## Spring Data JPA

O que é a para que Serve

## O que é?

O Spring Data JPA é um **Framework** de **Persistência de Dados** lançado nos primeiros meses de 2011 com a finalidade de facilitar o processo de implementação das classes direcionadas a operações de **CRUD** (Create, Read, Update, Delete) e a criação de repositórios

## O que é um Framework?

Framework uma ferramenta que ajuda o desenvolvedor a codificar melhor e mais rápido, unindo códigos comuns entre vários projetos de software, servindo como um suporte ou guia para a construção de algo.

# O que siginifica o termo: Persistência de Dados ?

Persistência de Dados é um termo utilizado para dizer que os dados foram gravados em algum lugar e que não se perderão quando o computador for desligado. Ou seja, qualquer tipo de aplicação que grave um estado possível de ser recuperado, está fazendo persistência.

### Para que serve o Spring Data JPA?

O Spring Data JPA libera o programador de ter que implementar as interfaces referentes aos repositórios e também deixa pré-implementado algumas funcionalidades como ordenação das consultas e paginação de registros. Já conta também com algumas classes e interfaces que reduzem a complexidade e a quantidade de código fonte.

Um projeto que não utiliza **Spring Data JPA**, geralmente tem uma interface com alguns métodos de consulta definidos que posteriormente são codificados em uma classe concreta. Quando se faz uso de **Spring Data**, basta incluir a assinatura do método de pesquisa na interface e adicionar a esta assinatura, uma anotação do tipo @Query, com a respectiva consulta no formato JPQL. A implementação deste método será realizada automaticamente pelo **Spring Data** em tempo de execução.

### Alguns exemplos de utilização

No **Spring Data** já existem todos os métodos necessários para fazer um **CRUD**. Mas antes, temos construir a entidade:

```
@Entity
     public class Produto implements Serializable {
 45
       private static final long serialVersionUID = 1L;
 6789
       @Id
       @GeneratedValue
       private Long id;
10
       private String nome;
11
12
13
       private String descricao;
14
15
       private boolean ativo;
16
17
       @Temporal(TemporalType.TIMESTAMP)
18
19
20
       private Date cadastro;
21
       private int quantidade;
22
23
       // getters e setters omitidos
```

# Exemplo de como criar o repositório para a entidade Produto e nome **Produtos**:

```
public interface Produtos extends JpaRepository<Produto, Long> {
}
```

A implementação da interface **Produtos**, será disponibilizada pelo próprio **Spring Data JPA** em tempo de execução.

Exemplo de **Ordenação de Registros** com um método recebendo o tipo Sort como parâmetro e permitindo receber da requisição, a propriedade pela qual a consulta será ordenada e em qual direção (ascendente ou descendente):

```
Public List<Produto> pesquisar(
    @RequestParam(defaultValue = "nome") String ordenacao,
    @RequestParam(defaultValue = "ASC") Sort.Direction direcao) {
    return produtos.findAll(new Sort(direcao, ordenacao));
}
```

A facilidade mais interessante desse framework, é a criação de consultas a partir da **Assinatura do Método**. Sendo assim, exemplificaremos uma consulta para retornar o produto pelo nome:

```
public interface Produtos extends JpaRepository<Produto, Long> {
    Produto findByNome(String nome);
}
```

Observe também que não necessário dar uma implementação para esse método findByNome

O recurso de criar consultas pela **assinatura do método** é bastante útil mas não resolverá todos os problemas. Eventualmente, ainda será necessário executar consultas com JPQL. Os desenvolvedores do **Spring Data JPA** sabendo disso, incluíram a anotação **@Query** para os usuários do Framework

# Segue um exemplo que também não precisa dar a implementação para o método:

```
public interface Produtos extends JpaRepository<Produto, Long> {
    ...

@Query("from Produto where nome like concat(?1, '%')")
    List<Produto> pesquisarProdutos(String nome);
}
```

Repare que a consulta e o método recebem um parâmetro para que o **Spring Data** possa utilizá-lo na hora a busca. O nome **pesquisarProdutos**, não precisa seguir uma regra pois a consulta válida é a que está na anotação.

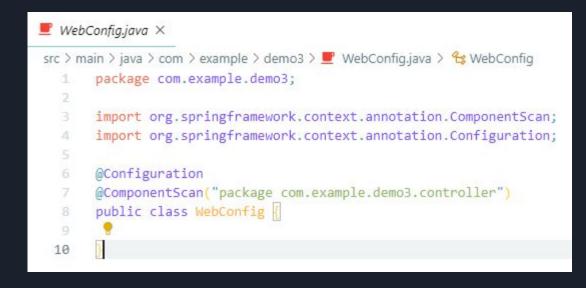
```
■ Aluno.java •
package com.example.demo3.model;
      import javax.persistence.Entity;
      import javax.persistence.GeneratedValue;
      import javax.persistence.GenerationType;
      import javax.persistence.Id;
      @Entity
      public class Aluno {
          @Id
          @GeneratedValue(strategy = GenerationType.AUTO)
          private Integer id;
          private String firstName;
          private String lastName;
          public Integer getId() {
              return id;
          public void setId(Integer id) {
              this.id = id;
          public String getFirstName() {
              return firstName;
          public void setFirstName(String firstName) {
              this.firstName = firstName;
          public String getLastName() {
              return lastName;
          public void setLastName(String lastName) {
              this.lastName = lastName;
          public Aluno(String firstName, String lastName) {
              this.firstName = firstName;
              this.lastName = lastName;
          public Aluno() {
          @Override
          public String toString() {
              return "Aluno [firstName=" + firstName + ", id=" + id + ", lastName=" + lastName + "]";
 46
```

```
■ Aluno,java ■ IlunoRepository,java ×

src > main > java > com > example > demo3 > repository > 💆 AlunoRepository.java > ...
       package com.example.demo3.repository;
       import com.example.demo3.model.Aluno;
       import org.springframework.data.repository.CrudRepository;
       // This will be AUTO IMPLEMENTED by Spring into a Bean called AlunoRepository
       // CRUD refers Create, Read, Update, Delete
       public interface AlunoRepository extends CrudRepository (Aluno, Integer) {
           Aluno findByLastName(String lastName);
           Aluno findById(long id);
 17
```

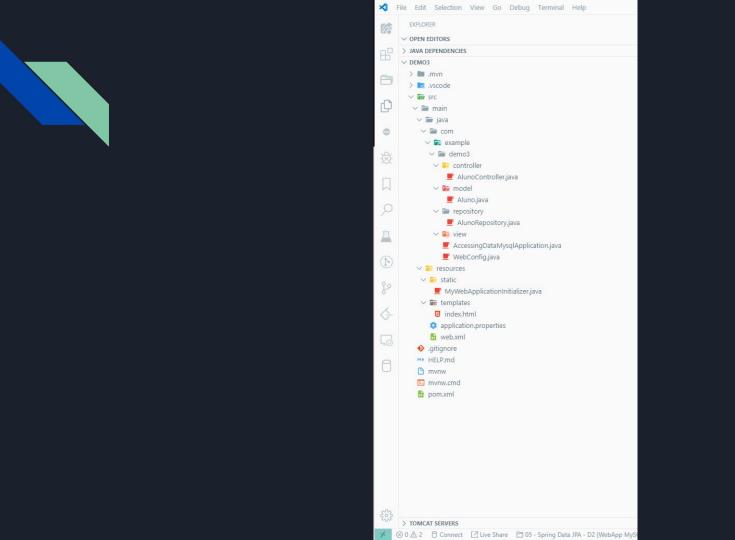
```
■ Aluno.java 
■ AlunoController.java 
■
src > main > java > com > example > demo3 > controller > P AlunoController.java > ...
       import org.springframework.beans.factory.annotation.Autowired;
       import org.springframework.stereotype.Controller;
       import org.springframework.ui.Model;
       import org.springframework.web.bind.annotation.GetMapping;
       import org.springframework.web.bind.annotation.PostMapping;
       import org.springframework.web.bind.annotation.RequestMapping;
      import org.springframework.web.bind.annotation.RequestParam;
       import org.springframework.web.bind.annotation.ResponseBody;
       import org.springframework.web.servlet.view.RedirectView;
       @Controller
       @RequestMapping("/")
       public class AlunoController {
           @Autowired // Which is auto-generated by Spring, we will use it to handle the data
           private AlunoRepository alunoRepository;
           @GetMapping(path = "/index")
           public String showPageNow(Model model, @RequestParam(defaultValue = "0") int page) {
               model.addAttribute("alunos", alunoRepository.findAll());
               return "index";
           @GetMapping(path = "/")
           public String showPage(Model model) {
               model.addAttribute("alunos", alunoRepository.findAll());
               return "index";
           @GetMapping(path = "/all")
           public @ResponseBody RedirectView Aluno(Model model) {
               model.addAttribute("alunos", alunoRepository.findAll());
               // return "index";
               // return new ModelAndView("forward:/index", model);
               return new RedirectView("index");
           @PostMapping(path = "/add")
           public @ResponseBody RedirectView addNewAluno(@RequestParam String firstName, @RequestParam String lastName) {
               Aluno a = new Aluno();
               a.setFirstName(firstName);
               a.setLastName(lastName);
               alunoRepository.save(a);
               // return "redirect:/":
               // return "redirect:/index";
               return new RedirectView("index");
           @PostMapping(path = "/delete")
           public @ResponseBody RedirectView delAluno(Integer id) {
               alunoRepository.deleteById(id);
               // return "redirect:/index";
               return new RedirectView("index");
```

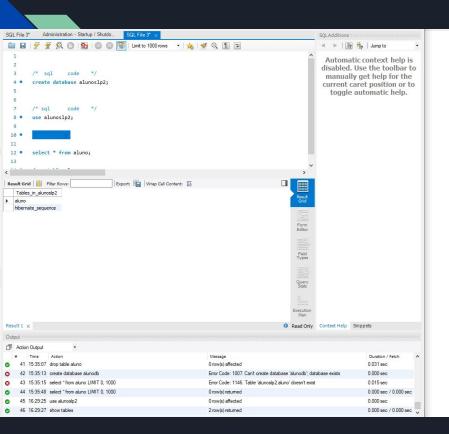
```
AccessingDataMysqlApplication.java ×
src > main > java > com > example > demo3 > 💆 AccessingDataMysqlApplication.java > 😘 AccessingDataMysqlApplication
       package com.example.demo3;
       import org.springframework.boot.SpringApplication;
       import org.springframework.boot.autoconfigure.SpringBootApplication;
       import org.springframework.context.annotation.ComponentScan;
       import org.springframework.context.annotation.Configuration;
       @Configuration
       @ComponentScan("package com.example.demo3.controller")
       @SpringBootApplication
       public class AccessingDataMysqlApplication {
           Run | Debug
           public static void main(String[] args) {
                SpringApplication.run(AccessingDataMysqlApplication.class, args);
  17
```



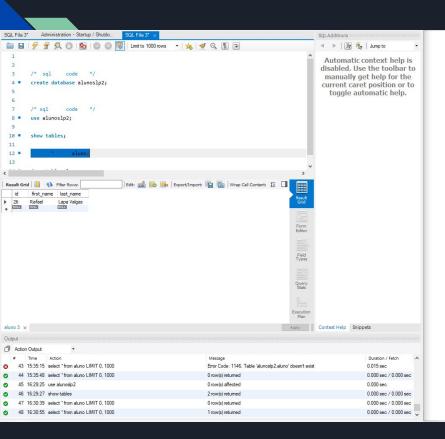
```
application.properties •
src > main > resources > application.properties
       ## MySQL
       #spring.datasource.url=jdbc:mysql://localhost:3306/springdatajpa?useTimezone=true&serverTimezone=UTC
       #spring.datasource.username=root
       #spring.datasource.password=entrar
       #`hibernate sequence' doesn't exist
       #spring.jpa.hibernate.use-new-id-generator-mappings=false
       # drop n create table, good for testing, comment this in production
       # spring.jpa.hibernate.ddl-auto=create
       #2 versão
       spring.jpa.hibernate.ddl-auto=update
       spring.datasource.url=jdbc:mysql://localhost:3306/alunoslp2?useTimezone=true&serverTimezone=UTC
       spring.datasource.username=root
       spring.datasource.password=entrar
          /* sal
                       code
               create database alunoslp2;
               use alunoslp2;
               show tables;
               select * from aluno;
               drop table aluno;
  34
```

```
src > main > resources > templates > 🥫 index.html > 🔗 html > 💬 body > 🤣 div.container..container-fluid > 🔗 section.container..container-fluid > 😭 table.table.t
      </head>
      <body>
         <div class="container .container-fluid">
             <section class="container .container-fluid">
                <div style="margin: 2em; margin-bottom: 4em;">
                   <h1>Linguagem de Programação 2<span></h1>
             </section>
             <section class="container .container-fluid">
                <thead class="thead-dark">
                       #
                       Nome
                       Sobrenome
                       <span style="color: ■ black;">.</span>
                   </thead>
                        Cadê os alunos...? 
                          <span th:text=" ${alunos.id}"> id</span>
                          <span th:text=" ${alunos.firstName}"> id</span>
                          <span th:text=" ${alunos.lastName}"> id</span>
                              <form action="/delete" method="post">
                                 <input type="text" name="id" th:value="${alunos.id}" style="display:none;">
 59
                                kinput class="btn btn-primary" type="submit" value="delete"
                              </form>
                       </section>
             <section class="container .container-fluid">
                <div class="form-group">
                   <form action="/add" method="post">
                       <div><input type="text" class="form-control" name="firstName" placeholder="First name"><br> </div>
                       <div><input type="text" class="form-control" name="lastName" placeholder="Last name"><br></div>
                       <div><input type="submit" value="add" class="btn btn-primary"</pre>
                             style="width: 20%; margin: auto; margin-top: 1em;"></div>
                   </form>
             </section>
         </div>
```

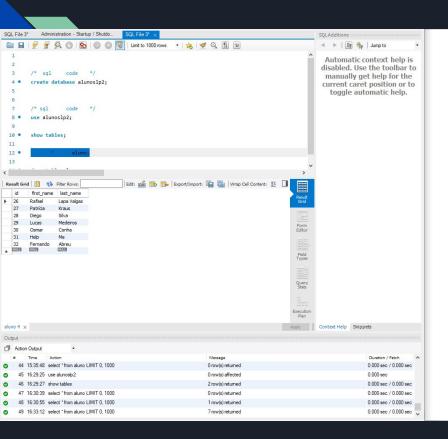




#	Nome
Cadê os alunos?	
-	
First name	
Last name	
add	

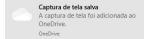


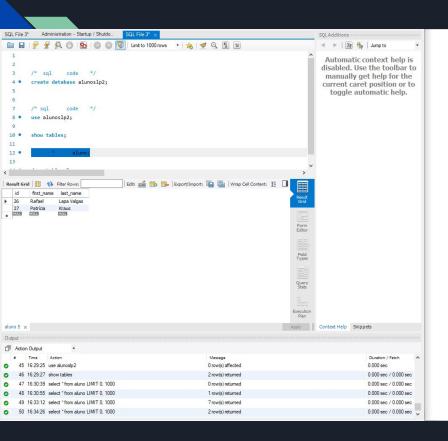
#	Nome	Sobrenome	
26	Rafael	Lapa Valgas	delete
First name			
Last name			
add			
add			



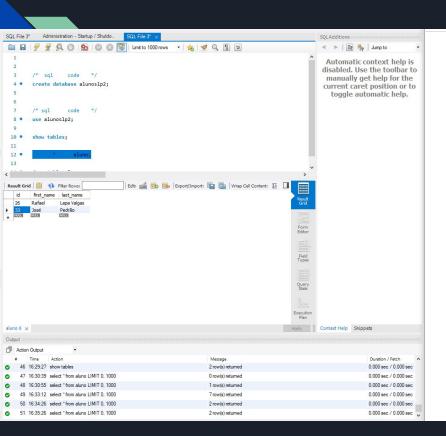
#	Nome	Sobrenome	,
26	Rafael	Lapa Valgas	delete
27	Patrícia	Kraus	delete
28	Diego	Silva	delete
29	Lucas	Medeiros	delete
30	Osmar	Conha	delete
31	Help	Ме	delete
32	Fernando	Abreu	delete
First name			
Last name			

add





#	Nome	Sobrenome	
26	Rafael	Lapa Valgas	delete
27	Patrícia	Kraus	delete
First name			
Last name			
add			



#	Nome	Sobrenome	
26	Rafael	Lapa Valgas	delete
33	José	Pedrão	delete
First name			
Last name			
add			

### VLW GALERA!! ;)

## Spring Data JPA

O que é a para que Serve

