Ejercicio: Gestión de Pokémons con Spring Boot

Descripción general

Se trata de desarrollar una aplicación web en Java utilizando Spring Boot, Spring Data JPA y MySQL. La aplicación gestiona un listado de Pokémons y ofrece distintos endpoints REST para consultar, buscar y categorizar los pokémons según sus atributos.

Requisitos funcionales

1. **Listado de Pokémons**
   - Endpoint: `GET /api/pokemons`
   - Devuelve una lista con todos los pokémons registrados.
2. **Categorizar Pokémons según puntos de vida**
   - Endpoint: `GET /api/categories`
   - La aplicación debe clasificar a los pokémons en dos categorías:
     - **"debil"**: aquellos con `hitPoints` menores a 100.
     - **"fuerte"**: aquellos con `hitPoints` mayores o iguales a 100.
   - Se retorna un `Map<String, List<Pokemon>>` con las categorías correspondientes.
3. **Búsqueda de Pokémons por prefijo**
   - Endpoint: `GET /api/pokemons/search`
   - Parámetro requerido: `prefix` (cadena de caracteres que representa el inicio del nombre).
   - Se debe devolver un objeto JSON con dos listas:
     - **normal**: pokémons que coinciden con el prefijo.
     - **buffed**: mismos pokémons, pero con un aumento del 10% en sus puntos de vida (`hitPoints`).

Requisitos técnicos

- **Entidad Pokémon**:

  La entidad debe incluir los siguientes atributos:

  - `id` (Long, autogenerado)
  - `name` (String, no nulo)
  - `type` (String, no nulo)
  - `hitPoints` (Long, no nulo)

- **Persistencia**:

  Utilizar Spring Data JPA para interactuar con la base de datos MySQL.

  Configurar la conexión a la base de datos en el archivo `application.properties` con las siguientes propiedades:

  - URL, usuario, contraseña y driver.
  - Generación automática de la estructura de la base de datos (`ddl-auto=create`).

- **Servicios y lógica de negocio**:

  - Un servicio (`PokemonService`) que se encargue de obtener todos los pokémons, buscar por prefijo y aplicar el buff (incremento del 10% en `hitPoints`).
  - Un servicio adicional (`PokemonCategoryService`) para categorizar los pokémons según sus puntos de vida.

- **Controlador REST**:

  Se debe implementar un controlador (`PokemonController`) que exponga los endpoints mencionados anteriormente y que coordine las llamadas a los servicios correspondientes.

Estructura del proyecto

El ejercicio se encuentra estructurado en varios paquetes:

- **controllers**: Contiene el controlador REST (`PokemonController.java`).
- **models**: Define la entidad `Pokemon` (`Pokemon.java`).

- **repositories**: Interfaz `PokemonRepository` para las operaciones de acceso a datos.
- **services**:
  - Interfaces `PokemonService` y `PokemonCategoryService`.
  - Implementaciones `PokemonServiceImpl` y `PokemonCategoryServiceImpl`.
- **configuración**: Archivo `application.properties` para la conexión a la base de datos y otras configuraciones de Spring Boot.
- **Clase principal**: `MvcCompletoApplication.java`, que arranca la aplicación.

Consideraciones adicionales

- **Buff de hitPoints**:

  Para el endpoint de búsqueda, se debe calcular el buffado de los puntos de vida multiplicando el valor original por 1.1 (redondeando o usando conversión a long según convenga).

- **Validación**:

  Asegurarse de que el parámetro `prefix` en el endpoint de búsqueda no sea nulo o vacío, implementando las validaciones necesarias.

- **Persistencia de datos**:

  Se introducirán estas líneas en `import.sql`

```
INSERT INTO pokemons (name, type, hit_points) VALUES ('Pikachu', 'Eléctrico', 90);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Charizard', 'Fuego', 150);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Bulbasaur', 'Planta', 80);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Squirtle', 'Agua', 70);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Gyarados', 'Agua', 200);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Jigglypuff', 'Normal', 120);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Alakazam', 'Psíquico', 110);
```

```
INSERT INTO pokemons (name, type, hit_points) VALUES ('Gengar', 'Fantasma', 130);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Machamp', 'Lucha', 160);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Snorlax', 'Normal', 300);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Dragonite', 'Dragón', 250);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Arcanine', 'Fuego', 180);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Lapras', 'Agua', 210);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Electabuzz', 'Eléctrico', 140);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Scyther', 'Bicho', 130);
INSERT INTO pokemons (name, type, hit_points) VALUES ('Metagross', 'Acero', 220);
```