

PÓS-GRADUAÇÃO ALFA



UNIALFA
CENTRO UNIVERSITÁRIO
ALVES FARIA

Diego Américo Guedes

Refatoração e Evolução de Software

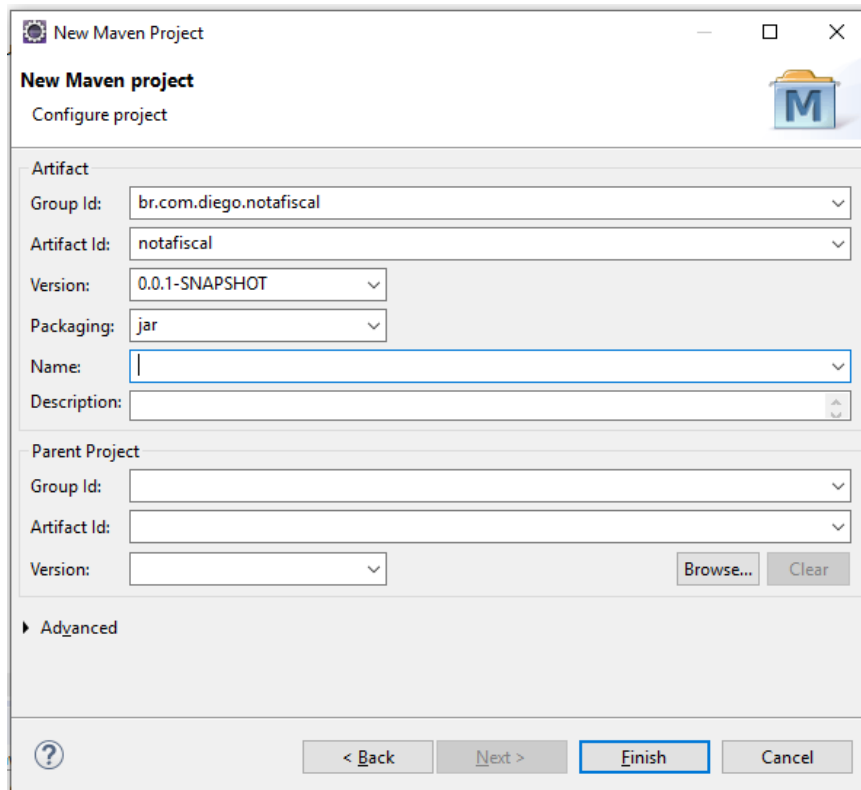


**GRUPO
JOSÉ ALVES**

- Um *framework* (ou arcabouço), em desenvolvimento de software, é uma abstração que une códigos comuns entre vários projetos de software provendo uma funcionalidade genérica.
- Um framework pode atingir uma funcionalidade específica, por configuração, durante a programação de uma aplicação.
- Ao contrário das bibliotecas, é o framework quem dita o fluxo de controle da aplicação, chamado de Inversão de Controle



- New project → New Maven project



New Maven Project
Configure project

Artifact

Group Id:

Artifact Id:

Version:

Packaging:

Name:

Description:

Parent Project

Group Id:

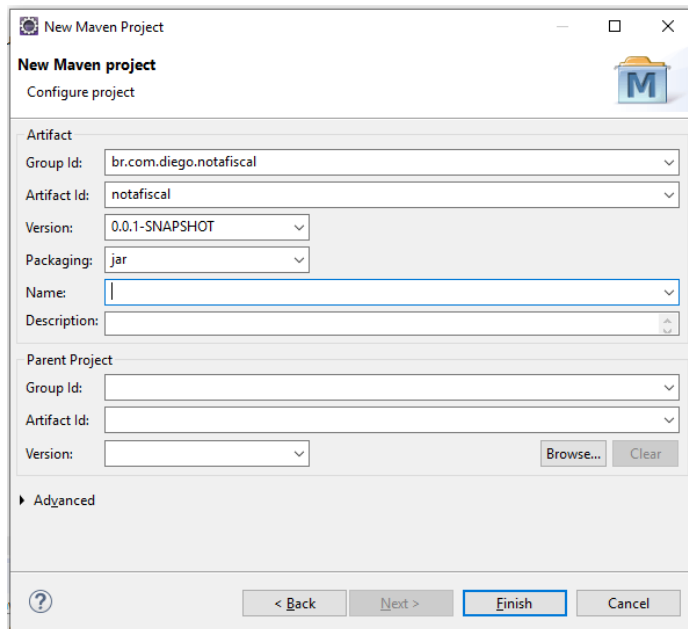
Artifact Id:

Version:

► **Advanced**



- Criar o pacote `br.com.diego.notafiscal`



New Maven Project

Configure project

Artifact

Group Id: `br.com.diego.notafiscal`

Artifact Id: `notafiscal`

Version: `0.0.1-SNAPSHOT`

Packaging: `jar`

Name:

Description:

Parent Project

Group Id:

Artifact Id:

Version:

Browse... Clear

Advanced

< Back Next > Finish Cancel



- Copiar para o pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
4.0.0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>br.com.diego.notafiscal</groupId>
<artifactId>notafiscal</artifactId>
<version>0.0.1-SNAPSHOT</version>
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
<version>1.3.6.RELEASE</version>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-thymeleaf</artifactId>
<version>1.3.6.RELEASE</version>
</dependency>
</dependencies>
</project>
```



- A vantagem de usar Spring Boot é que não precisamos mais nos preocupar com a instalação e configuração do projeto em um container (tomcat, JBOSS, etc).
- Precisamos apenas configurar o Spring Boot para que inicie um container automaticamente e gerencie todos os nossos Beans.
- Faremos isso por meio das classes Java.



```
package br.com.diego.notafiscal;

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

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

← → ↻ ⓘ localhost:8080

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Fri Sep 13 17:45:51 BRT 2019

There was an unexpected error (type=Not Found, status=404).

No message available


```
package br.com.diego.notafiscal;

import org.springframework.boot.SpringApplication;
import
org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

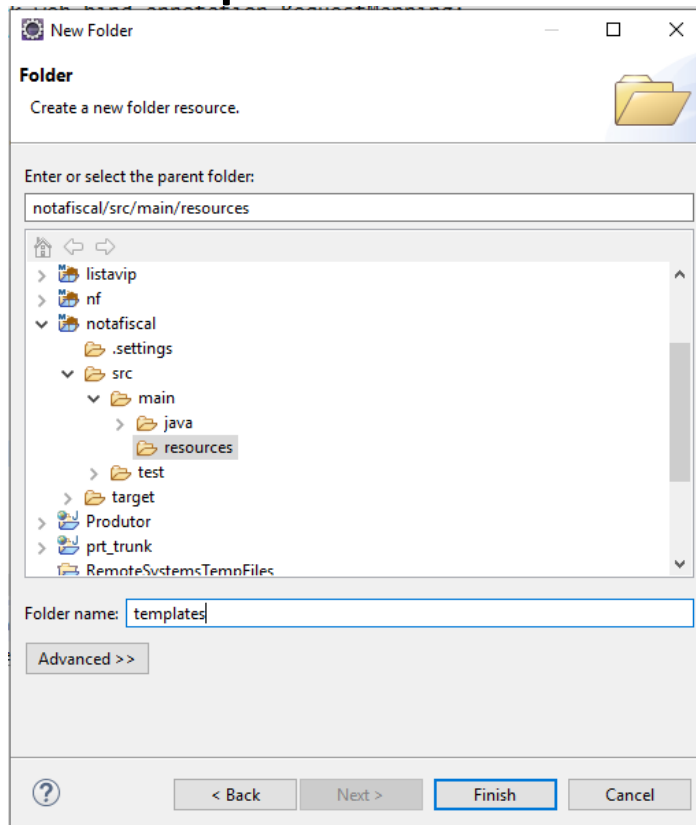
@SpringBootApplication
@Controller
public class WebApplication {

    @RequestMapping("/")
    @ResponseBody
    public String ola() {
        return "Ola, Bem vindo ao sistema de nota fiscal";
    }

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



- Criar um novo diretório (New->Folder) chamado templates



- Criando arquivo index.html em templates

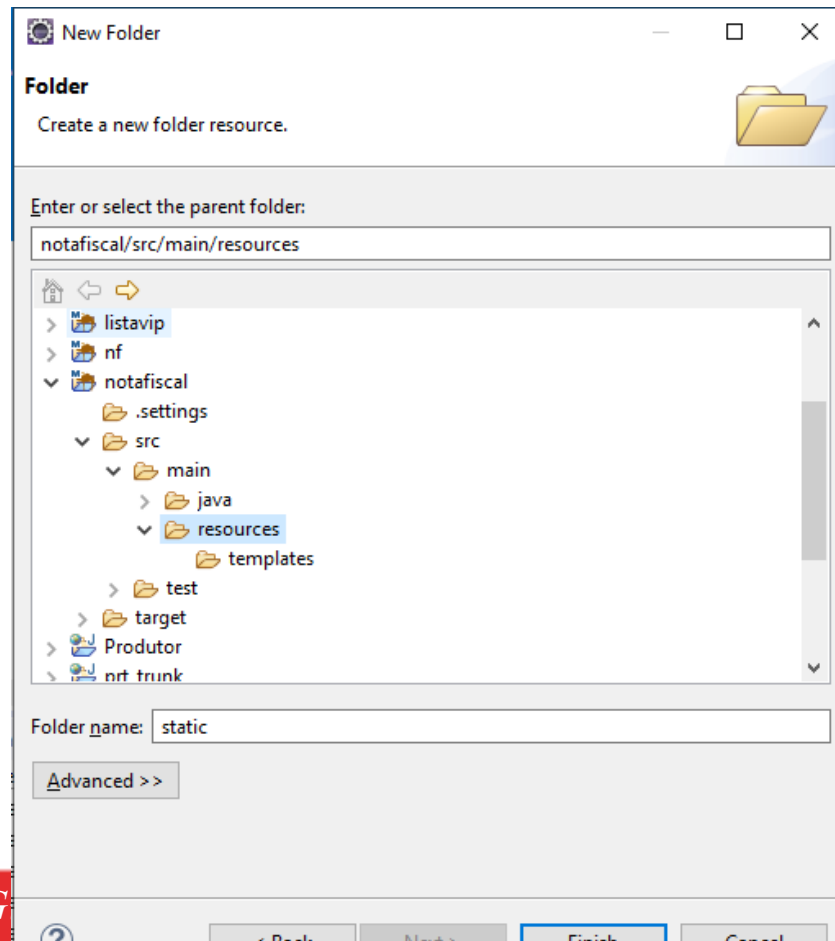
```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>ListaVIP</title>
<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
<div class="container" >
<div class="jumbotron" align="center" style="margin-top: 50px;">
<h1>Seja bem vindo ao Sistema de Notas Fiscais</h1>
<div align="center">
<a href="listanotasfiscais" class="btn btn-lg btn-primary">Clique aqui para ver as notas
fiscais</a>
</div>
</div>
</div>
</div>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script src="bootstrap/js/bootstrap.min.js"></script>

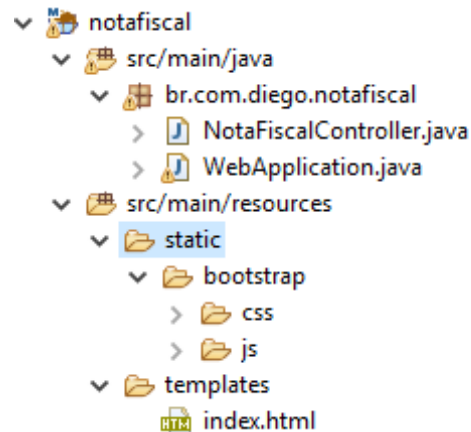
</body>
</html>
```



- Note que usa-se Bootstrap
- Crie um Folder *static* em resources



- Copie o código do bootstrap para ficar igual a imagem a baixo

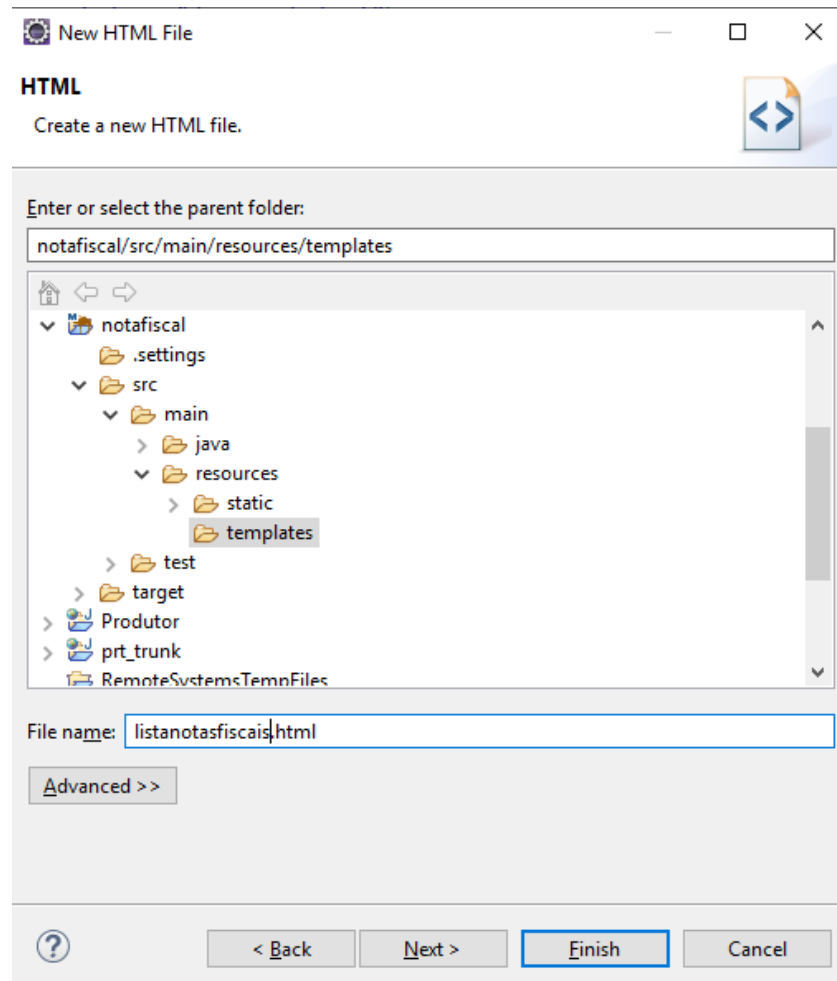


- Criar uma classe *NotaFiscalController* e nesta criaremos um método chamado *index* que mapeará a requisição em / e retornará como *String* o nome do template que criamos, o *index.html*

```
package br.com.diego.notafiscal;  
  
import org.springframework.stereotype.Controller;  
import  
org.springframework.web.bind.annotation.RequestMapping;  
  
@Controller  
public class NotaFiscalController {  
    @RequestMapping("/")  
    public String index() {  
        return "index";  
    }  
}
```



- Criar arquivo html em resources/templates chamado listanotasfiscais.html



- Copie o código abaixo para listanotasfiscais.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>Notas Fiscais</title>
<link href="bootstrap/css/bootstrap.min.css" rel="stylesheet" />
</head>
<body>
<div class="container">
<div id="listaDeNotasFiscais">
<table class="table table-hover">
<thead>
<tr>
<th>Empresa</th>
<th>Valor Bruto</th>
<th>Imposto</th>
</tr>
</thead>
<tr th:each="notafiscal : ${notasfiscais}">
<td><span th:text="${notafiscal.empresa}"></span></td>
<td><span th:text="${notafiscal.valor}"></span></td>
<td><span th:text="${notafiscal.imposto}"></span></td>
</tr>
</table>
</div>
</div>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.11.3/jquery.min.js"></script>
<script src="bootstrap/js/bootstrap.min.js"></script>
</body>
</html>
```



- Precisamos mapear a rota /listaconvidados para o novo template

```
package br.com.diego.notafiscal;

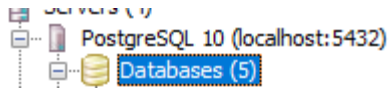
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class NotaFiscalController {
    @RequestMapping("/")
    public String index() {
        return "index";
    }

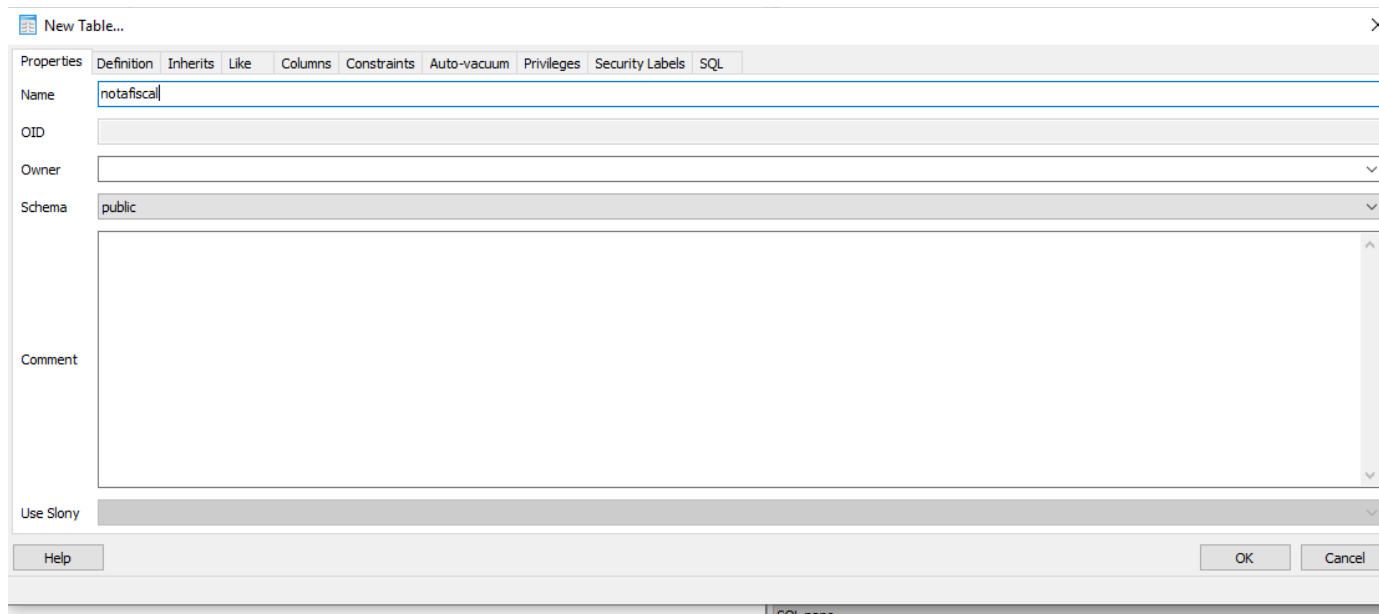
    @RequestMapping("listanotasfiscais")
    public String listaNotasFiscais(){
        return "listanotasfiscais";
    }
}
```



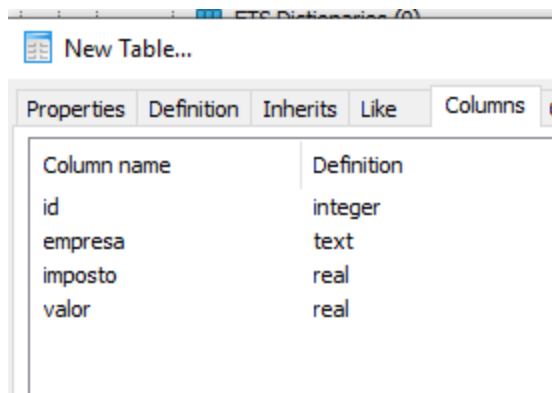
- Hora de criar o banco de dados



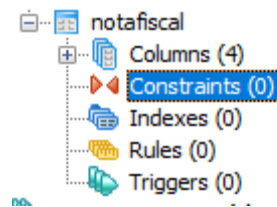
- Botão direito: new database
 - Coloque o nome de contabil
 - No BD contabil, crie uma tabela notafiscal



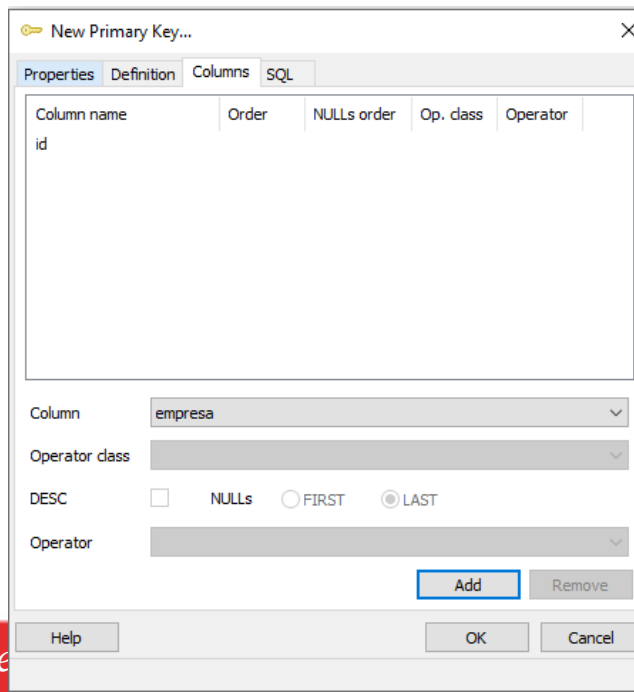
- Crie 4 colunas
 - id – inteiro
 - empresa – Text
 - imposto – real
 - Valor - real



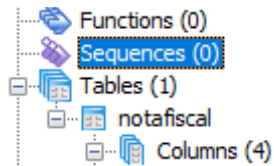
- Adicione uma constraints (chave primaria)



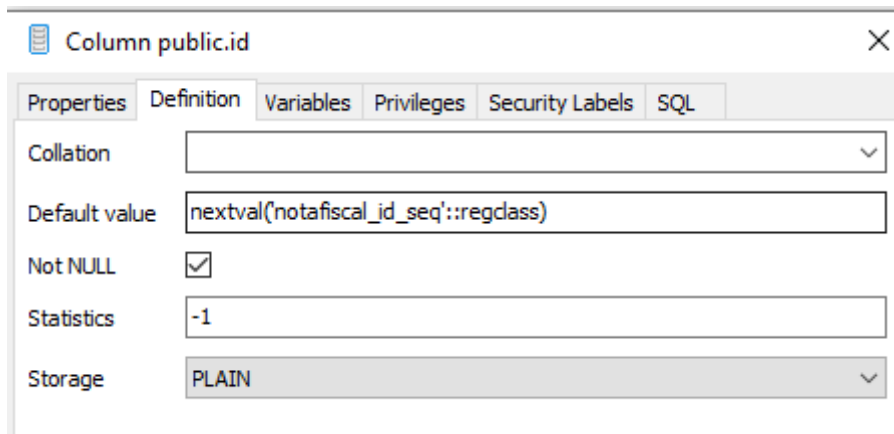
- Colocando id como PK



- Criando um sequence
- Coloque o nome de notafiscal_id_seq



- Atualizar campo id



- Montar esse novo pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-
  4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>br.com.diego.notafiscal</groupId>
    <artifactId>notafiscal</artifactId>
  <version>0.0.1-SNAPSHOT</version>

  <dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>1.3.6.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
    <version>1.3.6.RELEASE</version>
  </dependency>

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>1.3.6.RELEASE</version>
  </dependency>
  <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>9.7.1003-jdbc4</version>
```

Prof. Me. Diego Guedes



UNIALFA
CENTRO UNIVERSITÁRIO
ALVES FARIA

- Para a configuração do banco de dados, vamos utilizar mais um starter, esta chamado de Spring Boot Data JPA Starter, que configura todas as dependências com Hibernate e JPA.
- Na classe Configuracao.java, adicionaremos um novo método responsável por criar o DataSource de conexão ao banco, o anotaremos com @Bean para que este possa ser gerenciado pelo Spring.



- Nova classe WebApplication

```
package br.com.diego.notafiscal;

import javax.sql.DataSource;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

@SpringBootApplication

public class WebApplication {

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();
        dataSource.setUsername("postgres");
        dataSource.setPassword("diego");
        dataSource.setUrl("jdbc:postgresql://localhost:5432/contabil");
        dataSource.setDriverClassName("org.postgresql.Driver");
        return dataSource;
    }

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



- Já temos a página de apresentação as notas fiscais
- Além disso, temos o *controller* que exibe esta página
 - mas não temos uma entidade que representa as notas fiscais
 - Para isto criaremos uma nova classe chamada NotaFiscal



```
package br.com.diego.notafiscal;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity(name = "notafiscal")
public class NotaFiscal {
    @Id
    @GeneratedValue
    private Long id;

    private String empresa;
    private Double imposto;
    private Double valor;

    public Long getId() {
        return id;
    }

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

    public String getEmpresa() {
        return empresa;
    }
}
```

```
public void setEmpresa(String empresa)
{
    this.empresa = empresa;
}

public Double getImposto() {
    return imposto;
}

public void setImposto(Double imposto)
{
    this.imposto = imposto;
}

public Double getValor() {
    return valor;
}

public void setValor(Double valor) {
    this.valor = valor;
}
}
```



- Neste ponto, precisamos fazer com que o controller de convidados possa resgatar os registros no banco de dados e então deixa-los disponíveis para o página exibir.
- O Spring Boot tem disponível um CRUD genérico que permite que façamos isso de forma bem simples.

```
package br.com.diego.notafiscal;  
  
import org.springframework.data.repository.CrudRepository;  
  
public interface NotaFiscalRepository extends CrudRepository<NotaFiscal,  
Long>{  
  
}
```

- Para utilizar o CRUD precisaremos apenas de um atributo do tipo desta interface anotado com *@Autowired* para que o Spring disponibilize um objeto com as características de um repository capaz de retornar objetos de NotaFiscal



```

package br.com.diego.notafiscal;

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

@Controller
public class NotaFiscalController {
    @Autowired
    private NotaFiscalRepository rp;

    @RequestMapping("/")
    public String index() {
        return "index";
    }

    @RequestMapping("listanotasfiscais")
    public String listaNotasFiscais(Model model) {
        Iterable<NotaFiscal> nf = rp.findAll();

        model.addAttribute("notasfiscais", nf);

        return "listanotasfiscais";
    }
}

```



- Implementando a inclusão de uma nota fiscal

```
package br.com.diego.notaFiscal;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;

@Controller
public class NotaFiscalController {
    @Autowired
    private NotaFiscalRepository rp;

    @RequestMapping("/")
    public String index() {
        return "index";
    }

    @RequestMapping("listanotasfiscais")
    public String listaNotasFiscais(Model model) {
        Iterable<NotaFiscal> nf = rp.findAll();

        model.addAttribute("notasfiscais", nf);

        return "listanotasfiscais";
    }

    @RequestMapping(value = "salvar", method = RequestMethod.POST)
    public String salvar(@RequestParam("nome") String nome, @RequestParam("valor") Double valor, Model model) {
        Double i = 1.1;
        NotaFiscal nf = new NotaFiscal(nome, i, valor);
        rp.save(nf);

        Iterable<NotaFiscal> nf_list = rp.findAll();
        model.addAttribute("notasfiscais", nf_list);

        return "listanotasfiscais";
    }
}
```



- Crie 2 impostos (ISS e ICMS)
 - ICMS - 11% do valor bruto
 - ISS - 10% do valor bruto
- Criar um campo para informar por meio de qual imposto deve ser calculado o imposto da nota fiscal
- Adicionar um formulário para excluir uma nota fiscal a partir do id

