Entrega 1: Entidades JPA: Asset Management

1. Estructura General del Proyecto

En esta primera entrega tenemos una aplicación Spring Boot para gestión de activos y categorías. Utiliza Maven como herramienta de gestión de dependencias y ciclo de vida del proyecto, JPA para el mapeo objeto-relacional, y H2 como base de datos en memoria para desarrollo.

2. Maven y su integración con Spring

2.1 ¿Qué es Maven y cómo se relaciona con Spring?

Maven es una herramienta de gestión de proyectos que permite automatizar el proceso de construcción, gestionar dependencias y establecer una estructura estándar para tus proyectos Java. Spring Framework se integra perfectamente con Maven, ya que todos sus módulos están disponibles como dependencias en repositorios públicos.

2.2 Los scripts de Maven Wrapper (mvnw y mvnw.cmd)

Los archivos `mvnw` (para sistemas Unix) y `mvnw.cmd` (para Windows) son parte del Maven Wrapper, una característica que permite ejecutar Maven sin necesidad de instalarlo previamente en el sistema. Los archivos incluyen:

- .mvn/wrapper/maven-wrapper.properties: Define la versión de Maven a utilizar (3.9.9).
- mvnw y mvnw.cmd: Scripts que descargan e instalan automáticamente la versión correcta de Maven si no está disponible.

Esto es extremadamente útil porque:

1. Garantiza que todos los desarrolladores usen la misma versión de Maven.
2. Elimina la necesidad de instalaciones manuales.
3. Facilita la integración continua y el despliegue.

2.3 El archivo pom.xml

El POM (Project Object Model) es el archivo de configuración fundamental de Maven. Analicémoslo en detalle:

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
```

Esto establece el esquema XML que el archivo sigue, permitiendo a Maven validar su estructura.

```
<modelVersion>4.0.0</modelVersion>
<parent>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-parent</artifactId>
<version>3.4.4</version>
<relativePath/> <!-- lookup parent from repository -->
</parent>
```

- modelVersion: Indica la versión del modelo POM que se está utilizando.
- parent: Establece que este proyecto hereda configuraciones del starter parent de Spring Boot.
    - El spring-boot-starter-parent proporciona:
        - Configuraciones de plugins predeterminadas.
        - Versiones predefinidas para dependencias comunes (gestión de dependencias).
        - Configuración de compilación predeterminada.

```
<groupId>com.miniinfomates2003</groupId>
<artifactId>asset-management</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>asset-management</name>
<description>Microservicio para la gestión de activos y categorías en el contexto de cuentas y planes.</description>
```

- groupId: Identifica de manera única la organización (usando convención de nombres de dominio inversa).
- artifactId: Nombre base del artefacto principal (JAR) que produce el proyecto.
- version: La versión actual del proyecto (0.0.1-SNAPSHOT indica una versión en desarrollo).
- name y description: Metadatos descriptivos.

```
<properties>
<java.version>17</java.version>
</properties>
```

Define propiedades utilizadas en el proyecto, específicamente la versión de Java (17).

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<scope>runtime</scope>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

La sección de dependencias lista todos los módulos externos que requieren la aplicación:

- spring-boot-starter-data-jpa: Agrupa dependencias para usar Spring Data JPA, Hibernate y JDBC.
- h2: Driver para la base de datos H2 (en memoria) con alcance `runtime` (solo necesario durante la ejecución).
- spring-boot-starter-test: Proporciona utilidades para pruebas como JUnit, Mockito, etc., solo para la fase de pruebas.

```
<build>
<plugins>
<plugin>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
</plugins>
</build>
```

La sección de construcción configura cómo Maven debe compilar y empaquetar el proyecto:

- spring-boot-maven-plugin: Permite crear un JAR ejecutable que incluye todas las dependencias necesarias.

3. El archivo application.properties

El archivo application.properties es fundamental en las aplicaciones Spring Boot, ya que contiene configuraciones que sobrescriben los valores predeterminados del framework.

```
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=sa
spring.datasource.url=jdbc:h2:mem:testdb
```

Estas propiedades configuran la conexión a la base de datos:

- driver-class-name: Especifica el driver JDBC a utilizar (H2).
- username y password: Credenciales para conectarse a la BD (aquí son simples porque es una BD en memoria).
- url: Ubicación de la base de datos (formato: jdbc:h2:mem:testdb indica una BD H2 en memoria llamada "testdb").

```
spring.jpa.properties.jakarta.persistence.schema-generation.database.action=create
spring.jpa.properties.jakarta.persistence.schema-generation.scripts.action=create
spring.jpa.properties.jakarta.persistence.schema-generation.scripts.create-target=create.sql
```

Estas propiedades controlan la generación del esquema de base de datos:

- database.action=create: Indica a JPA que cree las tablas automáticamente al iniciar la aplicación.
- scripts.action=create: Genera scripts SQL para la creación del esquema.
- scripts.create-target=create.sql: Guarda el script generado en un archivo llamado "create.sql".

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
```

Define el dialecto SQL específico que Hibernate debe usar (en este caso, el de H2).

4. La clase principal de la aplicación

```
@SpringBootApplication
public class AssetManagementApplication implements CommandLineRunner {

public static void main(String[] args) {
SpringApplication.run(AssetManagementApplication.class, args);
}

@Override
public void run(String... args) throws Exception {
System.out.println("La aplicación ha arrancado y el esquema DDL se ha generado.");
}
}
```

- `@SpringBootApplication`: Esta anotación es fundamental y combina tres anotaciones:
  - `@Configuration`: Marca la clase como fuente de definiciones de bean.
  - `@EnableAutoConfiguration`: Le dice a Spring Boot que configure automáticamente Spring basándose en las dependencias del classpath.
  - `@ComponentScan`: Le indica a Spring que busque otros componentes en el paquete actual y subpaquetes.
- `CommandLineRunner`: Es una interfaz funcional que permite ejecutar código después de que el contexto de Spring se haya iniciado completamente. Implementar esta interfaz te obliga a proporcionar el método `run()`.
- `SpringApplication.run()`: Este método estático inicia la aplicación Spring Boot, creando el contexto de aplicación y registrando todos los componentes.

5. Las entidades JPA

5.1 La entidad Activo

```
@Entity
public class Activo {
@Id
@GeneratedValue
private Integer id;
@Column(nullable = false)
private String nombre;
private String tipo;
@Column(nullable = false)
private Integer tamanio;
private String url;

private Set<Integer> idProducto;
```

```
@ManyToMany(mappedBy = "activos")
private Set<Categoria> categorias;  // Relación bidireccional

// Getters, setters, hashCode, equals, toString
}
```

- `@Entity`: Indica a JPA que esta clase representa una tabla en la base de datos.
- `@Id`: Marca el campo como clave primaria.
- `@GeneratedValue`: La clave primaria se generará automáticamente (por defecto, utilizando una secuencia o un autoincremento).
- `@Column(nullable = false)`: Especifica que este campo no puede ser nulo en la base de datos.
- `@ManyToMany(mappedBy = "activos")`: Establece una relación muchos a muchos con la entidad Categoria, donde el "propietario" de la relación es la entidad Categoria (a través de su campo "activos").

5.2 La entidad Categoria

```
@Entity
public class Categoria {
@Id
@GeneratedValue
private Integer id;
@Column(nullable = false)
private String nombre;
@ManyToMany
@JoinTable(
name = "activo_categoria",
joinColumns = @JoinColumn(name = "categoria_id"),
inverseJoinColumns = @JoinColumn(name = "activo_id")
)
private Set<Activo> activos;  // Relación con activos

// Getters, setters, hashCode, equals, toString
}
```

- `@ManyToMany`: Define una relación muchos a muchos con la entidad Activo.
- `@JoinTable`: Especifica la tabla de unión para la relación muchos a muchos:
  - `name`: Nombre de la tabla de unión (activo_categoria).
  - `joinColumns`: Columna que hace referencia a la entidad actual (categoria_id).
  - `inverseJoinColumns`: Columna que hace referencia a la entidad relacionada (activo_id).

6. Los repositorios JPA

```
public interface ActivoRepository extends JpaRepository<Activo, Integer> {
}

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

- `JpaRepository<T, ID>`: Interfaz genérica que proporciona métodos CRUD estándar y funcionalidades de paginación.
  - `T`: Tipo de entidad gestionada (Activo o Categoria).
  - `ID`: Tipo de datos de la clave primaria (Integer en ambos casos).

Al extender JpaRepository, obtienes automáticamente métodos como:

- `save(entidad)`: Para crear o actualizar entidades.
- `findById(id)`: Para buscar por identificador.
- `findAll()`: Para recuperar todas las entidades.
- `delete(entidad)`: Para eliminar entidades.
- `count()`: Para contar entidades.

7. El proceso de arranque de la aplicación

Cuando ejecutas la aplicación (a través de los wrappers de Maven o directamente como una aplicación Java), ocurre lo siguiente:

1. **Fase de compilación**:
    - Maven compila las clases Java.
    - Procesa recursos y los coloca en el classpath.
    - Ejecuta pruebas (si existen).
2. **Inicio de Spring Boot**:
    - La clase principal con la anotación `@SpringBootApplication` se invoca.
    - Spring Boot inicializa el contexto de aplicación.
    - Se descubren y registran componentes a través del escaneo de componentes.
    - Se configura la base de datos H2 en memoria según application.properties.
3. **Configuración de JPA/Hibernate**:
    - Spring Data JPA inicializa las interfaces de repositorio.
    - Hibernate escanea las entidades anotadas con `@Entity`.
    - Se genera el esquema de base de datos basado en las entidades (tablas Activo, Categoria y activo_categoria).
4. **Ejecución del CommandLineRunner**:
    - Una vez que todo está configurado, se invoca el método `run()` implementado en la clase principal.
    - Se muestra el mensaje "La aplicación ha arrancado y el esquema DDL se ha generado."

8. La magia de Spring Boot: Autoconfiguration

Una de las características más potentes de Spring Boot es su capacidad de autoconfiguración. Basándose en las dependencias que has agregado en el pom.xml, Spring Boot realiza configuraciones sensatas por defecto:

- `spring-boot-starter-data-jpa`: Detecta que quieres usar JPA, por lo que configura automáticamente:
    - Un `EntityManagerFactory` para interactuar con la base de datos a través de JPA.
    - Un `TransactionManager` para gestionar transacciones.
    - Los repositorios JPA a partir de las interfaces que extiendan `JpaRepository`.
- `h2`: Detecta la presencia del driver H2 y, al no haber configurado otra base de datos:
    - Configura una base de datos H2 en memoria.
    - Expone una consola web H2 para acceder a la base de datos (accesible en http://localhost:8080/h2-console cuando se ejecuta la aplicación).

9. El ciclo de vida de una petición en Spring

Aunque el proyecto actual no incluye controladores REST, es útil entender cómo funcionaría el ciclo completo cuando lo amplíes:

1. **Cliente → Controlador**: Una petición HTTP llega a un controlador REST anotado con `@RestController`.
2. **Controlador → Servicio**: El controlador llama a métodos en la capa de servicio (generalmente anotados con `@Service`).
3. **Servicio → Repositorio**: La capa de servicio utiliza repositorios para acceder a datos.
4. **Repositorio → Base de datos**: Los repositorios interactúan con la base de datos usando JPA/Hibernate.
5. **Repositorio → Servicio → Controlador → Cliente**: La respuesta sigue el camino inverso hasta el cliente.

Conclusión

El proyecto de Asset Management es una excelente base para aprender Spring Boot. Demuestra la integración de Maven con Spring Boot, el uso de JPA para mapeo objeto-relacional, y cómo configurar una base de datos. Si bien es simple por ahora, contiene la estructura fundamental sobre la que se puede construir una aplicación robusta y completa.