PERSISTENCIA DE DATOS

CAPA SERVICE Y CAPA REPOSITORY

En este documento vamos a configurar nuestro proyecto Spring Boot para trabajar con JPA. Para esto necesitamos las dependencias de Spring Boot, Starter Web y Starter Data JPA. Como hemos definido al comienzo de nuestro proyecto la primera dependencia, solo nos queda buscar starter data jpa en la página mvnrepository.com y agregarla a nuestro pom.xml.

Para este ejemplo se utilizó la siguiente:

```
<!-- https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa -->
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-data-jpa</artifactId>
		</dependency>
```

Spring Data JPA

Para comprender un poco de porqué necesitamos el uso de este módulo, tenemos que saber que Spring Data JPA nos facilita el acceso a la capa de persistencia de datos y a los diferentes tipos de almacenes de persistencia, ya sea en sistemas de BD relacionales y BD no relacionales. Nos reduce una gran cantidad de código repetitivo usando JPA, por lo que las entidades(clases), las asociaciones entre ellas, las consultas y las funciones de almacenamiento(CRUD) se nos hace muy fácil a la hora de implementar.

CONFIGURACIÓN PARA LA CONEXIÓN A LA BASE DE DATOS

En este ejemplo se está trabajando con una base de datos MySQL, asi que vamos a agregar la dependencia mysql-connector-java, para que nuestro proyecto que está desarrollado con lenguaje Java pueda interactuar y conectarse con la base de datos Mysql.

Para este ejemplo se utilizó la siguiente:

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			 <artifactId>mysql-connector-java</artifactId>
			 <version>8.0.33</version>
		</dependency>
```

Ahora vamos a modificar nuestro archivo ***application.properties*** para agregar la configuración para la conexión con la base de datos.

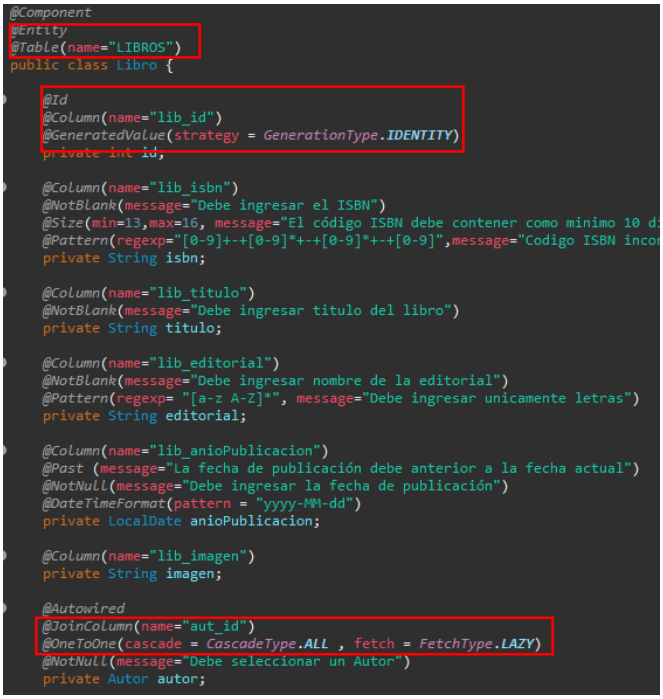
```
#Datos de conexión con la base de datos Mysql
spring.datasource.url = jdbc:mysql://localhost:3306/libro_autor?serverTimezone=UTC

spring.datasource.username = root
spring.datasource.password = root

#habilita el registro de declaraciones SQL
spring.jpa.show-sql = true
#definicion de mysql dialect
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
#iniciar el esquema
spring.jpa.generate-ddl = true
#configuracion para el acceso a la base de datos
spring.jpa.hibernate.ddl-auto =create

#spring.jpa.hibernate.ddl-auto= update
```

Lo siguiente será modificar nuestras clases, agregando las anotaciones como @Entity, @Column, @Table, @JoinColumn, @GeneratedValue, entre otras.

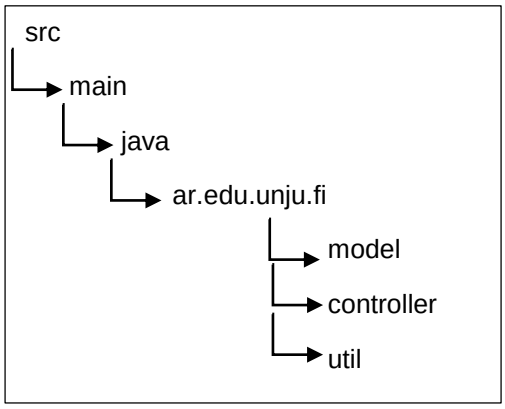
```
import jakarta.persistence.CascadeType;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.FetchType;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.OneToOne;
import jakarta.persistence.Table;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;
import jakarta.validation.constraints.Past;
import jakarta.validation.constraints.Pattern;
import jakarta.validation.constraints.Size;
```

```
@Component
@Entity
@Table(name="LIBROS")
public class Libro {

	@Id
	@Column(name="lib_id")
	@GeneratedValue(strategy = GenerationType.IDENTITY)
	private int id;

	@Column(name="lib_isbn")
	@NotBlank(message="Debe ingresar el ISBN")
	@Size(min=13,max=16, message="El código ISBN debe contener como minimo 10 digitos
y como maximo 13 digitos")
	@Pattern(regexp="[0-9]+-+[0-9]*+-+[0-9]*+-+[0-9]",message="Codigo ISBN incorrecto.
(Ej: 0-7645-2641-3)")
	private String isbn;

	@Column(name="lib_titulo")
	@NotBlank(message="Debe ingresar titulo del libro")
	private String titulo;
```

```java
@Column(name="lib_editorial")
@NotBlank(message="Debe ingresar nombre de la editorial")
@Pattern(regexp= "[a-z A-Z]*", message="Debe ingresar unicamente letras")
private String editorial;

@Column(name="lib_anioPublicacion")
@Past (message="La fecha de publicación debe anterior a la fecha actual")
@NotNull(message="Debe ingresar la fecha de publicación")
@DateTimeFormat(pattern = "yyyy-MM-dd")
private LocalDate anioPublicacion;

@Column(name="lib_imagen")
private String imagen;

@Autowired
@JoinColumn(name="aut_id")
@OneToOne(cascade = CascadeType.ALL , fetch = FetchType.LAZY)
@NotNull(message="Debe seleccionar un Autor")
private Autor autor;
```

ANOTACIONES EN LAS ENTIDADES	
	***@Entity:*** esta anotación define una clase como una entidad. ***@Table():***nos permite indicarle a JPA contra que tabla debe mapear una entidad al momento de persistir. ***@Id:*** indica cual campo será la clave principal de la entidad actual. ***@Column:*** esta anotación se utiliza para agregar nombre a la columna que representa un campo en una tabla de la base de datos. Otros atributos que puede incluir *@Column* aparte de *name*, son *nulleable* y *length*. ***@GeneratedValue:*** esta anotación la utilizamos para que las claves primarias de cada tabla sean autoincrementales. Utilizamos *GenerationType.IDENTITY* para que genere en la base de datos un incremento automático en cada operación de alta de la columna "*lib_id*". Esto es muy eficiente para general un valor para la clave principal de la entidad. ***@JoinColumn:*** esta anotación marca una columna como columna de unión entre entidades. Esta anotación junto a ***@OneToOne*** nos indica que esta columna determinada en la entidad principal "*Libro*" hace referencia a la clave principal de la entidad secundaria "*autor*". La propiedad *fetch*: nos permite determinar el tipo de carga que vamos a realizar con los datos. *LAZY* hace la carga de datos cuando se solicita traer de la base de datos, en cambio *EAGER* trae los datos de la entidad relacionada sin necesidad de solicitud.

Si recordamos, la estructura de nuestro proyecto se veía de la siguiente forma



El cual utilizábamos la carpeta útil para simular la persistencia de datos mediante el manejo de listas, e incluso toda la lógica de negocio se encontraba ahí, todas las funciones principales de nuestra aplicación. Pero ahora vamos a completar nuestro proyecto con dos capas más que son, la capa de servicio y la capa de repositorio.

La capa de servicio se encargará de aplicar la lógica de negocio, mientras que la capa de repositorio contendrá las interfaces que se extienden de JPA, y permitirán la conexión de las clases a la base de datos. Esta última capa gestiona la información solicitada a la base de datos.

CAPA DE REPOSITORIO

Para esto creamos una interface para cada clase determinada en nuestro proyecto en el paquete ar.edu.unju.fi.repository.

Luego a la interfaz la extendemos de la interfaz *CrudRepository*.

CrudRepository es un repositorio que nos ofrece Spring Data JPA, el cual maneja dos datos, el primero es la entidad(clase) que utilizará esta interfaz y segundo es el tipo de dato del Id de la entidad. Es por esto que no implementamos nada más que la extensión ya que Spring se encarga de implementar la interfaz con la entidad determinada y podemos realizar todas las operaciones de CRUD que vienen predefinida en ella.

Las operaciones que nos provee *CrudRepository* son:

save: guarda una entidad.

saveAll: guarda las entidades de una lista iterable.

findById: busca por el identificador(Id) que determinamos de la entidad.

existsById: verifica si existe un elemento con un determinado identificador(Id).

findAll: devuelve todos los elementos de una determinada entidad.

findAllById: busca todos los elementos que tengan el identificador(Id).

count: devuelve el total de registros de una entidad.

deleteById: elimina un registro de acuerdo al identificador(Id).

delete: elimina la entidad.

deleteAllById: elimina todos los elementos que correspondan con el id.

deleteAll(Iterable): elimina todos los elementos que se reciban en el parámetro

deleteAll(): elimina todos los elementos.

```
package ar.edu.unju.fi.repository;

import org.springframework.data.repository.CrudRepository;

import ar.edu.unju.fi.model.Autor;

public interface IAutorRepository extends CrudRepository<Autor,Integer>{

}
```

```
package ar.edu.unju.fi.repository;

import org.springframework.data.repository.CrudRepository;

import ar.edu.unju.fi.model.Libro;

public interface ILibroRepository extends CrudRepository<Libro,Integer>{

}
```

CAPA DE SERVICIO

INTERFACES

Ahora pasamos a crear las interfaces de servicio que nos permitirán determinar de forma clara las tareas de negocio que se podrá realizar con cada entidad.

Tanto para la interfaz de Libro como de Autor vamos a definir los métodos abstractos necesarios para realizar alta y modificación.

```java
package ar.edu.unju.fi.service;

import java.util.List;

import ar.edu.unju.fi.model.Libro;

public interface ILibroService {

	public void addLibro(Libro libro);

	public List<Libro> getAllLibros();

	public Libro findLibroById(int id);

}
```

```java
package ar.edu.unju.fi.service;

import java.util.List;

import ar.edu.unju.fi.model.Autor;

public interface IAutorService {

	public void addAutor(Autor autor);

	public List<Autor> getAllAutores();

	public Autor findAutorById(int id);


}
```

IMPLEMENTACIÓN DE SERVICIO

Continuando con servicios pasamos a implementar el código para describir la lógica del comportamiento de los métodos definidos en las interfaces.

```java
package ar.edu.unju.fi.service.imp;

import java.util.List;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ar.edu.unju.fi.model.Autor;
import ar.edu.unju.fi.repository.IAutorRepository;
import ar.edu.unju.fi.service.IAutorService;
@Service("autorServiceImp")
public class AutorServiceImp implements IAutorService {

	@Autowired
	private IAutorRepository autorRepository;

	@Autowired
	private Autor autor;
```

```
	@Override
	public void addAutor(Autor autor) {
		autorRepository.save(autor);
	}

	@Override
	public List<Autor> getAllAutores() {
	List<Autor> autores = (List<Autor>) autorRepository.findAll();
		return autores;
	}

	@Override
	public Autor findAutorById(int id) {
	      autor = autorRepository.findById(id).get();
	      return autor;
	}


}
```

```
package ar.edu.unju.fi.service.imp;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import ar.edu.unju.fi.model.Libro;
import ar.edu.unju.fi.repository.ILibroRepository;
import ar.edu.unju.fi.service.ILibroService;
@Service("libroServiceImp")
public class LibroServiceImp implements ILibroService {

	@Autowired
	private ILibroRepository libroRepository;

	@Autowired
	private Libro libro;
	@Override
	public void addLibro(Libro libro) {
	     libroRepository.save(libro);
	}

	@Override
	public List<Libro> getAllLibros() {
	   List<Libro> libros= (List<Libro>) libroRepository.findAll();
	   return libros;
	}

	@Override
	public Libro findLibroById(int id) {
	    libro= libroRepository.findById(id).get();
	    return libro;
	}


}
```

```java
package ar.edu.unju.fi.service.imp;

import java.util.List;

@Service("libroServiceImp")
public class LibroServiceImp implements ILibroService {

    @Autowired
    private ILibroRepository libroRepository;

    @Autowired
    private Libro libro;
    @Override
    public void addLibro(Libro libro) {
        libroRepository.save(libro);
    }

    @Override
    public List<Libro> getAllLibros() {
        List<Libro> libros= (List<Libro>) libroRepository.findAll();
        return libros;
    }

    @Override
    public Libro findLibroById(int id) {
        libro= libroRepository.findById(id).get();
        return libro;
    }
```

Inyectamos la interfaz del repositorio para poder realizar las operaciones de persistencia.
El método *addLibro()* nos permitirá tanto para agregar una nueva instancia como para actualizar una instancia, ya que el método *save()* de CrudRepository tiene la capacidad de persistir incluso si conocemos el identificador de la entidad.
Para el método *getAllLibros*() utilizamos el método *findAll()* de CrudRepository y lo casteamos a un objeto de tipo *List*, ya que por defecto devuelve un Iterable.
Como se puede ver, los métodos utilizados, como *findById(),* son métodos con nombres identificables que nos permite determinar claramente que acciones realizan.

MODIFICACIONES A LA CAPA CONTROLADOR

Finalmente vamos a hacer cambios en los controladores de una aplicación, ya que ahora el controlador utilizará los servicios que brinda la capa service que definimos anteriormente, y ya no tendrá que trabajar con las listas definidas en el paquete útil.

AutorController

```java
@Controller
@RequestMapping("/autores")
public class AutorController {

        @Autowired
        private Autor unAutor;

        @Autowired
        private AutorServiceImp autorServiceImp;

        @GetMapping("/lista")
        public ModelAndView getPageAutores() {
                ModelAndView mav= new ModelAndView("listaAutores");
                mav.addObject("autores", autorServiceImp.getAllAutores());
                return mav;
        }

        @GetMapping("/formulario")
        public ModelAndView getPageFormAutor() {
                unAutor= new Autor();
                ModelAndView mav= new ModelAndView("nuevoAutor");
                mav.addObject("unAutor", unAutor);
                return mav;
        }
```

```
        @PostMapping("/guardar")
        public ModelAndView postPageSaveAutor (@Valid  @ModelAttribute("unAutor") Autor
        autor,   BindingResult result) {
                ModelAndView mav;
                if(result.hasErrors()) {
                        mav= new ModelAndView("nuevoAutor");
                }else {
                    //ListaAutores.addAutor(autor);
                    autorServiceImp.addAutor(autor);
                    mav= new ModelAndView("listaAutores");
                    mav.addObject("autores", autorServiceImp.getAllAutores());
                }
                return mav;
        }

        @GetMapping("/modificar/{id}")
        public ModelAndView getPageEditAutor(@PathVariable("id") int id) {
                unAutor= autorServiceImp.findAutorById(id);
                ModelAndView mav= new ModelAndView("nuevoAutor");
                mav.addObject("unAutor", unAutor);
                return mav;
        }


}
```

LibroController

```
@Controller
@RequestMapping("/libros")
public class LibroController {

        @Autowired
        private Libro unLibro;

        @Autowired
        private Autor unAutor;

        @Autowired
        private UploadFile uploadFile;

        @Autowired
        private LibroServiceImp libroServiceImp;

        @Autowired
        private AutorServiceImp autorServiceImp;

        @GetMapping("/lista")
        public ModelAndView getPageLibros() {
                ModelAndView mav= new ModelAndView("listaLibros");
                mav.addObject("libros", libroServiceImp.getAllLibros());
                return mav;
        }

        @GetMapping("/formulario")
        public ModelAndView getPageFormLibro() {
                unLibro= new Libro();
                ModelAndView mav= new ModelAndView("nuevoLibro");
                mav.addObject("unLibro", unLibro);
                mav.addObject("autores",autorServiceImp.getAllAutores());
                return mav;
        }
```

```
        @PostMapping("/guardar")
        public ModelAndView postPageSaveLibro(@Valid @ModelAttribute("unLibro") Libro libro,
BindingResult result,
                    @RequestParam("file") MultipartFile image) throws Exception {
                 ModelAndView mav;
                if (result.hasErrors()) {
                        mav= new ModelAndView("nuevoLibro");
                        mav.addObject("autores",autorServiceImp.getAllAutores());
                }else {
                        //unAutor=ListaAutores.findAutorById(libro.getAutor().getId());
                        unAutor=autorServiceImp.findAutorById(libro.getAutor().getId());
                        libro.setAutor(unAutor);
                        if(libro.getId()>0) {
                                //unLibro= ListaLibros.findLibroById(libro.getId());
                                unLibro= libroServiceImp.findLibroById(libro.getId());
                                if (!image.isEmpty()) {
                                        if(unLibro.getImagen() != null &&
unLibro.getImagen().length() > 0)
                                                uploadFile.delete(unLibro.getImagen());
                                        String uniqueFileName = uploadFile.copy(image);
                                        libro.setImagen(uniqueFileName);
                                }else {
                                        if(unLibro.getImagen() != null)
                                                libro.setImagen(unLibro.getImagen());
                                }
                                //ListaLibros.libros.set(libro.getId()-1, libro);
                                //libroServiceImp.addLibro(libro);
                        }else {
                                if (!image.isEmpty()) {
                                        String uniqueFileName = uploadFile.copy(image);
                                        libro.setImagen(uniqueFileName);
                                }
                                //ListaLibros.addLibro(libro);

                        }
                        libroServiceImp.addLibro(libro);
                    mav= new ModelAndView("listaLibros");
                    //mav.addObject("libros", ListaLibros.getListLibros());
                    mav.addObject("libros", libroServiceImp.getAllLibros());
                }
                return mav;
        }

        @GetMapping("/modificar/{id}")
        public ModelAndView getPageEditLibro(@PathVariable("id") int id) {
                //unLibro= ListaLibros.findLibroById(id);
                unLibro= libroServiceImp.findLibroById(id);
                ModelAndView mav= new ModelAndView("nuevoLibro");
                mav.addObject("unLibro", unLibro);
                //mav.addObject("autores",ListaAutores.getListaAutores());
                mav.addObject("autores",autorServiceImp.getAllAutores());
                return mav;
        }

        @GetMapping("/detalle/{id}")
        public ModelAndView getPagePortadaLibro(@PathVariable("id") int id) {
                ModelAndView mav = new ModelAndView("detalleLibro");
                //unLibro= ListaLibros.findLibroById(id);
                unLibro= libroServiceImp.findLibroById(id);
                mav.addObject("titulo", "PORTADA DEL LIBRO "+unLibro.getTitulo());
                mav.addObject("filename",unLibro.getImagen());

                return mav;
        }
```

```
        @GetMapping("/uploads/{filename}")
        public ResponseEntity<Resource> goImage(@PathVariable String filename) {
                Resource resource = null;
                try {
                        resource = uploadFile.load(filename);
                } catch (MalformedURLException e) {
                        e.printStackTrace();
                }
                return ResponseEntity.ok()
                .header(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=\"" +
                resource.getFilename() + "\"")
                .body(resource);
        }


}
```

Cambios en los métodos de los controladores

```
@Autowired
private AutorServiceImp autorServiceImp;

@GetMapping("/lista")
public ModelAndView getPageAutores() {
    ModelAndView mav= new ModelAndView("listaAutores");
    //mav.addObject("autores", ListaAutores.getListaAutores());
    mav.addObject("autores", autorServiceImp.getAllAutores());
    return mav;
}

@GetMapping("/formulario")
public ModelAndView getPageFormAutor() {
    unAutor= new Autor();
    ModelAndView mav= new ModelAndView("nuevoAutor");
    mav.addObject("unAutor", unAutor);
    return mav;
}

@PostMapping("/guardar")
public ModelAndView postPageSaveAutor (@Valid  @ModelAttribute("unAutor") Autor autor,
       BindingResult result) {
    ModelAndView mav;
    if(result.hasErrors()) {
        mav= new ModelAndView("nuevoAutor");
    }else {
        autorServiceImp.addAutor(autor);
        mav= new ModelAndView("listaAutores");
        mav.addObject("autores", autorServiceImp.getAllAutores());
    }
    return mav;
}
```

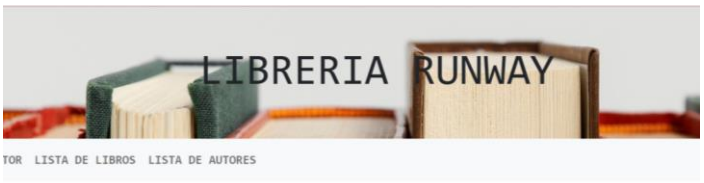
Inyectamos el service autorServiceImp.

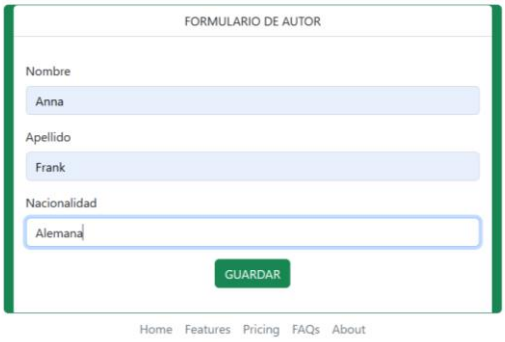
Los cambios que podemos observar son las líneas de código de los métodos que trabajan con las listas, es por esto que intercambiamos el uso de listas por nuestra clase de servicio.

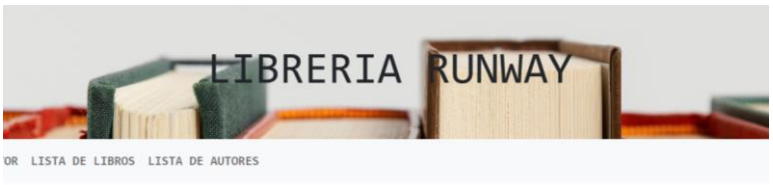
Otro cambio que se puede observar es en el método guardar. Ya que, como vimos en la capa de repositorio el método save() permite tanto agregar como actualizar un campo, y es por esto que se reduce el código en el método postPageSaveAutor().

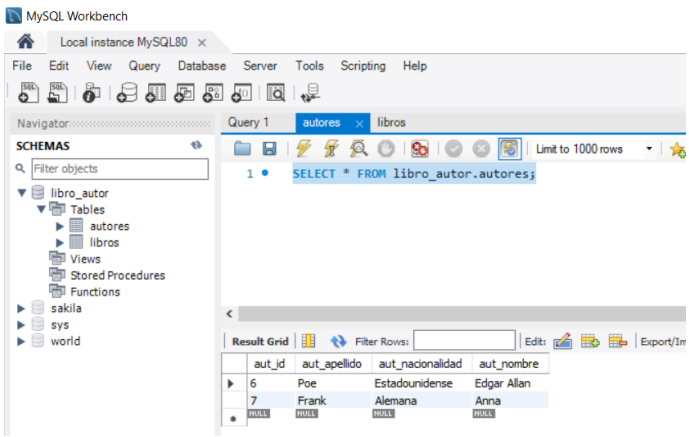
RESULTADO

Ejemplo Alta de un autor









Ejemplo Alta de un libro