Configurando Spring Boot Data JPA

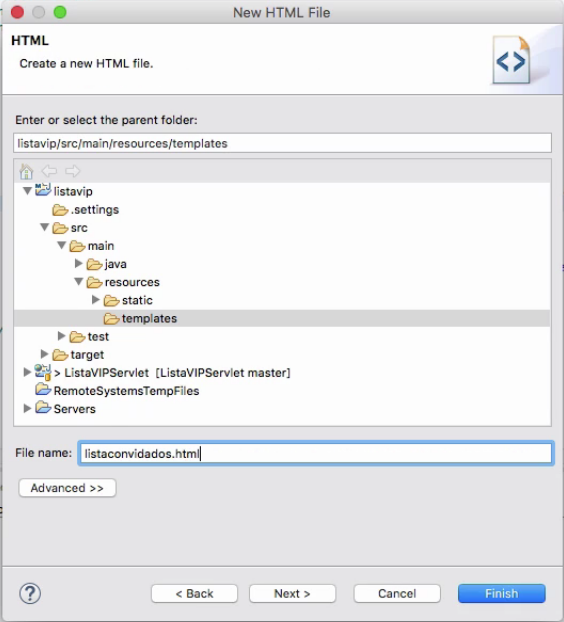
Se pensarmos em como a aplicação antiga funcionava, recordaremos que nela havia uma página onde existia uma listagem dos convidados e um formulário que nos permitia adicionar novos convidados à lista. O código da listagem era semelhante ao que está abaixo.

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>  
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">  
<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 id="listaDeConvidados">  
 <table class="table table-hover">  
 <thead>  
 <tr>  
 <th>Nome</th>  
 <th>Email</th>  
 <th>Nome</th>  
 </tr>  
 </thead>  
 <c:forEach var="convidado" items="${convidados}">  
 <tr>  
 <td>${convidado.nome}</td>  
 <td>${convidado.email}</td>  
 <td>${convidado.telefone}</td>  
 </tr>  
 </c:forEach>  
 </table>  
 </div>  
 <hr>  
 <form action="convidado">  
 <div class="form-group">  
 <label for="nome">Nome</label> <input type="text"  
 class="form-control" id="nome" name="nome" placeholder="Nome">  
 </div>  
 <div class="form-group">  
 <label for="email">Email</label> <input type="email"  
 class="form-control" id="email" name="email" placeholder="Email">  
 </div>  
 <div class="form-group">  
 <label for="telefone">Telefone</label> <input  
 type="text" class="form-control" id="telefone" name="telefone" placeholder="Telefone">  
 </div>  
 <button type="submit" class="btn btn-success">Convidar</button>  
 </form>  
 </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>

Havia uma tabela, onde através do forEach, percorria-se uma lista de convidados e imprimia-se, seu nome, email e telefone. Esta então será a segunda parte da nossa migração.

Começaremos criando uma página simples de teste, a fim de verificar que tudo está funcionando normalmente. Criaremos mais um arquivo na pasta de templateschamada listaconvidados.html e nesta apenas adicionaremos o seguinte código.

Criando listaconvidados.html



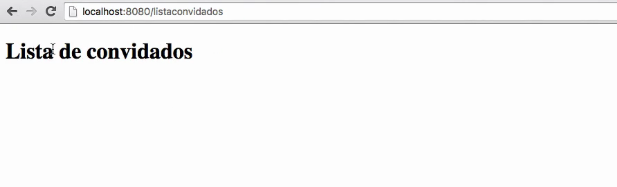
Código presente no listaconvidados.html

<!DOCTYPE html>  
<html>  
<head>  
 <meta charset="utf-8" />  
 <title>insert title here</title>  
</head>  
<body>  
 <h1>Lista de Convidados</h1>  
</body>  
</html>

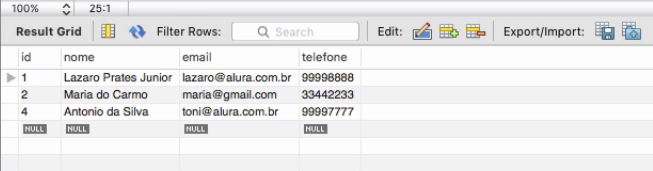
E por último, precisamos mapear a rota /listaconvidados para este *template*. Na classe ConvidadoController então, criaremos um novo método chamado listaConvidados que apenas retorna o nome do *template* da seguinte forma.

@RequestMapping("listaconvidados")  
public String listaConvidados(){  
 return "listaconvidados";  
}

Então, ao acessarmos localhost:8080/listaconvidados, teremos nossa página sendo exibida corretamente.



Mas se lembrarmos bem, a lista de convidados está salva no banco de dados e não temos acesso ao banco de dados ainda. Se fizermos um *select* no banco de dados, teremos os seguintes resultados.



A imagem anterior mostra como o MySQL Workbench apresenta os dados. Use a aplicação de acesso ao banco de dados que se sentir mais confortável. Caso não tenha conhecimentos de *SQL*, recomendamos também qee faça nossos cursos de [MySQL I](https://cursos.alura.com.br/course/introducao-a-banco-de-dados-e-sql) e [MySQL II](https://cursos.alura.com.br/course/banco-de-dados-e-sql-complexas)

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*. Antes precisávamos configurar tudo isso via *XML*, mas não precisamos mais disso. Adicionaremos as seguinte dependências ao pom.xml

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 <version>1.4.2.RELEASE</version>  
</dependency>  
  
<dependency>  
 <groupId>mysql</groupId>  
 <artifactId>mysql-connector-java</artifactId>  
 <version>5.1.40</version>  
</dependency>

A primeira para o *starter* da *JPA*, e a segunda do conector do *MySQL*. Agora precisamos configurar o acesso ao banco de dados, ou seja, definir o caminho para o banco, o usuário e a senha.

No arquivo src/main/resources/application.properties adicionar as propriedades abaixo:

**spring.datasource.url= jdbc:mysql://<ip>:3306/listavip**

**spring.datasource.username=root**

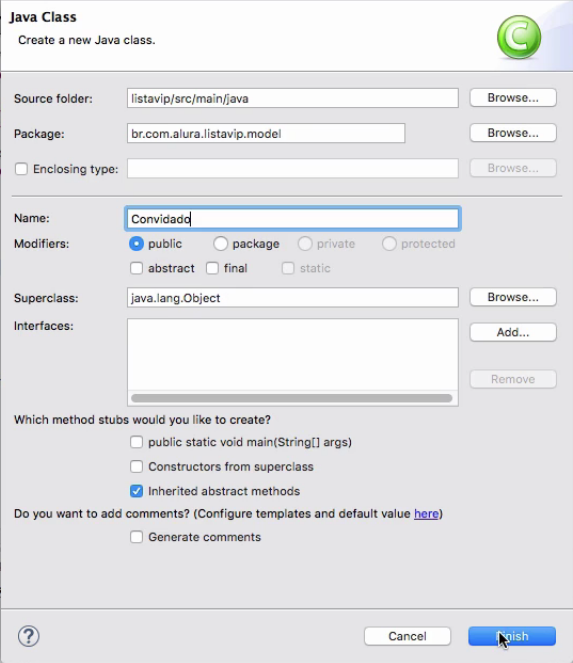
**spring.datasource.password=root**

**spring.jpa.show-sql=true**

**spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect**

Lembre-se de trocar as informações para estarem de acordo com suas configurações locais.

Já temos a página de apresentação dos convidados, temos o *controller* que exibe esta página, mas não temos uma entidade que representa os convidados na nossa aplicação. Para isto criaremos uma nova classe chamada Convidado que terá os atributos, id, nome, email e telefone.



Note que esta classe foi criada em um novo pacote chamado Model. Não é algo obrigatório, mas uma boa prática que ajuda a organizar o código.

@Entity(name = "convidado")  
public class Convidado {  
 @Id  
 @GeneratedValue  
 private Long id;  
  
 private String nome;  
 private String email;  
 private String telefone;  
  
 public Long getId() {  
 return id;  
 }  
  
 public void setId(Long 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 getTelefone() {  
 return telefone;  
 }  
  
 public void setTelefone(String telefone) {  
 this.telefone = telefone;  
 }  
}

A classe Convidado é bem simples e tem apenas algumas anotação da *JPA* e os atributos e método respectivos a cada coluna no banco de dados.

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.

Primeiro precisaremos criar uma *interface* que extenda a *CRUDRepository* do *Spring* e então, usar esta *interface* para acessar o banco de dados. A *interface* já herdará todos os método necessários, precisando apenas, indicar para ela qual é a entidade e qual é o identificador único de cada registro. Abaixo temos nossa *interface*ConvidadoRepository.

public interface ConvidadoRepository extends CrudRepository<Convidado, Long>{  
  
}

A interface ConvidadoRepository foi criada em um pacote diferente, por boas práticas, o pacote chama-se repository

A *interface* apenas tem os indicadores que quais classes são a entidade e o identificador único, que neste caso são: A classe Convidado e o atributo do tipo Long.

Para utilizá-la 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 Convidado. No ConvidadoController teremos:

@Autowired  
private ConvidadoRepository repository;

O passo a seguir é capturar todos os registros presentes no banco de dados usando o objeto repository e deixamos disponível para a página por meio de um outro objeto, chamado Model, que será recebido como parâmetro no método listaConvidados da classe ConvidadoController.

@RequestMapping("listaconvidados")  
public String listaConvidados(Model model){  
  
 Iterable<Convidado> convidados = repository.findAll();  
 model.addAttribute("convidados", convidados);  
  
 return "listaconvidados";  
}

O model será disponibilizado para a *view (página)* pelo *Spring*. O método usará o findAll do *repository* para retornar todos os registro em um Iterable por onde podemos iterar. Adicionamos os convidados como atributo de model e retornamos o nome do *template*.

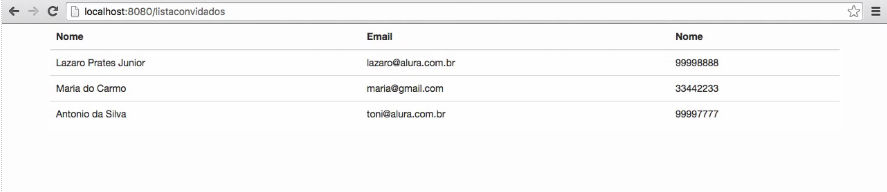
Como não estamos usando mais *JSP*, teremos que utilizar um outro *starter* do *Spring Boot* para que nossos templates possam capturar os objetos que estamos enviando para estes e fazer com que a página fique dinâmica. É aqui que começamos a usar o *Thymeleaf*. No pom.xml teremos mais uma dependência.

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-thymeleaf</artifactId>  
 <version>1.4.2.RELEASE</version>  
</dependency>

Para utilizar o *thymeleaf* em nossos *templates* precisamos apenas criar o *template*, que chamaremos de listaconvidados.html e no atributo html do *template* adicionaremos o atributo xmlns:th="http://www.thymeleaf.org". Desta forma, o *template engine* sabe que o *template* atual deve ser processado. Vejamos a listagem dos convidados como fica:

<!DOCTYPE html>  
<html xmlns:th="http://www.thymeleaf.org">  
<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 id="listaDeConvidados">  
 <table class="table table-hover">  
 <thead>  
 <tr>  
 <th>Nome</th>  
 <th>Email</th>  
 <th>telefone</th>  
 </tr>  
 </thead>  
 <tr th:each="convidado : ${convidados}">  
 <td> <span th:text="${convidado.nome}"></span> </td>  
 <td> <span th:text="${convidado.email}"></span> </td>  
 <td> <span th:text="${convidado.telefone}"></span> </td>  
 </tr>  
 </table>  
</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>

Perceba que não precisamos ficar usando *tags* específicas, apenas atributos nos elementos do *HTML*. O atributo th:each percorre uma lista de itens iteráveis, repetindo o próprio elemento e o atributo th:text imprime texto na página. Bem simples. E como resultado teremos:



**Observação:** Lembre-se de adicionar no template, os links para os scripts e estilos do *Bootstrap*. Também altere o link presente no index.html para ter a *URL* apontando para listaconvidados ao invés de listavip.