

1 – Web services para acesso a uma base de dados MySQL

Nesta ficha de trabalho é pressuposto que tem instalado na sua máquina o SGBD MySQL. Os exercícios desta ficha foram testados com o MySQL 8.0.28 e java 17.

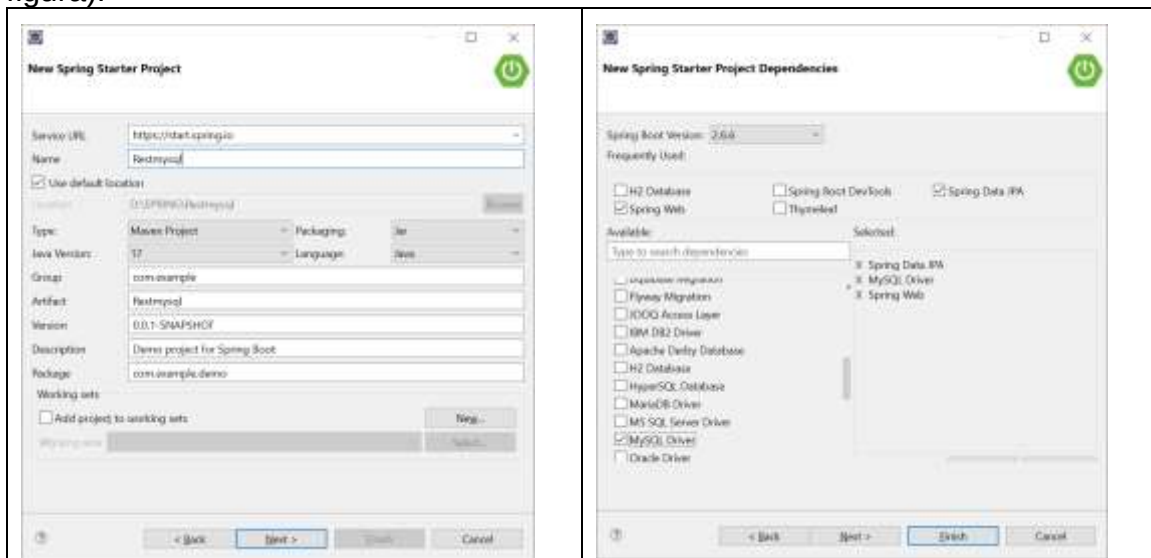
Para instalar a versão 8.0.28 (ou a mais recente disponível) em Windows, pode, por exemplo, seguir o vídeo: https://youtu.be/eq-e_n7lm2M

Vamos implementar um exemplo que cria e acede a uma base de dados MySQL seguindo o tutorial:

<https://spring.io/guides/gs/accessing-data-mysql/>

a) Como em FP08, em File, seleccionar “new / Spring starter project”.

- Dê um nome ao seu projeto, selecione a sua versão do Java e, após Next, selecione as dependências **Spring Web**, **Spring Data JPA** e **MySQL Driver** (ver figura).



b) Siga o tutorial nos passos: **Create the Database** e **Create the application.properties File** para criar a base de dados e o respetivo ficheiro de configuração.

Dependendo das versões que usar, no ficheiro application.properties pode ter que substituir a linha:

```
#5 spring.datasource.driver-class-name =com.mysql.jdbc.Driver
```

```
por: spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver)
```

- c) Criar agora a *entity User* que irá dar origem à tabela *User* na base de dados (passo **Create the @Entity Model**) do tutorial.
- d) Criar a interface *UserRepository* a partir do qual será gerado um bean (*userRepository*) com as operações *CRUD* de acesso à tabela *User* (passo **Create the Repository** do tutorial).
- e) Finalmente, **criar o controlador** que irá definir os “*endpoints*” que nos permitem aceder ao bean criado como um serviço web.
- f) A classe de inicialização não precisa de ser alterada e pode executar o seu exemplo com: Run as Spring Boot Application.

g) Testar os dois endpoints: GET localhost:8080/demo/all (obtém todos os utilizadores e POST localhost:8080/demo/add (permite adicionar um novo utilizador) Pode testar os dois URIs usando a ferramenta “client ULR” curl, pré-instalada no Windows 10 e 11.

Pode executá-los numa linha de comandos:

```
> curl localhost:8080/demo/add -d name=xxxx -d email= yyy@gmail.com  
> curl localhost:8080/demo/all
```

h) Acedendo ao MySQL Workbench, pode verificar que a base de dados foi criada. Pode ainda inserir novos dados e aceder aos dados inseridos.

2 – Interface Web para *web services* com acesso a base de dados MySQL

Notas: O exemplo foi construído com base no exemplo do tutorial do exemplo 1: (<https://spring.io/guides/gs/accessing-data-mysql/>), no exemplo ilustrado no vídeo (https://www.youtube.com/watch?v=_5sAmaRJd2c) e no site:

<https://www.javaguides.net/2020/05/spring-boot-crud-web-application-with-thymeleaf.html>

a) Comece por criar a base de dados db_escola. Em Windows, aceda ao mysql como root com o comando:

```
mysql -u root -p
```

Enter password: *****

De seguida criar a base de dados e um utilizador com privilégios para manipular essa base de dados:

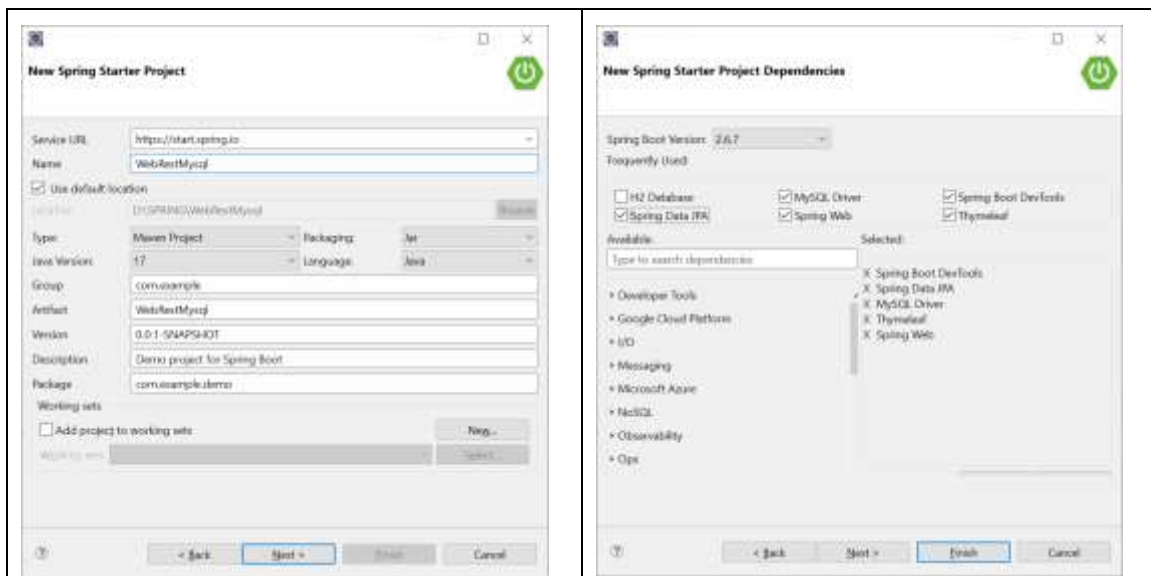
mysql> create database db_escola; -- Creates the new database

mysql> create user 'springuser2'@'%' identified by 'ThePassword'; -- Creates the user

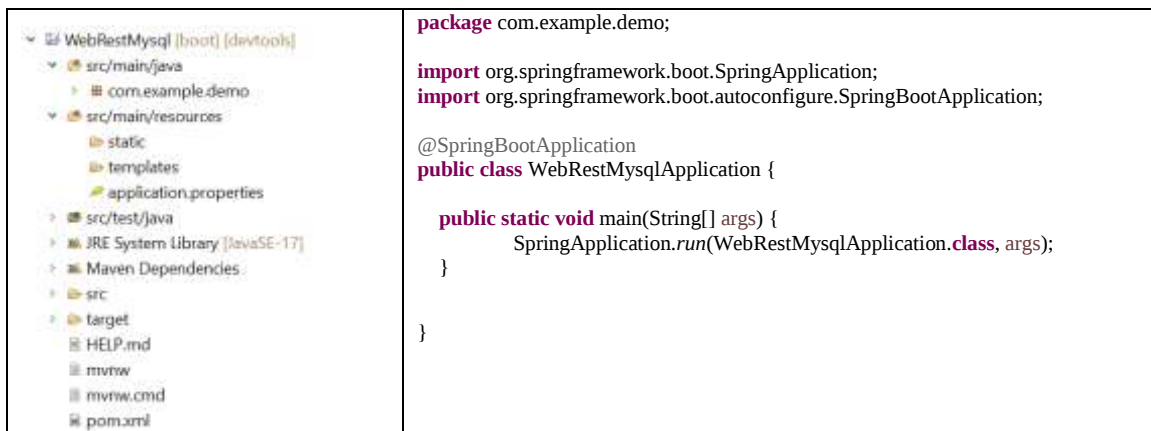
mysql> grant all on db_escola.* to 'springuser2'@'%'; -- Gives all privileges to the new user on the newly created database

b) Como anteriormente, em File, selecionar “new / Spring starter project”.

- Dê um nome ao seu projeto, WebRestMysql, selecione a sua versão do Java e após Next selecione as dependências **Spring Web**, **Spring Data JPA**, **MySQL Driver**, **Thymeleaf** e **Spring Boot DevTools** (ver figura).



Observe que foi criado um projeto com a estrutura da figura abaixo contendo a classe com o *main* da aplicação (*Application Class*).



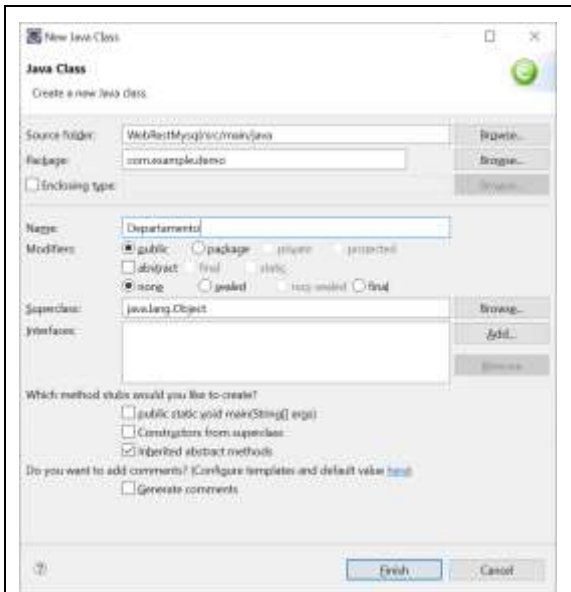
c) Definir a ligação à base de dados editando o ficheiro application.properties com o seguinte conteúdo:

```
spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://${MYSQL_HOST:localhost}:3306/db_escola
spring.datasource.username=springuser2
spring.datasource.password=ThePassword
spring.datasource.driver-class-name = com.mysql.cj.jdbc.Driver
```

Suponhamos uma base de dados que contém as tabelas Departamento e Professor. Um Departamento tem vários Professores e um Professor pertence a um Departamento.

d) Definir a entity Departamento

Criar uma nova classe e inserir o código abaixo (direita). Complete a classe gerando em “source” os getters, setters e o método toString. Esta entity irá dar origem à tabela departamento na base de dados db_escola.

	<pre>package com.example.demo; import jakarta.persistence.Entity; import jakarta.persistence.Id; @Entity // This tells Hibernate to make a table out of this class public class Departamento { @Id //depId is the primary key private Integer depId; private String nome;</pre>
---	--

e) Criar a interface DepartamentoRepository

Criar uma nova interface com o código da figura (à direita). A partir desta interface será gerado de forma automática o Bean depRepository a usar no controller.



```

package com.example.demo;

import org.springframework.data.repository.CrudRepository;

//This will be AUTO IMPLEMENTED by Spring into a
//Bean called departamentoRepository
//CRUD refers Create, Read, Update, Delete

public interface DepartamentoRepository extends
CrudRepository<Departamento, Integer> {

}

```

f) Criar a classe MainController, com o código abaixo. A classe é um Controller, declara o Bean depRepository e define o webService getAllDeps.

```

package com.example.demo;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller // This means that this class is a Controller
public class MainController {

    @Autowired // This means to get the bean called depRepository
                // Which is auto-generated by Spring, we will use it to handle
the data
    private DepartamentoRepository depRepository;

    @GetMapping(path="/")
    public String getAllDeps (Model model) {
        model.addAttribute("ListDeps" , depRepository.findAll());
        return "index";
    }
}

```

g) Criar o template index.html.

Este template, em *Thymeleaf*, permite gerar código html do lado do servidor e é retornado pelo método `getAllDeps` criado no Controller.

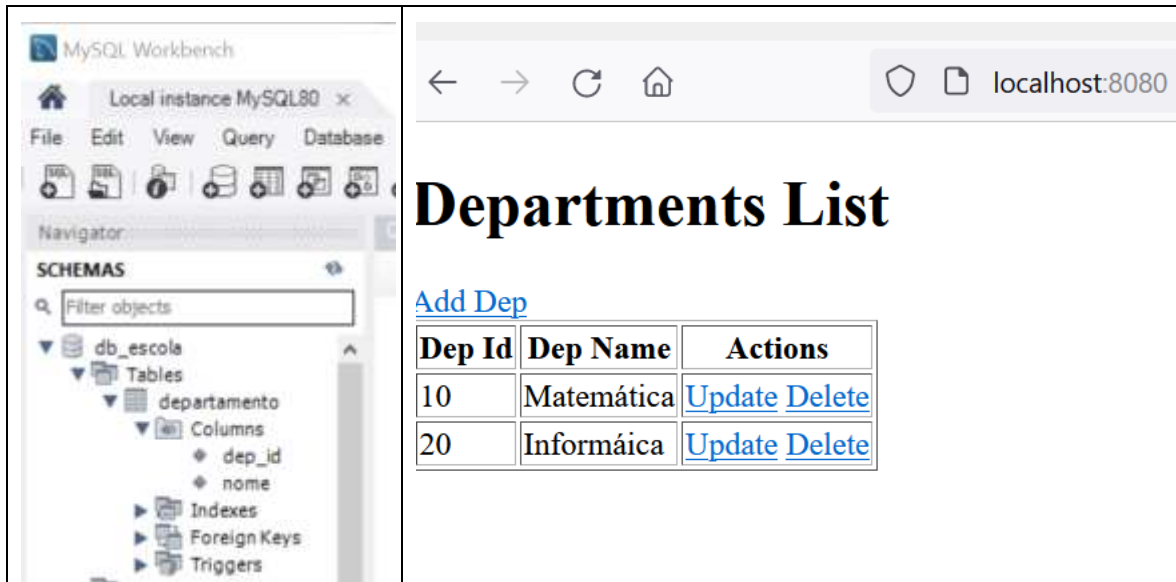
No package `resources/templates` criar um ficheiro `index.html` com o conteúdo ilustrado abaixo.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Departments management</title>
</head>
<body>
<div>
  <h1> Departments List </h1>
  <a th:href="@{/showNewDepForm}"> Add Dep </a>
  <table border="1" class="table table-striped table-responsive-md">
    <thead>
      <tr>
        <th>Dep Id</th>
        <th>Dep Name</th>
        <th> Actions </th>
      </tr>
    </thead>
    <tbody>
      <tr th:each="dep : ${ListDeps}">
        <td th:text="${dep.depId}"></td>
        <td th:text="${dep.nome}"></td>
        <td>
          <a th:href="@{/showUpdateDepForm/{id}(id=${dep.depId})}" >Update</a>
          <a th:href="@{/deleteDep/{id}(id=${dep.depId})}" >Delete</a>
        </td>
      </tr>
    </tbody>
  </table>
</div>
</body>
</html>
```

Estude o código em *thymeleaf*, e execute o projeto com `Run as Spring Boot Application`.

De seguida no seu browser aceda ao endereço: <http://localhost:8080/>, deverá obter um primeiro esquema da sua página web ainda sem dados.

- Aceda agora ao MySQL Workbench e verifique que na base de dados db_escola foi criada a tabela departamento, com as colunas dep_id e nome. Insira algumas linhas de dados e faça *refresh* no seu browser. Deverá agora obter a lista de todos os departamentos inseridos.



The image shows two side-by-side screenshots. The left screenshot is of MySQL Workbench, displaying the 'Schemas' pane with the 'db_escola' database selected. Under 'Tables', the 'departamento' table is visible, showing columns 'dep_id' and 'nome'. The right screenshot is a web browser at 'localhost:8080' displaying a page titled 'Departments List'. It includes a link 'Add Dep' and a table with the following data:

Dep Id	Dep Name	Actions
10	Matemática	Update Delete
20	Informática	Update Delete

h) Inserir novo Departamento

- Na classe MainController adicione os seguintes métodos showNewDepForm e saveDep listados abaixo.

```
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;
...
@GetMapping("/showNewDepForm")
public String showNewDepForm(Model model) {
    // create model attribute to bind form data
    Departamento depar = new Departamento();
    model.addAttribute("newdep", depar);
    return "new_dep";
}

@PostMapping("/saveDep")
public String saveDep (@ModelAttribute("newdep") Departamento dep ) {
    // save dep to database
    depRepository.save(dep);
    return "redirect:/";
}
```

- No package resources/templates criar um ficheiro html, new_dep.html com o conteúdo ilustrado abaixo.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Departments management</title>
</head>
<body>
<div>
    <h1>Department Management System</h1>
    <hr>
    <h2>Save Department</h2>

    <form action="#" th:action="@{/saveDep}" th:object="${newdep}"
method="POST">

        <input type="text" th:field="*{depId}" placeholder="Dep id" >

        <input type="text" th:field="*{nome}" placeholder="Dep Name" >

        <button type="submit" > Save Dep</button>
    </form>

    <hr>

    <a th:href="@{/}"> Back to Department List</a>

</div>
</body>
</html>
```

Estude os novos métodos e a nova página criada.

Pode agora executar e inserir novos departamentos pela interface criada:

Department Management System Save Department 40 Informática Save Dep Back to Departments List	Departments List Add Dep <table border="1"> <thead> <tr> <th>Dep Id</th> <th>Dep Name</th> <th>Actions</th> </tr> </thead> <tbody> <tr> <td>10</td> <td>Matemática</td> <td>Update Delete</td> </tr> <tr> <td>20</td> <td>Informática</td> <td>Update Delete</td> </tr> <tr> <td>30</td> <td>Física</td> <td>Update Delete</td> </tr> <tr> <td>40</td> <td>Química</td> <td>Update Delete</td> </tr> </tbody> </table>	Dep Id	Dep Name	Actions	10	Matemática	Update Delete	20	Informática	Update Delete	30	Física	Update Delete	40	Química	Update Delete
Dep Id	Dep Name	Actions														
10	Matemática	Update Delete														
20	Informática	Update Delete														
30	Física	Update Delete														
40	Química	Update Delete														

i) Eliminar um Departamento

- Na classe MainController adicione o método deleteDep ilustrado abaixo:

```
import org.springframework.web.bind.annotation.PathVariable;
...
@GetMapping("/deleteDep/{id}")
public String deleteDep(@PathVariable(value = "id") Integer id) {
    depRepository.deleteById(id);
    return "redirect:/";
}
```

- Teste a aplicação, eliminando alguns registos.

j) Atualizar dados dos departamentos

- Na classe MainController adicione o método showUpdateDepForm ilustrado abaixo.

```
import java.util.Optional;
...
@GetMapping("/showUpdateDepForm/{id}")
@GetMapping("/showUpdateDepForm/{id}")
public String showUpdateDepForm(@PathVariable(value = "id") Integer id,
Model model) {
    Optional < Departamento > optional = depRepository.findById(id);
    Departamento dep = null;
    if (optional.isPresent()) {
        dep = optional.get();
    } else {
        throw new RuntimeException(" Department not found for id :: " + id);
    }
    // set department as a model attribute to pre-populate the form
    model.addAttribute("depart", dep);
    return "update_dep";
}
```

- No package resources/templates criar um ficheiro html, update_dep.html, com o conteúdo ilustrado abaixo.

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta charset="ISO-8859-1">
<title>Users management</title>
</head>
<body>
<div>
    <h1>Department Management System</h1>
    <hr>
    <h2>Update Department</h2>

    <form action="#" th:action="@{/saveDep}" th:object="${depart}"
method="POST">

        <!-- Add hidden form field to handle update -->
        <input type="hidden" th:field="*{depId}" />

        <input type="text" th:field="*{nome}" >

        <button type="submit" > Update Dep</button>
    </form>

    <hr>

    <a th:href="@{/}"> Back to User List</a>

</div>
</body>
</html>
```

Estude o código, e teste aplicação.

- Depois de perceber o exemplo, explore como criar a entity Professor de forma a que a tabela professor que será criada na base de dados tenha uma associação many-to-one com a tabela Departamento. Complete a aplicação de forma a poder realizar as operações CRUD também na tabela professor.

Links para explorar:

Spring

<https://spring.io/>

Especificação do JAVA EE

<https://jakarta.ee/release/>

<https://jakarta.ee/specifications/platform/9/jakarta-platform-spec-9.pdf>

<https://eclipse-ee4j.github.io/jakartaee-tutorial/>

Java Persistence API

<https://jakarta.ee/specifications/persistence/3.0/jakarta-persistence-spec-3.0.html>

Thymeleaf

<https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>

Acesso à base de dados:

Após concluir o desenvolvimento das suas aplicações, deve alterar os direitos de acesso à sua base de dados.

(Retirado de <https://spring.io/guides/gs/accessing-data-mysql/>)

In application.properties: `spring.jpa.hibernate.ddl-auto=none`

```
mysql> revoke all on db_example.* from 'springuser'@'%';
```

```
mysql> grant select, insert, delete, update on db_example.* to 'springuser'@'%';
```

When you want to make changes to the database:

1. Regrant permissions.
2. Change the `spring.jpa.hibernate.ddl-auto` to `update`.
3. Re-run your applications.