida.

usuario\_pregunta\_partida(pregunta\_id) REFERENCES pregunta(id)

Relaciona una pregunta con la respuesta dada en una partida.

usuario\_pregunta\_partida(partida\_id) REFERENCES partida(id)

Relaciona una partida con las preguntas que se respondieron en ella.

**Resumen de Relaciones**

usuario → Se relaciona con partida y usuario\_pregunta\_partida.

categoria → Se relaciona con pregunta.

pregunta → Se relaciona con categoria, respuesta y usuario\_pregunta\_partida.

respuesta → Se relaciona con pregunta.

partida → Se relaciona con usuario y usuario\_pregunta\_partida.

usuario\_pregunta\_partida → Es una tabla intermedia que relaciona usuario, pregunta y partida.

- Restriccións e regras de integridade.

**1. Restriccións de Clave Primaria (PRIMARY KEY)**

Todas las tablas tienen una clave primaria (id) que asegura que cada registro es único.

**2. Restriccións de Clave Foránea (FOREIGN KEY)**

Se aplican en las relaciones entre tablas para mantener la integridad referencial.

Regla: No se puede insertar un valor en una columna foránea si no existe en la tabla referenciada.

Ejemplo: No se puede insertar una pregunta con un categoria\_id que no exista en categoria.

**3. Restricciones de Unicidad (UNIQUE)**

usuario.email → Evita que dos usuarios tengan el mismo correo.

categoria.nombre → Evita que existan categorías duplicadas.

**4. Restricciones de No Nulo (NOT NULL)**

Se aplica a campos esenciales para evitar valores vacíos.

Ejemplo: Un usuario debe tener nombre, email y contraseña.

Ejemplo: Una pregunta debe tener un texto y una respuesta debe tener texto y es\_correcta.

**5. Restricción de Tipo de Dato (ENUM)**

usuario.rol → Solo puede ser 'admin' o 'usuario'.

Regla: Evita que se inserten valores no válidos.

**6. Restricción de Valores Booleanos (BOOLEAN NOT NULL)**

respuesta.es\_correcta y usuario\_pregunta\_partida.acertada deben ser TRUE o FALSE.

**7. Restricciones de Integridad en la Relación usuario\_pregunta\_partida**

Asegura que cada entrada representa una relación válida entre usuario, pregunta y partida.

### 3. Tecnologías Empleadas e Configuración da Contorna

- Linguaxe de programación: Java 17 .

Es una versión LTS (Long-Term Support) del lenguaje de programación Java, lanzada en 2021. Ofrece mejoras en rendimiento, seguridad y nuevas características como sealed classes, pattern matching y records. Es ideal para desarrollo empresarial y aplicaciones robustas.

- Framework: Spring Boot 3.

Es un framework basado en Java que facilita la creación de aplicaciones web y microservicios. Simplifica la configuración de proyectos al ofrecer un enfoque de "convención sobre configuración". Spring Boot 3, que se basa en Spring Framework 6, ofrece soporte para Java 17+ y características modernas, como integración con Jakarta EE, mejoras en la seguridad y rendimiento, y mayor facilidad para crear aplicaciones independientes con configuración mínima.

- Base de datos: H2 (embebida).

Es una base de datos relacional en memoria, ligera y de código abierto. Está escrita en Java y se utiliza principalmente para pruebas o aplicaciones pequeñas. Puede funcionar tanto en modo embebido (sin servidor) como en modo cliente-servidor. H2 es muy rápida y fácil de configurar, lo que la convierte en una opción popular para el desarrollo y pruebas de aplicaciones, pero no suele ser usada en producción a gran escala.

- ORM: Spring Data JPA.

Es un proyecto de Spring que simplifica el uso de JPA (Java Persistence API) para interactuar con bases de datos. Proporciona una capa de abstracción que facilita la implementación de repositorios de acceso a datos, eliminando la necesidad de escribir mucho código repetitivo. Con Spring Data JPA, puedes definir interfaces de repositorio y el framework genera automáticamente las consultas SQL basadas en los métodos que defines, como `findById()`, `findByName()`, etc. Esto acelera el desarrollo y reduce la complejidad al trabajar con bases de datos en aplicaciones Java.

- Autenticación: Spring Security + JWT.

- Spring Security es un framework que proporciona autenticación y autorización en aplicaciones Java. Permite gestionar el acceso a recursos mediante diversas estrategias como autenticación por formulario, autenticación básica, o integración con OAuth2. Es altamente configurable y se adapta a diversas necesidades de seguridad.
- JWT (JSON Web Token) es un estándar abierto que permite la transmisión segura de información entre partes como un objeto JSON. En el contexto de autenticación, se usa para generar tokens firmados que permiten verificar la identidad del usuario de manera stateless (sin necesidad de mantener el estado del servidor).

- Frontend para administradores: Thymeleaf + Bootstrap.

- Thymeleaf es un motor de plantillas para aplicaciones web Java, utilizado principalmente con Spring Boot. Permite generar vistas HTML dinámicas con una sintaxis simple y natural. Thymeleaf se integra bien con Spring, permitiendo pasar objetos y datos del backend a las plantillas HTML, que luego se renderizan en el navegador del usuario.
- Bootstrap es un framework CSS de código abierto que facilita el desarrollo de interfaces web responsivas y modernas. Proporciona una serie de componentes predefinidos (como botones, tablas, formularios, navegación) y una cuadrícula flexible que se adapta automáticamente a diferentes tamaños de pantalla, mejorando la experiencia en dispositivos móviles.

- Herramientas de documentación: Swagger, Postman.

- Swagger es un conjunto de herramientas para documentar y probar APIs RESTful. Permite describir la estructura de las API mediante un archivo de especificación (OpenAPI), lo que facilita la generación automática de documentación interactiva y la creación de clientes y servidores. Además, proporciona una interfaz web donde los desarrolladores pueden probar las API directamente desde la documentación, lo que simplifica las pruebas y el desarrollo de las mismas.
- Postman es una herramienta popular para realizar pruebas y depuración de APIs. Permite enviar peticiones HTTP (GET, POST, PUT, DELETE, etc.) y visualizar las respuestas de manera estructurada. Postman también permite organizar las pruebas en colecciones, crear entornos de prueba, generar documentación y automatizar pruebas, facilitando la integración y el trabajo en equipo en el desarrollo de APIs.

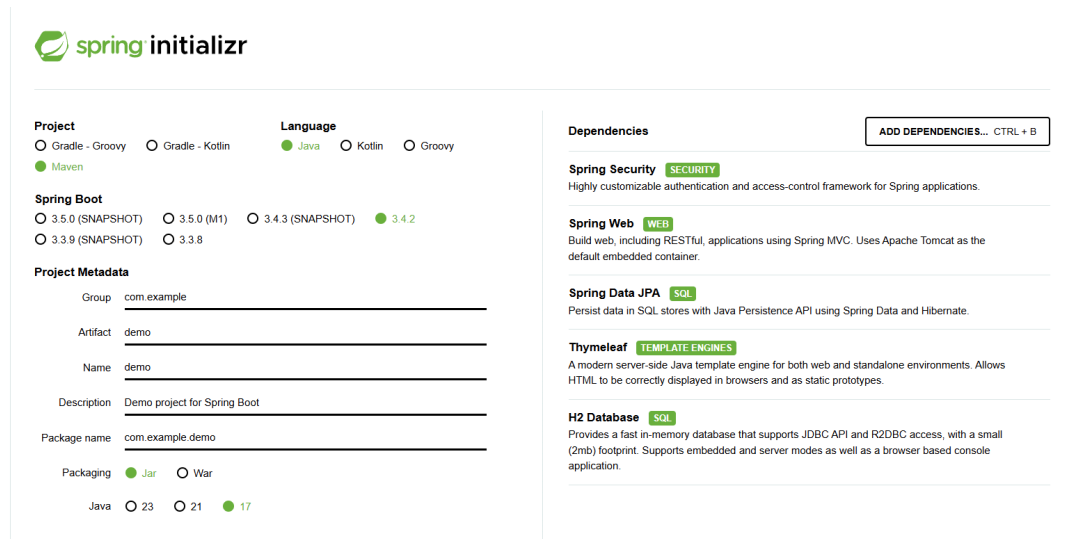
- Gestión de dependencias: Maven.

Maven es una herramienta de gestión de proyectos Java que maneja las dependencias automáticamente. Se usa un archivo llamado pom.xml, donde declaras las bibliotecas que tu proyecto necesita. Maven descarga estas dependencias desde repositorios, resuelve versiones automáticamente y maneja dependencias transitivas (es decir, las dependencias de las dependencias). Esto facilita la gestión y actualización de las bibliotecas sin intervención manual.

Incluir pasos para configurar o contorno:

- IDE recomendado IntelliJ.


Para crear el proyecto nos dirigimos a <https://start.spring.io/> y cargamos las dependencias



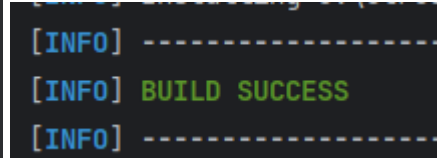
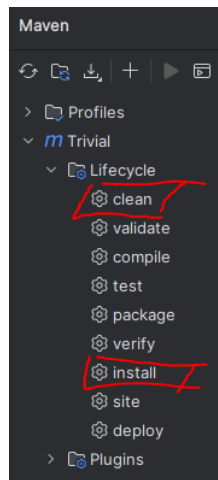
The screenshot shows the Spring Initializr web form. It includes sections for Project, Language, Spring Boot, Project Metadata, and Dependencies. The Project section has radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven (selected). The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for 3.5.0 (SNAPSHOT), 3.5.0 (M1), 3.4.3 (SNAPSHOT), and 3.4.2 (selected). The Project Metadata section includes fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). The Packaging section has radio buttons for Jar (selected) and War. The Dependencies section lists several dependencies: Spring Security, Spring Web, Spring Data JPA, Thymeleaf, and H2 Database, each with a button to add it.

Se agregaron dependencias a mayores mira el pom.xml y asegurate de tenerlos todos.

- Preparación de jar para ejecutar en equipos externos:

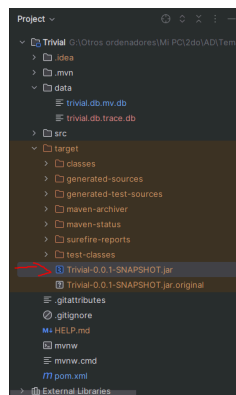
En el IntelliJ pulsamos en la M a la derecha  esto nos abrirá una pestaña donde tendremos que darle a clean y después a install para generar el .jar que será nuestro ejecutable





Es que esta todo OK y

tendremos el .jar en nuestros archivos.



Si abrimos una consola en la carpeta donde se encuentra ejecutaremos este comando `java -jar Trivial-0.0.1-SNAPSHOT.jar` y se ejecutará la aplicación.

## 4. Diseño do Backend – API REST con Spring Boot

Explicación da arquitectura do backend:

ad.Trivial

- config -> contiene la seguridad, la configuración del swagger y el cors
  - securiti -> Contiene las clases de seguridad y se generan los token
- controllers -> contiene los controladores del backend y del frontend
  - backend -> contiene las operaciones del thymeleaf
  - crudCompleto -> contiene las operaciones crud nativas accesibles por la api
  - frontend -> controladores rest para el frontend, no esta el crud completo
  - AuthController -> rest controller de registro y login
- models -> contiene los modelos de datos de la base de datos y los DTO
  - modelosDTO -> contiene los DTO todos de la aplicación
  - resto de modelos -> los modelos originales están sueltos
- repositories -> Contiene las interfaces con los métodos crud
- services -> Contiene los repositories implementados en services
- resources

- templates
  - admin -> Contiene todas los html del backend con el thymeleaf

### **Controladores (controllers):**

Se encargan de recibir peticiones del frontend o API y delegarlas a los servicios.

Están organizados en subpaquetes:

backend: Maneja operaciones con Thymeleaf.

crudCompleto: Expone las operaciones CRUD accesibles vía API REST.

frontend: Controladores REST sin CRUD completo.

AuthController: Controlador REST para autenticación (registro y login).

### **Servicios (services):**

Implementan la lógica de negocio.

Usan los repositorios (repositories) para interactuar con la base de datos.

Contienen métodos como obtenerTodas(), guardar(), actualizar(), eliminar(), etc.

### **Ejemplo de CRUD para la entidad Categoría (CategoriaCrudController):**

- Obtener todas las categorías (GET /crud/categorias)

@GetMapping

```
public List<Categoria> obtenerTodas() {  
    return categoriaService.obtenerTodas();  
}
```

- Guardar una nueva categoría (POST /crud/categorias)

@PostMapping

```
public Categoria guardar(@RequestBody Categoria categoria) {  
    return categoriaService.guardar(categoria);  
}
```

- Actualizar una categoría (PUT /crud/categorias)

@PutMapping

```
public Categoria actualizar(@RequestBody Categoria categoria) {  
    return categoriaService.guardar(categoria);  
}
```

- Eliminar una categoría por ID (DELETE /crud/categorias/{id})

@DeleteMapping("/{id}")

```
public void borrar(@PathVariable Long id) {  
    categoriaService.eliminar(id);  
}
```

- Servicio asociado (CategoriaService)

@Service

```
public class CategoriaService {  
    @Autowired  
    private CategoriaRepository categoriaRepository;  
    public List<Categoria> obtenerTodas() {  
        return categoriaRepository.findAll();  
    }  
    public Categoria guardar(Categoria categoria) {  
        return categoriaRepository.save(categoria);  
    }  
    public void eliminar(Long id) {  
        categoriaRepository.deleteById(id);  
    }  
}
```

- Repositorio (CategoriaRepository)

@Repository

```
public interface CategoriaRepository extends JpaRepository<Categoria, Long> {  
}
```

## 5. Implementación de Autenticación e Seguridad

### 1. Registro e inicio de sesión con JWT

#### Registro:

Cuando un usuario se registra, su contraseña se encripta con BCrypt antes de guardarse en la base de datos.

Esto se hace en UsuarioService con

```
BCryptPasswordEncoder().encode(usuario.getContraseña()).
```

#### Inicio de sesión:

Cuando un usuario inicia sesión, se genera un JWT con su nombre de usuario.

JwtService se encarga de crear el token con generateToken(username), estableciendo un tiempo de expiración y firmándolo con una clave secreta.

El token se devuelve al cliente y se usa en cada petición autenticada.

### 2. Protección de endpoints con roles (USER, ADMIN)

SpringSecurity define la seguridad de los endpoints en securityFilterChain():

#### Rutas públicas:

/auth/\*\* → Permite el registro e inicio de sesión sin autenticación.

/swagger-ui/\*\* → Permite acceso sin restricciones a la documentación de la API.

/h2-console/\*\* → Permite el acceso a la consola H2.

/api/partidas/mejores → Accesible sin autenticación.

#### Rutas protegidas:

/crud/\*\* → Solo accesible por usuarios con rol ADMIN.

/api/\*\* → Solo accesible por USER y ADMIN.

Cualquier otra petición debe estar autenticada.

### 3. Encriptación de contraseñas con BCrypt

BCryptPasswordEncoder encripta la contraseña antes de guardarla en la base de datos. En UserDetailsServiceImpl, cuando un usuario intenta autenticarse, se compara la contraseña encriptada con la ingresada.

Esto evita que las contraseñas se almacenen en texto plano, asegurando la seguridad de los datos de los usuarios.

### 4. Validación y uso del token JWT

JwtRequestFilter intercepta todas las peticiones (//\*).

Si hay un token en la cabecera Authorization, lo valida (jwtService.validateToken(token)).

Si es válido, obtiene el usuario (jwtService.getUsernameFromToken(token)) y lo autentica en el contexto de seguridad (SecurityContextHolder).

Así, cada petición protegida requiere un JWT válido.

## 6. Desenvolvimiento do Backend con Thymeleaf

Explicación do panel de administración con [Thymeleaf](#):

En mi panel de administración tengo una pantalla de inicio que solo muestra un mensaje de inicio y en el navbar tienes enlaces a todos los apartados que tiene para modificar:

Categoría, Preguntas, Usuarios, Partidas, Resultado de preguntas. en todas las páginas tenemos la misma estructura, una lista con un editar borrar y un formulario abajo para agregar o editar.

Link: <http://localhost:8080/admin/dashboard>

### Gestión de Categorías

#### Listado de Categorías

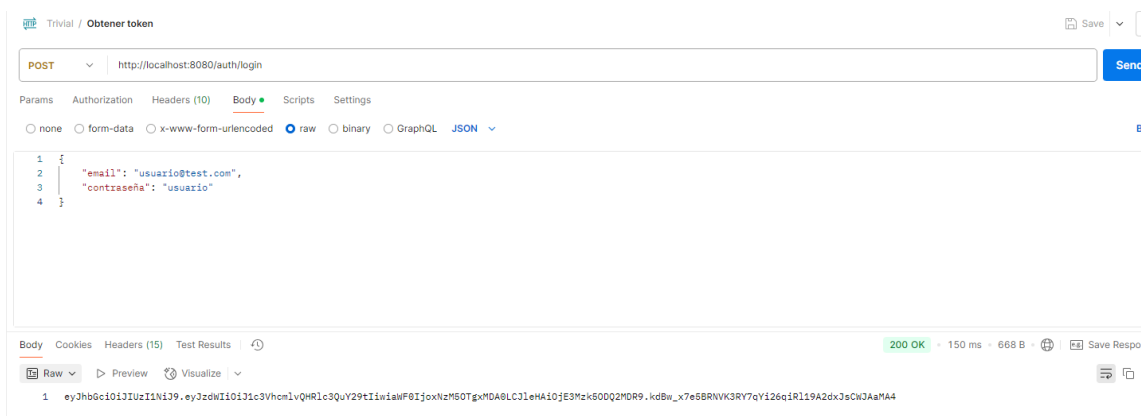
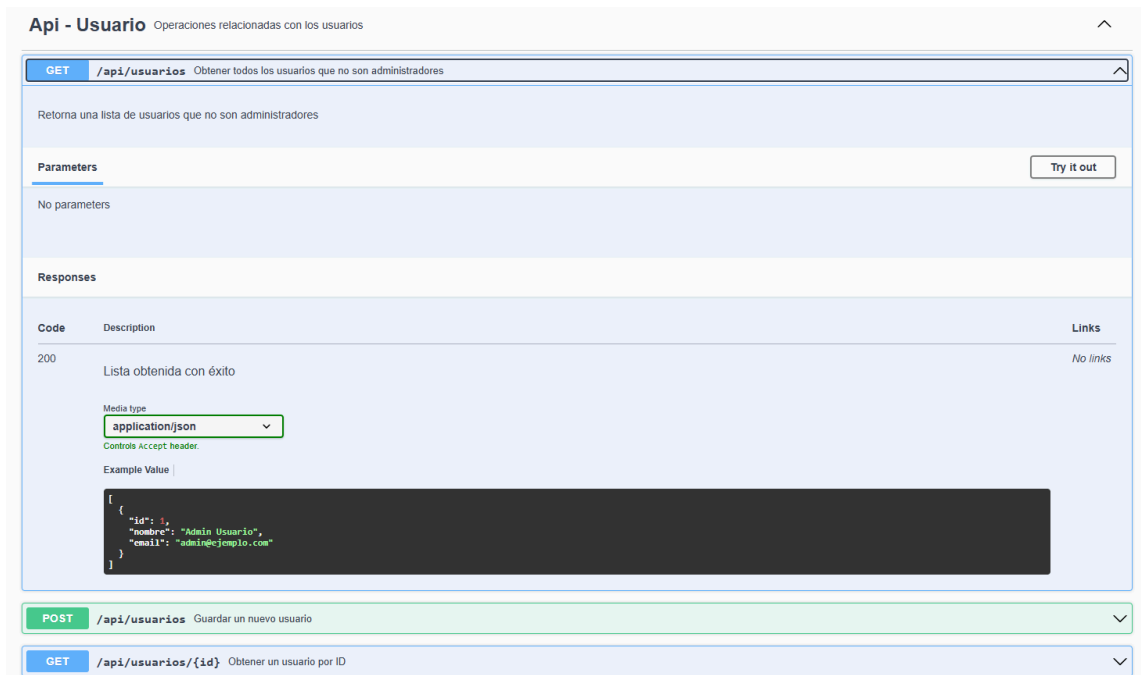
ID	Nombre	Descripción	Acciones
1	Historia	Preguntas relacionadas con eventos históricos y personajes importantes.	<a href="#">Editar</a> <a href="#">Eliminar</a>
2	Ciencia	Preguntas sobre temas científicos como física, química y biología.	<a href="#">Editar</a> <a href="#">Eliminar</a>
3	Geografía	Preguntas sobre países, continentes, ríos y montañas.	<a href="#">Editar</a> <a href="#">Eliminar</a>
4	Deportes	Preguntas sobre deportes y atletas famosos.	<a href="#">Editar</a> <a href="#">Eliminar</a>

#### Crear Nueva Categoría

Nombre:

Descripción:

En la documentación de [swagger](#) puedes ver todos los endpoints con sus ejemplos de enviar y lo que recibes



## 8. Conclusiones e Valoración Persoal

### Dificultades encontradas y cómo se resolvieron

Uno de los principales retos fue la falta de funciones en H2 en comparación con MySQL, ya que H2 es más limitado. Esto requirió adaptar el script de la base de datos para asegurar su compatibilidad.

Otro desafío importante fue la implementación de la seguridad. La gestión de usuarios con contraseñas encriptadas fue un punto clave, especialmente al necesitar registrar un usuario para realizar pruebas. Para simplificar el proceso, Thymeleaf se dejó accesible para todos, ya que solo se usa a nivel local y no requería autenticación.

### Mejoras para futuras versiones

Implementar login en Thymeleaf para mejorar la seguridad del entorno de administración. Optimizar el uso de DTOs para mejorar la eficiencia en la transferencia de datos y reducir la exposición innecesaria de información.

### Opinión sobre el uso de Spring Boot y sus tecnologías

El desarrollo del CRUD y la API con Spring Boot fue bastante sencillo. Su estructura clara y las herramientas integradas facilitaron la creación de endpoints, la gestión de la base de datos y la implementación de la seguridad.

## 9. Anexos

### Documentación en [Swagger](#)

Toda la información sobre la estructura de los JSON, tanto para las solicitudes como para las respuestas, está documentada en Swagger. Esto facilita la integración con otros sistemas y permite a los desarrolladores comprender cómo interactuar con la API de manera sencilla.

Link: <http://localhost:8080/swagger-ui/index.html>

### Carga de la base de datos en otro sistema

Si necesitas ejecutar la base de datos en otro entorno, en la carpeta **distribución/base** de datos encontrarás:

El script SQL para generar la estructura de la base de datos.

Datos iniciales pre configurados para facilitar el uso inmediato del sistema.

### **Despliegue en un servidor**

Para desplegar la aplicación en un servidor, se recomienda seguir estos pasos:

Configurar las variables de entorno: Asegurar que las propiedades de la base de datos y el JWT están correctamente definidas en el entorno del servidor.

Generar el archivo JAR:

```
mvn clean package
```

Esto generará un archivo .jar en la carpeta target/.

Ejecutar la aplicación en el servidor:

```
java -jar target/nombre-del-proyecto.jar
```

Puedes usar Docker o un servidor como Tomcat en caso de una aplicación más compleja.

Configurar el acceso externo: Si se usa una base de datos externa, asegurarse de que las credenciales y la URL están correctamente configuradas.

Seguridad y logs: Implementar un sistema de logs para monitorear errores y tráfico. Se recomienda Spring Boot Actuator para métricas y supervisión.