



# SPRING DATA

FAPESC – DESENVOLVEDORES PARA TECNOLOGIA DA INFORMAÇÃO

HERCULANO DE BIASI

[herculano.debiasi@unoesc.edu.br](mailto:herculano.debiasi@unoesc.edu.br)



# TÓPICOS

- Motivação
- Spring Data
- Repositórios
- Configuração
- *Query builder*
- Anotação `@Query`
- Paginação de resultados
- Exemplo H2

# MOTIVAÇÃO

- Muitos SGBDs relacionais e não relacionais (NoSQL) no mercado



# MOTIVAÇÃO

## ■ JDBC vs. JPA vs MongoDB: Vários modelos diferentes de programação

```
public void insere(TipoProduto tipoProduto){
    Connection con = new ConexaoUtil().criarConexao();
    PreparedStatement stmt;
    try {
        stmt = con.prepareStatement("insert into tip_produto "
            + " (NOMTIPPRODUTO, CLASSIFICACAO) "
            + " values (?,?)");
        stmt.setString(1, tipoProduto.getNome());
        stmt.setString(2, tipoProduto.getClassificacao());
        stmt.executeUpdate();
    } catch (SQLException ex) {
        //Aqui tratar erro
    }
}
```



```
private void inserir(ModelClass objeto) {
    JPAUtil.getEm().getTransaction().begin();
    JPAUtil.getEm().persist(objeto);
    JPAUtil.getEm().getTransaction().commit();
}

public List<TipoProduto> getTipoProduto(String nome){
    return JPAUtil.getEm().createQuery("select p from "
        + "TipoProduto p where p.nome like :nome")
        .setParameter("nome", nome)
        .getResultList();
}
```

```
Mongo mongo = new Mongo();
DB db = mongo.getDB("tutorial");
DBCollection employees = db.getCollection("employees");
employees.insert(new BasicDBObject().append("name", "Diana Hightower")
    .append("gender", "f").append("phone", "520-555-1212").append("age", 30));
DBCursor cursor = employees.find();
while (cursor.hasNext()) {
    DBObject object = cursor.next();
    System.out.println(object);
}
```

# MOTIVAÇÃO

- Unificação em torno de um único modelo



The diagram illustrates the motivation for Spring Data by showing a variety of databases grouped into two layers. The top layer, enclosed in a light green oval, contains logos for Hadoop, MongoDB, CouchDB, Neo4j, RAVENDB, and Redis. The bottom layer, also enclosed in a light green oval, contains logos for Apache HBASE, Cassandra, Oracle, MySQL, and PostgreSQL. A small cartoon character is visible in the bottom right corner of the diagram. The text 'Spring Data' is prominently displayed in the center, overlapping both layers.

# Spring Data

# MOTIVAÇÃO

- Unificação em torno de um único modelo

- MongoDB

```
@Document
public class Pessoa{

    @Id
    private BigInteger id;

    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

- Redis

```
@RedisHash("Pessoa")
public class Pessoa{

    @Id
    private BigInteger id;

    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

- Jpa (MariaDb)

```
@Entity
public class Pessoa{

    @Id
    private BigInteger id;

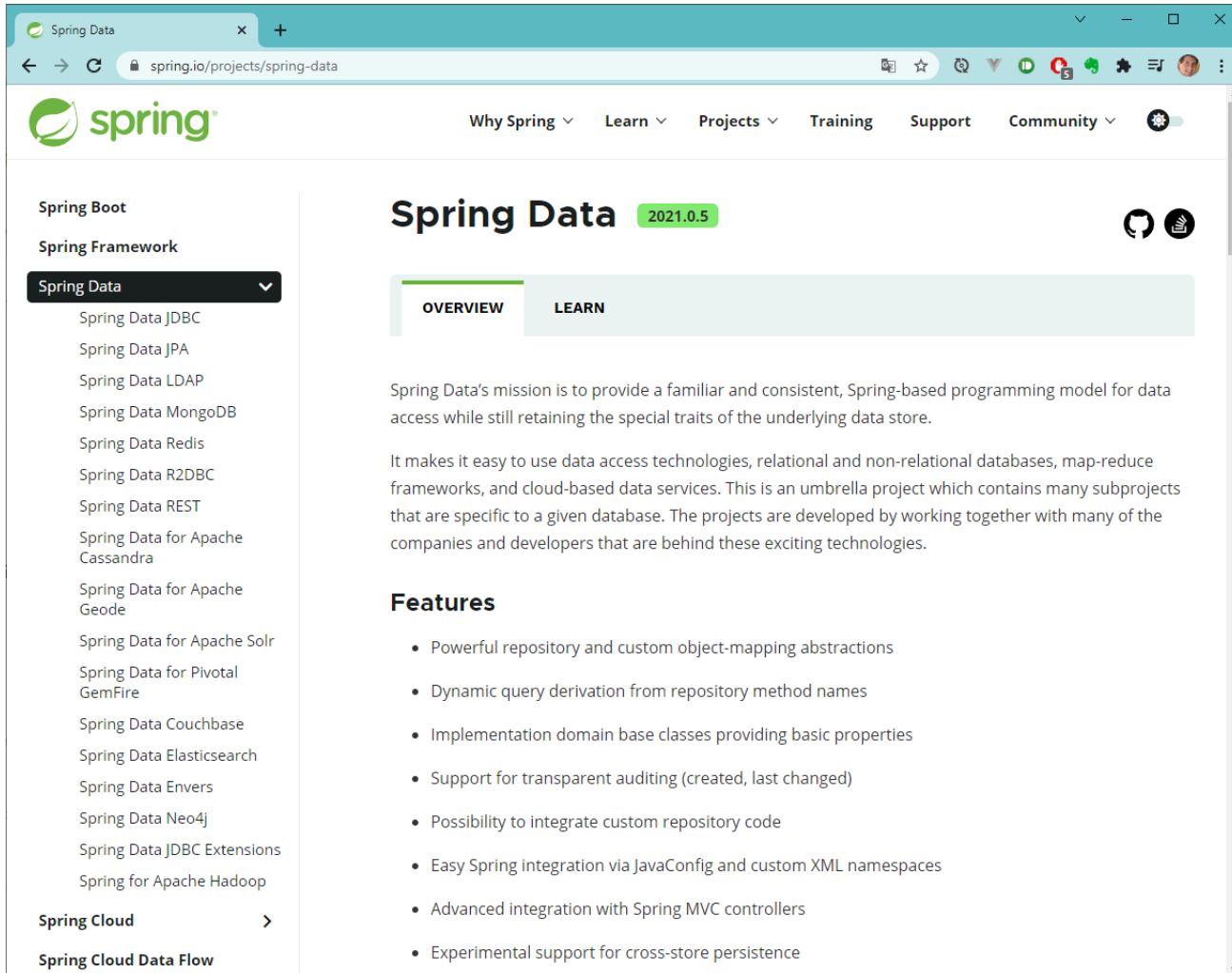
    private String nome;

    private int idade;

    //Metodos get/set e construtores
}
```

# SPRING DATA

- Spring Data é o modelo de programação dentro do Spring Framework para acesso e manipulação de dados



The screenshot shows the Spring Data website. The left sidebar contains a navigation menu with the following items: Spring Boot, Spring Framework, Spring Data (selected), Spring Data JDBC, Spring Data JPA, Spring Data LDAP, Spring Data MongoDB, Spring Data Redis, Spring Data R2DBC, Spring Data REST, Spring Data for Apache Cassandra, Spring Data for Apache Geode, Spring Data for Apache Solr, Spring Data for Pivotal GemFire, Spring Data Couchbase, Spring Data Elasticsearch, Spring Data Envers, Spring Data Neo4j, Spring Data JDBC Extensions, and Spring for Apache Hadoop. The main content area is titled "Spring Data" with a version badge "2021.0.5". It features two tabs: "OVERVIEW" and "LEARN". The "OVERVIEW" tab is active, displaying the mission statement: "Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store." It also includes a paragraph about the project's scope and a list of features.

## Spring Data 2021.0.5

### OVERVIEW

Spring Data's mission is to provide a familiar and consistent, Spring-based programming model for data access while still retaining the special traits of the underlying data store.

It makes it easy to use data access technologies, relational and non-relational databases, map-reduce frameworks, and cloud-based data services. This is an umbrella project which contains many subprojects that are specific to a given database. The projects are developed by working together with many of the companies and developers that are behind these exciting technologies.

### Features

- Powerful repository and custom object-mapping abstractions
- Dynamic query derivation from repository method names
- Implementation domain base classes providing basic properties
- Support for transparent auditing (created, last changed)
- Possibility to integrate custom repository code
- Easy Spring integration via JavaConfig and custom XML namespaces
- Advanced integration with Spring MVC controllers
- Experimental support for cross-store persistence

# SPRING DATA

- O objetivo do Spring Data é simplificar o desenvolvimento de aplicações que usam diferentes tecnologias de acesso a dados (bancos relacionais e não relacionais (NoSQL), *frameworks* [map-reduce](#), serviços de dados baseados em nuvem, etc), além do suporte aperfeiçoado a bancos de dados relacionais
- O Spring Data fornece grande suporte para as necessidades de persistência, trazendo já prontas operações CRUD, com as quais pode-se criar, ler, alterar e deletar dados, contando também com funcionalidades como paginação e ordenação de dados
- Alguns subprojetos do Spring Data
  - Spring Data JPA: Para bancos relacionais, como MySQL/MariaDB e PostgreSQL
  - Spring MongoDB: Banco de dados orientado a documentos
  - Spring Data Redis: Banco de dados do tipo chave-valor, em memória

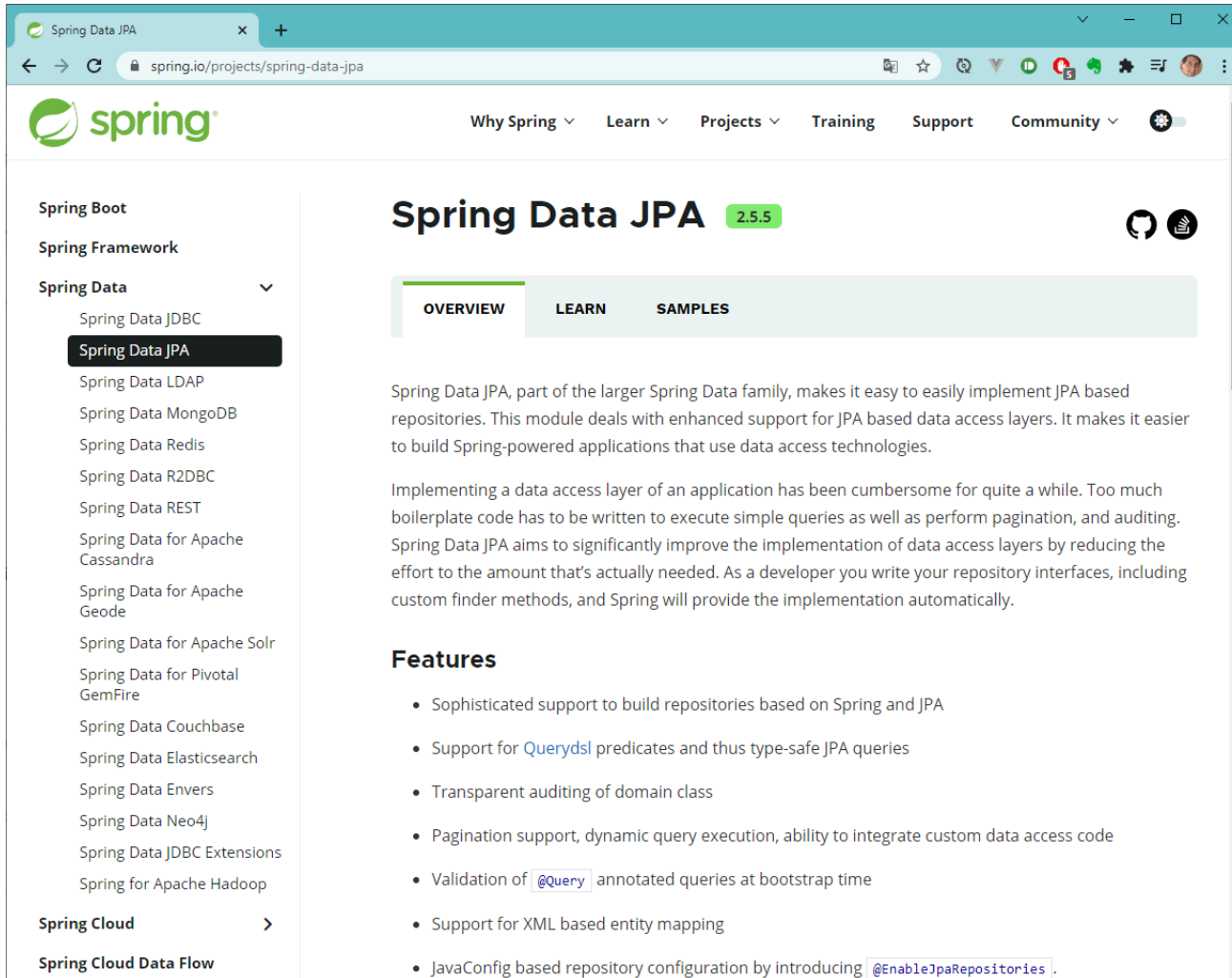


# SPRING DATA

- O Spring Data Commons é o projeto que provê a infraestrutura compartilhada por todos os outros projetos do Spring Data
  - Contém *interfaces* e meta modelos de dados para as classes de persistência Java
  - Geração de *queries* (consultas) dinâmicas
  - Suporte a auditoria transparente (criação, última mudança)
  - Possibilidade de integrar código customizado de repositórios

# SPRING DATA

- O principal componente do Spring Data é o [Spring Data JPA](#), que implementa o modelo em camadas baseado em repositórios



The screenshot shows the Spring Data JPA project page on the Spring.io website. The page is titled "Spring Data JPA" with a version indicator "2.5.5". The left sidebar lists various Spring Data projects, with "Spring Data JPA" highlighted. The main content area has tabs for "OVERVIEW", "LEARN", and "SAMPLES". The "OVERVIEW" tab is active, displaying a description of Spring Data JPA and a list of features.

**Spring Data JPA 2.5.5**

**OVERVIEW** | LEARN | SAMPLES

Spring Data JPA, part of the larger Spring Data family, makes it easy to easily implement JPA based repositories. This module deals with enhanced support for JPA based data access layers. It makes it easier to build Spring-powered applications that use data access technologies.

Implementing a data access layer of an application has been cumbersome for quite a while. Too much boilerplate code has to be written to execute simple queries as well as perform pagination, and auditing. Spring Data JPA aims to significantly improve the implementation of data access layers by reducing the effort to the amount that's actually needed. As a developer you write your repository interfaces, including custom finder methods, and Spring will provide the implementation automatically.

### Features

- Sophisticated support to build repositories based on Spring and JPA
- Support for [Querydsl](#) predicates and thus type-safe JPA queries
- Transparent auditing of domain class
- Pagination support, dynamic query execution, ability to integrate custom data access code
- Validation of `@Query` annotated queries at bootstrap time
- Support for XML based entity mapping
- JavaConfig based repository configuration by introducing `@EnableJpaRepositories`

# REPOSITÓRIOS

- Um repositório no Spring Data é uma abstração destinada a reduzir de forma significativa a quantidade de *boiler plate* necessária para implementar camadas de acesso a dados para diferentes sistemas de armazenamento de dados
- A interface `CrudRepository` provê funcionalidades de CRUD sofisticadas para a entidade que está sendo gerenciada

```
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);  
  
    // ... more functionality omitted.  
}
```

- ① Saves the given entity.
- ② Returns the entity identified by the given id.
- ③ Returns all entities.
- ④ Returns the number of entities.
- ⑤ Deletes the given entity.
- ⑥ Indicates whether an entity with the given id exists.

# REPOSITÓRIOS

- Implementação de um repositório JPA

```
public interface ProdutoRepository  
    extends JpaRepository<Produto, Integer> {
```

```
}
```

```
@Service  
public class ProdutoRegrasImpl implements ProdutoRegras {
```

```
@Autowired  
private ProdutoRepository produtoRepository;
```

```
public void salvar(Produto produto){  
    produtoRepository.save(produto);  
}
```

```
}
```

# CONFIGURAÇÃO

## ■ Dependências no *pom.xml*

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
  <groupId>org.postgresql</groupId>
  <artifactId>postgresql</artifactId>
</dependency>
```

## ■ Arquivo *application.properties*

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.jpa.show-sql=true
3 spring.jpa.properties.dialect=org.hibernate.dialect.PostgreSQLDialect
4 spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults=false
5 spring.datasource.url=jdbc:postgresql://localhost:5432/eleicao
6 spring.datasource.username=postgres
7 spring.datasource.password=postgres
8 spring.datasource.driverClassName=org.postgresql.Driver
```

# CONFIGURAÇÃO

- Há 5 opções do Spring para gerenciar a evolução do banco de dados
  - `update`: Sempre que a aplicação iniciar o Spring Data verifica se as classes entidades estão de acordo com o banco – caso não estejam, será feito um *update* no banco adicionando novas colunas na tabela dessa entidade
    - É importante notar que o *update* não remove ou renomeia colunas
    - Por isso, não é recomendado de jeito algum utilizar o Spring Data para gerenciar a evolução do seu banco, para isso existem ferramentas especializadas, por exemplo, o FlyWay
  - `create`: Sempre que a aplicação for iniciada o Spring Data vai apagar tudo e recriar novamente do zero
  - `create-drop`: Semelhante ao `create`, mas sempre que a aplicação é parada ele apaga tudo que foi criado
  - `validate`: Faz uma validação sempre que a aplicação inicia e verifica se o banco confere com as suas classes de entidade
  - `none`: Basicamente é nenhuma das opções acima, ou seja, significa que não se quer que o Spring Data faça alterações no banco – é a recomendada para produção, para que não ocorra problemas do Spring modificar o banco

# QUERY BUILDER

- *Query builder*: Construção de uma *query* utilizando somente o nome do método

```
public interface FilmeRepository
    extends JpaRepository<Filme, Long> {

    List<Filme> findByName(String nome);
}
```

# QUERY BUILDER

## ■ Métodos do query builder

Keyword	Sample
And	<code>findByLastnameAndFirstname</code>
Or	<code>findByLastnameOrFirstname</code>
Between	<code>findByStartDateBetween</code>
LessThan	<code>findByAgeLessThan</code>
GreaterThan	<code>findByAgeGreaterThan</code>
IsNull	<code>findByAgeIsNull</code>
IsNotNull,NotNull	<code>findByAge(Is)NotNull</code>
Like	<code>findByFirstnameLike</code>
NotLike	<code>findByFirstnameNotLike</code>
OrderBy	<code>findByAgeOrderByLastnameDesc</code>
Not	<code>findByLastnameNot</code>
In	<code>findByAgeIn(Collection&lt;Age&gt; ages)</code>
NotIn	<code>findByAgeNotIn(Collection&lt;Age&gt; age)</code>



# ANOTAÇÃO @QUERY

- Com a anotação @Query é possível utilizar **JPQL** (Java Persistence Query Language) para a escrita de consultas personalizadas

```
public interface UserRepository extends JpaRepository<User, Long> {  
  
    @Query("select u from User u where u.firstname = :firstname or u.lastname = :lastname")  
    User findByLastnameOrFirstname(@Param("lastname") String lastname,  
                                    @Param("firstname") String firstname);  
}
```

# PAGINAÇÃO DE RESULTADOS

- Com as classes `Page` e `Pageable` pode-se fazer paginação nos resultados das consultas de maneira bem fácil

```
public interface ClienteRepository
    extends JpaRepository<Cliente, Long> {

    Page<Cliente> findByNomeLike(String nome, Pageable pageable);

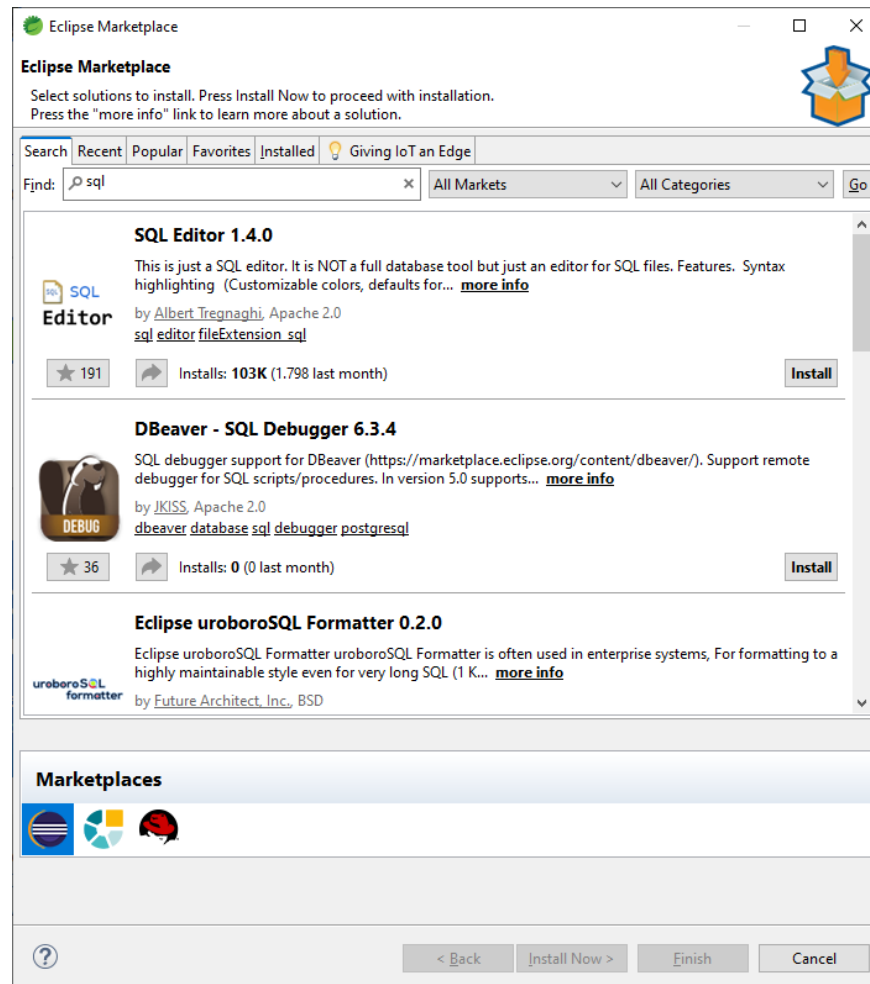
}
```

- Utilização

```
Pageable paginacao = new PageRequest(page, pageSize);
Page<Cliente> clientes = clienteRepository.findAll(paginacao);
List<Cliente> clientesList = clientes.getContent();
```

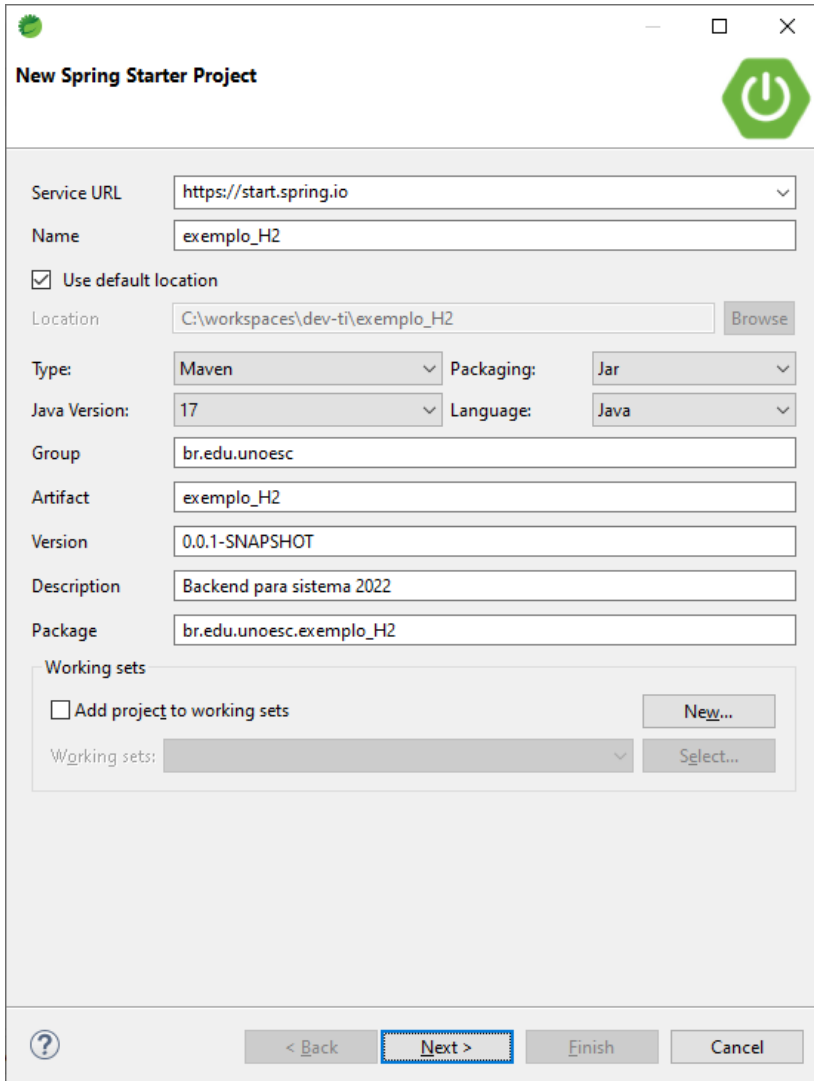
# EXEMPLO H2

- Instale o plug-in SQL Editor (*Help → Eclipse Marketplace*)



# EXEMPLO H2

## ■ Importe o arquivo como um projeto do Maven



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

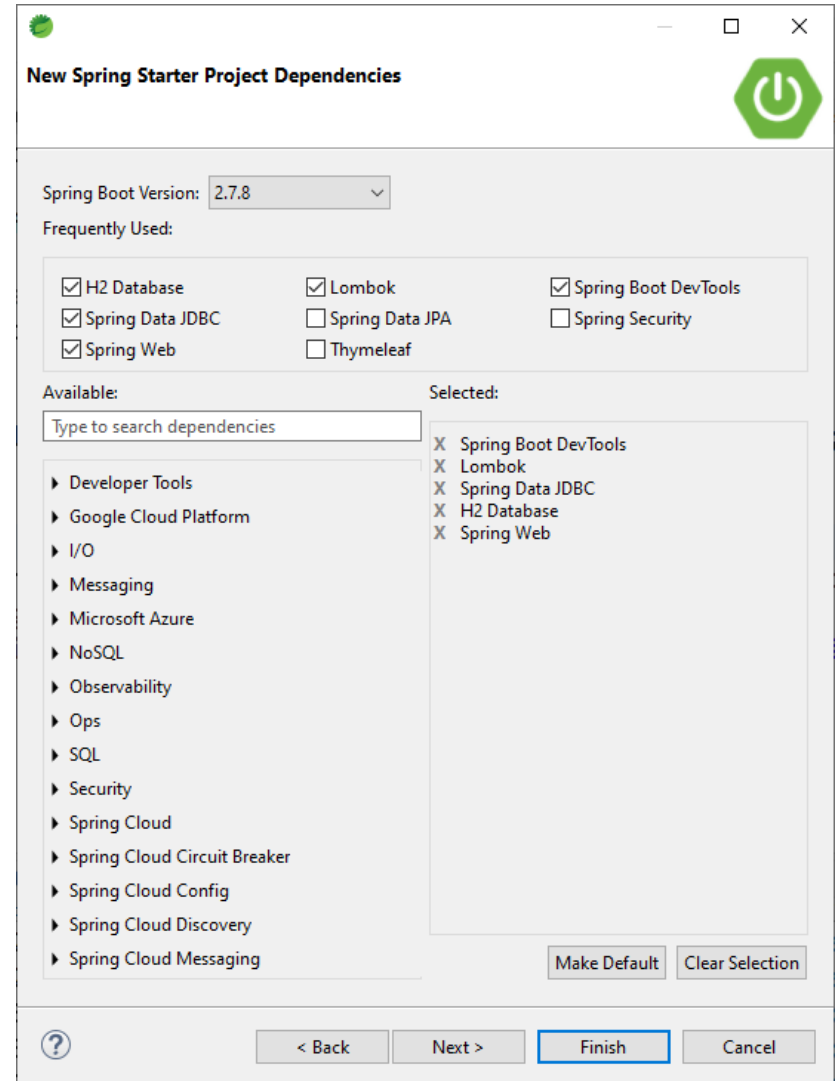
Description:

Package:

Working sets

☐ Add project to working sets

Working sets:



**New Spring Starter Project Dependencies**

Spring Boot Version:

Frequently Used:

<input checked="" type="checkbox"/> H2 Database	<input checked="" type="checkbox"/> Lombok	<input checked="" type="checkbox"/> Spring Boot DevTools
<input checked="" type="checkbox"/> Spring Data JDBC	<input type="checkbox"/> Spring Data JPA	<input type="checkbox"/> Spring Security
<input checked="" type="checkbox"/> Spring Web	<input type="checkbox"/> Thymeleaf	

Available:

Type to search dependencies

- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops
- SQL
- Security
- Spring Cloud
- Spring Cloud Circuit Breaker
- Spring Cloud Config
- Spring Cloud Discovery
- Spring Cloud Messaging

Selected:

- X Spring Boot DevTools
- X Lombok
- X Spring Data JDBC
- X H2 Database
- X Spring Web

# EXEMPLO H2

## ■ Dependências

- Webjars são bibliotecas web (como jQuery ou Bootstrap), empacotadas em arquivos *jar*
- São uma opção interessante para aplicações sem acesso à internet, pois não é preciso fazer nenhum *download*

The screenshot shows the WebJars website (https://www.webjars.org) in a browser. The page has a dark header with the WebJars logo and navigation links. Below the header, there's a section titled "WebJars are client-side web libraries (e.g. jQuery & Bootstrap) packaged into JAR (Java Archive) files." followed by a bulleted list of features and deployment information. Below this, a section titled "WebJars come in four flavors:" lists four categories: NPM WebJars, Bower GitHub WebJars, Classic WebJars, and Bower Original WebJars, each with specific details about their packaging and group/artifact IDs. At the bottom, there's a "Search Results" section with a search bar containing "jquery" and a filter for "Classic" selected. The search results table shows one result: "Alpaca - Easy Forms for jQuery" with version 1.5.24 and 6428 files. The table also includes a "Build Tool" column with various options like SBT, Maven, Ivy, etc.

WebJars - Web Libraries in Jars

https://www.webjars.org

WebJars Documentation Sponsored by Heroku

WebJars are client-side web libraries (e.g. jQuery & Bootstrap) packaged into JAR (Java Archive) files.

- Explicitly and easily manage the client-side dependencies in JVM-based web applications
- Use JVM-based build tools (e.g. Maven, Gradle, sbt, ...) to download your client-side dependencies
- Know which client-side dependencies you are using
- Transitive dependencies are automatically resolved and optionally loaded via RequireJS
- Deployed on [Maven Central](#)
- Public CDN, generously provided by: [JSDELIVR](#)

WebJars come in four flavors:

**NPM WebJars**

- Contents mirror NPM package
- GroupId: `org.webjars.npm`
- ArtifactId: NPM Package or URL-based Name

**Bower GitHub WebJars**

- Contents mirror Bower package
- GroupId: `org.webjars.bowergithub.[GITHUB_ORG]`
- ArtifactId: GitHub Repo Name

**Classic WebJars**

- Custom Built and Manually Deployed
- GroupId: `org.webjars`
- ArtifactId: Varies

**Bower Original WebJars**

- **Deprecated** Use Bower GitHub WebJars instead
- GroupId: `org.webjars.bower`
- ArtifactId: Bower Package or URL-based Name

Search Results

Add a WebJar

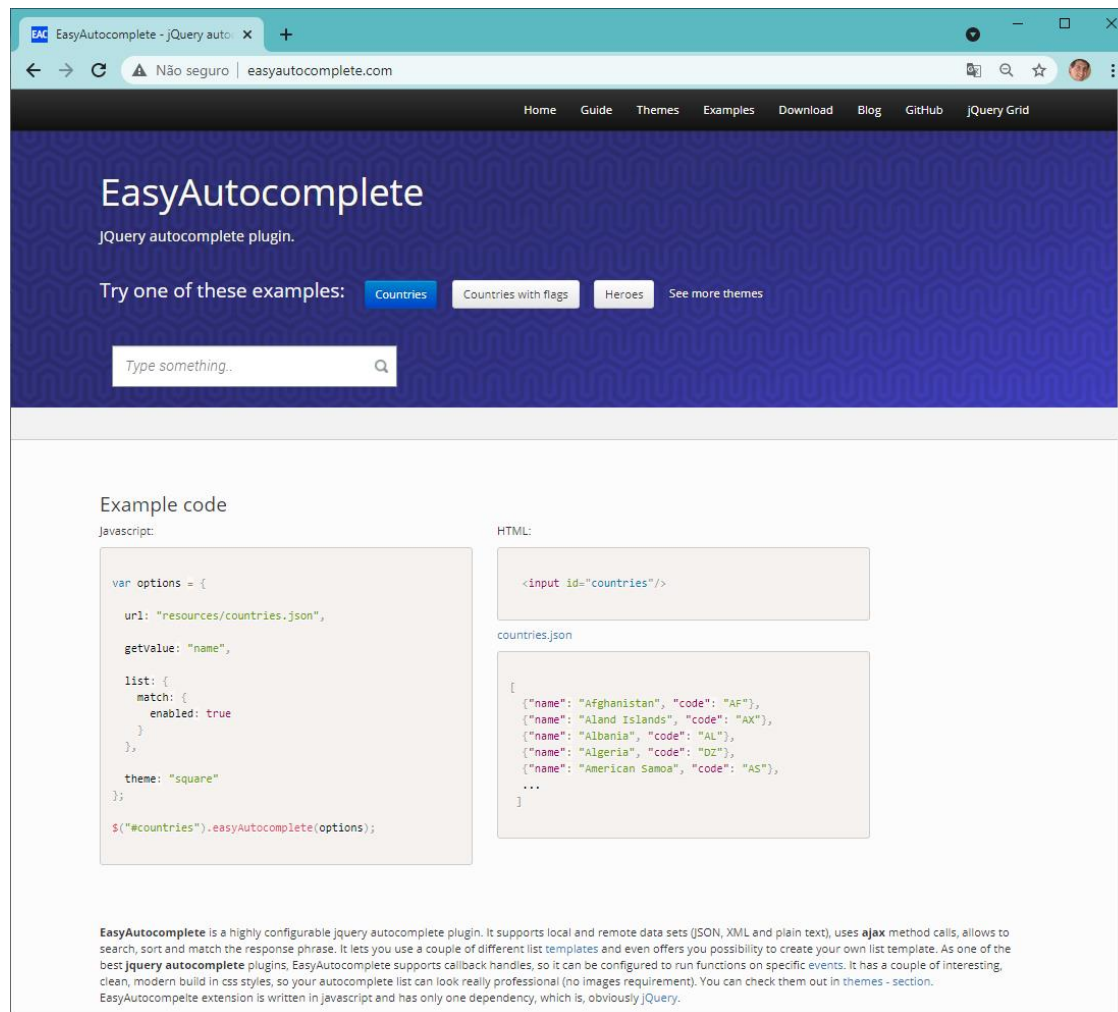
☐ NPM ☐ Bower GitHub ☐ Bower Original ☒ Classic

jquery

Name	Versions	Build Tool: SBT / Play 2 Maven Ivy Grape Gradle Buildr Leiningen	Files
<a href="#">Alpaca - Easy Forms for jQuery</a>	1.5.24	<pre>&lt;dependency&gt; &lt;groupId&gt;org.webjars&lt;/groupId&gt; &lt;artifactId&gt;alpaca&lt;/artifactId&gt; &lt;version&gt;1.5.24&lt;/version&gt; &lt;/dependency&gt;</pre>	6428 Files

# EXEMPLO H2

■ EasyAutocomplete é um *plugin autocomplete* para jQuery



The screenshot shows the EasyAutocomplete website. The header includes navigation links: Home, Guide, Themes, Examples, Download, Blog, GitHub, and jQuery Grid. The main content area features the title "EasyAutocomplete" and the subtitle "jQuery autocomplete plugin." Below this, there are buttons for "Countries", "Countries with flags", "Heroes", and "See more themes". A search input field with the placeholder "Type something.." is also present. The "Example code" section displays the following code:

**Javascript:**

```
var options = {  
  url: "resources/countries.json",  
  getValue: "name",  
  list: {  
    match: {  
      enabled: true  
    }  
  },  
  theme: "square"  
};  
$("#countries").easyAutocomplete(options);
```

**HTML:**

```
<input id="countries"/>
```

**countries.json**

```
[  
  {"name": "Afghanistan", "code": "AF"},  
  {"name": "Aland Islands", "code": "AX"},  
  {"name": "Albania", "code": "AL"},  
  {"name": "Algeria", "code": "DZ"},  
  {"name": "American Samoa", "code": "AS"},  
  ...  
]
```

**EasyAutocomplete** is a highly configurable jquery autocomplete plugin. It supports local and remote data sets (JSON, XML and plain text), uses **ajax** method calls, allows to search, sort and match the response phrase. It lets you use a couple of different list templates and even offers you possibility to create your own list template. As one of the best **jquery autocomplete** plugins, EasyAutocomplete supports callback handles, so it can be configured to run functions on specific events. It has a couple of interesting, clean, modern build in css styles, so your autocomplete list can look really professional (no images requirement). You can check them out in **themes** - section. EasyAutocomplete extension is written in javascript and has only one dependency, which is, obviously jQuery.

# EXEMPLO H2

## ■ Configuração do arquivo *pom.xml*

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
3   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://m
4   <modelVersion>4.0.0</modelVersion>
5   <parent>
6     <groupId>org.springframework.boot</groupId>
7     <artifactId>spring-boot-starter-parent</artifactId>
8     <version>2.7.8</version>
9     <relativePath/> <!-- lookup parent from repository -->
10  </parent>
11  <groupId>br.edu.unoesc</groupId>
12  <artifactId>exemplo_H2</artifactId>
13  <version>0.0.1-SNAPSHOT</version>
14  <name>exemplo_H2</name>
15  <description>Backend para sistema 2022</description>
16  <properties>
17    <java.version>17</java.version>
18  </properties>
19  <dependencies>
20    <dependency>
21      <groupId>org.springframework.boot</groupId>
22      <artifactId>spring-boot-devtools</artifactId>
23      <scope>runtime</scope>
24      <optional>true</optional>
25    </dependency>
26
27    <dependency>
28      <groupId>org.springframework.boot</groupId>
29      <artifactId>spring-boot-starter-data-jdbc</artifactId>
30    </dependency>
31    <dependency>
32      <groupId>com.h2database</groupId>
33      <artifactId>h2</artifactId>
34      <scope>runtime</scope>
35    </dependency>
36
37    <dependency>
38      <groupId>org.springframework.boot</groupId>
39      <artifactId>spring-boot-starter-web</artifactId>
40    </dependency>
```

```
41  </dependencies>
42
43  <dependency>
44    <groupId>org.projectlombok</groupId>
45    <artifactId>lombok</artifactId>
46    <optional>true</optional>
47  </dependency>
48  <dependency>
49    <groupId>org.springframework.boot</groupId>
50    <artifactId>spring-boot-starter-test</artifactId>
51    <scope>test</scope>
52  </dependency>
53
54  <dependency>
55    <groupId>org.webjars</groupId>
56    <artifactId>bootstrap</artifactId>
57    <version>5.1.3</version>
58  </dependency>
59  <dependency>
60    <groupId>org.webjars</groupId>
61    <artifactId>jquery</artifactId>
62    <version>3.6.3</version>
63  </dependency>
64  <dependency>
65    <groupId>org.webjars.bower</groupId>
66    <artifactId>EasyAutocomplete</artifactId>
67    <version>1.3.3</version>
68  </dependency>
69  </dependencies>
70
71  <build>
72    <plugins>
73      <plugin>
74        <groupId>org.springframework.boot</groupId>
75        <artifactId>spring-boot-maven-plugin</artifactId>
76        <configuration>
77          <excludes>
78            <exclude>
79              <groupId>org.projectlombok</groupId>
80              <artifactId>lombok</artifactId>
81            </exclude>
82          </excludes>
83        </configuration>
84      </plugin>
85    </plugins>
86  </build>
87 </project>
```

# EXEMPLO H2

- Configurações do arquivo *application.properties*

```
application.properties ×  
1 logging.level.com.zaxxer.hikari.HikariConfig=DEBUG
```



# EXEMPLO H2

## ■ Propriedades no console

```
dev-ti - exemplo_H2/src/main/resources/application.properties - Spring Tool Suite 4
File Edit Navigate Search Project Git Run Window Help
Problems Javadoc Declaration Console Terminal Git Staging History
exemplo_H2-1 - ExemploH2Application [Spring Boot App] C:\sts4\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe (8 de fev. de 2023 12:27:48) [pid: 37720]
ain] j.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
ain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
ain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.71]
ain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
ain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 316 ms
ain] com.zaxxer.hikari.HikariConfig : Driver class org.h2.Driver found in Thread context class loader jdk.internal.loader.ClassLoaders$AppClassLoader@2f333739
ain] com.zaxxer.hikari.HikariConfig : HikariPool-2 - configuration:
ain] com.zaxxer.hikari.HikariConfig : allowPoolSuspension.....false
ain] com.zaxxer.hikari.HikariConfig : autoCommit.....true
ain] com.zaxxer.hikari.HikariConfig : catalog.....none
ain] com.zaxxer.hikari.HikariConfig : connectionInitSql.....none
ain] com.zaxxer.hikari.HikariConfig : connectionTestQuery.....none
ain] com.zaxxer.hikari.HikariConfig : connectionTimeout.....30000
ain] com.zaxxer.hikari.HikariConfig : dataSource.....none
ain] com.zaxxer.hikari.HikariConfig : dataSourceClassName.....none
ain] com.zaxxer.hikari.HikariConfig : dataSourceJNDI.....none
ain] com.zaxxer.hikari.HikariConfig : dataSourceProperties.....{password=<masked>}
ain] com.zaxxer.hikari.HikariConfig : driverClassName....."org.h2.Driver"
ain] com.zaxxer.hikari.HikariConfig : exceptionOverrideClassName.....none
ain] com.zaxxer.hikari.HikariConfig : healthCheckProperties.....{}
ain] com.zaxxer.hikari.HikariConfig : healthCheckRegistry.....none
ain] com.zaxxer.hikari.HikariConfig : idleTimeout.....600000
ain] com.zaxxer.hikari.HikariConfig : initializationFailTimeout.....1
ain] com.zaxxer.hikari.HikariConfig : isolateInternalQueries.....false
ain] com.zaxxer.hikari.HikariConfig : jdbcUrl.....jdbc:h2:mem:d769fe3c-2049-46cd-a96f-d104f4e9cfe4;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
ain] com.zaxxer.hikari.HikariConfig : keepaliveTime.....0
ain] com.zaxxer.hikari.HikariConfig : leakDetectionThreshold.....0
ain] com.zaxxer.hikari.HikariConfig : maxLifetime.....1800000
ain] com.zaxxer.hikari.HikariConfig : maximumPoolSize.....10
ain] com.zaxxer.hikari.HikariConfig : metricRegistry.....none
ain] com.zaxxer.hikari.HikariConfig : metricsTrackerFactory.....none
ain] com.zaxxer.hikari.HikariConfig : minimumIdle.....10
ain] com.zaxxer.hikari.HikariConfig : password.....<masked>
ain] com.zaxxer.hikari.HikariConfig : poolName....."HikariPool-2"
ain] com.zaxxer.hikari.HikariConfig : readOnly.....false
ain] com.zaxxer.hikari.HikariConfig : registerMbeans.....false
ain] com.zaxxer.hikari.HikariConfig : scheduledExecutor.....none
ain] com.zaxxer.hikari.HikariConfig : schema.....none
ain] com.zaxxer.hikari.HikariConfig : threadFactory.....internal
ain] com.zaxxer.hikari.HikariConfig : transactionIsolation.....default
ain] com.zaxxer.hikari.HikariConfig : username....."sa"
ain] com.zaxxer.hikari.HikariConfig : validationTimeout.....5000
ain] com.zaxxer.hikari.HikariDataSource : HikariPool-2 - Starting...
ain] com.zaxxer.hikari.HikariDataSource : HikariPool-2 - Start completed.
ain] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:d769fe3c-2049-46cd-a96f-d104f4e9cfe4'
ain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInfo [name: default]
```

# EXEMPLO H2

## ■ Configurações do arquivo *application.properties*

### ■ Utilização de nome fixo em vez de GUID (*Globally Unique Identifier*) aleatório

```
1 #logging.level.com.zaxxer.hikari.HikariConfig=DEBUG
2
3 spring.h2.console.enabled=true
4 spring.datasource.generate-unique-name=false
```

```
com.zaxxer.hikari.HikariDataSource : HikariPool-3 - Starting...
com.zaxxer.hikari.HikariDataSource : HikariPool-3 - Start completed.
o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:testdb'
```

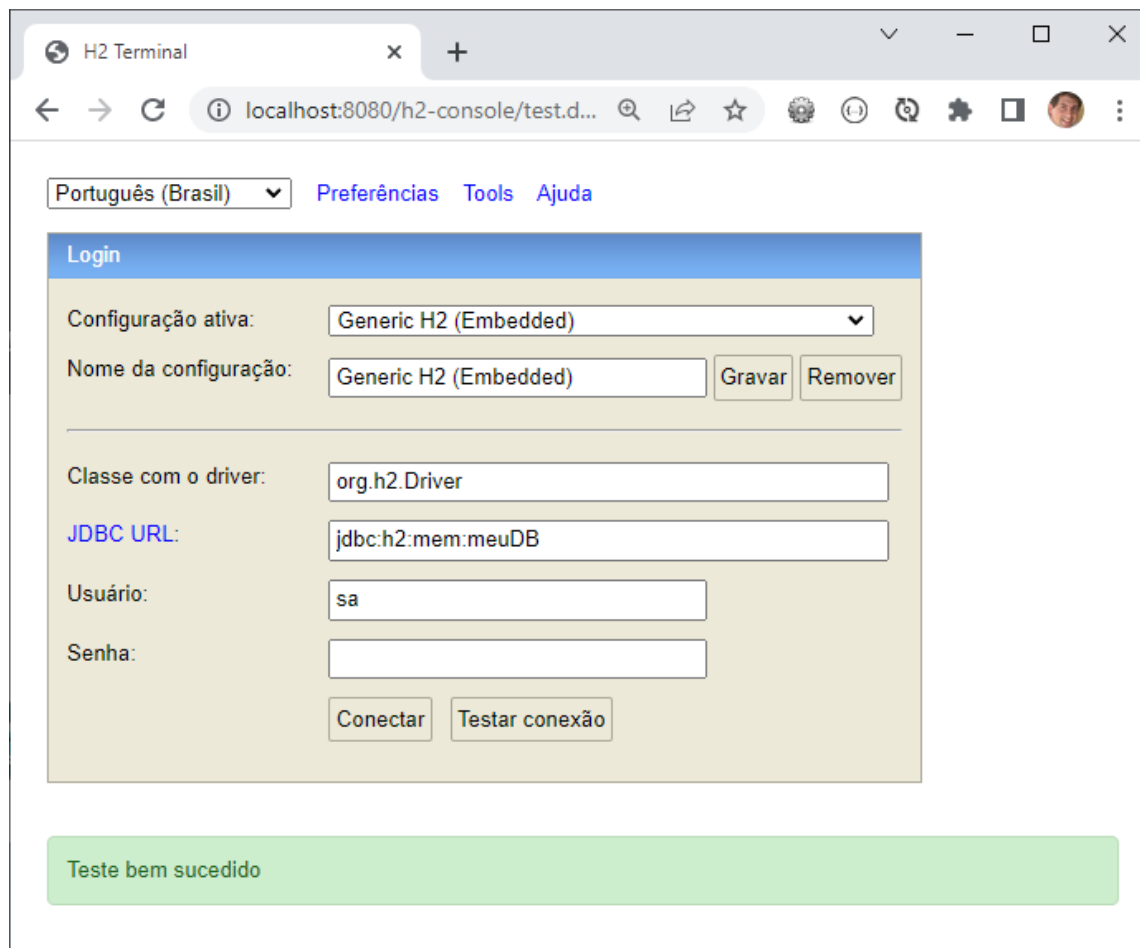
### ■ Configuração do nome do banco de dados

```
1 #logging.level.com.zaxxer.hikari.HikariConfig=DEBUG
2
3 spring.h2.console.enabled=true
4 spring.datasource.generate-unique-name=false
5 spring.datasource.name=meuDB
```

```
com.zaxxer.hikari.HikariDataSource : meuDB - Starting...
com.zaxxer.hikari.HikariDataSource : meuDB - Start completed.
o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:meuDB'
```

# EXEMPLO H2

- Console no endereço <http://localhost:8080/h2-console>



The screenshot shows a web browser window titled "H2 Terminal" with the address bar displaying "localhost:8080/h2-console/test.d...". The interface includes a language dropdown set to "Português (Brasil)" and links for "Preferências", "Tools", and "Ajuda". The main section is titled "Login" and contains the following fields and buttons:

- Configuração ativa:** A dropdown menu showing "Generic H2 (Embedded)".
- Nome da configuração:** A text input field containing "Generic H2 (Embedded)", with "Gravar" and "Remover" buttons to its right.
- Classe com o driver:** A text input field containing "org.h2.Driver".
- JDBC URL:** A text input field containing "jdbc:h2:mem:meuDB".
- Usuário:** A text input field containing "sa".
- Senha:** An empty text input field.
- Buttons:** "Conectar" and "Testar conexão" buttons are located below the password field.

A green message box at the bottom of the form area displays the text "Teste bem sucedido".

# EXEMPLO H2

■ Console no endereço <http://localhost:8080/h2-console>

The screenshot shows the H2 Terminal web console in a browser. The address bar shows the URL `localhost:8080/h2-console/login.do?sessionId=da9163877844dcb744e438a6bba7dda0`. The interface includes a sidebar with a tree view showing the database structure: `jdbc:h2:mem:meuDB`, `INFORMATION_SCHEMA`, `Usuários`, `SA`, and `Administrador`. The main area has a toolbar with buttons for `Executar comando`, `Executar selecionado`, `Auto complete`, `Limpar`, and `Comando SQL:`. Below the toolbar is a large text area for entering SQL commands. At the bottom, there are sections for **Comandos importantes** (important commands) and **Scripts de exemplo** (example scripts).

**Comandos importantes**

Ícone	Descrição
?	Mostrar esta página de ajuda
📜	Mostrar o histórico de comandos
▶	Ctrl+Enter Executar o comando SQL corrente
👤	Shift+Enter Executes the SQL statement defined by the text selection
⌨	Ctrl+Space Auto complete
🔌	Fechar conexão à Base de Dados

**Scripts de exemplo**

Ação	SQL
Apagar a tabela, caso ela exista	DROP TABLE IF EXISTS TEST;
Criar uma tabela nova com as colunas ID e NAME	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Adicionar uma linha nova	INSERT INTO TEST VALUES(1, 'Hello');
Adicionar outra linha	INSERT INTO TEST VALUES(2, 'World');
Pesquisar uma tabela	SELECT * FROM TEST ORDER BY ID;
Alterar os dados de uma linha	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Apagar uma linha	DELETE FROM TEST WHERE ID=2;
Ajuda	HELP ...

**Adicionar drivers de Base de Dados**

É possível registrar outros drivers, adicionando o arquivo JAR respectivo na variável de ambiente H2DRIVERS ou CLASSPATH. Exemplo (Windows): Para adicionar o driver que está em `C:/Programs/hsqldb/lib/hsqldb.jar` altere o valor da variável de ambiente H2DRIVERS para `C:/Programs/hsqldb/lib/hsqldb.jar`.

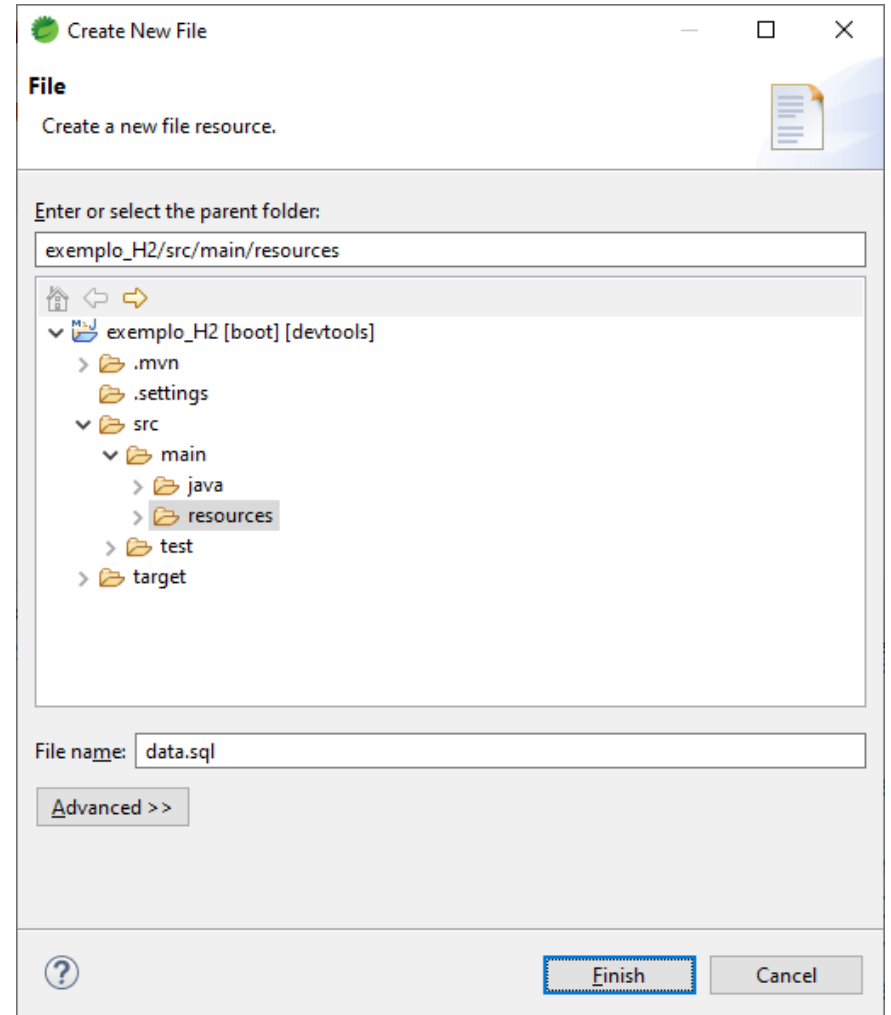
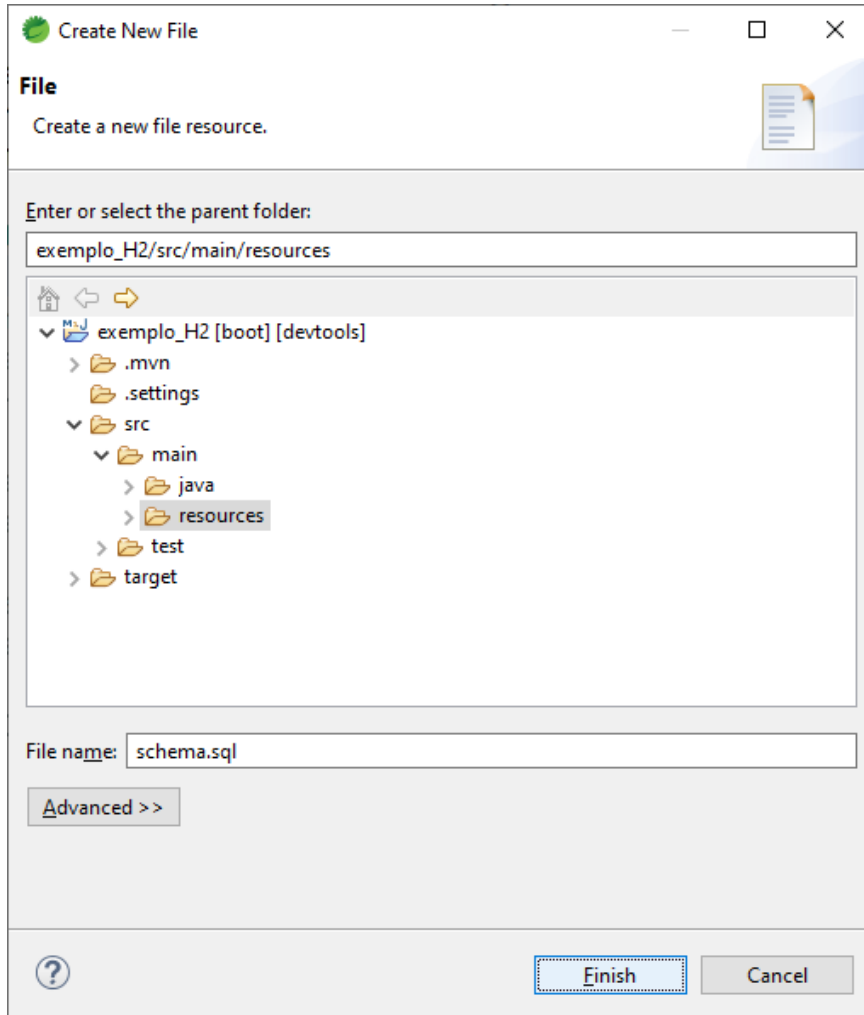
# EXEMPLO H2

## ■ Configurações do arquivo *application.properties*

```
1 #logging.level.com.zaxxer.hikari.HikariConfig=DEBUG
2
3 spring.h2.console.enabled=true
4 spring.datasource.generate-unique-name=false
5 spring.datasource.name=meuDB
6
7 spring.jpa.show-sql=true
8 spring.jpa.properties.hibernate.format_sql=true
9 logging.level.org.springframework.jdbc=DEBUG
```

# EXEMPLO H2

## ■ Criação de scripts de inicialização (*schema.sql* e *data.sql*)



# EXEMPLO H2

## ■ Scripts de inicialização

### ■ Arquivo *schema.sql*

```
1 CREATE TABLE livro (  
2     id INT AUTO_INCREMENT,  
3     titulo VARCHAR(255) NOT NULL,  
4     paginas INT NOT NULL,  
5     autor VARCHAR(255) NOT NULL  
6 );
```

### ■ Arquivo *data.sql*

```
1 INSERT INTO livro (titulo, paginas, autor) VALUES ('O senhor dos anéis', 1000, 'Tolkien');  
2 INSERT INTO livro (titulo, paginas, autor) VALUES ('Spring Boot', 328, 'Mark Heckler');
```

## EXEMPLO H2

## ■ Log de inicialização

The screenshot displays the Spring Tool Suite 4 IDE interface. The top menu bar includes File, Edit, Source, Navigate, Search, Project, Git, Run, Window, and Help. The toolbar contains various icons for file operations, development tools, and running the application. The console window shows the following output:

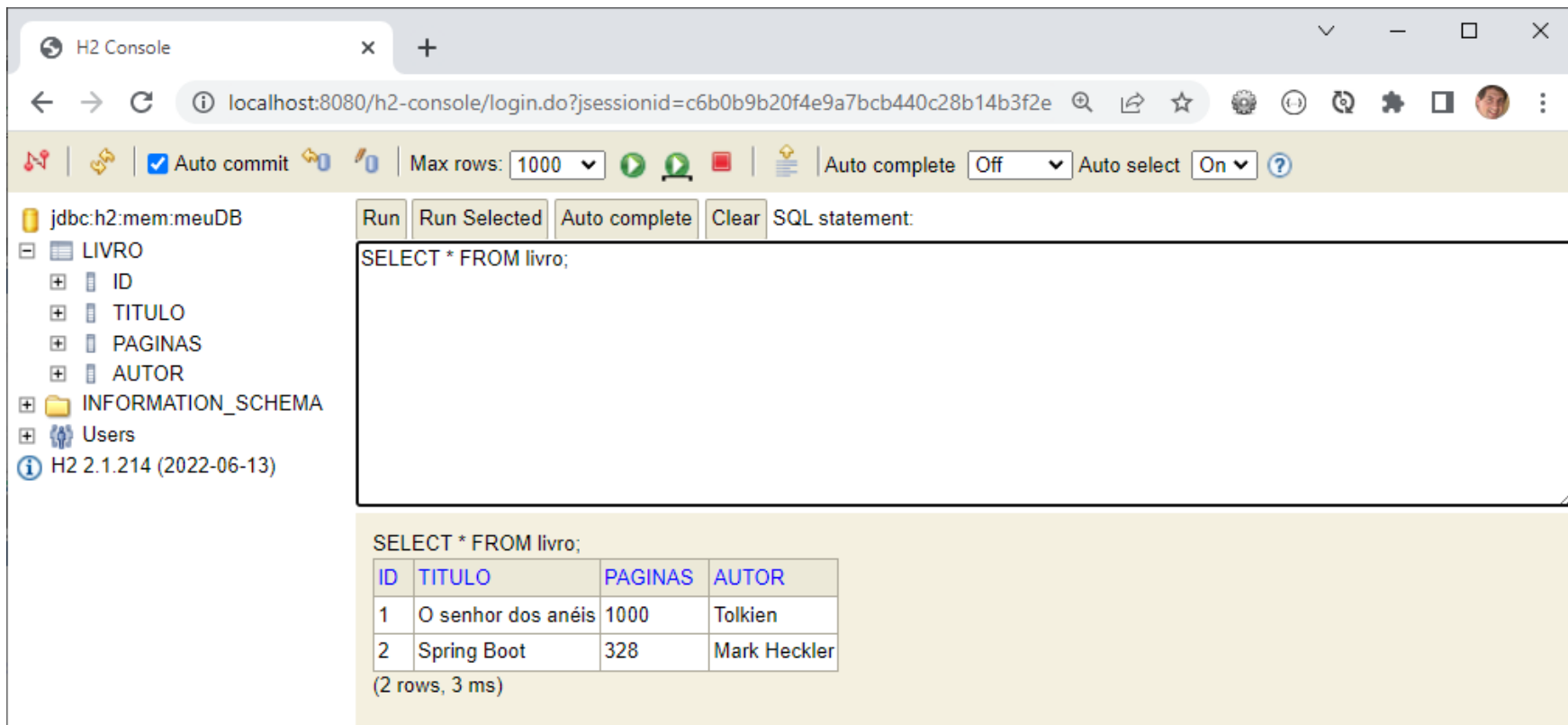
```
: No active profile set, falling back to 1 default profile: "default"
: Devtools property defaults active! Set 'spring.devtools.add-properties' to 'false' to disable
: For additional web related logging consider setting the 'logging.level.web' property to 'DEBUG'
: Bootstrapping Spring Data JPA repositories in DEFAULT mode.
: Finished Spring Data repository scanning in 9 ms. Found 0 JPA repository interfaces.
: Tomcat initialized with port(s): 8080 (http)
: Starting service [Tomcat]
: Starting Servlet engine: [Apache Tomcat/9.0.71]
: Initializing Spring embedded WebApplicationContext
: Root WebApplicationContext: initialization completed in 1396 ms
: meuDB - Starting...
: meuDB - Start completed.
: H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:meuDB'
: Fetching JDBC Connection from DataSource
: Fetching JDBC Connection from DataSource
: Executing SQL script from URL [file:C:/workspaces/dev-ti/exemplo_H2/target/classes/schema.sql]
: 0 returned as update count for SQL: CREATE TABLE livro ( id INT AUTO_INCREMENT, titulo VARCHAR(255) NOT NULL, paginas INT NOT NULL, autor VARCHAR(255) N
: Executed SQL script from URL [file:C:/workspaces/dev-ti/exemplo_H2/target/classes/schema.sql] in 17 ms.
: Fetching JDBC Connection from DataSource
: Fetching JDBC Connection from DataSource
: Executing SQL script from URL [file:C:/workspaces/dev-ti/exemplo_H2/target/classes/data.sql]
: 1 returned as update count for SQL: INSERT INTO livro (titulo, paginas, autor) VALUES ('O senhor dos anéis', 1000, 'Tolkien')
: 1 returned as update count for SQL: INSERT INTO livro (titulo, paginas, autor) VALUES ('Spring Boot', 328, 'Mark Heckler')
: Executed SQL script from URL [file:C:/workspaces/dev-ti/exemplo_H2/target/classes/data.sql] in 7 ms.
: Fetching JDBC Connection from DataSource
: HHH000204: Processing PersistenceUnitInfo [name: default]
: HHH000412: Hibernate ORM core version 5.6.14.Final
: HCANN000001: Hibernate Commons Annotations {5.1.2.Final}
: HHH000400: Using dialect: org.hibernate.dialect.H2Dialect
: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
: Initialized JPA EntityManagerFactory for persistence unit 'default'
: spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure spring.jpa.open-
: LiveReload server is running on port 35729
: Tomcat started on port(s): 8080 (http) with context path ''
: Started ExemploH2Application in 3.699 seconds (JVM running for 4.591)
```

The status bar at the bottom indicates the search for TypeDeclar...in: none (78%).



# EXEMPLO H2

## Console



The screenshot shows the H2 Console web interface in a browser. The address bar shows the URL `localhost:8080/h2-console/login.do?jsessionid=c6b0b9b20f4e9a7bcb440c28b14b3f2e`. The interface includes a toolbar with options like `Auto commit` (checked), `Max rows: 1000`, `Auto complete` (Off), and `Auto select` (On). On the left, a tree view shows the database structure: `jdbc:h2:mem:meuDB` containing a table `LIVRO` with columns `ID`, `TITULO`, `PAGINAS`, and `AUTOR`, and a schema `INFORMATION_SCHEMA`. The main area shows the SQL statement `SELECT * FROM livro;` and its results in a table with 2 rows and 4 columns.

SQL statement:

```
SELECT * FROM livro;
```

ID	TITULO	PAGINAS	AUTOR
1	O senhor dos anéis	1000	Tolkien
2	Spring Boot	328	Mark Heckler

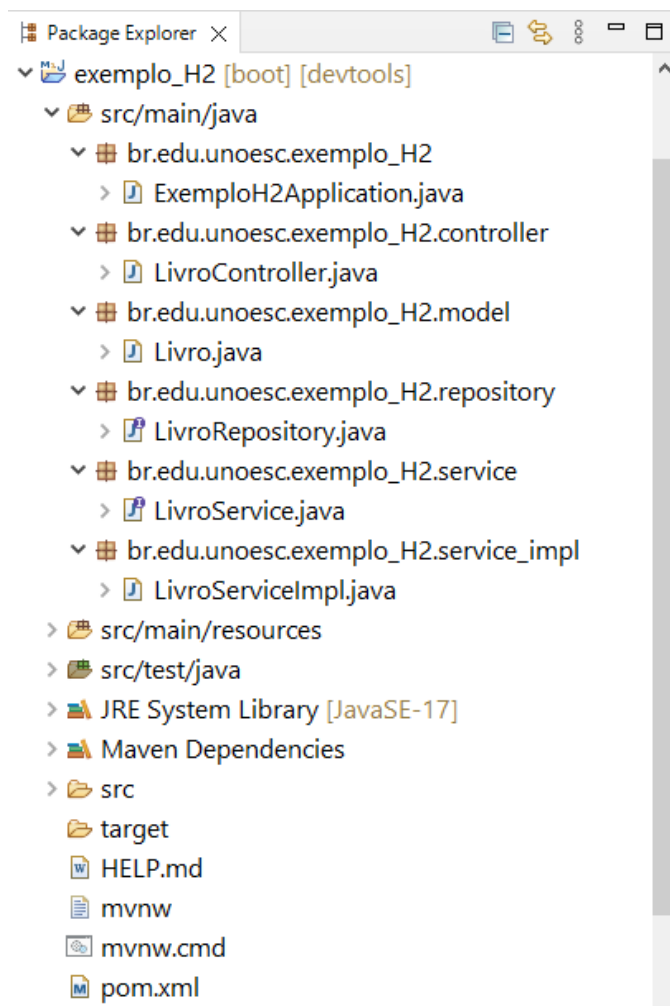
(2 rows, 3 ms)

# EXEMPLO H2

## ■ Criação dos pacotes

- model
- controller
- repository
- service
- service\_impl

## ■ Cópia dos recursos



# EXEMPLO H2

## ■ Registro (record) Livro

```
1 package br.edu.unoesc.exemplo_H2.model;  
2  
3 import org.springframework.data.annotation.Id;  
4  
5 public record Livro(  
6     @Id  
7     Integer id,  
8     String titulo,  
9     Integer paginas,  
10    String autor  
11 ) {  
12  
13 }
```

# EXEMPLO H2

- Registro (`record`) é um recurso que apareceu pela primeira vez na versão 14 como experimental e foi liberado de forma definitiva no Java 16
  - Um registro é imutável, ou seja, após criado, um `record` não pode mais ser alterado
  - Record oferece uma sintaxe compacta para declarar classes que são portadores transparentes para dados imutáveis, reduzindo significativamente o detalhamento dessas classes e irá melhorando a capacidade de leitura e manutenção do código
- Características
  - Um `Record` não possui uma cláusula `extends`
  - Um `Record` não pode ser abstrato
  - Os atributos derivados da classe `Record` são todos finais
  - Não se pode declarar campos de instância e nem métodos nativos
  - Uma instância de `Record` é criada com a expressão `new`
  - Pode ser declarada como um tipo genérico
  - Pode declarar métodos, atributos e inicializadores estáticos
  - Pode declarar métodos de instância
  - Pode implementar *interfaces*
  - Pode utilizar *annotations*
  - Pode ser serializado e desserializado

# EXEMPLO H2

## ■ *Interface* LivroRepository

```
1 package br.edu.unoesc.exemplo_H2.repository;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import br.edu.unoesc.exemplo_H2.model.Livro;
6
7 public interface LivroRepository extends CrudRepository<Livro, Integer> {
8
9 }
```

# EXEMPLO H2

## ■ Programa principal

### ■ Utilização do método commandLineRunner() para testes

```
1 package br.edu.unoesc.exemplo_H2;
2
3 import java.util.Optional;
4
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.boot.SpringApplication;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8 import org.springframework.context.annotation.Bean;
9
10 import br.edu.unoesc.exemplo_H2.model.Livro;
11 import br.edu.unoesc.exemplo_H2.repository.LivroRepository;
12
13 @SpringBootApplication
14 public class ExemploH2Application {
15
16     public static void main(String[] args) {
17         SpringApplication.run(ExemploH2Application.class, args);
18     }
19
20     @Bean
21     CommandLineRunner commandLineRunner(LivroRepository repositorio) {
22         return args -> {
23             Livro l = new Livro(null, "O senhor dos pastéis", 666, "Tolkien");
24             l = repositorio.save(l);
25
26             Livro n = new Livro(l.id(), "The Lord of the Rings", l.paginas(), l.autor());
27             repositorio.save(n);
28             // repositorio.delete(l);
29
30             repositorio.save(new Livro(null, "O hobbit", 42, "J.R.R.Tolkien"));
31             Optional<Livro> p = repositorio.findById(4);
32             // repositorio.delete(p.get());
33
34             System.out.println(repositorio.findAll());
35             for (var livro : repositorio.findAll()) {
36                 System.out.println(livro);
37             }
38         };
39     }
40 }
```

# EXEMPLO H2

## ■ Refatorações

1. Remova o *starter* do JDBC e acrescente o do JPA
2. Acrescente a seguinte configuração no arquivo *application.properties*  
`spring.jpa.hibernate.ddl-auto=update`
3. Modifique o registro (*record*) `Livro` para uma classe (entidade) JPA

# EXEMPLO H2

## ■ Classe Livro

```
1 package br.edu.unoesc.exemplo_H2.model;
2
3 import java.io.Serializable;
4
5 import javax.persistence.Entity;
6 import javax.persistence.GeneratedValue;
7 import javax.persistence.GenerationType;
8 import javax.persistence.Id;
9
10 import lombok.AllArgsConstructor;
11 import lombok.Data;
12 import lombok.NoArgsConstructor;
13
14 @Data
15 @Entity
16 @AllArgsConstructor
17 @NoArgsConstructor
18 public class Livro implements Serializable {
19     private static final long serialVersionUID = 1L;
20
21     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
22     private Integer id;
23
24     private String titulo;
25     private Integer paginas;
26     private String autor;
27 }
```



# EXEMPLO H2

## ■ Classe principal

- Possibilidade de alterar os valores das entidades

- Teste com a classe Optional

```
1 package br.edu.unoesc.exemplo_H2;
2
3 import java.util.Optional;
4
5 import org.springframework.boot.CommandLineRunner;
6 import org.springframework.boot.SpringApplication;
7 import org.springframework.boot.autoconfigure.SpringBootApplication;
8 import org.springframework.context.annotation.Bean;
9
10 import br.edu.unoesc.exemplo_H2.model.Livro;
11 import br.edu.unoesc.exemplo_H2.repository.LivroRepository;
12
13 @SpringBootApplication
14 public class ExemploH2Application {
15
16     public static void main(String[] args) {
17         SpringApplication.run(ExemploH2Application.class, args);
18     }
19
20     @Bean
21     CommandLineRunner commandLineRunner(LivroRepository repositorio) {
22         return args -> {
23             Livro l = new Livro(null, "O senhor dos pastéis", 666, "Tolkien");
24             repositorio.save(l);
25
26             repositorio.save(new Livro(null, "O hobbit", 42, "J.R.R.Tolkien"));
27
28             Optional<Livro> p = repositorio.findById(6);
29             if (p.isPresent()) {
30                 repositorio.delete(p.get());
31             } else {
32                 System.out.println("Registro não encontrado!");
33             }
34
35             l = repositorio.findById(2).get();
36             l.setTitulo("O livro do Spring Boot");
37             l.setPaginas(100);
38             repositorio.save(l);
39         };
40     }
41 }
```

# EXEMPLO H2

## ■ Novos métodos no repositório

```
1 package br.edu.unoesc.exemplo_H2.repository;
2
3 import java.util.List;
4
5 import org.springframework.data.jpa.repository.Query;
6 import org.springframework.data.repository.CrudRepository;
7 import org.springframework.data.repository.query.Param;
8
9 import br.edu.unoesc.exemplo_H2.model.Livro;
10
11 public interface LivroRepository extends CrudRepository<Livro, Integer> {
12     public List<Livro> findByAutorContainingIgnoreCase(String autor);
13
14     @Query("Select l from Livro l where l.paginas >= :paginas")
15     public List<Livro> porQtdPaginas(@Param("paginas") Integer paginas);
16
17     @Query("Select l from Livro l where upper(l.titulo) like upper(concat('%', :filtro, '%')) order by titulo")
18     public List<Livro> findByFiltro(@Param("filtro") String filtro);
19 }
```

# EXEMPLO H2

■ *Controller* (*LivroController.java*) utilizando diretamente o repositório

■ Injeção de dependência através da anotação `@Autowired`

```
1 package br.edu.unoesc.exemplo_H2.controller;
2
3 import java.util.List;
4
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RequestParam;
9 import org.springframework.web.bind.annotation.RestController;
10
11 import br.edu.unoesc.exemplo_H2.model.Livro;
12 import br.edu.unoesc.exemplo_H2.repository.LivroRepository;
13
14 @RestController
15 @RequestMapping("/api/livros")
16 public class LivroController {
17     @Autowired
18     private LivroRepository repositorio;
19
20     @GetMapping("/find")
21     List<Livro> findByFiltro(@RequestParam("filtro") String filtro) {
22         return repositorio.findByFiltro(filtro);
23     }
24
25     @GetMapping
26     public Iterable<Livro> findAll() {
27         return repositorio.findAll();
28     }
29 }
```

# EXEMPLO H2

## ■ Consultas

The screenshot displays the H2 Console interface with two browser tabs. The active tab is `localhost:8080/api/livros/find?filtro=senhor`. The address bar shows the URL `localhost:8080/api/livros/find?filtro=senhor`. The response is displayed in a JSON viewer with 'Raw' and 'Parsed' tabs. The 'Parsed' tab is selected, showing the following JSON data:

```
[{"id": 1, "titulo": "O senhor dos anéis", "paginas": 1000, "autor": "Tolkien"}, {"id": 3, "titulo": "O senhor dos pastéis", "paginas": 666, "autor": "Tolkien"}]
```

The background tab shows `localhost:8080/api/livros` and displays a partial JSON response:

```
[{"id": ..., "titulo": ..., "paginas": ..., "autor": ...}, {"id": ..., "titulo": ..., "paginas": ..., "autor": ...}, {"id": ..., "titulo": ..., "paginas": ..., "autor": ...}]
```

# EXEMPLO H2

## ■ Página *index.html* (início e fim)

```
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3 <head>
4   <meta charset="UTF-8">
5   <title>Spring Boot : Livros</title>
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7
8   <link rel="stylesheet" type="text/css" href="webjars/bootstrap/5.1.3/css/bootstrap.min.css" />
9   <link rel="stylesheet" type="text/css" href="webjars/EasyAutocomplete/1.3.3/dist/easy-autocomplete.themes.css" />
10 </head>
11 <body>
12   <div class="container">
13     <script type="text/javascript" src="webjars/jquery/3.6.3/jquery.min.js"></script>
14     <script type="text/javascript" src="webjars/bootstrap/5.1.3/js/bootstrap.min.js"></script>
15     <script type="text/javascript" src="webjars/EasyAutocomplete/1.3.3/dist/jquery.easy-autocomplete.js"></script>
16     .
17     .
18     .
19
20     <h1 class="text-center mt-2">
21       
22       Spring Boot : Livros
23     </h1>
24
25     <form class="mt-1">
26       Buscar:
27       <input type="text" id="filtro" autofocus="autofocus" class="form-control" />
28     </form>
29   </div>
30 </body>
31 </html>
```

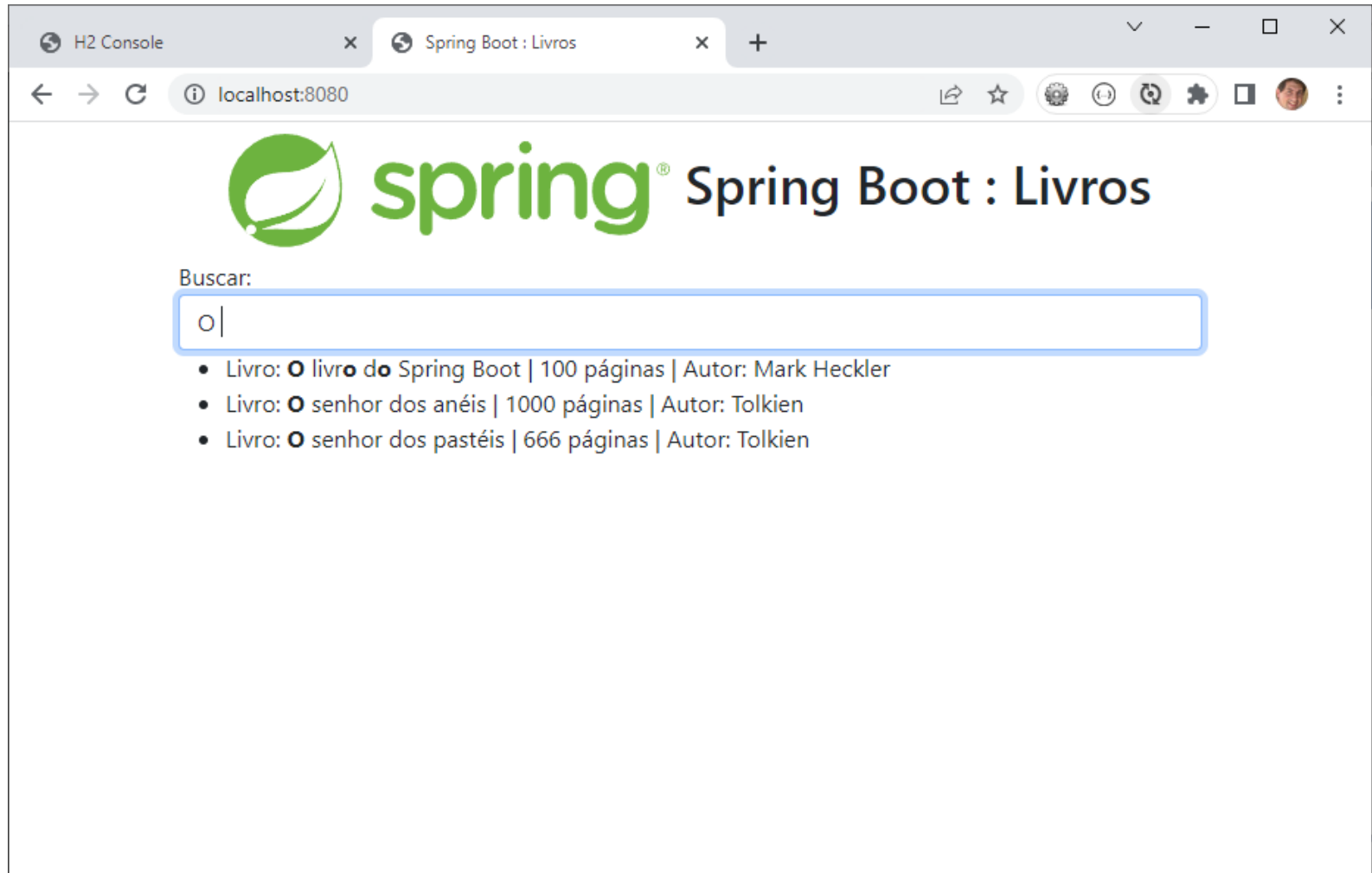
# EXEMPLO H2

## ■ Página *index.html*

```
17<script type="text/javascript">
18    $(document).ready(function () {
19        let opcoes = {
20            url: function (filtro) {
21                return '/api/livros/find/';
22            },
23            getValue: function (elemento) {
24                return elemento.titulo;
25            },
26            ajaxSettings: {
27                dataType: 'json',
28                method: 'GET',
29                data: { }
30            },
31            template: {
32                type: "custom",
33                method: function (value, item) {
34                    return "Livro: " + value + " | " + item.paginas + " páginas | Autor: " + item.autor;
35                }
36            },
37            preparePostData: function (data) {
38                data.filtro = $('#filtro').val();
39                return data;
40            },
41            theme: "dark",
42            list: {
43                showAnimation: {
44                    type: "slide"
45                },
46                hideAnimation: {
47                    type: "slide"
48                },
49                match: {
50                    enabled: true
51                },
52                maxNumberOfElements: 15
53            },
54            //requestDelay: 100
55        };
56
57        $('#filtro').easyAutocomplete(opcoes);
58    });
59</script>
```

# EXEMPLO H2

## ■ Exemplo



# EXEMPLO H2

## ■ Página *index\_ajax\_tabela.html*

```
1 <!DOCTYPE html>
2 <html lang="pt-BR">
3 <head>
4     <meta charset="UTF-8">
5     <title>Spring Boot : Livros</title>
6     <meta name="viewport" content="width=device-width, initial-scale=1.0">
7
8     <link rel="stylesheet" type="text/css" href="webjars/bootstrap/5.1.3/css/bootstrap.min.css" />
9     <link rel="stylesheet" type="text/css" href="webjars/EasyAutocomplete/1.3.3/dist/easy-autocomplete.themes.css" />
10 </head>
11 <body>
12     <div class="container">
13         <script type="text/javascript" src="webjars/jquery/3.6.3/jquery.min.js"></script>
14         <script type="text/javascript" src="webjars/bootstrap/5.1.3/js/bootstrap.min.js"></script>
15         <script type="text/javascript" src="webjars/EasyAutocomplete/1.3.3/dist/jquery.easy-autocomplete.js"></script>
```



# EXEMPLO H2

## ■ Página *index\_ajax\_tabela.html*

```
17<script type="text/javascript">
18    $(document).ready(function () {
19        let filtroForm = document.getElementById('filtro');
20
21        filtroForm.addEventListener('input', (filtro) => {
22            $.ajax({
23                type : 'GET',
24                url : '/api/livros/find?filtro=' + filtroForm.value,
25                data : '$format=json',
26                dataType : 'json',
27                success : function(dados) {
28                    let total = 0;
29                    $('#livros tbody').empty();
30                    $.each(dados, function(d, resultado) {
31                        $('#livros tbody').append(
32                            '<tr>' +
33                                '<td class="text-nowrap">' + resultado.titulo + '</td>' +
34                                "&td class=\"text-nowrap\">" + resultado.paginas + '</td>' +
35                                "&td class=\"text-nowrap\">" + resultado.autor + '</td>' +
36                                '</tr>');
37                        total++;
38                    });
39
40                    if (total === 0) {
41                        $('#resultados').text('Nenhum registro encontrado!');
42                    } else if (total === 1) {
43                        $('#resultados').text('1 registro encontrado!');
44                    } else {
45                        $('#resultados').text(total + ' registros encontrados!');
46                    }
47                }
48            });
49        });
50    });
51</script>
```

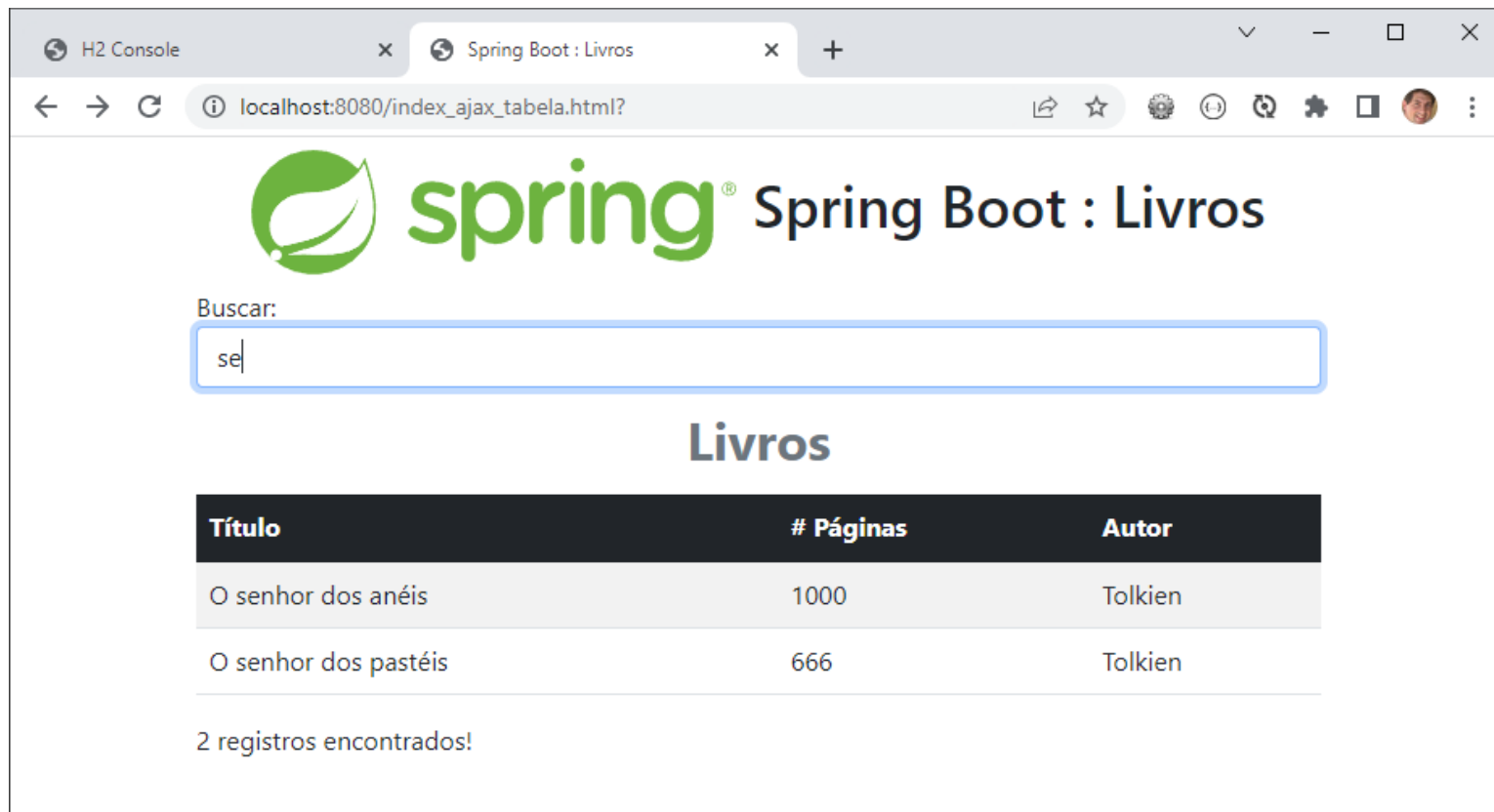
# EXEMPLO H2

## ■ Página `index_ajax_tabela.html`

```
53<h1 class="text-center mt-2">
54  
55  Spring Boot : Livros
56</h1>
57
58<form class="mt-1">
59  Buscar:
60  <input type="text" id="filtro" autofocus="autofocus" class="form-control" />
61</form>
62
63<table id="livros" class="table table-striped table-condensed table-hover table-responsive">
64  <caption style="font-size: 2em; font-weight: bold; text-align: center; caption-side: top">
65    Livros
66  </caption>
67
68  <thead class="table-dark">
69    <tr>
70      <th>Título</th>
71      <th># Páginas</th>
72      <th>Autor</th>
73    </tr>
74  </thead>
75
76  <tbody></tbody>
77</table>
78
79  <div id="resultados"></div>
80</div>
81</body>
82</html>
```


# EXEMPLO H2

## ■ Exemplo



H2 Console x Spring Boot : Livros x +

localhost:8080/index\_ajax\_tabela.html?

 **spring**® Spring Boot : Livros

Buscar:

se

### Livros

Título	# Páginas	Autor
O senhor dos anéis	1000	Tolkien
O senhor dos pastéis	666	Tolkien

2 registros encontrados!

# EXEMPLO H2

- Uma boa prática de desenvolvimento é separar o sistema em camadas e dividir as responsabilidades nos componentes corretos
  - A camada *controller* tem a finalidade de apenas orientar a requisição para a lógica de negócio e devolver a resposta
  - A tarefa do componente controlador é receber as requisições, chamar os componentes de negócios e devolver as respostas, não devendo implementar regras de negócios
  - As regras de negócio podem ser colocadas dentro das entidades, dessa forma ela não fica sendo somente uma estrutura de dados mas passa a ter responsabilidades
    - Normalmente nela ficam regras de negócio mais básicas, que não envolvem interações com outras entidades
  - Quando há interação com outras entidades ou com repositórios costuma-se criar uma camada ou componente de serviço
  - Métodos de consulta podem ser implementados na classe controladora ou de serviço

# EXEMPLO H2

## ■ Arquitetura

Front end (app)

↕ JSON (Requisições HTTP / REST)

Controladores REST

↕ Data Transfer Objects (DTO)

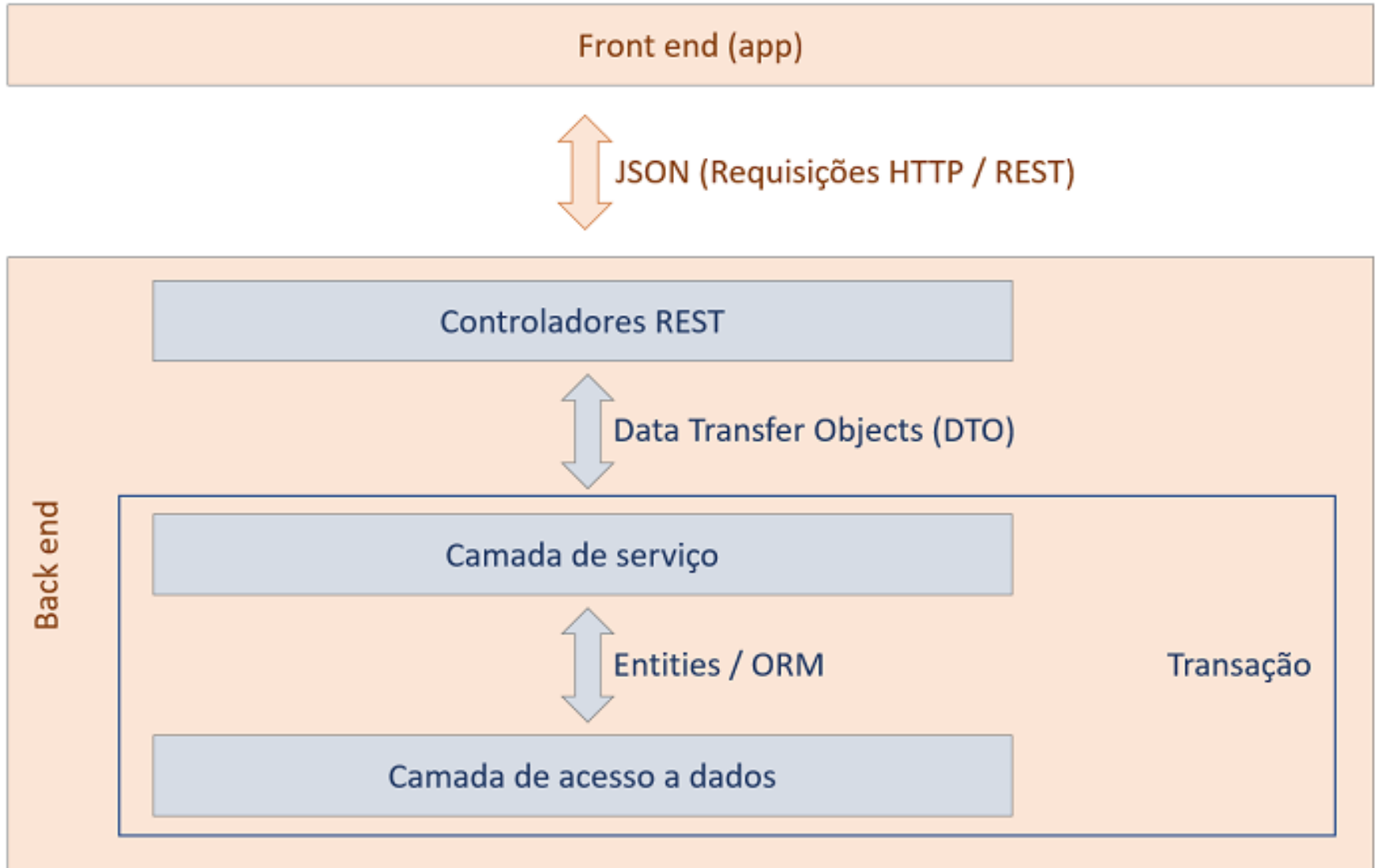
Camada de serviço

↕ Entities / ORM

Camada de acesso a dados

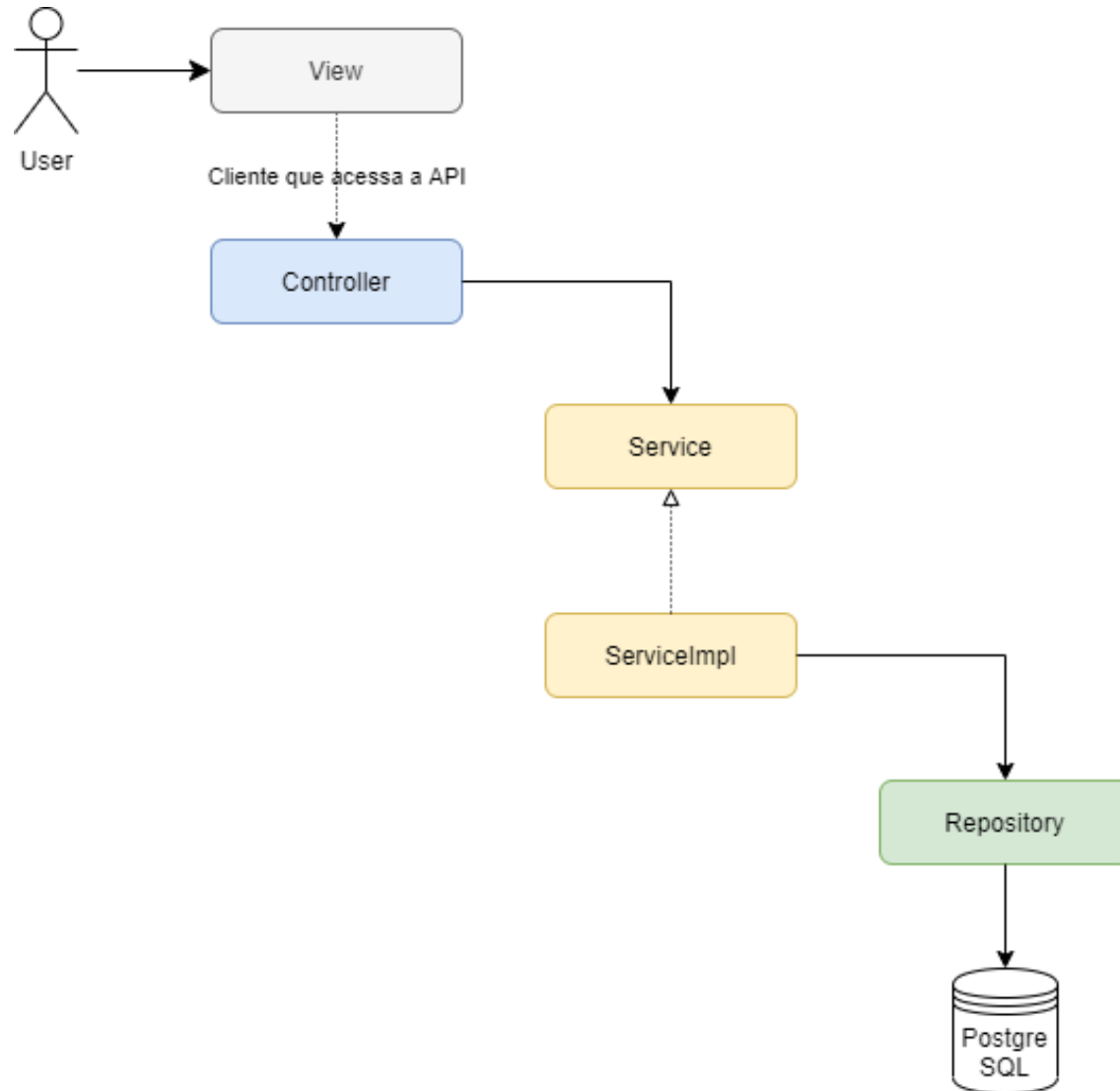
Transação

Back end



# EXEMPLO H2

## ■ Arquitetura



# EXEMPLO H2

- Acrescentando uma camada de serviço: *Interface* LivroService

```
1 package br.edu.unoesc.exemplo_H2.service;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import br.edu.unoesc.exemplo_H2.model.Livro;
7
8 public interface LivroService {
9     Livro salvar(Livro livro);
10    void excluir(Integer id);
11
12    Livro buscar(Integer id);
13    Livro buscarPorCodigo(Integer id);
14    Optional<Livro> porCodigo(Integer id);
15
16    List<Livro> buscarPorTitulo(String titulo);
17    List<Livro> buscarPorAutor(String autor);
18    List<Livro> buscarPorQtdPaginas(Integer qtdPaginas);
19
20    Iterable<Livro> listar();
21 }
```

# EXEMPLO H2

## ■ Implementação concreta da interface: LivroServiceImpl

```
1 package br.edu.unoesc.exemplo_H2.service_impl;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.hibernate.ObjectNotFoundException;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.stereotype.Service;
9
10 import br.edu.unoesc.exemplo_H2.model.Livro;
11 import br.edu.unoesc.exemplo_H2.repository.LivroRepository;
12 import br.edu.unoesc.exemplo_H2.service.LivroService;
13
14 @Service
15 public class LivroServiceImpl implements LivroService {
16     @Autowired
17     private LivroRepository repositorio;
18
19     @Override
20     public Livro salvar(Livro livro) {
21         return repositorio.save(livro);
22     }
23
24     @Override
25     public void excluir(Integer id) {
26         if (repositorio.existsById(id)) {
27             repositorio.deleteById(id);
28         } else {
29             throw new ObjectNotFoundException("Objeto não encontrado! Id: " + id +
30                 ", Tipo: " + Livro.class.getName(),
31                 null);
32         }
33     }
34
35     @Override
36     public Iterable<Livro> listar() {
37         return repositorio.findAll();
38     }
39 }
```



# EXEMPLO H2

- Implementação concreta da *interface*: `LivroServiceImpl`

```
40 @Override
41 public Livro buscar(Integer id) {
42     Optional<Livro> obj = repositorio.findById(id);
43     return obj.orElseThrow(() ->
44         new ObjectNotFoundException("Objeto não encontrado! Id: " + id +
45                                     ", Tipo: " + Livro.class.getName(),
46                                     null
47     ));
48 }
49
50
51 @Override
52 public Optional<Livro> porCodigo(Integer id) {
53     return repositorio.findById(id);
54 }
55
56 @Override
57 // Retorna um novo objeto caso id não exista
58 public Livro buscarPorCodigo(Integer id) {
59     return repositorio.findById(id).orElse(new Livro());
60 }
61
62 @Override
63 public List<Livro> buscarPorTitulo(String titulo) {
64     return repositorio.findByFiltro(titulo);
65 }
66
67 @Override
68 public List<Livro> buscarPorAutor(String autor) {
69     return repositorio.findByAutorContainingIgnoreCase(autor);
70 }
71
72 @Override
73 public List<Livro> buscarPorQtdPaginas(Integer qtdPaginas) {
74     return repositorio.porQtdPaginas(qtdPaginas);
75 }
76 }
```

## EXEMPLO H2

- Refatorando a classe principal a fim de utilizar a classe service e testando os novos métodos

```
1 package br.edu.unoesc.exemplo_H2;
2
3 import java.util.Optional;
4
5 import org.hibernate.ObjectNotFoundException;
6 import org.springframework.boot.CommandLineRunner;
7 import org.springframework.boot.SpringApplication;
8 import org.springframework.boot.autoconfigure.SpringBootApplication;
9 import org.springframework.context.annotation.Bean;
10
11 import br.edu.unoesc.exemplo_H2.model.Livro;
12 import br.edu.unoesc.exemplo_H2.service.LivroService;
13
14 @SpringBootApplication
15 public class ExemploH2Application {
16     public static void main(String[] args) {
17         SpringApplication.run(ExemploH2Application.class, args);
18     }
19
20     @Bean
21     CommandLineRunner commandLineRunner(LivroService servico) {
22         return args -> {
23             Livro l = new Livro(null, "O senhor dos pastéis", 666, "Tolkien");
24             servico.salvar(l);
25
26             servico.salvar(new Livro(null, "O hobbit", 42, "J.R.R.Tolkien"));
27
28             try {
29                 servico.excluir(6);
30             } catch (ObjectNotFoundException e) {
31                 System.out.println(e);
32             }
33
34             // Teste com a classe Optional
35             Optional<Livro> o = servico.porCodigo(3);
36             if (!o.isPresent()) {
37                 System.out.println("Livro não existe!");
38             } else {
39                 System.out.println(o);
40             }
41
42             // Cria nova instância pois objeto com id 10 não existe
43             l = servico.buscarPorCodigo(10);
44             l.setTitulo("O livro do Spring Boot");
45             l.setPaginas(100);
46             l.setAutor("Zé das Couves");
47             servico.salvar(l);
48         };
49     }
50 }
```

# EXEMPLO H2

## ■ Refatorando a classe *controller*

```
1 package br.edu.unoesc.exemplo_H2.controller;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.hibernate.ObjectNotFoundException;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.HttpStatus;
9 import org.springframework.http.ResponseEntity;
10 import org.springframework.web.bind.annotation.DeleteMapping;
11 import org.springframework.web.bind.annotation.GetMapping;
12 import org.springframework.web.bind.annotation.PathVariable;
13 import org.springframework.web.bind.annotation.PostMapping;
14 import org.springframework.web.bind.annotation.PutMapping;
15 import org.springframework.web.bind.annotation.RequestBody;
16 import org.springframework.web.bind.annotation.RequestMapping;
17 import org.springframework.web.bind.annotation.RequestParam;
18 import org.springframework.web.bind.annotation.ResponseStatus;
19 import org.springframework.web.bind.annotation.RestController;
20
21 import br.edu.unoesc.exemplo_H2.model.Livro;
22 import br.edu.unoesc.exemplo_H2.service.LivroService;
23
24 @RestController
25 @RequestMapping("/api/livros")
26 public class LivroController {
27     @Autowired
28     private LivroService servico;
29
30     @GetMapping
31     public Iterable<Livro> listar() {
32         return servico.listar();
33     }
34
35     @GetMapping("/{id}")
36     public ResponseEntity<Livro> buscarPorCodigo(@PathVariable("id") Integer id) {
37         Optional<Livro> livro = servico.porCodigo(id);
38
39         if (livro.isPresent()) {
40             return ResponseEntity.ok(livro.get());
41         }
42
43         return ResponseEntity.notFound().build();
44     }
45
46     @GetMapping("/find")
47     public List<Livro> findByFiltro(@RequestParam("filtro") String filtro) {
48         return servico.buscarPorTitulo(filtro);
49     }
50
51     @GetMapping("/autor")
52     public List<Livro> buscarPorAutor(@RequestParam("filtro") String filtro) {
53         return servico.buscarPorAutor(filtro);
54     }
55
56     @GetMapping("/qtdpaginas")
57     public List<Livro> buscarPorQtdPaginas(@RequestParam("filtro") Integer filtro) {
58         return servico.buscarPorQtdPaginas(filtro);
59     }
60 }
```

# EXEMPLO H2

## ■ Refatorando a classe *controller*

```
61 @GetMapping(value={"/livroporqtdpgs", "/livroporqtdpgs/{paginas}"})
62 public List<Livro> listarPorQuantidade(@PathVariable Optional<Integer> paginas) {
63     return servico.buscarPorQtdPaginas(paginas.orElse(0));
64 }
65
66 @PostMapping()
67 @ResponseStatus(HttpStatus.CREATED)
68 public Livro incluir(@RequestBody Livro livro) {
69     return servico.salvar(livro);
70 }
71
72 @PutMapping("/{id}")
73 @ResponseStatus(HttpStatus.CREATED)
74 public ResponseEntity<Livro> atualizar(@PathVariable("id") Integer id,
75                                     @RequestBody Livro livro) {
76     if (servico.porCodigo(id).isEmpty()) {
77         return ResponseEntity.notFound().build();
78     }
79
80     livro.setId(id);
81     livro = servico.salvar(livro);
82
83     return ResponseEntity.ok(livro);
84 }
85
86 @DeleteMapping("/{id}")
87 public ResponseEntity<Void> excluir(@PathVariable("id") Integer id) {
88     try {
89         servico.excluir(id);
90     } catch (ObjectNotFoundException e) {
91         return ResponseEntity.notFound().build();
92     }
93
94     return ResponseEntity.noContent().build();
95 }
96 }
```

# EXEMPLO H2

- Refatorando: Acréscimo de código para tratamento de erros e códigos de retorno

- A classe `ResponseEntity` representa toda a resposta HTTP, incluindo o código de *status*, cabeçalho (*header*) e corpo (*body*)

```
1 package br.edu.unoesc.exemplo_H2.controller;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.hibernate.ObjectNotFoundException;
7 import org.springframework.beans.factory.annotation.Autowired;
8 import org.springframework.http.HttpStatus;
9 import org.springframework.http.ResponseEntity;
10 import org.springframework.web.bind.annotation.DeleteMapping;
11 import org.springframework.web.bind.annotation.GetMapping;
12 import org.springframework.web.bind.annotation.PathVariable;
13 import org.springframework.web.bind.annotation.PostMapping;
14 import org.springframework.web.bind.annotation.PutMapping;
15 import org.springframework.web.bind.annotation.RequestBody;
16 import org.springframework.web.bind.annotation.RequestMapping;
17 import org.springframework.web.bind.annotation.RequestParam;
18 import org.springframework.web.bind.annotation.ResponseStatus;
19 import org.springframework.web.bind.annotation.RestController;
20
21 import br.edu.unoesc.exemplo_H2.model.Livro;
22 import br.edu.unoesc.exemplo_H2.service.LivroService;
23
24 @RestController
25 @RequestMapping("/api/livros")
26 public class LivroController {
27     @Autowired
28     private LivroService servico;
29
30     @GetMapping
31     public Iterable<Livro> listar() {
32         return servico.listar();
33     }
34
35     @GetMapping("/{id}")
36     public ResponseEntity<Livro> buscarPorCodigo(@PathVariable("id") Integer id) {
37         Optional<Livro> livro = servico.porCodigo(id);
38
39         if (livro.isPresent()) {
40             return ResponseEntity.ok(livro.get());
41         }
42
43         return ResponseEntity.notFound().build();
44     }
45 }
```

# EXEMPLO H2

- Refatorando: Acréscimo de código para tratamento de erros e códigos de retorno
- A classe `ResponseEntity` representa toda a resposta HTTP, incluindo o código de *status*, cabeçalho (*header*) e corpo (*body*)

```
46 @GetMapping("/find")
47 List<Livro> findByFiltro(@RequestParam("filtro") String filtro) {
48     return servico.buscarPorTitulo(filtro);
49 }
50
51 @GetMapping("/autor")
52 List<Livro> buscarPorAutor(@RequestParam("filtro") String filtro) {
53     return servico.buscarPorAutor(filtro);
54 }
55
56 @GetMapping("/qtdpaginas")
57 List<Livro> buscarPorQtdPaginas(@RequestParam("filtro") Integer filtro) {
58     return servico.buscarPorQtdPaginas(filtro);
59 }
60
61 @PostMapping()
62 @ResponseStatus(HttpStatus.CREATED)
63 public Livro incluir(@RequestBody Livro livro) {
64     return servico.salvar(livro);
65 }
66
67 @PutMapping("/{id}")
68 @ResponseStatus(HttpStatus.CREATED)
69 public ResponseEntity<Livro> atualizar(@PathVariable("id") Integer id,
70                                     @RequestBody Livro livro) {
71     if (servico.porCodigo(id).isEmpty()) {
72         return ResponseEntity.notFound().build();
73     }
74
75     livro.setId(id);
76     livro = servico.salvar(livro);
77
78     return ResponseEntity.ok(livro);
79 }
80
81 @DeleteMapping("/{id}")
82 public ResponseEntity<Void> excluir(@PathVariable("id") Integer id) {
83     try {
84         servico.excluir(id);
85     } catch (ObjectNotFoundException e) {
86         return ResponseEntity.notFound().build();
87     }
88
89     return ResponseEntity.noContent().build();
90 }
91 }
```