**Informática 2: API REST - Harry Potter**
17/4/2023

# Índice



# 1. Introducción

El alumno implementará un API REST con Node.JS que complete la especificación indicada en el presente documento.

# 2. Base de datos

El API almacenará los datos persistentes en una base de datos MySQL o MariaDB (a elección del alumno) y se usarán las siguientes tablas:

- Películas
- Personajes

## 2.1. Colecciones

De cara a poder definir las columnas a usar en las tablas, es necesario conocer las colecciones de datos.

Destacar que si el alumno necesita almacenar datos extra, podrá hacerlo sin problema alguno, la siguiente lista de columnas es meramente orientativa:

#### 2.1.1. Película

- id - Identificador único (autoincremental)
- title - Título de la película
- length - Longitud en minutos de la película
- sinopsis - Resumen de la película
- year - Año de emisión

#### 2.1.2. Personaje

- id - Identificador único (autoincremental)
- name - Nombre del personaje
- desc - Descripción del personaje

### 2.2. pelicula_personajes

Esta tabla se usará para cruzar datos entre las dos anteriores

- idPelicula - Identificador de película
- idPersonaje - Identificador de personaje

## 3. API REST

El API debe ofrecer un CRUD [1] completo sobre las colecciones:

- /peliculas
- /personajes

Los métodos HTTP a utilizar deberán ser los adecuados para cada operación

Además, se ofrecerá la consulta en las colecciones:

- /peliculas/:idPelicula/personajes - Mostrar los personajes de la película
- /personajes/:idPersonajes/peliculas - Mostrar las películas dónde sale el personaje

[1] CRUD - Create/Creación, Read/Consulta, Update/Edición y Delete/Borrado

### 3.1. API Combate

Finalmente, implementar una serie de recursos virtuales que nos permitan simular combates entre los personajes.

Los combates serán definidos por el alumno (juego de Rol), y cada personaje almacenará una serie de atributos (magia, fuerza, poder, ...) a elección del alumno.

Cada combate aplicará un factor aleatorio además de varias iteracciones de combate.

Devolverá el estado final de cada personaje tras el combate.

- GET /combate/:idPersonaje - Obtener los atributos del personaje actuales
- POST /combate?p1=idPersonaje1&p2=idPersonaje2 - Realizar un nuevo combate entre ambos personajes
- PUT /combate/:idPersonaje - Poder cambiar atributos al personaje (curación); en el cuerpo del mensaje se enviarán los nuevos atributos

Obviamente, tras un combate, al volver a consultar el GET, obtendremos el estado del personaje tras el combate.

## 4. Autenticación y control de acceso

El API para consultas será de libre acceso, pero para añadir o modificar datos así como realizar combates, el usuario deberá estar autenticado.

A modo de guía, se recomienda:

- Crear una tabla de usuarios con login y password en la base de datos. La contraseña debería estar cifrada con un HASH (SHA1)
- Agregar una ruta al API: POST /login que reciba en el cuerpo el login y la password, valide contra la BBDD y devuelva un TOKEN (se recomienda utilizar JWT)[2]
- El resto de peticiones enviarán en la cabecera 'Authorization' el valor 'Bearer TokenRecibidoTrasElLogin', para lo que se recomienda utilizar la librería 'express-bearer-token'[3] que nos añadirá a req.token este identificador y con JWT podremos validar si es correcto y dar acceso a los elementos de edición del API

[2] `https://www.npmjs.com/package/jsonwebtoken`
[3] `https://www.npmjs.com/package/express-bearer-token`

# 5. Web de pruebas

Implementar una sencilla web para probar el API, no es necesario que sea 'bonita', simplemente que ofrezca una serie de botones y formularios para poder probar el API cómodamente.