Dotzena d'Exercicis Guiats Progressius en Spring Boot

Introducció

Aquesta col·lecció d'exercicis està dissenyada per a estudiants que comencen amb Spring Boot des de zero. Cada exercici inclou instruccions detallades pas a pas, amb tot el codi necessari i explicacions clares.

Prerequisits:

- Java JDK 17 o superior instal·lat
- Un IDE (IntelliJ IDEA, Eclipse o Visual Studio Code)
- Maven 3.8.6 o superior
- Coneixements bàsics de Java i programació orientada a objectes

Exercici 1: Crear el Teu Primer Projecte Spring Boot

Objectiu: Aprendre a crear un projecte Spring Boot bàsic i executar-lo.

Passos detallats:

Pas 1.1: Accedir a Spring Initializr

1. Obre el navegador i ves a https://start.spring.io/
2. Veuràs una interfície amb diverses opcions de configuració

Pas 1.2: Configurar el projecte

1. **Project:** Selecciona "Maven Project"
2. **Language:** Selecciona "Java"
3. **Spring Boot:** Selecciona la versió més recent (per exemple, 3.2.0)
4. **Project Metadata:**
   - Group: `com.exemple`
   - Artifact: `primer-projecte`
   - Name: `primer-projecte`
   - Description: `El meu primer projecte Spring Boot`
   - Package name: `com.exemple.primerprojecte`
   - Packaging: `Jar`
   - Java: `17`

Pas 1.3: Afegir dependències

1. Fes clic al botó "ADD DEPENDENCIES"
2. Cerca i selecciona "Spring Web"
3. Aquesta dependència inclou tot el necessari per crear aplicacions web

Pas 1.4: Generar i descarregar el projecte

1. Fes clic al botó "GENERATE"
2. Es descarregarà un fitxer ZIP anomenat `primer-projecte.zip`
3. Descomprimeix aquest fitxer en una carpeta del teu ordinador

Pas 1.5: Importar el projecte a l'IDE

Per **IntelliJ IDEA:**

1. Obre IntelliJ IDEA
2. Selecciona "Open" al menú principal
3. Navega fins a la carpeta descomprimida
4. Selecciona el fitxer `pom.xml` i fes clic a "Open"
5. Marca l'opció "Open as Project"
6. Espera que IntelliJ descarregui totes les dependències (pot trigar uns minuts)

Per **Eclipse:**

1. Obre Eclipse
2. Selecciona "File" → "Import"
3. Selecciona "Maven" → "Existing Maven Projects"
4. Navega fins a la carpeta descomprimida
5. Fes clic a "Finish"

Pas 1.6: Explorar l'estructura del projecte

El projecte tindrà aquesta estructura:

```
primer-projecte/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/exemple/primerprojecte/
│   │   │       └── PrimerProjecteApplication.java
│   │   └── resources/
│   │       └── application.properties
│   └── test/
├── pom.xml
└── mvnw
```

Pas 1.7: Examinar la classe principal

Obre el fitxer `PrimerProjecteApplication.java`. Hauries de veure:

```
package com.exemple.primerprojecte;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class PrimerProjecteApplication {

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

Explicació del codi:

- `@SpringBootApplication`: Aquesta anotació combina tres anotacions importants i configura automàticament Spring Boot
- `main()`: El punt d'entrada de l'aplicació
- `SpringApplication.run()`: Inicia l'aplicació Spring Boot

Pas 1.8: Executar l'aplicació

Opció A - Des de l'IDE:

1. Fes clic dret sobre `PrimerProjecteApplication.java`
2. Selecciona "Run 'PrimerProjecteApplication'"
3. Mira la consola, hauries de veure missatges similars a:

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v3.2.0)

...
Started PrimerProjecteApplication in 2.536 seconds
```

Opció B - Des de la terminal:

1. Obre una terminal
2. Navega fins a la carpeta del projecte
3. Executa: `./mvnw spring-boot:run` (Linux/Mac) o `mvnw.cmd spring-boot:run` (Windows)

Pas 1.9: Verificar que l'aplicació funciona

1. Obre el navegador

2. Ves a http://localhost:8080
3. Veuràs una pàgina d'error (això és normal, encara no hem creat cap endpoint!)

Resultat esperat:
L'aplicació s'inicia correctament i escolta al port 8080. Encara no fa res visible, però és un bon començament!

Exercici 2: Crear el Teu Primer Controlador REST

Objectiu: Aprendre a crear un controlador que respongui a peticions HTTP.

Passos detallats:

Pas 2.1: Crear el paquet per als controladors

1. Dins de `src/main/java/com/exemple/primerprojecte/`
2. Crea un nou paquet anomenat `controller`
3. La ruta completa serà: `src/main/java/com/exemple/primerprojecte/controller/`

Pas 2.2: Crear la classe del controlador

1. Dins del paquet `controller`, crea una nova classe Java
2. Anomena-la `HolaController`
3. Escriu el següent codi:

```
package com.exemple.primerprojecte.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class HolaController {

    @GetMapping("/hola")
    public String saludar() {
        return "Hola, benvingut a Spring Boot!";
    }
}
```

Explicació del codi:

- `@RestController`: Indica que aquesta classe és un controlador REST que retornarà dades directament (no vistes HTML)
- `@GetMapping("/hola")`: Mapeja peticions HTTP GET a la ruta `/hola`
- `public String saludar()`: Mètode que es crida quan algú accedeix a `/hola`
- `return "Hola..."`: El text que es retornarà al navegador

Pas 2.3: Executar l'aplicació

1. Atura l'aplicació si encara s'està executant (botó vermell de stop)
2. Torna a executar l'aplicació
3. Spring Boot detectarà automàticament el nou controlador

Pas 2.4: Provar l'endpoint

1. Obre el navegador
2. Ves a http://localhost:8080/hola
3. Hauries de veure: `Hola, benvingut a Spring Boot!`

Pas 2.5: Afegir més endpoints

Afegeix aquest mètode a la classe `HolaController`:

```
@GetMapping("/adeu")
public String acomiadar() {
    return "Fins aviat!";
}
```

Pas 2.6: Provar el nou endpoint

1. No cal reiniciar (si tens Spring Boot DevTools)
2. Ves a http://localhost:8080/adeu
3. Hauries de veure: `Fins aviat!`

Resultat esperat:
Tens dos endpoints funcionant que retornen text simple. Has après a crear controladors REST bàsics!

Exercici 3: Treballar amb Paràmetres d'URL

Objectiu: Aprendre a acceptar i utilitzar paràmetres en les peticions HTTP.

Passos detallats:

Pas 3.1: Afegir un endpoint amb paràmetre de ruta

Afegeix aquest mètode a `HolaController`:

```
import org.springframework.web.bind.annotation.PathVariable;

@GetMapping("/hola/{nom}")
public String saludarPersonalitzat(@PathVariable String nom) {
    return "Hola, " + nom + "! Com estàs?";
}
```

Explicació del codi:

- `{nom}`: Placeholder a la URL que acceptarà qualsevol valor

- `@PathVariable String nom`: Captura el valor de la URL i l'assigna a la variable `nom`
- El mètode utilitza aquesta variable per personalitzar la resposta

Pas 3.2: Provar l'endpoint amb diferents valors

1. Ves a http://localhost:8080/hola/Maria
   - Resposta: `Hola, Maria! Com estàs?`
2. Ves a http://localhost:8080/hola/Joan
   - Resposta: `Hola, Joan! Com estàs?`
3. Prova amb el teu nom!

Pas 3.3: Afegir paràmetres de consulta (query parameters)

Afegeix aquest nou mètode:

```
import org.springframework.web.bind.annotation.RequestParam;

@GetMapping("/salutacio")
public String salutacioCompleta(
    @RequestParam String nom,
    @RequestParam(required = false, defaultValue = "amig") String tractament
) {
    return "Hola, estimat " + tractament + " " + nom + "!";
}
```

Explicació del codi:

- `@RequestParam String nom`: Paràmetre obligatori de la URL (com `?nom=Maria`)
- `required = false`: Fa que el paràmetre sigui opcional
- `defaultValue = "amig"`: Valor per defecte si no s'especifica

Pas 3.4: Provar els paràmetres de consulta

1. Ves a http://localhost:8080/salutacio?nom=Anna
   - Resposta: `Hola, estimat amig Anna!`
2. Ves a http://localhost:8080/salutacio?nom=Pere&tractament=professor
   - Resposta: `Hola, estimat professor Pere!`

Pas 3.5: Combinar paràmetres de ruta i consulta

Crea un endpoint més complex:

```
@GetMapping("/usuari/{id}/perfil")
public String mostrarPerfil(
    @PathVariable Long id,
    @RequestParam(required = false, defaultValue = "false") boolean detallat
) {
    String resposta = "Perfil de l'usuari amb ID: " + id;
    if (detallat) {
```

```
        resposta += " (Mode detallat activat)";
    }
    return resposta;
}
```

Pas 3.6: Provar l'endpoint combinat

1. http://localhost:8080/usuari/123/perfil
    - Resposta: `Perfil de l'usuari amb ID: 123`
2. http://localhost:8080/usuari/123/perfil?detallat=true
    - Resposta: `Perfil de l'usuari amb ID: 123 (Mode detallat activat)`

Resultat esperat:
Ara saps treballar amb paràmetres tant a la ruta com a la consulta de la URL!

Exercici 4: Retornar Objectes JSON

Objectiu: Aprendre a crear classes de model i retornar-les com a JSON.

Passos detallats:

Pas 4.1: Crear el paquet per als models

1. Crea un nou paquet: `com.exemple.primerprojecte.model`
2. Aquí guardarem totes les nostres classes de dades

Pas 4.2: Crear la classe Usuari

Crea una nova classe anomenada `Usuari.java`:

```
package com.exemple.primerprojecte.model;

public class Usuari {
    private Long id;
    private String nom;
    private String email;
    private int edat;

    // Constructor buit
    public Usuari() {
    }

    // Constructor amb paràmetres
    public Usuari(Long id, String nom, String email, int edat) {
        this.id = id;
        this.nom = nom;
        this.email = email;
        this.edat = edat;
    }

    // Getters i Setters
```

```
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getNom() {
        return nom;
    }

    public void setNom(String nom) {
        this.nom = nom;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public int getEdat() {
        return edat;
    }

    public void setEdat(int edat) {
        this.edat = edat;
    }
}
```

Explicació del codi:

- Aquesta és una classe POJO (Plain Old Java Object)
- Té atributs privats amb getters i setters públics
- Spring Boot convertirà automàticament aquesta classe a JSON

Pas 4.3: Crear un controlador per als usuaris

Crea `UsuariController.java` al paquet `controller`:

```
package com.exemple.primerprojecte.controller;

import com.exemple.primerprojecte.model.Usuari;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

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

```
    @GetMapping("/{id}")
    public Usuari obtenirUsuari(@PathVariable Long id) {
        // Creem un usuari de prova
        return new Usuari(id, "Maria Garcia", "maria@exemple.com", 25);
    }
}
```

Explicació del codi:

- `@RequestMapping("/api/usuaris")`: Totes les rutes d'aquest controlador començaran amb `/api/usuaris`
- El mètode retorna un objecte `Usuari`
- Spring Boot automàticament el converteix a JSON

Pas 4.4: Provar l'endpoint

1. Ves a http://localhost:8080/api/usuaris/1
2. Hauries de veure aquest JSON:

```
{
  "id": 1,
  "nom": "Maria Garcia",
  "email": "maria@exemple.com",
  "edat": 25
}
```

Pas 4.5: Retornar múltiples objectes

Afegeix aquest mètode a `UsuariController`:

```
import java.util.Arrays;
import java.util.List;

@GetMapping
public List&lt;Usuari&gt; obtenirTotsElsUsuaris() {
    return Arrays.asList(
        new Usuari(1L, "Maria Garcia", "maria@exemple.com", 25),
        new Usuari(2L, "Joan Martí", "joan@exemple.com", 30),
        new Usuari(3L, "Anna Puig", "anna@exemple.com", 28)
    );
}
```

Pas 4.6: Provar la llista d'usuaris

1. Ves a http://localhost:8080/api/usuaris
2. Hauries de veure un array JSON amb tres usuaris:

```
[
  {
    "id": 1,
```

```
    "nom": "Maria Garcia",
    "email": "maria@exemple.com",
    "edat": 25
  },
  {
    "id": 2,
    "nom": "Joan Martí",
    "email": "joan@exemple.com",
    "edat": 30
  },
  {
    "id": 3,
    "nom": "Anna Puig",
    "email": "anna@exemple.com",
    "edat": 28
  }
]
```

Resultat esperat:
Ja saps crear classes de model i retornar-les com a JSON. Spring Boot s'encarrega automàticament de la conversió!

Exercici 5: Implementar un Servei

Objectiu: Aprendre a separar la lògica de negoci del controlador utilitzant serveis.

Passos detallats:

Pas 5.1: Crear el paquet per als serveis

1. Crea un nou paquet: `com.exemple.primerprojecte.service`
2. Aquí guardarem la lògica de negoci

Pas 5.2: Crear la classe de servei

Crea `UsuariService.java`:

```
package com.exemple.primerprojecte.service;

import com.exemple.primerprojecte.model.Usuari;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

@Service
public class UsuariService {

    // Simulem una base de dades amb una llista
    private List&lt;Usuari&gt; usuaris = new ArrayList&lt;&gt;();
    private Long proximId = 1L;
```

```
    // Constructor que inicialitza alguns usuaris
    public UsuariService() {
        usuaris.add(new Usuari(proximId++, "Maria Garcia", "maria@exemple.com", 25));
        usuaris.add(new Usuari(proximId++, "Joan Martí", "joan@exemple.com", 30));
        usuaris.add(new Usuari(proximId++, "Anna Puig", "anna@exemple.com", 28));
    }

    // Obtenir tots els usuaris
    public List&lt;Usuari&gt; obtenirTotsElsUsuaris() {
        return new ArrayList&lt;&gt;(usuaris);
    }

    // Buscar usuari per ID
    public Optional&lt;Usuari&gt; obtenirUsuariPerId(Long id) {
        return usuaris.stream()
                      .filter(u -&gt; u.getId().equals(id))
                      .findFirst();
    }
}
```

Explicació del codi:

- `@Service`: Marca aquesta classe com un servei de Spring
- `List<Usuari> usuaris`: Simula una base de dades en memòria
- `Optional<Usuari>`: Permet gestionar casos on l'usuari no existeix
- Els mètodes encapsulen la lògica de negoci

Pas 5.3: Modificar el controlador per utilitzar el servei

Actualitza `UsuariController.java`:

```
package com.exemple.primerprojecte.controller;

import com.exemple.primerprojecte.model.Usuari;
import com.exemple.primerprojecte.service.UsuariService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/api/usuaris")
public class UsuariController {

    @Autowired
    private UsuariService usuariService;

    @GetMapping
    public List&lt;Usuari&gt; obtenirTotsElsUsuaris() {
```

```
        return usuariService.obtenirTotsElsUsuaris();
    }

    @GetMapping("/{id}")
    public ResponseEntity&lt;Usuari&gt; obtenirUsuari(@PathVariable Long id) {
        return usuariService.obtenirUsuariPerId(id)
                .map(ResponseEntity::ok)
                .orElse(ResponseEntity.notFound().build());
    }
}
```

Explicació del codi:

- `@Autowired`: Spring injecta automàticament el servei
- `ResponseEntity<Usuari>`: Permet retornar diferents codis HTTP
- `.map(ResponseEntity::ok)`: Si l'usuari existeix, retorna 200 OK
- `.orElse(ResponseEntity.notFound().build())`: Si no existeix, retorna 404 Not Found

Pas 5.4: Provar els endpoints

1. Ves a http://localhost:8080/api/usuaris
   - Hauries de veure la llista completa
2. Ves a http://localhost:8080/api/usuaris/1
   - Hauries de veure l'usuari amb ID 1
3. Ves a http://localhost:8080/api/usuaris/999
   - Hauries de veure un error 404 (Not Found)

Pas 5.5: Afegir mètode per cercar per nom

Afegeix aquest mètode a `UsuariService`:

```
public List&lt;Usuari&gt; cercarPerNom(String nom) {
    return usuaris.stream()
                  .filter(u -&gt; u.getNom().toLowerCase().contains(nom.toLowerCase()))
                  .toList();
}
```

Afegeix aquest endpoint a `UsuariController`:

```
@GetMapping("/cercar")
public List&lt;Usuari&gt; cercarUsuaris(@RequestParam String nom) {
    return usuariService.cercarPerNom(nom);
}
```

Pas 5.6: Provar la cerca

1. Ves a http://localhost:8080/api/usuaris/cercar?nom=Maria
   - Hauries de veure només l'usuari Maria

2. Prova amb altres noms!

Resultat esperat:
Has après a separar la lògica de negoci en serveis, fent el codi més organitzat i reutilitzable!

Exercici 6: Crear Usuaris amb POST

Objectiu: Aprendre a acceptar dades del client i crear nous recursos.

Passos detallats:

Pas 6.1: Afegir el mètode al servei

Afegeix aquest mètode a `UsuariService`:

```
public Usuari crearUsuari(Usuari usuari) {
    usuari.setId(proximId++);
    usuaris.add(usuari);
    return usuari;
}
```

Explicació del codi:

- Assigna automàticament un ID nou
- Afegeix l'usuari a la llista
- Retorna l'usuari creat amb el seu nou ID

Pas 6.2: Afegir l'endpoint POST al controlador

Afegeix aquest mètode a `UsuariController`:

```
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.ResponseStatus;

@PostMapping
@ResponseStatus(HttpStatus.CREATED)
public Usuari crearUsuari(@RequestBody Usuari usuari) {
    return usuariService.crearUsuari(usuari);
}
```

Explicació del codi:

- `@PostMapping`: Gestiona peticions HTTP POST
- `@RequestBody`: Indica que les dades vindran al cos de la petició en format JSON
- `@ResponseStatus(HttpStatus.CREATED)`: Retorna el codi 201 (Created) en lloc de 200

Pas 6.3: Provar amb Postman o curl

Opció A - Utilitzant Postman:

1. Obre Postman
2. Crea una nova petició:
   - Mètode: POST
   - URL: http://localhost:8080/api/usuaris
   - Headers: `Content-Type: application/json`
   - Body (raw, JSON):

```
{
  "nom": "Pere Soler",
  "email": "pere@exemple.com",
  "edat": 35
}
```

3. Envia la petició
4. Hauries de rebre una resposta amb l'usuari creat i el seu nou ID

Opció B - Utilitzant curl (terminal):

```
curl -X POST http://localhost:8080/api/usuaris \
  -H "Content-Type: application/json" \
  -d '{"nom":"Pere Soler","email":"pere@exemple.com","edat":35}'
```

Pas 6.4: Verificar que s'ha creat

1. Ves a http://localhost:8080/api/usuaris
2. Hauries de veure el nou usuari a la llista

Pas 6.5: Crear un client HTTP a l'IDE

Si utilitzes IntelliJ IDEA Ultimate, pots crear un fitxer `test-api.http`:

```
### Obtenir tots els usuaris
GET http://localhost:8080/api/usuaris

### Crear un nou usuari
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Laura Vidal",
  "email": "laura@exemple.com",
  "edat": 27
}

### Obtenir un usuari específic
GET http://localhost:8080/api/usuaris/1
```

Pots executar cada petició fent clic a la icona verda que apareix.

Resultat esperat:
Ja saps com crear nous recursos utilitzant peticions POST i el cos de la petició!

Exercici 7: Actualitzar i Eliminar Usuaris

Objectiu: Completar les operacions CRUD (Create, Read, Update, Delete).

Passos detallats:

Pas 7.1: Afegir mètode d'actualització al servei

Afegeix aquests mètodes a `UsuariService`:

```
public Optional&lt;Usuari&gt; actualitzarUsuari(Long id, Usuari usuariActualitzat) {
    for (int i = 0; i &lt; usuaris.size(); i++) {
        Usuari usuari = usuaris.get(i);
        if (usuari.getId().equals(id)) {
            usuariActualitzat.setId(id);
            usuaris.set(i, usuariActualitzat);
            return Optional.of(usuariActualitzat);
        }
    }
    return Optional.empty();
}

public boolean eliminarUsuari(Long id) {
    return usuaris.removeIf(u -&gt; u.getId().equals(id));
}
```

Explicació del codi:

- `actualitzarUsuari`: Cerca l'usuari per ID i el reemplaça amb les noves dades
- `eliminarUsuari`: Elimina l'usuari si existeix i retorna true/false

Pas 7.2: Afegir endpoints PUT i DELETE

Afegeix aquests mètodes a `UsuariController`:

```
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.DeleteMapping;

@PutMapping("/{id}")
public ResponseEntity&lt;Usuari&gt; actualitzarUsuari(
        @PathVariable Long id,
        @RequestBody Usuari usuari) {
    return usuariService.actualitzarUsuari(id, usuari)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
}
```

```
@DeleteMapping("/{id}")
public ResponseEntity&lt;Void&gt; eliminarUsuari(@PathVariable Long id) {
    boolean eliminat = usuariService.eliminarUsuari(id);
    if (eliminat) {
        return ResponseEntity.noContent().build();
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

Explicació del codi:

- `@PutMapping`: Gestiona peticions HTTP PUT (actualització completa)
- `@DeleteMapping`: Gestiona peticions HTTP DELETE
- `ResponseEntity.noContent()`: Retorna 204 No Content (èxit sense contingut)

Pas 7.3: Provar l'actualització

Amb Postman:

1. Mètode: PUT
2. URL: http://localhost:8080/api/usuaris/1
3. Body (JSON):

```
{
  "nom": "Maria Garcia Actualitzada",
  "email": "maria.nova@exemple.com",
  "edat": 26
}
```

Amb curl:

```
curl -X PUT http://localhost:8080/api/usuaris/1 \
  -H "Content-Type: application/json" \
  -d '{"nom":"Maria Garcia Actualitzada","email":"maria.nova@exemple.com","edat":26}'
```

Pas 7.4: Provar l'eliminació

Amb Postman:

1. Mètode: DELETE
2. URL: http://localhost:8080/api/usuaris/1

Amb curl:

```
curl -X DELETE http://localhost:8080/api/usuaris/1
```

Pas 7.5: Verificar l'eliminació

1. Ves a http://localhost:8080/api/usuaris
2. L'usuari amb ID 1 ja no hauria d'aparèixer a la llista

Pas 7.6: Crear un fitxer de test complet

Crea `usuaris-api-test.http`:

```
### Llistar tots els usuaris
GET http://localhost:8080/api/usuaris

### Crear un usuari
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Test User",
  "email": "test@exemple.com",
  "edat": 25
}

### Obtenir un usuari
GET http://localhost:8080/api/usuaris/1

### Actualitzar un usuari
PUT http://localhost:8080/api/usuaris/1
Content-Type: application/json

{
  "nom": "Test User Actualitzat",
  "email": "test.nou@exemple.com",
  "edat": 26
}

### Eliminar un usuari
DELETE http://localhost:8080/api/usuaris/1
```

Resultat esperat:
Ja tens un CRUD complet! Pots crear, llegir, actualitzar i eliminar usuaris.

Exercici 8: Validació de Dades

Objectiu: Aprendre a validar les dades que reben els endpoints.

Passos detallats:

Pas 8.1: Afegir la dependència de validació

Si no la tens, afegeix aquesta dependència al `pom.xml`:

```
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
```

```
    &lt;artifactId&gt;spring-boot-starter-validation&lt;/artifactId&gt;
&lt;/dependency&gt;
```

Després, actualitza el projecte Maven (IntelliJ: botó dret sobre el projecte → Maven → Reload Project).

Pas 8.2: Afegir anotacions de validació al model

Modifica la classe `Usuari`:

```
package com.exemple.primerprojecte.model;

import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

public class Usuari {
    private Long id;

    @NotBlank(message = "El nom no pot estar buit")
    private String nom;

    @NotBlank(message = "L'email no pot estar buit")
    @Email(message = "L'email ha de tenir un format vàlid")
    private String email;

    @NotNull(message = "L'edat és obligatòria")
    @Min(value = 18, message = "L'edat mínima és 18 anys")
    private Integer edat;

    // Constructors, getters i setters...
    // (mantén els que ja tenies)
}
```

Explicació de les anotacions:

- `@NotBlank`: El camp no pot ser null, buit ni només espais
- `@Email`: Valida que tingui format d'email
- `@NotNull`: El camp no pot ser null
- `@Min`: El valor mínim acceptat
- `message`: Missatge personalitzat d'error

Pas 8.3: Activar la validació al controlador

Modifica els mètodes POST i PUT de `UsuariController`:

```
import jakarta.validation.Valid;

@PostMapping
@ResponseStatus(HttpStatus.CREATED)
```

```
public Usuari crearUsuari(@Valid @RequestBody Usuari usuari) {
    return usuariService.crearUsuari(usuari);
}

@PutMapping("/{id}")
public ResponseEntity&lt;Usuari&gt; actualitzarUsuari(
        @PathVariable Long id,
        @Valid @RequestBody Usuari usuari) {
    return usuariService.actualitzarUsuari(id, usuari)
            .map(ResponseEntity::ok)
            .orElse(ResponseEntity.notFound().build());
}
```

Explicació del codi:

- `@Valid`: Activa la validació de l'objecte segons les anotacions

Pas 8.4: Provar validacions amb dades incorrectes

Test 1 - Nom buit:

```
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "",
  "email": "test@exemple.com",
  "edat": 25
}
```

Resposta esperada: Error 400 Bad Request amb el missatge "El nom no pot estar buit"

Test 2 - Email invàlid:

```
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Test",
  "email": "email-invalid",
  "edat": 25
}
```

Resposta esperada: Error amb "L'email ha de tenir un format vàlid"

Test 3 - Edat menor de 18:

```
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Test",
  "email": "test@exemple.com",
```

```
  "edat": 15
}
```

Resposta esperada: Error amb "L'edat mínima és 18 anys"

Pas 8.5: Millorar les respostes d'error

Crea una classe per gestionar errors de validació. Crea el paquet `exception` i dins crea `GlobalExceptionHandler`:

```
package com.exemple.primerprojecte.exception;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; handleValidationErrors(
            MethodArgumentNotValidException ex) {

        Map&lt;String, String&gt; errors = new HashMap&lt;&gt;();
        ex.getBindingResult().getAllErrors().forEach((error) -&gt; {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });

        Map&lt;String, Object&gt; response = new HashMap&lt;&gt;();
        response.put("timestamp", LocalDateTime.now());
        response.put("status", HttpStatus.BAD_REQUEST.value());
        response.put("errors", errors);

        return new ResponseEntity&lt;&gt;(response, HttpStatus.BAD_REQUEST);
    }
}
```

Explicació del codi:

- `@RestControllerAdvice`: Gestiona excepcions de forma global
- `@ExceptionHandler`: Captura un tipus específic d'excepció
- Retorna un JSON estructurat amb tots els errors de validació

Pas 8.6: Provar les respostes millorades

Envia dades invàlides i veuràs una resposta com aquesta:

```
{
  "timestamp": "2025-10-22T23:00:00",
  "status": 400,
  "errors": {
    "nom": "El nom no pot estar buit",
    "email": "L'email ha de tenir un format vàlid",
    "edat": "L'edat mínima és 18 anys"
  }
}
```

Resultat esperat:
Ara tens validació completa de dades amb missatges d'error clars i estructurats!

Exercici 9: Integrar Base de Dades H2

Objectiu: Aprendre a utilitzar una base de dades real (en memòria) amb JPA.

Passos detallats:

Pas 9.1: Afegir les dependències necessàries

Afegeix aquestes dependències al `pom.xml`:

```
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-data-jpa&lt;/artifactId&gt;
&lt;/dependency&gt;

&lt;dependency&gt;
    &lt;groupId&gt;com.h2database&lt;/groupId&gt;
    &lt;artifactId&gt;h2&lt;/artifactId&gt;
    &lt;scope&gt;runtime&lt;/scope&gt;
&lt;/dependency&gt;
```

Actualitza Maven després d'afegir-les.

Pas 9.2: Configurar la base de dades H2

Obre `src/main/resources/application.properties` i afegeix:

```
# Configuració de H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

# Configuració JPA
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true

# Consola H2
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

Explicació de la configuració:

- `jdbc:h2:mem:testdb`: Base de dades H2 en memòria anomenada "testdb"
- `ddl-auto=update`: Crea/actualitza les taules automàticament
- `show-sql=true`: Mostra les queries SQL a la consola
- H2 Console: Interfície web per veure la base de dades

Pas 9.3: Convertir Usuari en una entitat JPA

Modifica la classe `Usuari`:

```
package com.exemple.primerprojecte.model;

import jakarta.persistence.*;
import jakarta.validation.constraints.Email;
import jakarta.validation.constraints.Min;
import jakarta.validation.constraints.NotBlank;
import jakarta.validation.constraints.NotNull;

@Entity
@Table(name = "usuaris")
public class Usuari {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotBlank(message = "El nom no pot estar buit")
    @Column(nullable = false)
    private String nom;

    @NotBlank(message = "L'email no pot estar buit")
    @Email(message = "L'email ha de tenir un format vàlid")
    @Column(nullable = false, unique = true)
    private String email;

    @NotNull(message = "L'edat és obligatòria")
    @Min(value = 18, message = "L'edat mínima és 18 anys")
    @Column(nullable = false)
    private Integer edat;

    // Constructors
    public Usuari() {
    }

    public Usuari(String nom, String email, Integer edat) {
        this.nom = nom;
```

```
        this.email = email;
        this.edat = edat;
    }

    // Getters i Setters (mantén els que ja tenies)
}
```

Explicació de les anotacions JPA:

- `@Entity`: Marca aquesta classe com una entitat de base de dades
- `@Table(name = "usuaris")`: Nom de la taula a la base de dades
- `@Id`: Marca el camp com a clau primària
- `@GeneratedValue`: La base de dades generarà automàticament l'ID
- `@Column`: Especifica propietats de la columna (nullable, unique, etc.)

Pas 9.4: Crear el repositori

Crea un nou paquet `repository` i dins crea `UsuariRepository`:

```
package com.exemple.primerprojecte.repository;

import com.exemple.primerprojecte.model.Usuari;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.List;
import java.util.Optional;

@Repository
public interface UsuariRepository extends JpaRepository&lt;Usuari, Long&gt; {

    // Spring Data JPA crea automàticament la implementació
    Optional&lt;Usuari&gt; findByEmail(String email);
    List&lt;Usuari&gt; findByNomContainingIgnoreCase(String nom);
}
```

Explicació del codi:

- `JpaRepository<Usuari, Long>`: Proporciona mètodes CRUD automàtics
- `findByEmail`: Spring crea automàticament la query SQL
- `findByNomContainingIgnoreCase`: Cerca parcial sense distingir majúscules

Pas 9.5: Actualitzar el servei per utilitzar el repositori

Modifica `UsuariService`:

```
package com.exemple.primerprojecte.service;

import com.exemple.primerprojecte.model.Usuari;
import com.exemple.primerprojecte.repository.UsuariRepository;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;
import java.util.Optional;

@Service
public class UsuariService {

    @Autowired
    private UsuariRepository usuariRepository;

    public List&lt;Usuari&gt; obtenirTotsElsUsuaris() {
        return usuariRepository.findAll();
    }

    public Optional&lt;Usuari&gt; obtenirUsuariPerId(Long id) {
        return usuariRepository.findById(id);
    }

    public Usuari crearUsuari(Usuari usuari) {
        return usuariRepository.save(usuari);
    }

    public Optional&lt;Usuari&gt; actualitzarUsuari(Long id, Usuari usuariActualitzat) {
        return usuariRepository.findById(id)
                .map(usuari -&gt; {
                    usuari.setNom(usuariActualitzat.getNom());
                    usuari.setEmail(usuariActualitzat.getEmail());
                    usuari.setEdat(usuariActualitzat.getEdat());
                    return usuariRepository.save(usuari);
                });
    }

    public boolean eliminarUsuari(Long id) {
        if (usuariRepository.existsById(id)) {
            usuariRepository.deleteById(id);
            return true;
        }
        return false;
    }

    public List&lt;Usuari&gt; cercarPerNom(String nom) {
        return usuariRepository.findByNomContainingIgnoreCase(nom);
    }

    public Optional&lt;Usuari&gt; obtenirUsuariPerEmail(String email) {
        return usuariRepository.findByEmail(email);
    }
}
```

Pas 9.6: Executar i provar

1. Inicia l'aplicació
2. A la consola veuràs les queries SQL que creen la taula

3. Prova els endpoints com abans:
    - POST per crear usuaris
    - GET per veure'ls
    - Etc.

Pas 9.7: Accedir a la consola H2

1. Ves a http://localhost:8080/h2-console
2. Configuració:
    - JDBC URL: `jdbc:h2:mem:testdb`
    - User Name: `sa`
    - Password: (deixa-ho buit)
3. Fes clic a "Connect"
4. Pots veure la taula `USUARIS` i fer queries SQL directament!

Pas 9.8: Afegir dades inicials

Crea una classe per inicialitzar dades. Al paquet arrel, crea `DataInitializer`:

```
package com.exemple.primerprojecte;

import com.exemple.primerprojecte.model.Usuari;
import com.exemple.primerprojecte.repository.UsuariRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class DataInitializer implements CommandLineRunner {

    @Autowired
    private UsuariRepository usuariRepository;

    @Override
    public void run(String... args) throws Exception {
        // Crear alguns usuaris inicials
        usuariRepository.save(new Usuari("Maria Garcia", "maria@exemple.com", 25));
        usuariRepository.save(new Usuari("Joan Martí", "joan@exemple.com", 30));
        usuariRepository.save(new Usuari("Anna Puig", "anna@exemple.com", 28));

        System.out.println("Dades inicials carregades!");
    }
}
```

Resultat esperat:
Ara tens una base de dades real que persisteix les dades! Pots crear, modificar i eliminar usuaris, i les dades es guarden a H2.

Exercici 10: Gestió d'Excepcions Avançada

Objectiu: Aprendre a gestionar errors de forma professional amb excepcions personalitzades.

Passos detallats:

Pas 10.1: Crear excepcions personalitzades

Crea aquestes classes al paquet `exception`:

UsuariNoTrobatException.java:

```
package com.exemple.primerprojecte.exception;

public class UsuariNoTrobatException extends RuntimeException {
    public UsuariNoTrobatException(Long id) {
        super("No s'ha trobat l'usuari amb ID: " + id);
    }
}
```

EmailDuplicatException.java:

```
package com.exemple.primerprojecte.exception;

public class EmailDuplicatException extends RuntimeException {
    public EmailDuplicatException(String email) {
        super("Ja existeix un usuari amb l'email: " + email);
    }
}
```

Pas 10.2: Crear una classe per a respostes d'error

Crea `ErrorResponse.java` al paquet `exception`:

```
package com.exemple.primerprojecte.exception;

import java.time.LocalDateTime;

public class ErrorResponse {
    private LocalDateTime timestamp;
    private int status;
    private String error;
    private String message;
    private String path;

    public ErrorResponse(int status, String error, String message, String path) {
        this.timestamp = LocalDateTime.now();
        this.status = status;
        this.error = error;
        this.message = message;
        this.path = path;
    }
```

```
    // Getters i Setters
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }

    public int getStatus() {
        return status;
    }

    public void setStatus(int status) {
        this.status = status;
    }

    public String getError() {
        return error;
    }

    public void setError(String error) {
        this.error = error;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }

    public String getPath() {
        return path;
    }

    public void setPath(String path) {
        this.path = path;
    }
}
```

Pas 10.3: Ampliar el GlobalExceptionHandler

Actualitza `GlobalExceptionHandler` per gestionar les noves excepcions:

```
package com.exemple.primerprojecte.exception;

import jakarta.servlet.http.HttpServletRequest;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ExceptionHandler;
```

```
import org.springframework.web.bind.annotation.RestControllerAdvice;

import java.time.LocalDateTime;
import java.util.HashMap;
import java.util.Map;

@RestControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(UsuariNoTrobatException.class)
    public ResponseEntity&lt;ErrorResponse&gt; handleUsuariNoTrobat(
            UsuariNoTrobatException ex,
            HttpServletRequest request) {

        ErrorResponse error = new ErrorResponse(
                HttpStatus.NOT_FOUND.value(),
                "Usuari No Trobat",
                ex.getMessage(),
                request.getRequestURI()
        );

        return new ResponseEntity&lt;&gt;(error, HttpStatus.NOT_FOUND);
    }

    @ExceptionHandler(EmailDuplicatException.class)
    public ResponseEntity&lt;ErrorResponse&gt; handleEmailDuplicat(
            EmailDuplicatException ex,
            HttpServletRequest request) {

        ErrorResponse error = new ErrorResponse(
                HttpStatus.CONFLICT.value(),
                "Email Duplicat",
                ex.getMessage(),
                request.getRequestURI()
        );

        return new ResponseEntity&lt;&gt;(error, HttpStatus.CONFLICT);
    }

    @ExceptionHandler(MethodArgumentNotValidException.class)
    public ResponseEntity&lt;Map&lt;String, Object&gt;&gt; handleValidationErrors(
            MethodArgumentNotValidException ex,
            HttpServletRequest request) {

        Map&lt;String, String&gt; errors = new HashMap&lt;&gt;();
        ex.getBindingResult().getAllErrors().forEach((error) -&gt; {
            String fieldName = ((FieldError) error).getField();
            String errorMessage = error.getDefaultMessage();
            errors.put(fieldName, errorMessage);
        });

        Map&lt;String, Object&gt; response = new HashMap&lt;&gt;();
        response.put("timestamp", LocalDateTime.now());
        response.put("status", HttpStatus.BAD_REQUEST.value());
        response.put("error", "Validació Fallida");
        response.put("errors", errors);
```

```
        response.put("path", request.getRequestURI());

        return new ResponseEntity&lt;&gt;(response, HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(Exception.class)
    public ResponseEntity&lt;ErrorResponse&gt; handleGeneralException(
            Exception ex,
            HttpServletRequest request) {

        ErrorResponse error = new ErrorResponse(
                HttpStatus.INTERNAL_SERVER_ERROR.value(),
                "Error Intern del Servidor",
                ex.getMessage(),
                request.getRequestURI()
        );

        return new ResponseEntity&lt;&gt;(error, HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Pas 10.4: Modificar el servei per llançar excepcions

Actualitza `UsuariService`:

```
public Usuari crearUsuari(Usuari usuari) {
    // Comprovar si l'email ja existeix
    if (usuariRepository.findByEmail(usuari.getEmail()).isPresent()) {
        throw new EmailDuplicatException(usuari.getEmail());
    }
    return usuariRepository.save(usuari);
}

public Optional&lt;Usuari&gt; obtenirUsuariPerId(Long id) {
    return Optional.ofNullable(
        usuariRepository.findById(id)
            .orElseThrow(() -&gt; new UsuariNoTrobatException(id))
    );
}

public Optional&lt;Usuari&gt; actualitzarUsuari(Long id, Usuari usuariActualitzat) {
    Usuari usuari = usuariRepository.findById(id)
            .orElseThrow(() -&gt; new UsuariNoTrobatException(id));

    // Comprovar si l'email nou ja existeix en un altre usuari
    usuariRepository.findByEmail(usuariActualitzat.getEmail())
            .ifPresent(u -&gt; {
                if (!u.getId().equals(id)) {
                    throw new EmailDuplicatException(usuariActualitzat.getEmail());
                }
            });

    usuari.setNom(usuariActualitzat.getNom());
    usuari.setEmail(usuariActualitzat.getEmail());
    usuari.setEdat(usuariActualitzat.getEdat());
```

```
    return Optional.of(usuariRepository.save(usuari));
}

public boolean eliminarUsuari(Long id) {
    if (!usuariRepository.existsById(id)) {
        throw new UsuariNoTrobatException(id);
    }
    usuariRepository.deleteById(id);
    return true;
}
```

Pas 10.5: Provar les excepcions

Test 1 - Usuari no trobat:

```
GET http://localhost:8080/api/usuaris/999
```

Resposta esperada:

```
{
  "timestamp": "2025-10-22T23:15:00",
  "status": 404,
  "error": "Usuari No Trobat",
  "message": "No s'ha trobat l'usuari amb ID: 999",
  "path": "/api/usuaris/999"
}
```

Test 2 - Email duplicat:

Primer, crea un usuari:

```
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Test",
  "email": "test@exemple.com",
  "edat": 25
}
```

Després, intenta crear-ne un altre amb el mateix email:

```
POST http://localhost:8080/api/usuaris
Content-Type: application/json

{
  "nom": "Altre Test",
  "email": "test@exemple.com",
```

```
  "edat": 30
}
```

Resposta esperada:

```
{
  "timestamp": "2025-10-22T23:16:00",
  "status": 409,
  "error": "Email Duplicat",
  "message": "Ja existeix un usuari amb l'email: test@exemple.com",
  "path": "/api/usuaris"
}
```

Resultat esperat:
Tens un sistema complet de gestió d'errors amb respostes clares i codis HTTP apropiats!

Exercici 11: Paginació i Ordenació

Objectiu: Aprendre a gestionar llistes grans de dades amb paginació i ordenació.

Passos detallats:

Pas 11.1: Actualitzar el repositori

El repositori ja hereta de `JpaRepository`, que ja suporta paginació. No cal canviar res!

Pas 11.2: Actualitzar el servei

Afegeix aquests mètodes a `UsuariService`:

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;

public Page&lt;Usuari&gt; obtenirUsuarisPaginats(Pageable pageable) {
    return usuariRepository.findAll(pageable);
}

public List&lt;Usuari&gt; obtenirUsuarisOrdenats(String camp, String direccio) {
    Sort sort = direccio.equalsIgnoreCase("asc")
        ? Sort.by(camp).ascending()
        : Sort.by(camp).descending();
    return usuariRepository.findAll(sort);
}
```

Pas 11.3: Afegir endpoints al controlador

Afegeix aquests mètodes a `UsuariController`:

```
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Pageable;
import org.springframework.data.domain.Sort;

@GetMapping("/pagina")
public Page&lt;Usuari&gt; obtenirUsuarisPaginats(
        @RequestParam(defaultValue = "0") int pagina,
        @RequestParam(defaultValue = "10") int mida,
        @RequestParam(defaultValue = "id") String ordenarPer,
        @RequestParam(defaultValue = "asc") String direccio) {

    Sort sort = direccio.equalsIgnoreCase("asc")
            ? Sort.by(ordenarPer).ascending()
            : Sort.by(ordenarPer).descending();

    Pageable pageable = PageRequest.of(pagina, mida, sort);
    return usuariService.obtenirUsuarisPaginats(pageable);
}

@GetMapping("/ordenat")
public List&lt;Usuari&gt; obtenirUsuarisOrdenats(
        @RequestParam(defaultValue = "nom") String camp,
        @RequestParam(defaultValue = "asc") String direccio) {
    return usuariService.obtenirUsuarisOrdenats(camp, direccio);
}
```

Explicació dels paràmetres:

- `pagina`: Número de pàgina (comença a 0)
- `mida`: Quants elements per pàgina
- `ordenarPer`: Camp pel qual ordenar
- `direccio`: "asc" (ascendent) o "desc" (descendent)

Pas 11.4: Crear dades de prova

Modifica `DataInitializer` per crear més usuaris:

```
@Override
public void run(String... args) throws Exception {
    usuariRepository.save(new Usuari("Maria Garcia", "maria@exemple.com", 25));
    usuariRepository.save(new Usuari("Joan Martí", "joan@exemple.com", 30));
    usuariRepository.save(new Usuari("Anna Puig", "anna@exemple.com", 28));
    usuariRepository.save(new Usuari("Pere Soler", "pere@exemple.com", 35));
    usuariRepository.save(new Usuari("Laura Vidal", "laura@exemple.com", 27));
    usuariRepository.save(new Usuari("Marc Font", "marc@exemple.com", 32));
    usuariRepository.save(new Usuari("Núria Bosch", "nuria@exemple.com", 29));
    usuariRepository.save(new Usuari("David Roca", "david@exemple.com", 31));
    usuariRepository.save(new Usuari("Sofia Camps", "sofia@exemple.com", 26));
    usuariRepository.save(new Usuari("Albert Serra", "albert@exemple.com", 33));
```

```
    System.out.println("Dades inicials carregades: " + usuariRepository.count() + " usuar
}
```

Pas 11.5: Provar la paginació

Pàgina 1 (primers 5 elements):

```
GET http://localhost:8080/api/usuaris/pagina?pagina=0&amp;mida=5
```

Pàgina 2 (següents 5 elements):

```
GET http://localhost:8080/api/usuaris/pagina?pagina=1&amp;mida=5
```

Resposta esperada:

```
{
  "content": [
    {
      "id": 1,
      "nom": "Maria Garcia",
      "email": "maria@exemple.com",
      "edat": 25
    },
    ...
  ],
  "pageable": {
    "pageNumber": 0,
    "pageSize": 5
  },
  "totalElements": 10,
  "totalPages": 2,
  "last": false,
  "first": true
}
```

Pas 11.6: Provar l'ordenació

Ordenar per nom (ascendent):

```
GET http://localhost:8080/api/usuaris/pagina?ordenarPer=nom&amp;direccio=asc
```

Ordenar per edat (descendent):

```
GET http://localhost:8080/api/usuaris/pagina?ordenarPer=edat&amp;direccio=desc
```

Pas 11.7: Combinar paginació i ordenació

```
GET http://localhost:8080/api/usuaris/pagina?pagina=0&amp;mida=3&amp;ordenarPer=edat&amp;
```

Això retornarà els 3 usuaris més grans.

Resultat esperat:
Pots gestionar grans quantitats de dades de forma eficient amb paginació i ordenació!

Exercici 12: Seguretat Bàsica amb Spring Security

Objectiu: Aprendre a protegir l'aplicació amb autenticació bàsica.

Passos detallats:

Pas 12.1: Afegir la dependència de Spring Security

Afegeix al `pom.xml`:

```
&lt;dependency&gt;
    &lt;groupId&gt;org.springframework.boot&lt;/groupId&gt;
    &lt;artifactId&gt;spring-boot-starter-security&lt;/artifactId&gt;
&lt;/dependency&gt;
```

Actualitza Maven i reinicia l'aplicació.

Pas 12.2: Provar l'aplicació

1. Intenta accedir a http://localhost:8080/api/usuaris
2. Veuràs una pàgina de login!
3. Spring Security s'ha activat automàticament

Credencials per defecte:

- Usuari: `user`
- Contrasenya: La trobaràs a la consola, busca una línia com:

```
Using generated security password: a1b2c3d4-e5f6-7890-abcd-ef1234567890
```

Pas 12.3: Crear una configuració de seguretat personalitzada

Crea un nou paquet `config` i dins crea `SecurityConfig`:

```
package com.exemple.primerprojecte.config;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity
import org.springframework.security.core.userdetails.User;
```

```
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.provisioning.InMemoryUserDetailsManager;
import org.springframework.security.web.SecurityFilterChain;

@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Bean
    public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
        http
            .csrf(csrf -&gt; csrf.disable()) // Desactivar CSRF per simplificar (NO fer e
            .authorizeHttpRequests(auth -&gt; auth
                .requestMatchers("/h2-console/**").permitAll() // Permetre accés a H2
                .requestMatchers("/api/usuaris/**").authenticated() // Requereix autentic
                .anyRequest().permitAll()
            )
            .httpBasic(); // Autenticació HTTP Basic

        // Permetre H2 Console en frames
        http.headers(headers -&gt; headers.frameOptions(frame -&gt; frame.disable()));

        return http.build();
    }

    @Bean
    public UserDetailsService userDetailsService() {
        UserDetails admin = User.builder()
                .username("admin")
                .password(passwordEncoder().encode("admin123"))
                .roles("ADMIN")
                .build();

        UserDetails user = User.builder()
                .username("user")
                .password(passwordEncoder().encode("user123"))
                .roles("USER")
                .build();

        return new InMemoryUserDetailsManager(admin, user);
    }

    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

Explicació del codi:

- `@EnableWebSecurity`: Activa Spring Security
- `securityFilterChain`: Defineix les regles de seguretat

- `.requestMatchers("/api/usuaris/**").authenticated()`: Requereix autenticació per a aquests endpoints
- `userDetailsService`: Crea usuaris en memòria
- `BCryptPasswordEncoder`: Xifra les contrasenyes de forma segura

Pas 12.4: Provar l'autenticació

Amb Postman:

1. Crea una petició GET a http://localhost:8080/api/usuaris
2. A la pestanya "Authorization":
   - Type: Basic Auth
   - Username: `admin`
   - Password: `admin123`
3. Envia la petició

Amb curl:

```
curl -u admin:admin123 http://localhost:8080/api/usuaris
```

Pas 12.5: Afegir autorització per rols

Modifica `SecurityConfig`:

```
@Bean
public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {
    http
        .csrf(csrf -&gt; csrf.disable())
        .authorizeHttpRequests(auth -&gt; auth
            .requestMatchers("/h2-console/**").permitAll()
            .requestMatchers("/api/usuaris/**").hasAnyRole("USER", "ADMIN")
            .requestMatchers("/api/admin/**").hasRole("ADMIN")
            .anyRequest().permitAll()
        )
        .httpBasic();

    http.headers(headers -&gt; headers.frameOptions(frame -&gt; frame.disable()));

    return http.build();
}
```

Pas 12.6: Crear un endpoint només per administradors

Crea `AdminController`:

```
package com.exemple.primerprojecte.controller;

import com.exemple.primerprojecte.model.Usuari;
import com.exemple.primerprojecte.repository.UsuariRepository;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.Map;

@RestController
@RequestMapping("/api/admin")
public class AdminController {

    @Autowired
    private UsuariRepository usuariRepository;

    @GetMapping("/estadistiques")
    public Map&lt;String, Object&gt; obtenirEstadistiques() {
        long totalUsuaris = usuariRepository.count();
        double edatMitjana = usuariRepository.findAll().stream()
                .mapToInt(Usuari::getEdat)
                .average()
                .orElse(0.0);

        return Map.of(
                "totalUsuaris", totalUsuaris,
                "edatMitjana", edatMitjana
        );
    }

    @DeleteMapping("/eliminar-tots")
    public Map&lt;String, String&gt; eliminarTotsElsUsuaris() {
        usuariRepository.deleteAll();
        return Map.of("missatge", "Tots els usuaris han estat eliminats");
    }
}
```

Pas 12.7: Provar els permisos

Amb usuari normal (user/user123):

```
# Això funcionarà
curl -u user:user123 http://localhost:8080/api/usuaris

# Això donarà error 403 Forbidden
curl -u user:user123 http://localhost:8080/api/admin/estadistiques
```

Amb administrador (admin/admin123):

```
# Això funcionarà
curl -u admin:admin123 http://localhost:8080/api/admin/estadistiques
```

Pas 12.8: Afegir informació de l'usuari autenticat

Afegeix aquest endpoint a `UsuariController`:

```
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;

@GetMapping("/jo")
public Map&lt;String, String&gt; usuariActual() {
    Authentication auth = SecurityContextHolder.getContext().getAuthentication();
    return Map.of(
            "usuari", auth.getName(),
            "rols", auth.getAuthorities().toString()
    );
}
```

Prova-ho:

```
curl -u admin:admin123 http://localhost:8080/api/usuaris/jo
```

Resposta:

```
{
  "usuari": "admin",
  "rols": "[ROLE_ADMIN]"
}
```

Resultat esperat:
Tens una aplicació segura amb autenticació i autorització bàsiques! Els usuaris han d'identificar-se i només els administradors poden accedir a certes funcions.

Resum i Propers Passos

Felicitats! Has completat els 12 exercicis guiats de Spring Boot. Ara saps:

1. ✅ Crear projectes Spring Boot
2. ✅ Crear controladors REST
3. ✅ Treballar amb paràmetres
4. ✅ Retornar JSON
5. ✅ Implementar serveis
6. ✅ Crear recursos amb POST
7. ✅ Actualitzar i eliminar (CRUD complet)
8. ✅ Validar dades
9. ✅ Utilitzar bases de dades amb JPA
10. ✅ Gestionar excepcions
11. ✅ Paginar i ordenar resultats
12. ✅ Implementar seguretat bàsica

Propers passos suggerits:

1. **Aprendre sobre relacions entre entitats** (One-to-Many, Many-to-Many)
2. **Implementar JWT** per autenticació més moderna
3. **Crear tests unitaris** amb JUnit i Mockito
4. **Documentar l'API** amb Swagger/OpenAPI
5. **Connectar amb bases de dades reals** (MySQL, PostgreSQL)
6. **Aprendre Docker** per desplegar l'aplicació
7. **Implementar microserveis** amb Spring Cloud

Recursos addicionals:

- Documentació oficial: https://spring.io/projects/spring-boot
- Spring Guides: https://spring.io/guides
- Baeldung Spring tutorials: https://www.baeldung.com/spring-boot

Autor: Exercicis creats per a estudiants de desenvolupament web
Data: Octubre 2025
Versió: 1.0