Backend Spring Boot: Gestión de Equipos, Personas y Estadios

1. Introducción

Este documento describe la arquitectura y el diseño del backend Spring Boot para la gestión de equipos, personas y estadios. Proporciona información detallada sobre las entidades, sus relaciones, los controladores y los endpoints REST expuestos. El objetivo es ofrecer una guía completa para desarrolladores que deseen entender, mantener o ampliar este backend.

2. Arquitectura

El backend sigue una arquitectura RESTful basada en Spring Boot. Las entidades JPA gestionan la persistencia de datos y los controladores REST manejan las peticiones HTTP. Actualmente no se definen los servicios intermedios ni los repositories (DAO).

3. Entidades

3.1. Entidad Estadio

Representa un estadio donde juegan los equipos.

Tabla de Atributos:

Atributo	Tipo	Nullable	Descripción
---	---	---	---

Atributo	Tipo	Nullable	Descripción
id	Integer		Identificador único del estadio. Es necesario configurar GenerationType.IDENTI en el código.
nombre	String	false	Nombre del estadio.
equipo	Equipo		Relación Many-to-One con la entidad Equipo.

Relaciones:

- **Many-to-One:** Relación con la entidad `Equipo`. Un estadio pertenece a un equipo.

Código Java:

```java
import jakarta.persistence.*;
import java.io.Serializable;
import back.entity.equipo;

@Entity
@Table(name = "estadio")
public class estadio implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id // Falta la anotación @Id
    @Column
    private Integer id;
    @Column(nullable = false)
    private String nombre;
    @ManyToOne
    @JoinColumn(name = "equipo_id")
    private equipo equipo;

    public estadio() {}
```

```
    public Integer getId() { return this.id; }
    public void setId(Integer id) { this.id = id; }

    public String getNombre() { return this.nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public equipo getEquipo() { return equipo; }
    public void setEquipo(equipo equipo) { this.equipo = equipo; }
}
```

Nota: Se añadió la anotación `@Id` faltante en el campo `id` y los getters/setters para el campo equipo. Además, es importante agregar `@GeneratedValue(strategy = GenerationType.IDENTITY)` al campo `id` si se desea que la base de datos gestione su generación.

3.2. Entidad Persona

Representa una persona (entrenador, jugador, etc.) relacionada con un equipo.

Tabla de Atributos:

Atributo	Tipo	Nullable	Descripción
---	---	---	---
ci	String		Cédula de identidad (identificador único).
nombre	String	false	Nombre de la persona.
equipoList	List\<Equipo\>		Relación One-to-Many con la entidad Equipo.

Relaciones:

- **One-to-Many:** Relación con la entidad `Equipo`. Una persona puede estar asociada a varios equipos.

Código Java:

```java
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.List;

@Entity
@Table(name = "persona")
public class persona implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @Column
    private String ci;
    @Column(nullable = false)
    private String nombre;
    @OneToMany(mappedBy = "persona")
    private List<equipo> equipoList;

    public persona() {}

    public String getCi() { return this.ci; }
    public void setCi(String ci) { this.ci = ci; }

    public String getNombre() { return this.nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public List<equipo> getEquipoList() { return equipoList; }
    public void setEquipoList(List<equipo> equipoList) { this.equipoList = equipoList; }
}
```

Nota: Se añadió el getter/setter para el campo `equipoList`.

3.3. Entidad Equipo

Representa un equipo deportivo.

Tabla de Atributos:

Atributo	Tipo	Nullable	Descripción
---	---	---	---
id	Integer		Identificador único del equipo.
nombre	String	false	Nombre del equipo.
persona	Persona		Relación Many-to-One con la entidad Persona.
estadioList	List\<Estadio\>		Relación One-to-Many con la entidad Estadio.

Relaciones:

- **Many-to-One:** Relación con la entidad `Persona`. Un equipo está asociado a una persona.
- **One-to-Many:** Relación con la entidad `Estadio`. Un equipo puede tener varios estadios.

Código Java:

```java
import jakarta.persistence.*;
import java.io.Serializable;
import java.util.List;

@Entity
@Table(name = "equipo")
```

```
public class equipo implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column
    private Integer id;
    @Column(nullable = false)
    private String nombre;
    @ManyToOne
    @JoinColumn(name = "persona_id")
    private persona persona;
    @OneToMany(mappedBy = "equipo")
    private List<estadio> estadioList;

    public equipo() {}

    public Integer getId() { return this.id; }
    public void setId(Integer id) { this.id = id; }

    public String getNombre() { return this.nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }

    public persona getPersona() { return persona; }
    public void setPersona(persona persona) { this.persona = persona; }

    public List<estadio> getEstadioList() { return estadioList; }
    public void setEstadioList(List<estadio> estadioList) { this.estadioList = estadioList; }
}
```

Nota: Se añadieron los getters/setters para los campos `persona` y `estadioList`.

4. Controladores REST

Los controladores REST exponen los endpoints para interactuar con las entidades.

Nota: Los controladores mostrados aquí solo contienen el esqueleto. Es necesario implementar la lógica CRUD (Crear, Leer, Actualizar, Borrar) utilizando los repositorios de Spring Data JPA.

4.1. EstadioController

```java
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/estadio")
public class estadioController {
    // Métodos CRUD pueden ser implementados aquí
}
```

Endpoints REST:

Método	Endpoint	Descripción
GET	/api/estadio	Obtiene todos los estadios.
GET	/api/estadio/{id}	Obtiene un estadio por su ID.
POST	/api/estadio	Crea un nuevo estadio.
PUT	/api/estadio/{id}	Actualiza un estadio existente.
DELETE	/api/estadio/{id}	Elimina un estadio.

Ejemplo: Crear un estadio (POST /api/estadio)

Request (JSON):

```json
{
  "nombre": "Estadio Ejemplo",
```

```
  "equipo_id": 1
}
```

Request (Bash):

```
[BASH]
curl -X POST -H "Content-Type: application/json" -d '{"nombre": "Estadio Ejemplo", "equipo_id": 1}'
http://localhost:8080/api/estadio
```

Response (JSON):

```
[JSON]
{
  "id": 1,
  "nombre": "Estadio Ejemplo",
  "equipo_id": 1
}
```

4.2. PersonaController

```
[JAVA]
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/persona")
public class personaController {
    // Métodos CRUD pueden ser implementados aquí
}
```

Endpoints REST:

Método	Endpoint	Descripción
GET	/api/persona	Obtiene todas las personas.
GET	/api/persona/{ci}	Obtiene una persona por su CI.
POST	/api/persona	Crea una nueva persona.
PUT	/api/persona/{ci}	Actualiza una persona existente.
DELETE	/api/persona/{ci}	Elimina una persona.

Ejemplo: Crear una persona (POST /api/persona)

Request (JSON):

```
[JSON]
{
  "ci": "12345678",
  "nombre": "Juan Perez"
}
```

Request (Bash):

```
[BASH]
curl -X POST -H "Content-Type: application/json" -d '{"ci": "12345678", "nombre": "Juan Perez"}' http://localhost:8080/api/persona
```

Response (JSON):

```[JSON]
{
  "ci": "12345678",
  "nombre": "Juan Perez"
}
```

4.3. EquipoController

```[JAVA]
import org.springframework.web.bind.annotation.*;
import java.util.List;
@RestController
@RequestMapping("/api/equipo")
public class equipoController {
    // Métodos CRUD pueden ser implementados aquí
}
```

Endpoints REST:

Método	Endpoint	Descripción
GET	/api/equipo	Obtiene todos los equipos.
GET	/api/equipo/{id}	Obtiene un equipo por su ID.
POST	/api/equipo	Crea un nuevo equipo.
PUT	/api/equipo/{id}	Actualiza un equipo existente.
DELETE	/api/equipo/{id}	Elimina un equipo.

Ejemplo: Crear un equipo (POST /api/equipo)

Request (JSON):

```
[JSON]
{
  "nombre": "Equipo Ejemplo",
  "persona_id": "12345678"
}
```

Request (Bash):

```
[BASH]
curl -X POST -H "Content-Type: application/json" -d '{"nombre": "Equipo Ejemplo", "persona_id": "12345678"}' http://localhost:8080/api/equipo
```

Response (JSON):

```
[JSON]
{
  "id": 1,
  "nombre": "Equipo Ejemplo",
  "persona_id": "12345678"
}
```

5. Próximos pasos

- **Implementar Repositorios (DAO):** Utilizar Spring Data JPA para crear repositorios para cada entidad. Esto simplificará el acceso a la base de datos y la implementación de la lógica CRUD.
- **Implementar Servicios:** Crear capas de servicio entre los controladores y los repositorios para desacoplar la lógica de negocio del manejo de peticiones HTTP.

- **Implementar Validación:** Agregar validaciones a las entidades utilizando anotaciones de JSR-303 (Bean Validation) para asegurar la integridad de los datos.
- **Manejo de Excepciones:** Implementar un manejo de excepciones global para retornar respuestas HTTP apropiadas en caso de errores.
- **Paginación y Ordenamiento:** Agregar soporte para paginación y ordenamiento en los endpoints que retornan listas de entidades.
- **Seguridad:** Implementar seguridad utilizando Spring Security para proteger los endpoints REST.
- **Tests:** Escribir tests unitarios y de integración para asegurar la calidad del código.
- **Documentación con Swagger:** Integrar Swagger para generar documentación interactiva de la API REST.
- **Configuración:** Externalizar la configuración (especialmente la información de la base de datos) a un archivo `application.properties` o `application.yml`.

6. Consideraciones Adicionales

- **Manejo de Relaciones Bidireccionales:** Si se requiere navegar las relaciones en ambas direcciones, se deben agregar anotaciones como `@JsonManagedReference` y `@JsonBackReference` para evitar bucles infinitos en la serialización JSON.
- **Lazy Loading vs Eager Loading:** Considerar el uso de `FetchType.LAZY` o `FetchType.EAGER` en las relaciones para optimizar el rendimiento y evitar problemas de `LazyInitializationException`.
- **Transacciones:** Utilizar la anotación `@Transactional` para asegurar la consistencia de los datos en operaciones que involucran múltiples entidades.