

Mini Aplicació Spring Boot, Jpa i H2. REST API

Exemple Bàsic JPA CRUD. Empresa.	1
Què farem	1
Introducció REST	1
Generar el projecte Spring Boot	2
Crear la classe Empresa	5
Crear la classe Repository	9
VEURE: Interface CrudRepository<T,ID> (No la creeu!!!!!!)	9
Crear el Controller	11
Provar l'Api	13
Provar l'Api amb POSTMAN	15
Ampliacions de la pràctica	17
Ús de la classe Optional de Java 8	17
Control d'excepcions	18
Documentació de la RESTApi amb SWAGGER	20
Maven	21

Exemple Bàsic JPA CRUD. Empresa.

Què farem

La nostra aplicació tindrà bàsicament les següents classes:

- ProjecteDamDawApplication.java. Defineix l'aplicació SpringBoot...
- Empresa.java. JPAEntity. L'objecte/entitat que consultem. I que voldrem fer "persistent".
- EmpresaRepository.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
- EmpresaController.java. Exposar els mètodes que permeten connectar-se a l'aplicació a través de crides http.

Introducció REST

REST: (Representational State Transfer) és un estil d'arquitectura per dissenyar serveis web:

- els programes **interactuen amb crides HTTP senzilles**
- no manté l'estat
- exposa les URI's en estructura de directoris
- transfereix XML o JSON

Les crides http:

- Per **crear** un recurs: farem servir un HTTP POST
- Per **consultar** un recurs (o una col.lecció): farem servir un HTTP GET
- Per **actualitzar** un recurs: farem servir un HTTP PUT
- Per **eliminar** un recurs: farem servir un HTTP DELETE

Recordeu que HTTP també defineix respostes estàndar

- 200 - SUCCESS
- 404 - RESOURCE NOT FOUND
- 400 - BAD REQUEST
- 201 - CREATED
- 401 - UNAUTHORIZED
- 415 - UNSUPPORTED TYPE - Representation not supported for the resource
- 500 - SERVER ERROR

El concepte clau d'un servei REST són els recursos exposats o endpoints. Un llibre, un usuari de facebook, una adreça de googleMaps, una event en un Calendari de Google...

Aquest recursos son accessibles a través de URIs més o menys ben definides.

```
insert POST /calendars/calendarId/events Creates an event.  
get GET /calendars/calendarId/events/eventId Returns an event.
```

Perquè JSON

Aquesta és una de les decisions que pren SpringBoot (recordem que és un framework “**convention over configuration**”). Com que el més habitual és JSON, si no li diem el contrari inclou les dependències necessàries per treballar amb JSON. Si volem treballar amb XML només haurem d’afegir les dependències necessàries.

Qualsevol `@RestController` d’una aplicació SpringBoot per defecte retorna respostes JSON. (Si la llibreria Jackson2 està al classpath). I en una aplicació SpringBoot Web [spring-boot-starter-web] s’inclou automàticament

Generar el projecte Spring Boot

Genereu el projecte Spring Boot.

- Trieu les dependències Web, Jpa, H2 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’ anotació `@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-logging' to 'true' for additional web related logging consider setting the 'logging.level.org.springframework.boot.autoconfigure' to 'DEBUG'
  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 repository interfaces.
  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-boot-h2-db;DB_CLOSE_ON_EXIT=FALSE
  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 'jdbc:h2:file:~/spring-boot-h2-db'
  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

```

```

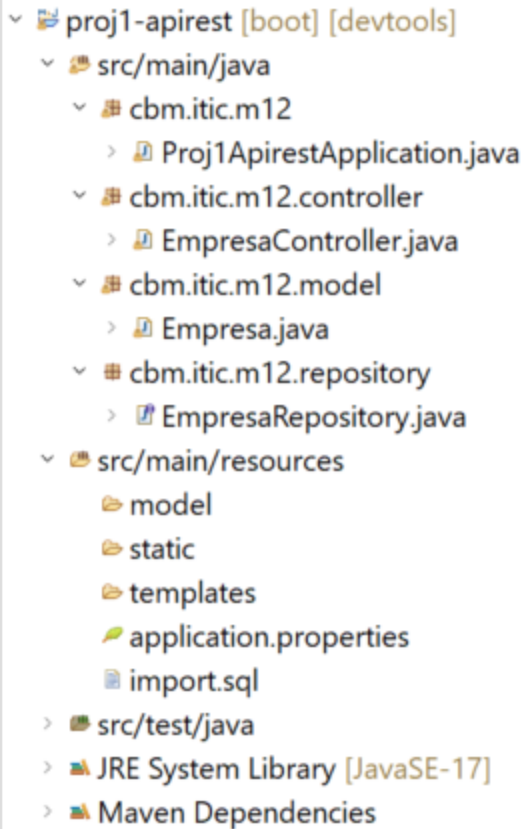
spring.jpa.defer-datasource-initialization=true
spring.sql.init.mode = always
spring.jpa.hibernate.ddl-auto=create
spring.jpa.show-sql=true

```

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:



```
proj1-apirest [boot] [devtools]
├── src/main/java
│   ├── cbm.itic.m12
│   │   ├── Proj1ApirestApplication.java
│   │   ├── cbm.itic.m12.controller
│   │   │   ├── EmpresaController.java
│   │   │   ├── cbm.itic.m12.model
│   │   │   │   ├── Empresa.java
│   │   │   ├── cbm.itic.m12.repository
│   │   │   │   ├── EmpresaRepository.java
│   │   ├── src/main/resources
│   │   │   ├── model
│   │   │   ├── static
│   │   │   ├── templates
│   │   │   ├── application.properties
│   │   │   └── import.sql
│   ├── src/test/java
│   ├── JRE System Library [JavaSE-17]
│   └── Maven Dependencies
```

Crear la classe Empresa

Creeu la classe Empresa.java

```
@Entity
public class Empresa {
    @Id //jakarta.persistence.Id
    private long id;

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

    @Column(nullable = false)
    private String cif;

    // A PARTIR D'AQUÍ CODI GENERAT DES DEL MENÚ "Source"
    // Els 2 constructors i els getters i setters

    ...
}
```

Feu els imports necessaris

DES DEL MENÚ SOURCE

GENEREU ELS CONSTRUCTORS!!!!!!!!!!!!!!

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.

DE MOMENT HO HE TRET.

PROVEU-HO

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

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

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

```
insert into empresa values (1, 'A12312312', 'Indra');
insert into empresa values (2, 'B45645678', 'itic');
insert into empresa values (3, 'A98798765', 'Blanc i Roig SL');
```

Si ara executem l'aplicació, ja veurem més coses

```
2023-11-05T13:22:59.092+01:00 INFO 16084 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-11-05T13:22:59.103+01:00 INFO 16084 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-11-05T13:22:59.104+01:00 INFO 16084 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.15]
2023-11-05T13:22:59.200+01:00 INFO 16084 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-11-05T13:22:59.204+01:00 INFO 16084 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1472 ms
2023-11-05T13:22:59.269+01:00 INFO 16084 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2023-11-05T13:22:59.513+01:00 INFO 16084 --- [ restartedMain] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection conn0: url=jdbc:h2:file:~/spring-boot
2023-11-05T13:22:59.515+01:00 INFO 16084 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2023-11-05T13:22:59.529+01:00 INFO 16084 --- [ restartedMain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h:
2023-11-05T13:22:59.832+01:00 INFO 16084 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
2023-11-05T13:22:59.954+01:00 INFO 16084 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 6.2.13.Final
2023-11-05T13:22:59.956+01:00 INFO 16084 --- [ restartedMain] org.hibernate.cfg.Environment : HHH000406: Using bytecode reflection optimizer
2023-11-05T13:23:00.223+01:00 INFO 16084 --- [ restartedMain] o.s.c.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2023-11-05T13:23:00.993+01:00 INFO 16084 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction.jta
Hibernate: drop table if exists empresa cascade
Hibernate: create table empresa (id bigint not null, cif varchar(255) not null, nom varchar(255) not null unique, primary key (id))
Hibernate: insert into empresa values (1, 'A12312312', 'Indra')
Hibernate: insert into empresa values (2, 'B45645678', 'itic')
Hibernate: insert into empresa values (3, 'A98798765', 'Blanc i Roig SL')
```

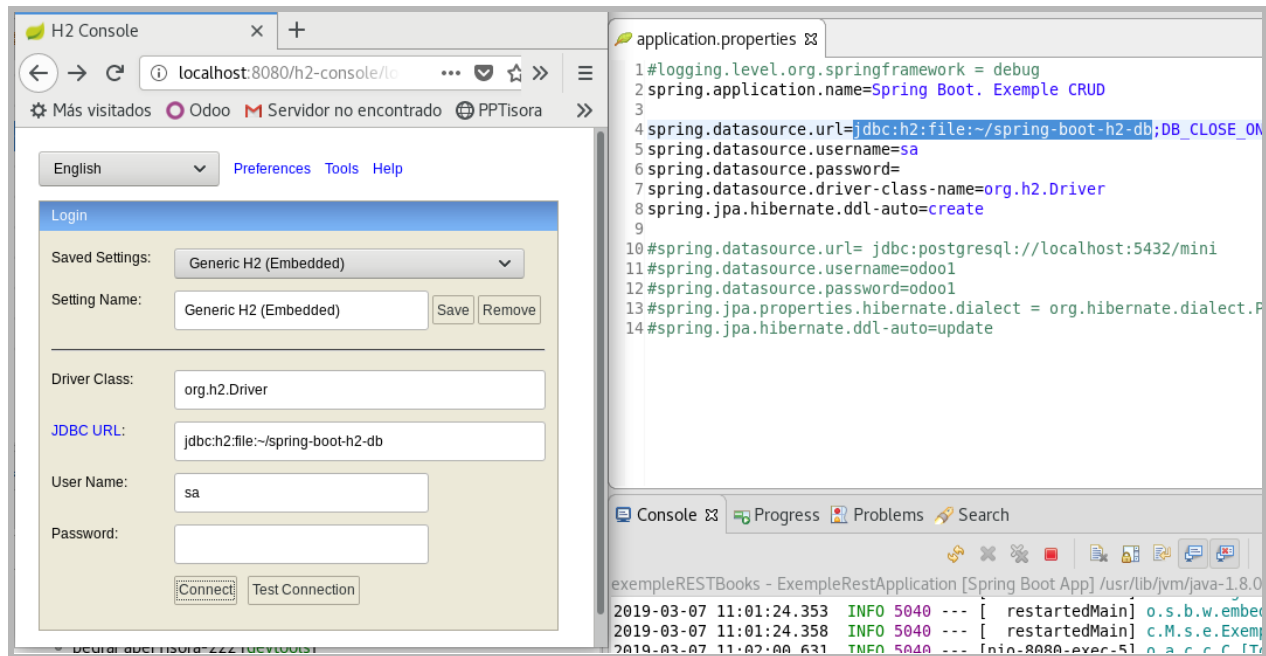
Per exemple ens diu:

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 on the left and the application.properties file on the right. The H2 Console is displaying the login form with the following details:

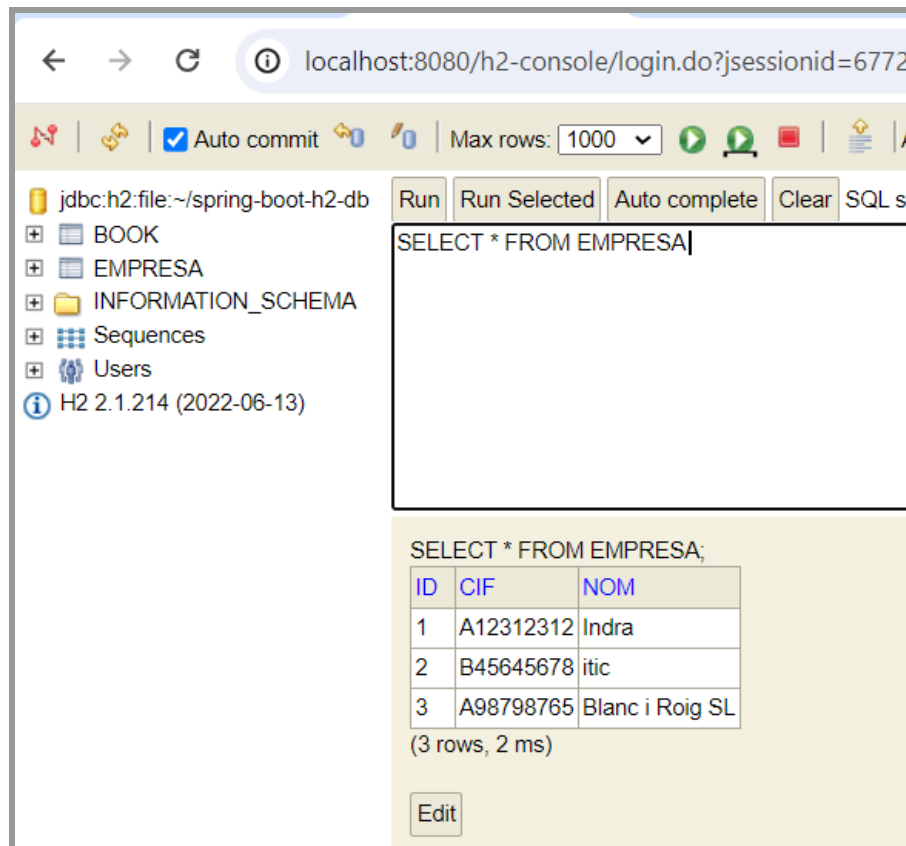
- Language: English
- Preferences: Tools Help
- Login:
 - Saved Settings: Generic H2 (Embedded)
 - Setting Name: Generic H2 (Embedded)
 - Driver Class: org.h2.Driver
 - JDBC URL: jdbc:h2:file:~/spring-boot-h2-db
 - User Name: sa
 - Password: (empty)
 - Buttons: Connect, Test Connection

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
```

Below the properties file, the IDE console shows the following log messages:

```
2019-03-07 11:01:24.353 INFO 5040 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat initialized with port(s): 8080 (http)
2019-03-07 11:01:24.358 INFO 5040 --- [ restartedMain] c.M.s.e.ExempleRESTApplication: ExampleRESTApplication started.
2019-03-07 11:02:00.631 INFO 5040 --- [main-8080-exec-5] o.a.c.c.C.[Tomcat]: Initializing Spring DispatcherServlet 'dispatcherServlet'
2019-03-07 11:02:00.631 INFO 5040 --- [main-8080-exec-5] o.s.w.s.m.m.a.RequestMappingHandlerMapping: Mapped URL [<!-- ... -->
```



The screenshot shows the H2 Console interface with the SQL query results displayed. The query executed is:

```
SELECT * FROM EMPRESA;
```

The results are shown in a table with 3 rows and 3 columns (ID, CIF, NOM). The table data is as follows:

ID	CIF	NOM
1	A12312312	Indra
2	B45645678	itic
3	A98798765	Blanc i Roig SL

Below the table, it indicates "(3 rows, 2 ms)". There is an "Edit" button at the bottom.

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 EmpresaRepository extends CrudRepository<Empresa, 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(empresa) farà un insert o update de l'objecte empresa 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>

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

Crear el Controller

Crearem una classe EmpresaController.java

```
@RestController
@RequestMapping("/api")
public class EmpresaController {

    @Autowired
    private EmpresaRepository empresaRepository;
```

@RestController: combinació de **@Controller** que ja vam veure i **@ResponseBody**. Els objectes (beans) es converteixen automàticament en/a JSON/XML...

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

Crearem els mètodes corresponents al

- **@GetMapping("/empresa")**
Veure tots els llibres
- **@GetMapping("/empresa/{id}")**
Buscar per id
- **@PostMapping("/empresa")**
Afegir una empresa nova
- **@DeleteMapping("/empresa/{id}")**
Esborrar una empresa nova
- **@PutMapping("/empresa/{id}")**
Modificar una empresa

El codi complet:

```
@RestController
@RequestMapping("/api")
public class EmpresaController {
    @Autowired
    private EmpresaRepository empresaRepository;

    @GetMapping("/empresa")
    public Iterable<Empresa> buscaTots() {
        return empresaRepository.findAll();
    }
    @GetMapping("/empresa/{id}")
    public Optional<Empresa> buscaPerId(@PathVariable long id) {
        return empresaRepository.findById(id);
    }
    @PostMapping("/empresa")
    @ResponseStatus(HttpStatus.CREATED)
    public Empresa afegeix(@RequestBody Empresa empresa) {
        Empresa empresa1 = empresaRepository.save(empresa);
        return empresa1;
    }
    @DeleteMapping("/empresa/{id}")
    public void esborra(@PathVariable long id) {
        empresaRepository.deleteById(id);
    }
}
```

Les anotacions del controller:

@RestController: combinació de **@Controller** and **@ResponseBody**

@RequestBody: (un paràmetre). Spring mapejarà el cos del request HTTP (que la URL correspongui al **@RequestMapping**) al paràmetre amb aquest anotació. Internament farà servir HTTP Message Converter per “des-serializar” el cos del request a l’objecte corresponent.

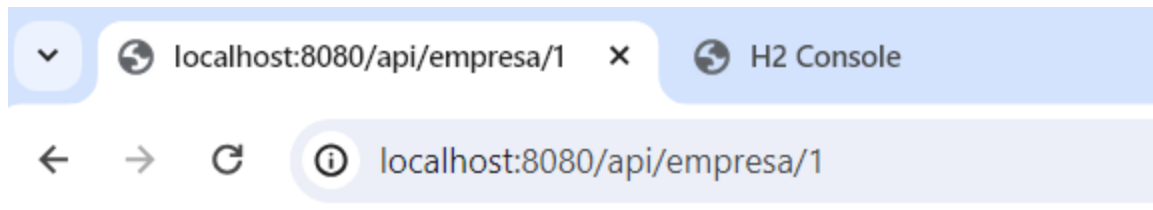
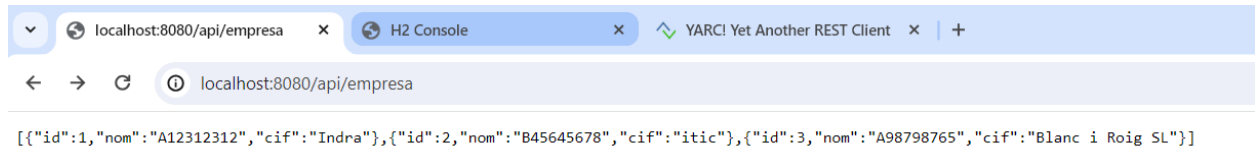
@ResponseBody: (al revés). El valor retornat es mapeja a la sortida HTTP.

@PathVariable: (en un paràmetre) indica que el paràmetre s’ha de mapejar al tros corresponent de la URL delimitat per {}

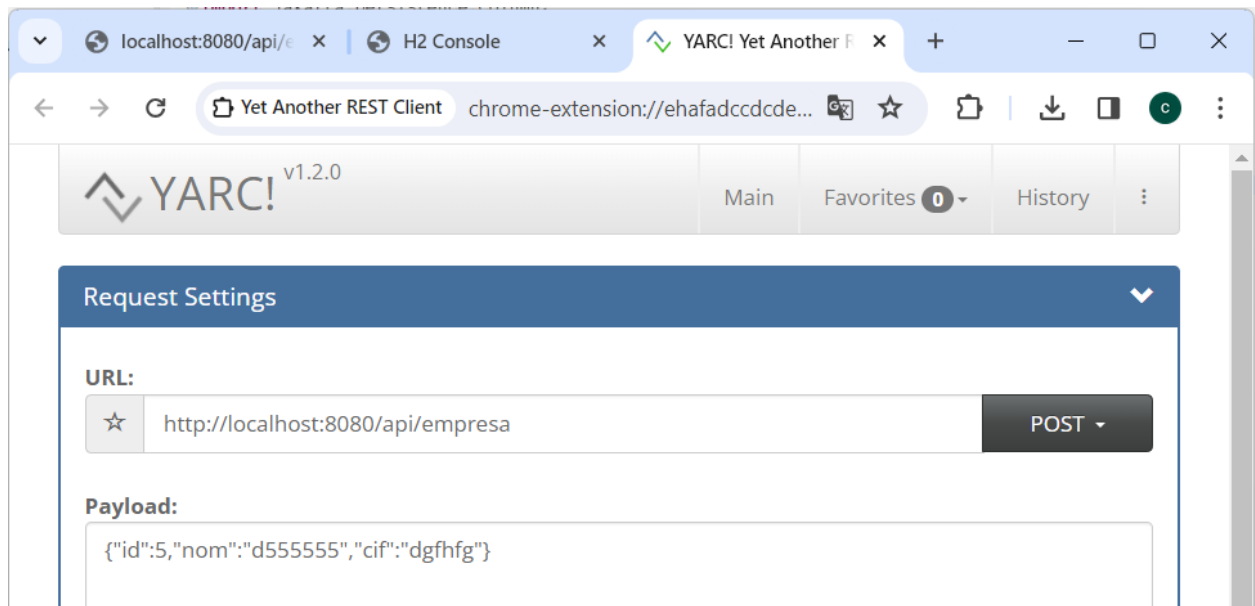
@ResponseStatus. Afegirem a la nostra resposta el HTTP.status especificat

Provar l'Api

Els gets són fàcils de provar:



Però per provar un post ens hauríem de construir un formulari HTML amb els camps que té una empresa i enviar-lo. (que és el que vam fer en el projecte anterior). Ara, com que el projecte només va de construir la API que exposem públicament, ho provarem amb una eina.



I la resposta:

Response

Request URL: <http://localhost:8080/api/empresa>
Request Method: POST
Response Time: 0.351 seconds
Response Status: 201 - Created

201

Response Body

Response Body (RAW)

Response Headers

Request Details

```
{
  "id": 5,
  "nom": "d555555",
  "cif": "dgfhfg"
}
```

Si intento inserir una altra empresa “Indra” em donarà error, pq hem definit el camp com a unique.

← → ↻

Yet Another REST Client

chrome-extension://ehafadccdcde...

☆

📁

⬇

📄

⚙

YARCI! v1.2.0

Main

Favorites 0

History

⋮

Request Settings

URL:
☆ POST

Payload:

```
{"id":5,"nom":"Indra","cif":"dgfhfg"}
```

Response

Request URL: <http://localhost:8080/api/empresa>
Request Method: POST
Response Time: 0.272 seconds
Response Status: 500 - Internal Server Error

500

Response Body

Response Body (RAW)

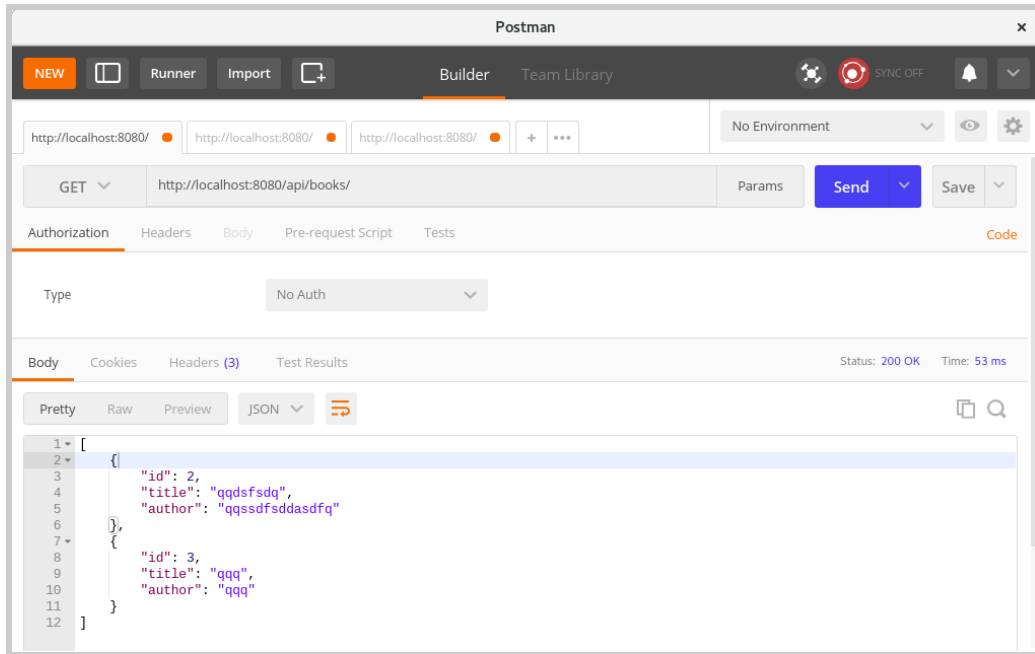
Response Headers

Request Details

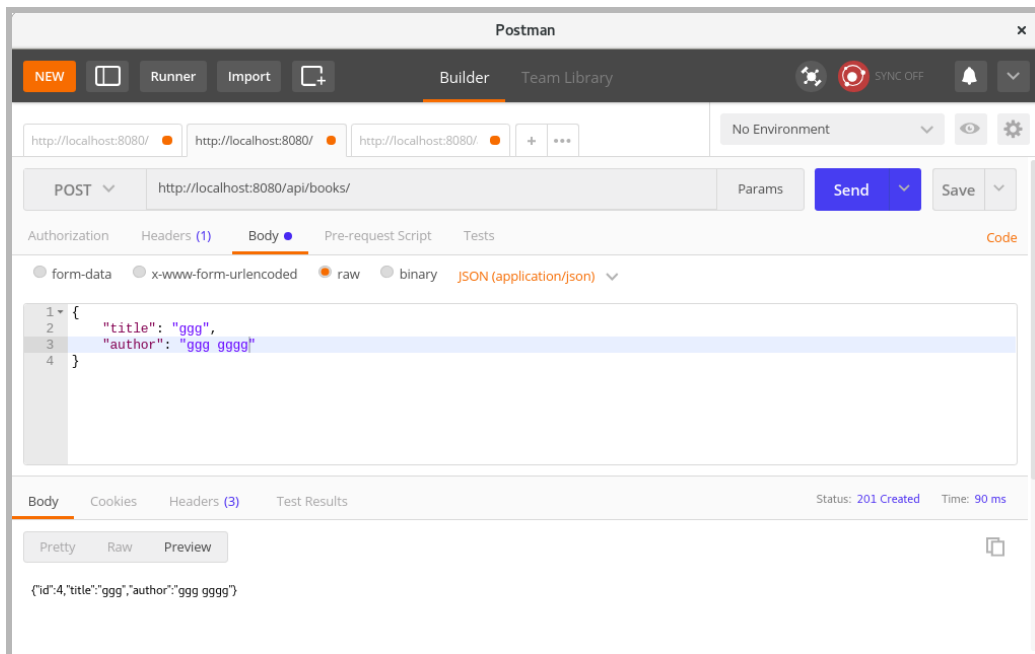
```
{
  "timestamp": "2023-11-05T11:51:07.778+00:00",
  "status": 500,
  "error": "Internal Server Error",
  "trace": "org.springframework.dao.DataIntegrityViolationException: could not execute st
atement [Violación de índice de Unicidad ó Clave primaria: \"PUBLIC.CONSTRAINT_INDEX_C ON
PUBLIC.EMPRESA(NOM NULLS FIRST) VALUES ( /* 1 */ 'Indra' )\"\\nUnique index or primary key
violation: \"PUBLIC.CONSTRAINT_INDEX_C ON PUBLIC.EMPRESA(NOM NULLS FIRST) VALUES ( /* 1
*/ 'Indra' )\": SQL statement:\\ninsert into empresa (cif,nom,id) values (?,?.?) [23505-21
```

Provar l'Api amb POSTMAN

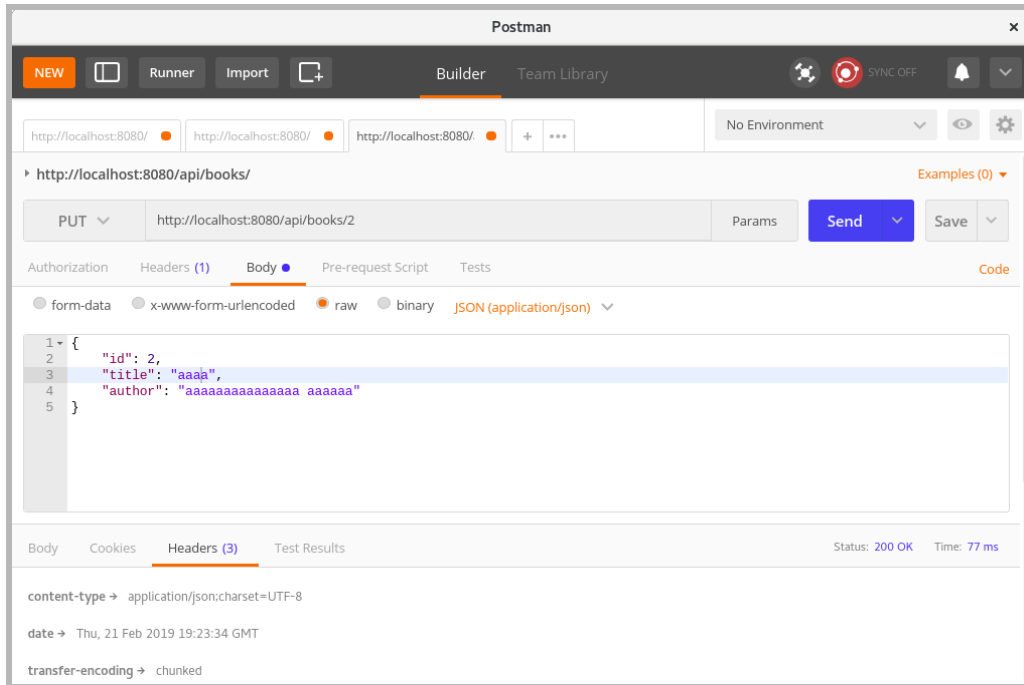
Per provar l'aplicació farem servir POSTMAN
GET per veure tots els resultats:



POST per afegir un registre



PUT per modificar un registre existent



Ampliacions de la pràctica

Ús de la classe Optional de Java 8

VEURE: <https://docs.oracle.com/javase/8/docs/api/java/util/Optional.html>

<https://www.arquitecturajava.com/que-es-un-java-optional/>

<https://www.adictosaltrabajo.com/2015/03/02/optional-java-8/>

Nota SEO

```
<meta property="article:published_time" content="2015-03-02T00:00:00+00:00" />
<meta property="article:modified_time" content="2021-02-15T11:54:43+00:00" />
```

Veieu que

```
empresaRepository.findById(id)
```

retorna un objecte Optional. Mireu la declaració de la findById

Sobre aquest object podem aplicar els mètodes de la classe Optional. Per exemple:

```
void ifPresent(Consumer<? super T> consumer)
If a value is present, invoke the specified consumer with the value, otherwise do
nothing.

boolean isPresent()
Return true if there is a value present, otherwise false.

T orElse(T other)
Return the value if present, otherwise return other.

T orElseThrow(Supplier<? extends X> exceptionSupplier)
Return the contained value, if present, otherwise throw an exception to be created
by the provided supplier.
```

Control d'excepcions

En el codi que hem vist fins ara no hi ha ni el mínim control d'errors.

SpringBoot proporciona una bona implementació per defecte de la gestió d'excepcions per serveis REST. (Proveu peticions errònies amb Postman). Però podem definir noves excepcions o personalitzar la gestió d'errors.

Afegiu la classe:

```
package cbm.itic.m12.exception;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;
@ResponseStatus(value = HttpStatus.NOT_FOUND)
public class EmpresaNotFoundException extends RuntimeException {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public EmpresaNotFoundException() {
        super("L'empresa no existeix");
    }
}
```

I la fem servir:

```
// buscaPerId v1
// -----
@GetMapping("/empresa/{id}")
public Optional<Empresa> buscaPerId(@PathVariable long id) {
    return empresaRepository.findById(id);
}

// buscaPerId v4
// -----
@GetMapping("/empresa/{id}")
Empresa buscaPerId(@PathVariable Long id) {
    return empresaRepository.findById(id)
        .orElseThrow(() -> new EmpresaNotFoundException());
}
```

Quan la fem servir ja no ens volca la traça de l'error. Ens diu què ha passat.

Response

Request URL: <http://localhost:8080/api/empresa/44>
Request Method: GET
Response Time: 0.029 seconds
Response Status: 404 - Not Found

404

Response Body

Response Body (RAW)

Response Headers

Request Details

```
{
  "timestamp": "2023-11-05T13:08:50.513+00:00",
  "status": 404,
  "error": "Not Found",
  "trace": "cbm.itic.m12.exception.EmpresaNotFoundException: L'empresa no existeix\r\n\tat cbm.",
  "message": "L'empresa no existeix",
  "path": "/api/empresa/44"
}
```

Documentació de la RESTapi amb SWAGGER

Maven

Aneu a la pàgina:

<https://mvnrepository.com/>

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

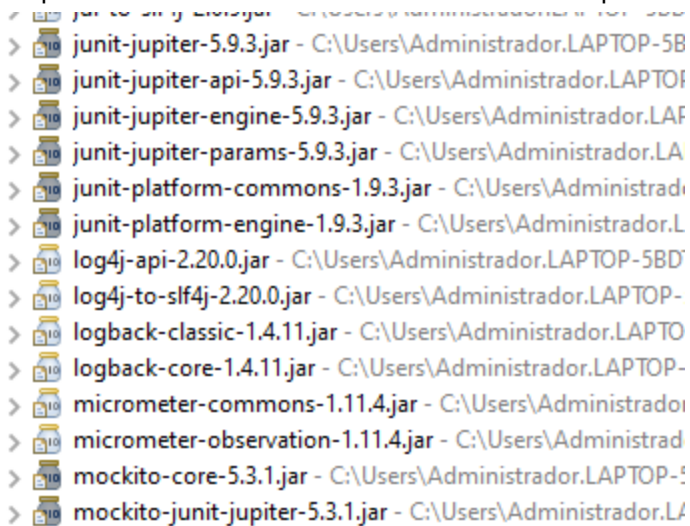
Aneu a la pàgina:

<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.LA...
- > junit-jupiter-params-5.9.3.jar - C:\Users\Administrador.LA...
- > junit-platform-commons-1.9.3.jar - C:\Users\Administrador.L...
- > 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.