



# CRUD - Spring Boot

Herysson R. Figueiredo  
herysson.figueiredo@ufn.edu.br



# Spring initializr

- Dependências
  - Spring web;
  - MySQL Driver
  - Spring Data JPA
  - Thymeleaf
  - Lombok



# Spring initializr

Lombok notation:

<https://projectlombok.org/features/Data>

## Project

☐ Gradle - Groovy

☐ Gradle - Kotlin

☒ Maven

## Language

☒ Java

☐ Kotlin

☐ Groovy

## Spring Boot

☐ 3.1.1 (SNAPSHOT)

☒ 3.1.0

☐ 3.0.8 (SNAPSHOT)

☐ 3.0.7

☐ 2.7.13 (SNAPSHOT)

☐ 2.7.12

## Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ Jar ☐ War

Java ☐ 20 ☒ 17 ☐ 11 ☐ 8

## Dependencies

ADD DEPENDENCIES... CTRL + B

### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

### MySQL Driver SQL

MySQL JDBC driver.

### Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

### Thymeleaf TEMPLATE ENGINES

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

### Lombok DEVELOPER TOOLS

Java annotation library which helps to reduce boilerplate code.

## Project

- ☐ Gradle - Groovy ☒ **Maven** ☒ **Java** ☐ Kotlin

## Spring Boot

- ☐ 3.1.1 (SNAPSHOT) ☒ **3.1.0** ☐ 3.0.8 (SNAPSHOT)
- ☐ 3.0.7 ☐ 2.7.13 (SNAPSHOT) ☐ 2.7.12

## Project Metadata

Group

Artifact

Name

Description

Package name

Packaging ☒ **Jar** ☐ War

Java ☐ 20 ☒ **17** ☐ 11 ☐ 8

## Dependencies

ADD DEPENDENCIES... CTRL + B

### Spring Web **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

### MySQL Driver **SQL**

MySQL JDBC driver.

### Spring Data JPA **SQL**

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

### Thymeleaf **TEMPLATE ENGINES**

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

### Lombok **DEVELOPER TOOLS**

Java annotation library which helps to reduce boilerplate code.

Project ▾



▾ **CRUD** C:\IntelliJ-Workspace\CRUD

> .idea

> .mvn

▾ src

▾ main

▾ java

▾ com

▾ example

▾ CRUD

CrudApplication.java

> resources

> test

.gitignore

HELP.md

mvnw

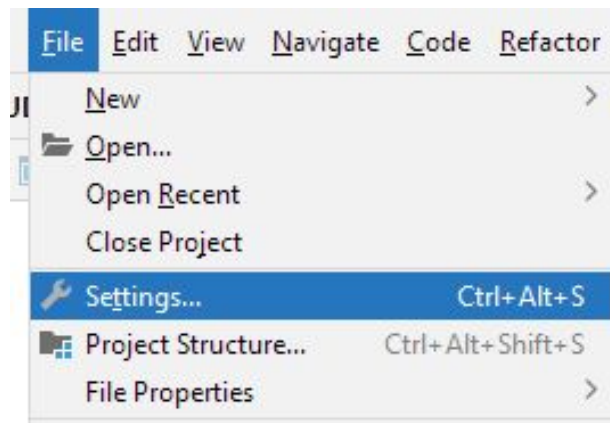
mvnw.cmd

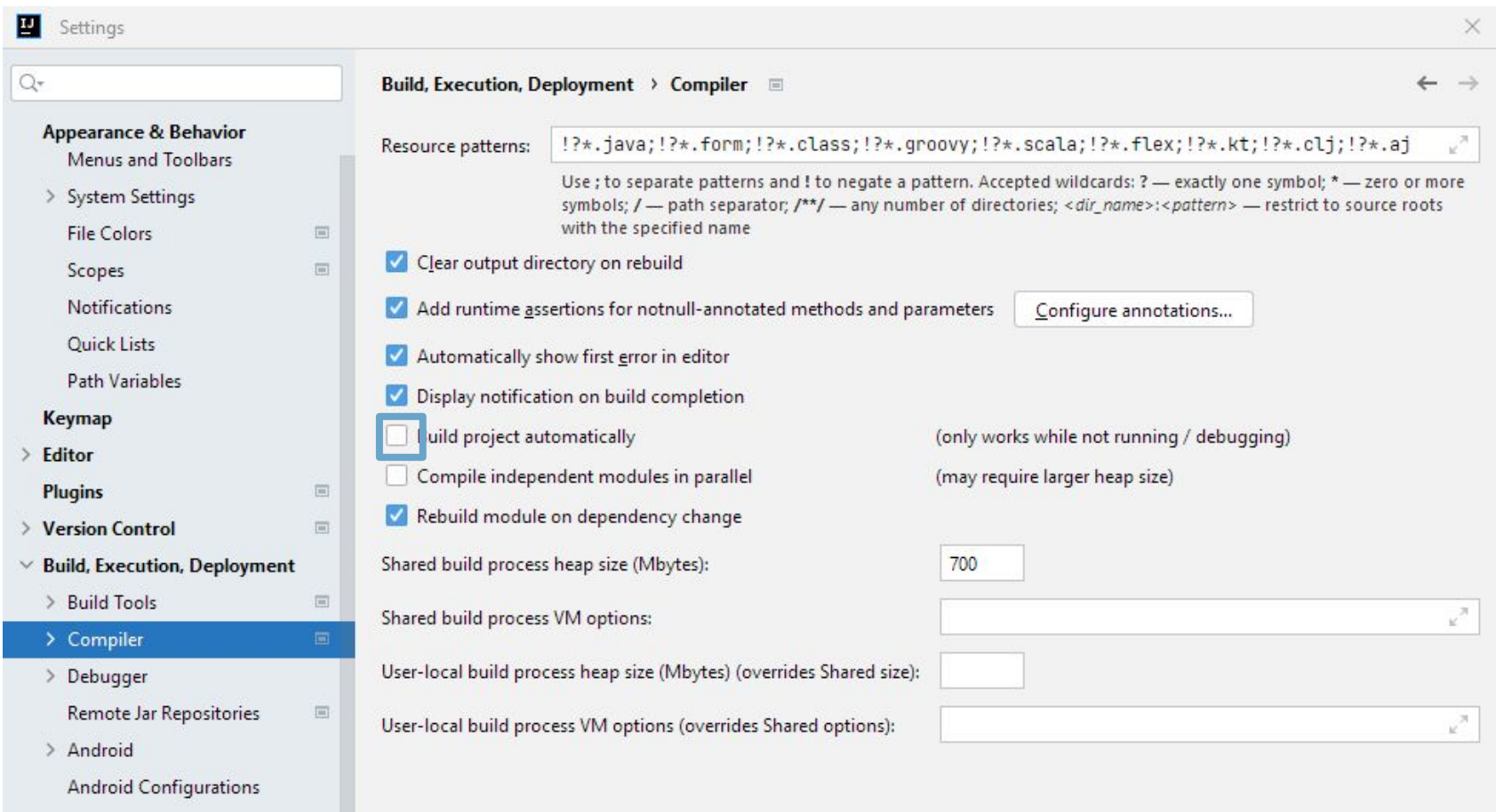
pom.xml

External Libraries

Scratches and Consoles

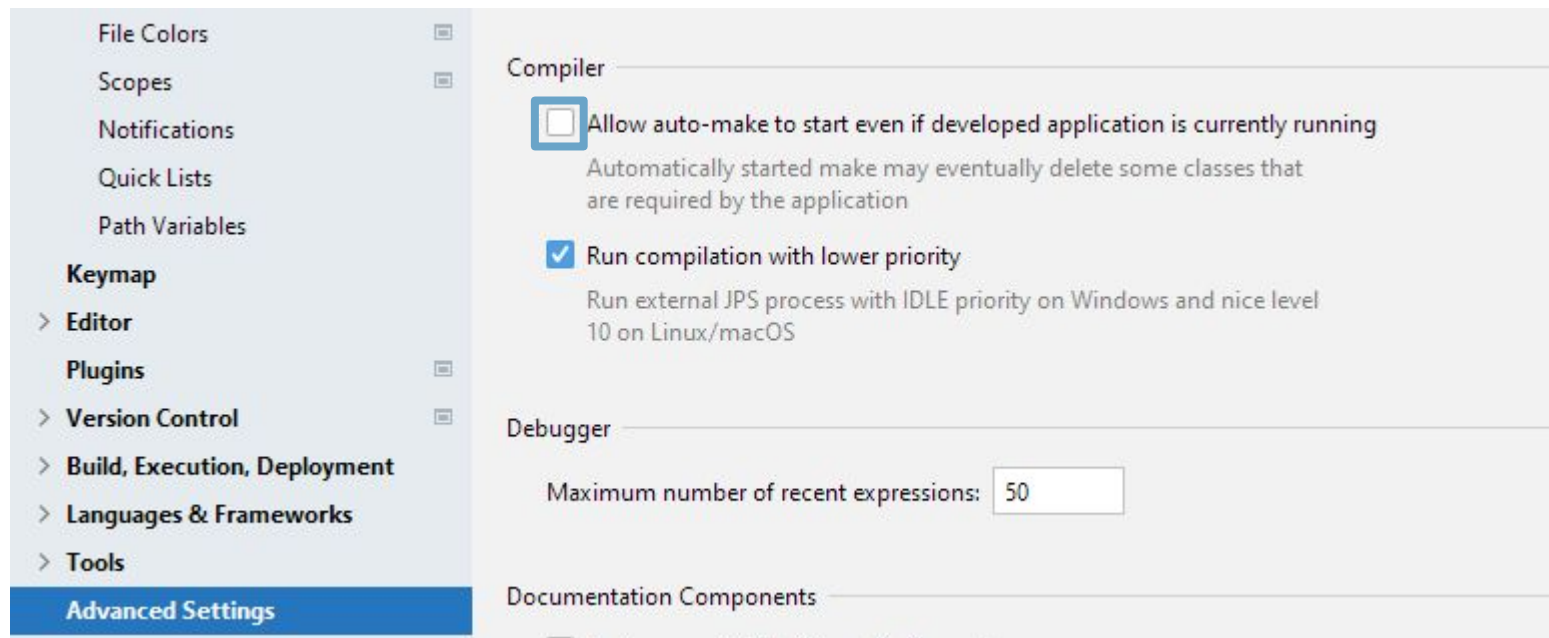
# Reload Automático







# Reload Automático



The screenshot displays the IntelliJ IDEA settings interface. On the left, a sidebar lists various settings categories: File Colors, Scopes, Notifications, Quick Lists, Path Variables, Keymap, Editor, Plugins, Version Control, Build, Execution, Deployment, Languages & Frameworks, Tools, and Advanced Settings (highlighted in blue). The main panel shows the 'Compiler' section. It contains two settings: 'Allow auto-make to start even if developed application is currently running' (unchecked) and 'Run compilation with lower priority' (checked). The 'Run compilation with lower priority' setting has a description: 'Run external JPS process with IDLE priority on Windows and nice level 10 on Linux/macOS'. Below the 'Compiler' section is the 'Debugger' section, which includes a 'Maximum number of recent expressions' input field set to 50. At the bottom, the 'Documentation Components' section is partially visible.

File Colors

Scopes

Notifications

Quick Lists

Path Variables

**Keymap**

> Editor

Plugins

> Version Control

> Build, Execution, Deployment

> Languages & Frameworks

> Tools

**Advanced Settings**

**Compiler**

☐ Allow auto-make to start even if developed application is currently running

Automatically started make may eventually delete some classes that are required by the application

☒ Run compilation with lower priority

Run external JPS process with IDLE priority on Windows and nice level 10 on Linux/macOS

**Debugger**

Maximum number of recent expressions: 50

**Documentation Components**



# Reload Automático - Dependência

```
<dependency>  
  
  <groupId>org.springframework.boot</ groupId>  
  
  <artifactId>spring-boot-devtools</ artifactId>  
  
  <scope>runtime</ scope>  
  
  <optional>true</ optional>  
  
</dependency>
```



## Model - Pessoa

- Crie o package model
  - Crie a Classe Pessoa

Project ▾



▾ **CRUD** C:\IntelliJ-Workspace\CRUD

- > .idea
- > .mvn
- ▾ src
  - ▾ main
    - ▾ java
      - ▾ com
        - ▾ example
          - ▾ CRUD
- > resources
- > test
- .gitignore
- HELP.md
- mvnw
- mvnw.cmd
- pom.xml
- External Libraries
- Scratches and Consoles

New >

Cut Ctrl+X

Copy Ctrl+C

Copy Path/Reference...

Paste Ctrl+V

Find Usages Alt+F7

Find in Files... Ctrl+Shift+F

Replace in Files... Ctrl+Shift+R

**F**ile

**S**cratch File Ctrl+Alt+Shift+Insert

**D**irectory

**H**TML File

**S**tylesheet

**J**avaScript File

**T**ypeScript File

**p**ackage.json

**D**ockerfile



## Pessoa.java

```
import jakarta.persistence.*;

@Entity //Isso fala para o JPA - Hibernate fazer uma tabale no banco
public class Pessoa {
    3 usages
    @Id // diz para o banco que o atributo abaixo é um PK
    @GeneratedValue(strategy = GenerationType.AUTO)//auto_increment
    private Integer id;
    4 usages
    @Column(nullable = false)//coluna não pode ser nulla
    private String nome;
    4 usages
    @Column(nullable = false, unique = true)//Coluna email é unica e não nulla
    private String email;
    4 usages
    @Column(length = 15)// Coluna abaixo é um varhcar(15)
    private String fone;
```



## Pessoa.java

```
public Pessoa() {  
}  
  
public Pessoa(Integer id, String nome, String email, String fone) {  
    this.id = id;  
    this.nome = nome;  
    this.email = email;  
    this.fone = fone;  
}  
  
public Pessoa(String nome, String email, String fone) {  
    this.nome = nome;  
    this.email = email;  
    this.fone = fone;  
}
```



## Pessoa.java

```
public Integer getId() {  
    return id;  
}  
  
public void setId(Integer id) {  
    this.id = id;  
}  
  
public String getNome() {  
    return nome;  
}  
  
public void setNome(String nome) {  
    this.nome = nome;  
}  
  
public String getEmail() {  
    return email;  
}  
  
public void setEmail(String email) {  
    this.email = email;  
}  
  
public String getFone() {  
    return fone;  
}  
  
public void setFone(String fone) {  
    this.fone = fone;  
}  
}
```



## Repository - Pessoa

- Crie o package repository
  - Crie a Interface Pessoa





# PessoaRepository.java

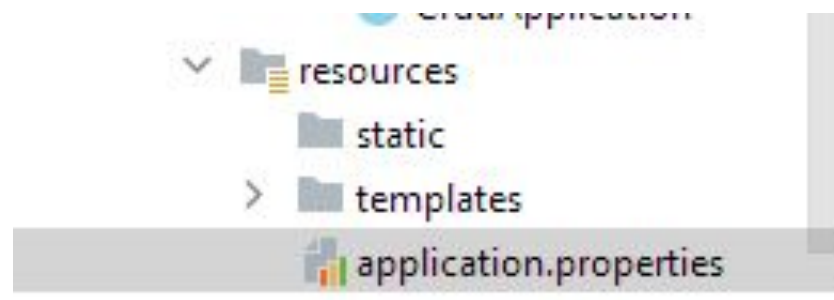
New Java Class	
I	Name
C	Class
I	Interface
R	Record
E	Enum
@	Annotation



## PessoaRepository.java

```
import com.example.CRUD.model.Pessoa;  
import org.springframework.data.repository.CrudRepository;  
//CRUD Creat, Read, Update, Delete da de objetos da classe Pessoa  
public interface PessoaRepository extends CrudRepository<Pessoa, Integer> {  
  
}
```

## Configuração JPA





# Configuração JPA

*#Update the database schema based on the entities*

```
spring.jpa.hibernate.ddl-auto =update
```

*#create database if not exists mysql true spring boot*

```
spring.datasource.url =jdbc:mysql://localhost:3306/CRUD?createDatabaseIfNotExist=true
```

*#DB User*

```
spring.datasource.username =root
```

*#DB password*

```
spring.datasource.password =laboratorio
```



## Controller - Pessoa

- Crie o package controller
  - Crie a classe PessoaController.java



## PessoaController.java

```
import com.example.CRUD.model.Pessoa;
import com.example.CRUD.repository.PessoaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller//Isso significa que esta classe é uma controller
@RequestMapping(path = "/pessoa")//Este mapping acessa esta classe
public class PessoaController {
```



## PessoaController.java

```
import com.example.CRUD.model.Pessoa;
import com.example.CRUD.repository.PessoaRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@Controller//Isso significa que esta classe é uma controller
@RequestMapping(path = "/pessoa")//Este mapping acessa esta classe
public class PessoaController {
```



# PessoaController.java

```
@GetMapping ("/cadastrar")

public String cadastrarPessoa (Model model){

    //maadar um objeto do tipo Pessoa para a pagina cadastro

    model.addAttribute("pessoa", new Pessoa());

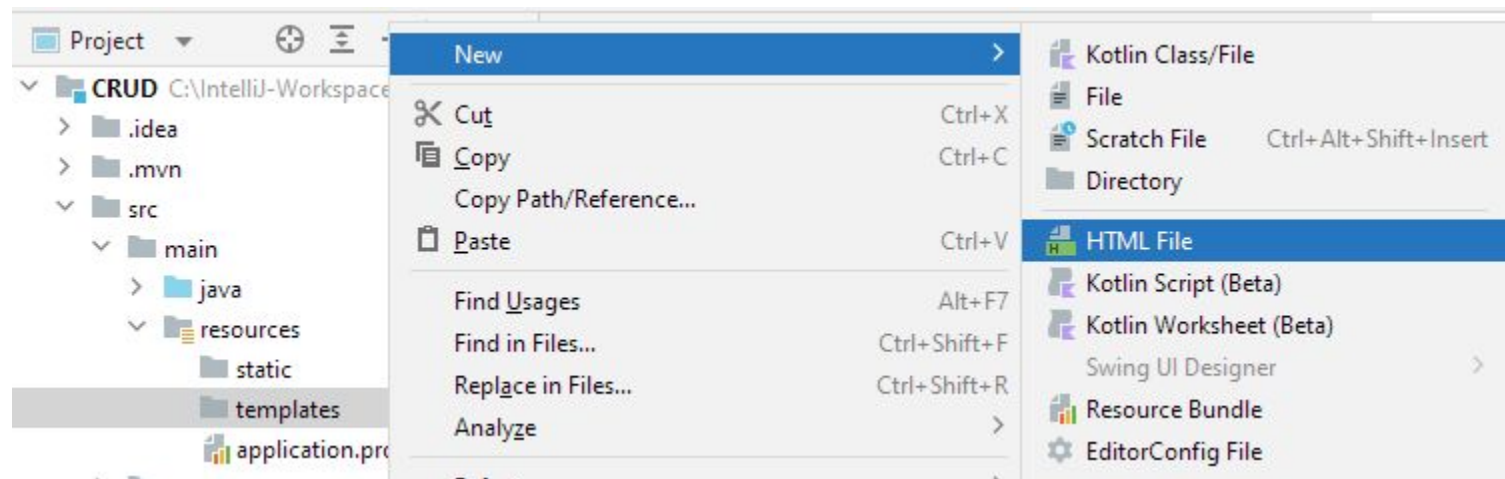
    //redirecionando para pagina cadastrarPessoa.html

    return "cadastrarPessoa";

}
```



# cadastrarPessoa.html



cada

```
<body>
<form action="#" th:action="@{/pessoa/save}" th:object="${pessoa}" method="post">
  <table>
    <tr>
      <td>Nome*: </td>
      <td><input type="text" th:field="*{nome}" required/></td>
    </tr>
    <tr>
      <td>E-mail*: </td>
      <td><input type="text" th:field="*{email}" required/></td>
    </tr>
    <tr>
      <td>Fone*: </td>
      <td><input type="text" th:field="*{fone}" required/></td>
    </tr>
  </table>
  <input type="submit"/>
</form>
</body>
```



# PessoaController.Java

```
@GetMapping("/listar")

public String listarProdutos(@ModelAttribute Pessoa pessoa, Model model) {

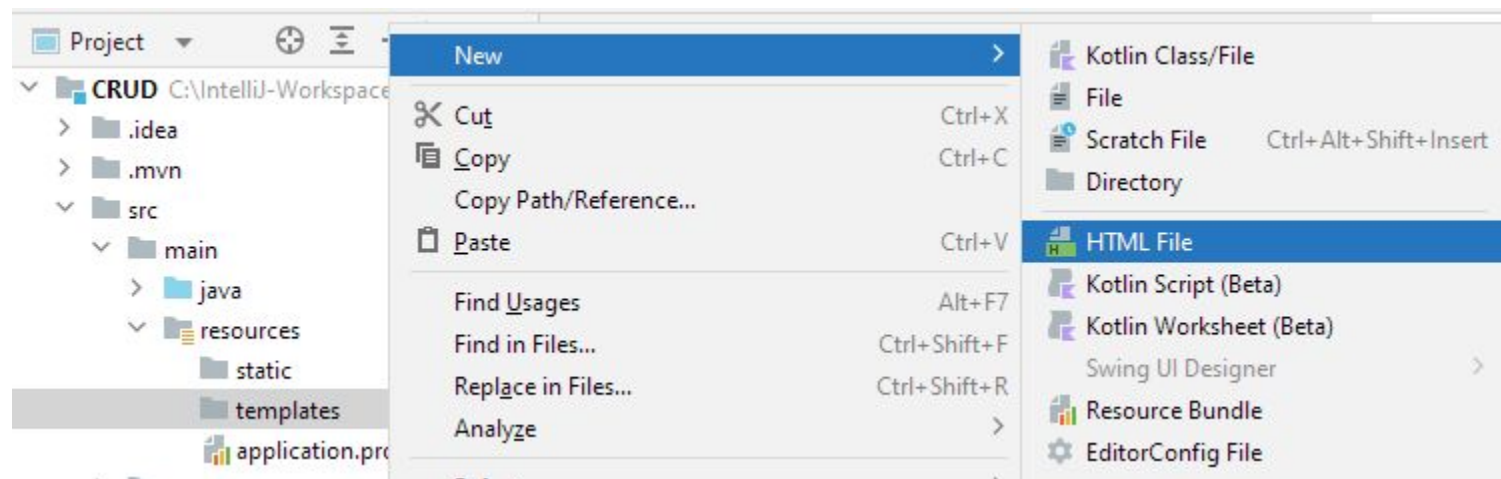
    List<Pessoa> listaPessoa = (List<Pessoa>) pessoaRepository.findAll();

    model.addAttribute("pessoas", listaPessoa);

    return "listarPessoas";

}
```

# listarPessoas.html



listarF

```
<h1>Lista de Pessoas</h1>
<table>
  <tr>
    <td>ID</td>
    <td>Nome</td>
    <td>Email</td>
    <td>Fone</td>
  </tr>
  <tr th:each="p : ${pessoas}">
    <td th:text="${p.id}"></td>
    <td th:text="${p.nome}"></td>
    <td th:text="${p.email}"></td>
    <td th:text="${p.fone}"></td>
    <td>
      <a th:href="@{/pessoa/alterar/{id}(id=${p.id})}">Editar</a>
      <a th:href="@{/pessoa/excluir/{id}(id=${p.id})}"> Excluir</a>
    </td>
  </tr>
</table>
<a href="/pessoa/cadastrar">Cadastrar outra pessoa</a>
```



# PessoaController.Java

```
@PostMapping ("/save")

public String salvarPessoa (@ModelAttribute Pessoa pessoa, Model model){

    //Salva no banco o objeto do tipo Pessoa com as informações da pagina cadastro

    pessoaRepository .save (pessoa) ;

    //Cria uma lista atualizada das pessoas cadastradas

    List<Pessoa> listaPessoa = (List<Pessoa>) pessoaRepository .findAll();

    model.addAttribute( "pessoas", listaPessoa );

    return "listarPessoas" ;

}
```



## PessoaController.Java

```
@GetMapping ("/excluir/{id}")

public String excluirPessoa (Model model, @PathVariable Long id) {

    pessoaRepository.deleteById(Math.toIntExact(id));

    List<Pessoa> listaPessoa = (List<Pessoa>) pessoaRepository.findAll();

    model.addAttribute("pessoas", listaPessoa);

    return "listarPessoas";

}
```



# PessoaController.Java

```
@GetMapping("/alterar/{id}")

public String altPessoa(@PathVariable Long id, Model model) {

    Pessoa p = pessoaRepository.findById(Math.toIntExact(id)).get();

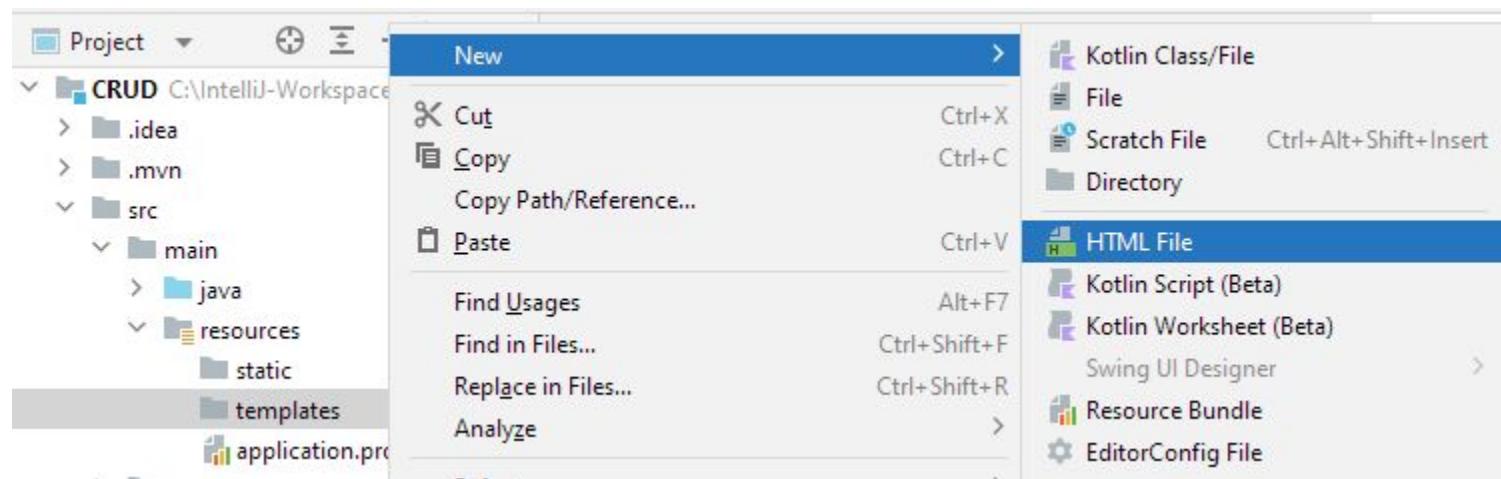
    model.addAttribute("pessoa", p);

    return "alterarPessoa";

}
```



## alterarPessoa.html



## alterarPessoa

```
<h1>Alterar Pessoa</h1>
<form action="#" th:action="@{/pessoa/alterar}" th:object="${pessoa}" method="post">
  <table>
    <tr>
      <td>ID:</td>
      <td><input type="text" th:field="*{id}" readonly/></td>
    </tr>
    <tr>
      <td>Nome*:</td>
      <td><input type="text" th:field="*{nome}" required/></td>
    </tr>
    <tr>
      <td>E-mail*:</td>
      <td><input type="text" th:field="*{email}" required/></td>
    </tr>
    <tr>
      <td>Fone*:</td>
      <td><input type="text" th:field="*{fone}" required/></td>
    </tr>
  </table>
  <input type="submit" value="Alterar"/>
</form>
```



# PessoaController.Java

```
@PostMapping ("/alterar")

public String alterarProduto (@ModelAttribute Pessoa novaPessoa, Model model) {

    pessoaRepository.save(novaPessoa);

    List<Pessoa> listaPessoa = (List<Pessoa>) pessoaRepository.findAll();

    model.addAttribute("pessoas", listaPessoa);

    return "listarPessoas";

}
```



## Bibliografia

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. UML: guia do usuário. 2. ed. Rio de Janeiro: Campus, 2006.

PRESSMAN, Roger S. Engenharia de software. 5. Ed. Rio de Janeiro: McGraw-Hill, 2002.

SOMMERVILLE, I. Engenharia de software. 8. Ed. São Paulo, SP: Addison Wesley, 2007

BEZERRA, Eduardo. Princípios de Análise e Projeto de Sistemas com UML. Rio de Janeiro: Campus, 2006.