CRUD con JPA

IT BOARDING

BOOTCAMP

Índice

IT BOARDING

BOOTCAMP

// Paso a Paso para un CRUD

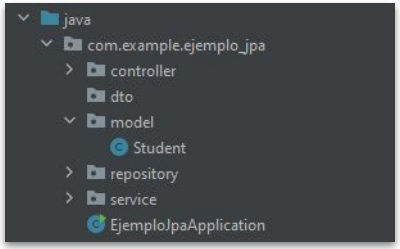
IT BOARDING

BOOTCAMP

Configuraciones para un CRUD

1) Creamos un nuevo proyecto y armamos las correspondientes capas mediante packages:

- controller
- dto
- model
- repository
- service

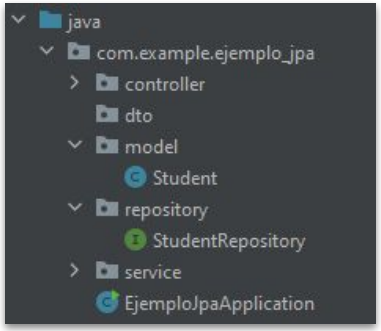


2) Generaremos una clase Student dentro de nuestra capa model y la mapeamos:

```java
import lombok.Getter;
import lombok.Setter;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Getter @Setter
@Entity
public class Student {

    @Id
    @GeneratedValue (strategy = GenerationType.SEQUENCE)
    private Long id;
    private String dni;
    private String name;
    private String lastname;

}
```

Configuraciones para un CRUD

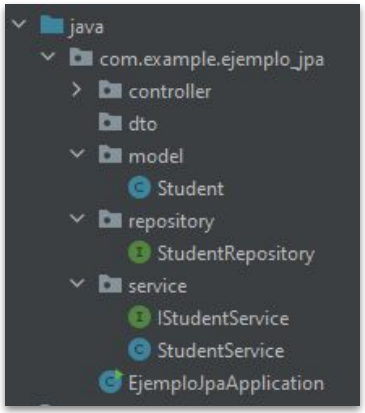
3) Configuramos nuestro repository creando la correspondiente interfaz




```java
package com.example.ejemplo_jpa.repository;

import com.example.ejemplo_jpa.model.Student;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface StudentRepository extends JpaRepository <Student, Long> {



}
```

Configuraciones para un CRUD

4) Configuramos nuestro service creando la correspondiente interfaz y su implementación

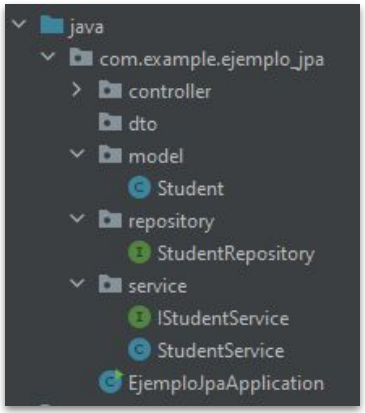


5) En la interfaz, crearemos estos 4 métodos para luego implementar el correspondiente CRUD:

```java
package com.example.ejemplo_jpa.service;

import com.example.ejemplo_jpa.model.Student;
import java.util.List;

public interface IStudentService {

    public List<Student> getStudents ();
    public void saveStudent (Student stu);
    public void deleteStudent (long id);
    public Student findStudent (long id);
```

Configuraciones para un CRUD

6) En la implementación StudentService vamos a instanciar al repositorio:




```java
@Service
public class StudentService implements IStudentService{

private final StudentRepository stuRepo;

 //Si implemento con constructor es así:
    public StudentService (StudentRepository stuRepo) {
        this.stuRepo = stuRepo;
    }
```

Configuraciones para un CRUD

7) Luego implementaremos cada uno de los métodos CRUD.

8) READ (para todos los registros) mediante el método **findAll()**

```
@Override
@Transactional (readOnly = true)
public List<Student> getStudents() {
    List<Student> studentList = stuRepo.findAll();
    return studentList;
}
```

9) READ (para un registro en particular) mediante el método **findById()**

```
@Override
@Transactional (readOnly = true)
public Student findStudent(long id) {
    //acá si no encuentro el student, devuelvo null, eso hace el orElse
    Student stu=stuRepo.findById(id).orElse( other: null);
    return stu;
}
```

Configuraciones para un CRUD

10) CREATE (dar de alta un registro) mediante el método **save()**

```
@Override
@Transactional
public void saveStudent(Student stu) {

    stuRepo.save(stu);
}
```

11) DELETE (para un registro en particular) mediante el método **deleteById()**

```
@Override
@Transactional
public void deleteStudent(long id) {
    stuRepo.deleteById(id);
}
```

// ¿Qué pasa con el UPDATE?

IT BOARDING

BOOTCAMP

¿Qué pasa con el UPDATE?

- En **Hibernate** como **proveedor de JPA** no existe un método exclusivo para editar o modificar registros (como en el caso de otros proveedores donde existen métodos edit o modify).

- El método que se utiliza en **JPA** con **Hibernate** para realizar modificaciones, es el mismo método **save()** utilizado para las altas.

- **Save** significa **Guardar**, por lo cual sirve tanto para **guardar nuevos registros** como así también para **guardar cambios** sobre los mismos.

// Implementación del CONTROLLER

IT BOARDING

BOOTCAMP

¿Cómo implementamos los end-points?

- En el controller generaremos una instancia a nuestro servicio.

```
@RestController
public class Controller {

    //implemento la interface
    @Autowired
    private IStudentService stuServ;
```

- Luego implementaremos un end-point por cada operación CRUD que queramos llevar a cabo.

CREATE

- Endpoint para crear un nuevo estudiante

```
@PostMapping ("/create")
  public String createStudent(@RequestBody Student student) {
    stuServ.saveStudent(student);
    return "El estudiante fue agregado correctamente";
}
```

READ

- Endpoint para traer a todos los estudiantes

```
@GetMapping ("/students")
public List<Student> getStudents () {

    List<Student> studentList = stuServ.getStudents();
    return studentList;

}
```

UPDATE

- Endpoint para realizar modificaciones a los estudiantes

```
@PostMapping ("edit/{id}")
public Student editStudent (@PathVariable long id,
                            @RequestParam ("name") String newName,
                            @RequestParam ("lastname") String newLastname) {
    Student stu = stuServ.findStudent(id);
    //esto puede ir en el service
    stu.setLastname(newName);
    stu.setName(newLastname);
    stuServ.saveStudent(stu);
    return stu;
}
```

DELETE

- Endpoint para eliminar estudiantes

```
@PostMapping("delete/{id}")
public String deleteStudent (@PathVariable long id) {
    stuServ.deleteStudent(id);
    return "El estudiante fue borrado correctamente";
}
```

// Mediante estos pasos vas a poder realizar un CRUD con JPA

¡Llegó el momento que intentes replicar los pasos! ¡VAMOS!

IT BOARDING

BOOTCAMP