

Mini Aplicació Spring Boot, Jpa i H2

Exemple Bàsic JPA CRUD. Projecte Book.	1
Què farem	1
Generar el projecte Spring Boot	1
Crear la classe Book	4
Crear la classe Repository	7
VEURE: Interface CrudRepository<T,ID> (No la creeu!!!!!!)	7
Crear el Controller	9
Creem la capa de presentació. llista.html	10
Maven	11
Annex.	12
ProjecteDamDawApplication.java	12
Controlador.java	12
Book.java	13
BookRepository.java	14
llista.html	14

Exemple Bàsic JPA CRUD. Projecte Book.

Què farem

La nostra aplicació tindrà bàsicament les següents classes:

- ProjecteDamDawApplication.java. Defineix l'aplicació SpringBoot...
- Book.java. JPAEntity. L'objecte/entitat que consultem. I que voldrem fer "persistent".
- BookRepository.java. Extendrà CRUDRepository o JPARepository. **Defineix els mètodes d'accés a la base de dades** (o a la capa de persistència).
 - els mètodes add, borrar, modifica que vam fer amb l'ArrayList ja estarien implementats a CRUDRepository. Que a més, no els inserta, llegeix o esborra d'una llista, sinó d'una BD. No els haurem de crear. Els podem fer servir directament.
 - JPA és un **ORM**. Mapeja estructures d'una BD relacional a objectes Java.
 - En aquest cas només farem el llistat de llibres findAll
- BookController.java. Exposa els mètodes que permeten connectar-se a l'aplicació a través de crides http. En aquest cas **NOMÉS** llistarem els llibres

Generar el projecte Spring Boot

Genereu el projecte Spring Boot.

- Trieu les dependències Web, Jpa, H2, Thymeleaf i Devtools. Spring s'encarregarà de buscar totes les llibreries necessàries.
- Trieu **Maven**

Spring Boot ja ens crea una classe nomProjecteApplication amb un mètode main i amb l'annotació @SpringBootApplication. Que recordem que correspon a

@Configuration,

@EnableAutoConfiguration (Spring configura l'aplicació en funció de les dependències del pom),

i @ComponentScan (Spring escanejarà el package buscant components)

Nota: Ara ja podeu executar el projecte des de la classe principal.

Veiem que ha aixecat el tomcat i el h2. I ens diu a quines adreces els podem trobar.

```

  59:23.556+02:00 INFO 1952 --- [ restartedMain] c.e.demo.ProjecteDawApplication : Starting ProjecteDawApplication using Java 17.0.8.1 with PID
  59:23.564+02:00 INFO 1952 --- [ restartedMain] c.e.demo.ProjecteDawApplication : No active profile set, falling back to 1 default profile: "default"
  59:23.642+02:00 INFO 1952 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-property-file' to
  59:24.286+02:00 INFO 1952 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
  59:24.314+02:00 INFO 1952 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 15 ms. Found 0 JPA r
  59:25.081+02:00 INFO 1952 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
  59:25.099+02:00 INFO 1952 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
  59:25.100+02:00 INFO 1952 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.13]
  59:25.188+02:00 INFO 1952 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
  59:25.190+02:00 INFO 1952 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1545 ms
  59:25.274+02:00 INFO 1952 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
  59:25.630+02:00 INFO 1952 --- [ restartedMain] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:file:~/spring
  59:25.631+02:00 INFO 1952 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
  59:25.643+02:00 INFO 1952 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jd
  59:26.063+02:00 INFO 1952 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH0000204: Processing PersistenceUnitInfo [name: default]
  59:26.180+02:00 INFO 1952 --- [ restartedMain] org.hibernate.Version : HHH0000412: Hibernate ORM core version 6.2.9.Final

```

Fixeu-vos en l'estructura del projecte creat

De fet ja podeu accedir a

<http://localhost:8080> (donarà error...pq?)

<http://localhost:8080/h2-console/> (encara no hi trobarem res)

Per començar treballarem amb la base de dades H2. Per simplificar. Després podem servir Postgres o MySQL, afegint la dependència i canviant el properties.

Revisarem el fitxer .properties:

```

spring.application.name=Spring Boot. Exemple CRUD
spring.datasource.url=jdbc:h2:file:~/spring-boot-h2-db;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

```

```

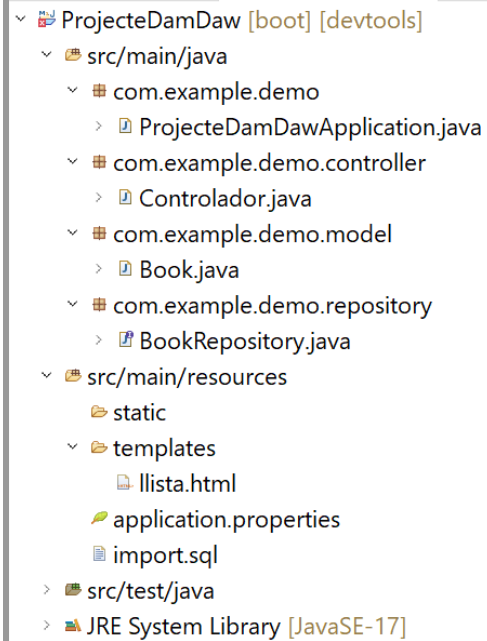
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode = always
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true

```

spring.jpa.hibernate.ddl-auto=create (la base de dades es crearà cada cop que executem l'aplicació). Si posem "update", només es modificarà quan fem canvis al model.
spring.jpa.show-sql=true Així veiem per consola le operacions que s'executen a la BD

Ara començarem a crear el codi, perquè faci alguna cosa

L'estructura del projecte serà la següent:



```
ProjecteDamDaw [boot] [devtools]
├── src/main/java
│   ├── com.example.demo
│   │   ├── ProjecteDamDawApplication.java
│   │   └── com.example.demo.controller
│   │       ├── Controlador.java
│   │       └── com.example.demo.model
│   │           ├── Book.java
│   │           └── com.example.demo.repository
│   │               └── BookRepository.java
│   └── src/main/resources
│       ├── static
│       ├── templates
│       │   ├── llista.html
│       │   ├── application.properties
│       │   └── import.sql
│       └── src/test/java
└── JRE System Library [JavaSE-17]
```

Aquesta és una estructura clàssica del MVC (Model-Vista-Controlador)

Crear la classe Book

Creeu la classe Book.java

```
@Entity
public class Book {

    @Id                                //jakarta.persistence.Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;

    @Column(nullable = false, unique = true)
    private String title;

    @Column(nullable = false)
    private String author;

    public Book() {
        super();
    }

    public Book(String title, String author) {
        super();
        this.title = title;
        this.author = author;
    }

    ...
}
```

Feu els imports necessaris

Genereu també els getters i setters!!!!!!

`@Entity`: la classe és una Entitat. En el nostre cas es transformarà en una taula a la base de dades.

`@Column (nullable ..., unique...)` camps de la taula de la base de dades

`@Id`: indica que és clau primària

`@GeneratedValue`: l'especificació de la generació de la clau primària.

`public Book()`: El constructor que espera Hibernate/Jpa.

Com també espera els **getters i setters!**

Per carregar alguns registres a la classe Book, podem crear un fitxer **/resources/import.sql** que Spring carregarà automàticament.

```
insert into book values (1, 'Harper Lee', 'Matar a un ruiseñor');  
insert into book values (2, 'Isabel Allende', 'La casa de los espíritus');
```

Si ara executem l'aplicació, ja veurem més coses

```
2023-10-16T10:13:58.688+02:00 INFO 7328 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'.  
2023-10-16T10:13:58.968+02:00 INFO 7328 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnit1  
2023-10-16T10:13:59.120+02:00 INFO 7328 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version  
2023-10-16T10:13:59.128+02:00 INFO 7328 --- [ restartedMain] org.hibernate.cfg.Environment : HHH000406: Using bytecode reflection c  
2023-10-16T10:13:59.489+02:00 INFO 7328 --- [ restartedMain] o.h.b.i.BytecodeProviderInitiator : HHH000021: Bytecode provider name : by  
2023-10-16T10:13:59.758+02:00 INFO 7328 --- [ restartedMain] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA  
2023-10-16T10:14:00.401+02:00 INFO 7328 --- [ restartedMain] o.h.b.i.BytecodeProviderInitiator : HHH000021: Bytecode provider name : by  
2023-10-16T10:14:01.591+02:00 INFO 7328 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implement  
Hibernate: drop table if exists book cascade  
Hibernate: drop sequence if exists book_seq  
Hibernate: create sequence book_seq start with 1 increment by 50  
Hibernate: create table book (id bigint not null, author varchar(255) not null, title varchar(255) not null unique, primary key (id))  
Hibernate: insert into book (id, author, title) values (1,'Harper Lee', 'Matar a un ruiseñor')  
Hibernate: insert into book (id, author, title) values (2, 'Isabel Allende', 'La casa de los espíritus')
```

Per exemple ens diu:

H2 console available at '/h2-console'. Database available at 'jdbc:h2:file:~/spring-boot-h2-db'

Per veure les taules que s'han creat a la BD i les dades que anem afegint podem fer servir la Consola H2.

<http://localhost:8080/h2-console/>

A JDBC URL poseu `jdbc:h2:file:~/spring-boot-h2-db`

The screenshot shows the H2 Console interface at localhost:8080/h2-console/. The 'Login' section is configured with 'Generic H2 (Embedded)' as the saved setting, 'Generic H2 (Embedded)' as the setting name, 'org.h2.Driver' as the driver class, 'jdbc:h2:file:~/spring-boot-h2-db' as the JDBC URL, 'sa' as the user name, and an empty password field. The 'Connect' button is highlighted.

The 'application.properties' file on the right contains the following configuration:

```
1 #logging.level.org.springframework = debug
2 spring.application.name=Spring Boot. Exemple CRUD
3
4 spring.datasource.url=jdbc:h2:file:~/spring-boot-h2-db;DB_CLOSE_ON_EXIT=FALSE
5 spring.datasource.username=sa
6 spring.datasource.password=
7 spring.datasource.driver-class-name=org.h2.Driver
8 spring.jpa.hibernate.ddl-auto=create
9
10 #spring.datasource.url= jdbc:postgresql://localhost:5432/mini
11 #spring.datasource.username=odoo1
12 #spring.datasource.password=odoo1
13 #spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.PostgreSQLDialect
14 #spring.jpa.hibernate.ddl-auto=update
```

The console output at the bottom shows the application starting successfully with the message: '2019-03-07 11:01:24.353 INFO 5040 --- [restartedMain] o.s.b.w.embedded.jetty.servlet.ServletContextHandler: Initializing Spring root WebApplicationContext'.

The H2 Console interface also shows a tree view of the database schema with 'BOOK' as the selected table. The 'SQL statement' field contains 'SELECT * FROM BOOK;'. The query result is displayed as follows:

ID	AUTHOR	TITLE
1	Harper Lee	Matar a un ruiseñor
2	Isabel Allende	La casa de los espíritus

(2 rows, 8 ms)

SpringBoot detecta que estem fent servir la base de dades interna H2 i automàticament hi genera les taules necessàries fent servir JPA/Hibernate, basant-se en les Entitats definides. (Etiquetades com a @Entity).

Per defecte, la base de dades es guarda en un fitxer a la carpeta personal de l'usuari. Com que al datasource del properties hem posat "file" ens guardarà la BD en un fitxer. Al directori d'usuari.

Busqueu-lo. Si poséssim "mem" treballariem amb una "base de dades" en memòria.

NOTA: Es creen les taules i camps associats a les entitats definides. Però amb una altra BD, la base de dades i l'usuari de connexió els heu de crear abans. Amb H2, no cal. Es genera el fitxer o la base en memòria i es fa servir un usuari predefinit.

Crear la classe Repository

```
@Repository  
public interface BookRepository extends CrudRepository<Book, Long> {  
}
```

Creeu una nova interfície que extengui CRUDRepository
Podríem extendre JpaRepository si necessitem una capa més de funcionalitats.

VEURE: Interface CrudRepository<T,ID> (No la creeu!!!!!!!)

VEURE:

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

Com que extenem CrudRepository, ja disposem dels següents mètodes:

```
public interface CrudRepository<T, ID extends Serializable>  
    extends Repository<T, ID> {  
  
    <S extends T> S save(S entity);  
  
    T findOne(ID primaryKey);  
  
    Iterable<T> findAll();  
  
    Long count();  
  
    void delete(T entity);  
  
    boolean exists(ID primaryKey);  
  
    // ...  
}
```

findAll ens retornarà tots els elements de la Taula
save(book) farà un insert o update de l'objecte book que li passem....

En aquest exemple només fem el findAll

Veure també:

[https://docs.spring.io/spring-data/jpa/docs/current/api/org.springframework.data.jpa.repository/JpaRepository.html](https://docs.spring.io/spring-data/jpa/docs/current/api/org.springframework.data.jpa.repository.JpaRepository.html)

Veureu que té molts més mètodes que CRUDRepository. Per fer accions més complicades contra la base de dades

Li indiquem a Spring on trobar l'objecte/entitat i la classe del repository. Afegim anotacions a la classe principal del projecte.

```
@EnableJpaRepositories("com.example.demo.repository")
@EntityScan("com.example.demo.model")
@SpringBootApplication
public class ProjecteDamDawApplication{
```

*Nota: Només caldria indicar-ho si no les hem posat a dins del package demo. Per defecte Spring escaneja els components que es troben a partir de la carpeta arrel.
Recordeu que Spring és Convention over Configuration*

Crear el Controller

Crearem una classe BookController.java

```
@Controller
@RequestMapping("")
public class Controlador{

    @Autowired
    private BookRepository bookRepository;
```

@Controller per gestionar una petició web.

@Autowired: "instanciem" l'objecte repositori que ens permet consultar/mantenir la BD.

Fixeu-vos que estem definint l'objecte, no fem un New. Això ho farà l'anotació

@Autowired

Crearem només el mètode corresponents al

- @GetMapping("/")
Veure tots els llibres
 - Farà servir el mètode findAll del nostre BookRepository per recuperar tots els llibres. Ni l'hem definit. L'heredem de CRUDRepository
 - Els passem a la plantilla Thymeleaf llista.html que els mostrarà

El codi complet:

```
@Controller
@RequestMapping("")
public class Controlador{

    @Autowired
    private BookRepository bookRepository;

    @GetMapping("/")
    public String buscaTots(Model model) {
        Iterable<Book> books = bookRepository.findAll();
        model.addAttribute("libros",books);
        return "llista";
    }
}
```

Creem la capa de presentació. llista.html

Creeu una pàgina html a la carpeta de templates.
Amb un thymeleaf bàsic

```
<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">
  <body>
    <h1>Llistat de llibres</h1>
    <table>
      <tr th:each="libro : ${libros}" >
        <td><span th:text="${libro.id}"></td>
        <td><span th:text="${libro.title}"></td>
        <td><span th:text="${libro.author}"></td>
      </tr>
    </table>
  </body>
</html>
```

Ara ja podeu executar l'aplicació completa.

Quan accediu a localhost:8080,

- com que no posem res més a la url s'executa el mètode **buscaTots** amb `@GetMapping("/")`
- aquest mètode executa la funció `findAll` del `CRUDRepository`. L'hauríem pogut redefinir, o estendre, però per aquest exemple bàsic ens és suficient.
- **Si mireu la consola, veieu que en el fons estem fent un `select * from book` a la base de dades. Però la funció **buscaTots** retorna una llista d'objectes de la classe **Book****
- Finalment li diem que ho envii al template "llista", que Spring indentifica com a `llista.html`, li passa la llista de llibres i ens la mostra

Maven

Aneu a la pàgina:

<https://mvnrepository.com/>

D'aquí baixa Spring (o qualsevol projecte fet amb Maven) les dependències necessàries.

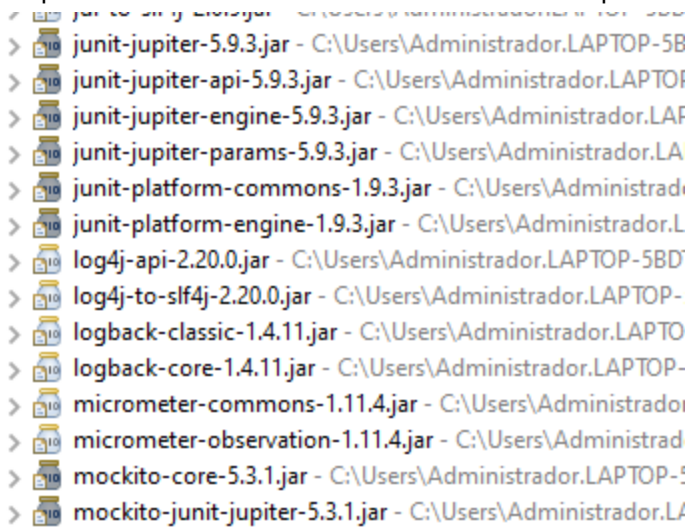
Aneu a la pàgina:

<https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa>

Si cliqueu en un num de versió, una de les coses que podeu veure són les dependències. I si cliqueu a la dependència... Així successivament. Per exemple d'aquí ve hibernate que és la libreria que realment fa la transformació entre sentències SQL i objectes Java

Per això tenim tantes llibreries a la carpeta de Maven Dependencies.

Per exemple també tenim totes les llibreries necessàries per fer tests:



> junit-jupiter-5.9.3.jar - C:\Users\Administrador.LAPTOP-5B
> junit-jupiter-api-5.9.3.jar - C:\Users\Administrador.LAPTO
> junit-jupiter-engine-5.9.3.jar - C:\Users\Administrador.LAF
> junit-jupiter-params-5.9.3.jar - C:\Users\Administrador.LA
> junit-platform-commons-1.9.3.jar - C:\Users\Administrador
> junit-platform-engine-1.9.3.jar - C:\Users\Administrador.L
> log4j-api-2.20.0.jar - C:\Users\Administrador.LAPTOP-5BD
> log4j-to-slf4j-2.20.0.jar - C:\Users\Administrador.LAPTOP-
> logback-classic-1.4.11.jar - C:\Users\Administrador.LAPTO
> logback-core-1.4.11.jar - C:\Users\Administrador.LAPTOP-
> micrometer-commons-1.11.4.jar - C:\Users\Administrador
> micrometer-observation-1.11.4.jar - C:\Users\Administrad
> mockito-core-5.3.1.jar - C:\Users\Administrador.LAPTOP-
> mockito-junit-jupiter-5.3.1.jar - C:\Users\Administrador.L

Fixeu-vos que una de les carpetes que crea Spring quan crea el projecte és la carpeta
/src/test/java paral·lela a la /src/main/java.

Annex.

ProjecteDamDawApplication.java

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.autoconfigure.domain.EntityScan;
import org.springframework.data.jpa.repository.config.EnableJpaRepositories;
@EnableJpaRepositories("com.example.demo.repository")
@EntityScan("com.example.demo.model")
@SpringBootApplication
public class ProjecteDamDawApplication {
    public static void main(String[] args) {
        SpringApplication.run(ProjecteDamDawApplication.class, args);
    }
}
```

Controlador.java

```
package com.example.demo.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import com.example.demo.model.Book;
import com.example.demo.repository.BookRepository;
@Controller
@RequestMapping("/")
public class Controlador {

    @Autowired
    private BookRepository bookRepository;
    @GetMapping("/")
    public String buscaTots(Model model) {
        Iterable<Book> books = bookRepository.findAll();
        model.addAttribute("libros", books);
        return "llista";
    }
}
```

Book.java

```
package com.example.demo.model;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private long id;
    @Column(nullable = false, unique = true)
    private String title;
    @Column(nullable = false)
    private String author;
    public Book() {
        super();
    }
    public Book(String title, String author) {
        super();
        this.title = title;
        this.author = author;
    }
    public long getId() {
        return id;
    }
    public void setId(long id) {
        this.id = id;
    }
    public String getTitle() {
        return title;
    }
    public void setTitle(String title) {
        this.title = title;
    }
    public String getAuthor() {
        return author;
    }
    public void setAuthor(String author) {
```

```
        this.author = author;
    }
}
```

BookRepository.java

```
package com.example.demo.repository;

import org.springframework.data.repository.CrudRepository;
import org.springframework.stereotype.Repository;

import com.example.demo.model.Book;

@Repository
public interface BookRepository extends CrudRepository<Book, Long> {
}
```

llista.html

```
<!DOCTYPE html>
<html lang="es" xmlns:th="http://www.thymeleaf.org">
    <body>
        <h1>Llistat de llibres</h1>
        <table>
            <tr th:each="libro : ${libros}" >
                <td><span th:text="${libro.id}"></td>
                <td><span th:text="${libro.title}"></td>
                <td><span th:text="${libro.author}"></td>
            </tr>
        </table>
    </body>
</html>
```