Universidad del Cauca
Laboratorio de Ingeniería de Software II
Brayan Daniel Perdomo, Libardo Pantoja

Taller: Mapeo de Entidades con JPA en Spring Boot

Objetivo

- Mapear correctamente un conjunto de clases en Java utilizando JPA en Spring Boot, respetando las relaciones dadas en el diagrama de clases e incluyendo herencia entre entidades.

Descripción del Taller

1. Crear un proyecto Spring Boot con Spring Data JPA y una base de datos H2 (también se puede usar MariaDB, MySQL o PostgreSQL).
2. Implementar las entidades del proyecto de curso más importantes y definir las relaciones usando anotaciones de JPA, incluyendo herencia. Tener en cuenta al menos las siguientes relaciones (pueden haber más):
    a. Un trabajo de grado está asociado a uno o dos estudiantes.
    b. Un trabajo de grado tiene un director. Un director puede tener varios trabajos de grado.
    c. Un trabajo de grado tiene uno o más co-directores.
    d. Un trabajo de grado tiene asociado un tipo (investigación, práctica profesional, plan co-terminal). Un tipo tiene asociado varios trabajos de grado.
    e. etc.
3. Implementar un repositorio para cada entidad usando JpaRepository.
4. Crear una clase *CommandLineRunner* para poblar la base de datos con datos de prueba.
5. Probar la persistencia y relaciones mediante consultas GET usando Spring Data JPA.
6. Probar distintas maneras de generación de mapeo de entidades a tablas: create, update, create-drop. Ejemplo, en el archivo **application.properties**: spring.jpa.hibernate.ddl-auto=**update**

Se puede ver la base de datos H2 creada, con los siguientes pasos:

Abrir la Consola Web de H2

1. **Ejecuta** tu aplicación Spring Boot.
2. **Abre tu navegador** y ve a:

```
None
http://localhost:8080/h2-console
```

3. **Configura la conexión**:
   - **JDBC URL:** `jdbc:h2:mem:testdb` (según la configuración que definiste).
   - **User:** `sa`
   - **Password:** (déjalo vacío si no definiste una).
4. **Haz clic en "Connect"** y podrás ver la base de datos, ejecutar consultas SQL y explorar las tablas.

Desde IntelliJ también se puede explorar la base de datos abriendo la pestaña “Database”, clic en “+” > “Data Source” >”H2”.

A continuación, algunas pistas de la implementación.

```
//Persona.java (Superclase)
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public abstract class Persona {
	@Id
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private Long id;

	private String nombres;
	private String apellidos;
	//…
}
```

```
//Estudiante.java
@Entity
public class Estudiante extends Persona {
	private String codigoEstudiante;

	@OneToMany(mappedBy = "estudiante")
	private List<TrabajoGrado> trabajos ;
	// Getters y setters
}
```

```
//Profesor.java
@Entity
public class Profesor extends Persona {
	private Enum departamento;

	@OneToMany(mappedBy = "profesor")
	private List<TrabajoGrado> trabajos;
	// Getters y setters
}
```

```
//Repositorios
public interface PersonaRepository extends JpaRepository<Persona, Long> {}
public interface EstudianteRepository extends JpaRepository<Estudiante, Long> {}
public interface ProfesorRepository extends JpaRepository<Profesor, Long> {}
public interface TrabajoGrado extends JpaRepository<TrabajoGrado, Long> {}
```

```
//DataLoader.java
@Component
public class DataLoader implements CommandLineRunner {
@Autowired
private EstudianteRepository estudianteRepository;
@Autowired
private ProfesorRepository profesorRepository;
@Autowired
private TrabajoGrado trabajoGradoRepository;
@Autowired


public void run(String... args) {
      Estudiante estudiante = new Estudiante();
      estudiante.setNombres("Juan Carlos");
      estudiante.setApellidos("Cardenas Muñoz");
      estudiante.setCodigoEstudiante("11736344");
      //…

      Profesor profesor = new Profesor();
      profesor.setNombres("Julio Ariel");
      profesor.setApellidos("Hurtado Alegria");
      //…

      TrabajoGrado trabajo = new TrabajoGrado();
      trabajo.setTitulo("Modelo de medición de habilidades en arquitectos de..");
      trabajo.setEstudiante1(estudiante);
      trabajo.setEstudiante2(null);
      trabajo.setDirector(profesor);
      //…

      estudianteRepository.save(estudiante);
      profesorRepository.save(profesor);
      trabajoRepository.save(profesor);

}
}
```

Para verificar que el mapeo funciona, puedes crear un endpoint en un RestController para consultar los datos:

```
@RestController
@RequestMapping("/estudiantes")
public class EstudianteController{

	@Autowired
	private EstudianteRepository estudianteRepository;

	@GetMapping
	public List<Estudiante> getAllEstudiantes() {
		return estudianteRepository.findAll();
	}
}
```

Lanzar la aplicación y acceder a http://localhost:8080/estudiantes debería devolver los datos insertados.

Nota: Si las respuestas JSON al invocar desde Postman los endpoints, dan “bucles infinitos”, tienen las siguientes opciones para corregir este problema:

1. No usar relaciones bidireccionales, sino relaciones en una sola dirección.
2. Si necesitas mantener la relación bidireccional, pero no quieres que se serialice en JSON, usa @JsonIgnore en uno de los lados.
3. Otra solución es usar las anotaciones @JsonManagedReference y @JsonBackReference, para indicar qué lado de la relación debe serializarse.
4. Si quieres más control sobre la respuesta JSON, una mejor práctica es usar un DTO (Data Transfer Object) en lugar de devolver la entidad directamente.

Rúbrica de evaluación del taller

Criterios de Evaluación:

- Correctamente mapeadas las entidades con JPA
- Correctamente mapeadas las relaciones JPA.

- Correctamente inicializados datos en el DataLoader

Rangos de evaluación:

- **Excelente (5.0):** el taller cumple con todos los criterios de evaluación.
- **Bueno (4.0):** el taller tiene algunos criterios por mejorar.
- **Regular (3.0):** el taller tiene importantes criterios por mejorar.
- **Deficiente (1 o 2):** el taller no cumple con los criterios de evaluación.

El proyecto se debe trabajar en los equipos definidos y se deberá ser sustentado al docente, la próxima clase. Se elige al azar la persona de cada equipo.

¡Exitos!