



# DESAFIO VACINA



## DESAFIO

Criar uma API REST que consegue cadastrar os usuários e a aplicação de vacinas. Para este desafio usaremos o Java como linguagem principal junto com o framework Spring Boot com Hibernate.

**Nome:** Cleiton Ortega dos Santos

**E-mail:** c.ortega200935@gmail.com

## Introdução a API REST

Primeiro vou falar sobre o que é uma API REST. A primeira coisa a se falar sobre API REST é que API (Application Programming Interface ou em português Interface de Programação de Aplicação) é um padrão estabelecido para um software para poder utilizar suas funcionalidades. Em outras palavras API é um programa com funcionalidades embutidas que o usuário utiliza. Já o conceito REST (Representational State Transfer ou em português Transferência Representacional de Estado) é um modelo de arquitetura para que sistemas WEB se comuniquem de maneira correta.

---

### Quais tecnologias utilizaremos?

Para este projeto utilizaremos as tecnologias de Back-End: Java, como linguagem, Spring Boot como framework e MySQL como banco de dados.

Para o Front-End utilizaremos: Angular, HTML, CSS e JavaScript.

### Quais programas utilizaremos?

Para podermos utilizar as tecnologias precisamos de alguns programas. Para este projeto utilizaremos o Eclipse como IDE, para criar o Back-End; XAMPP para simular um Servidor em nossa máquina e o Visual Studio Code como editor de texto para as tecnologias do Front-End

---



Visual Studio Code



## O que é o Spring Boot?

O Spring Boot é um framework para o Java que cuida da parte de infra-estrutura do projeto, facilitando para que não tenhamos que cuidar da estrutura do projeto. Assim podemos focar direto na criação da API.

### Quais tecnologias do Spring Boot foram utilizadas

Para este projeto utilizaremos as tecnologias:

**MySQL Driver** – Para a criação do banco de dados;

**Spring WEB** – Para criar um WEB Service;

**Spring Boot DevTools** – Para que a aplicação atualize automaticamente após alterar e salvar algum dado. Então não precisemos parar a aplicação e rodar novamente;

**Spring Data JPA** – Para usar recursos do JPA e Hibernate;

**Validation** – Para fazer a validação de alguns dados.

### Criando um projeto Spring

Para criarmos o Projeto Spring utilizaremos o site do Spring Initializr (<https://start.spring.io>), pois ele consegue fazer a criação do nosso projeto Spring para depois precisarmos apenas importá-lo para o Java e focar no projeto! No site do Spring Initializr, à direita da foto, em Vermelho, temos as dependências que ele adicionará ao projeto. À esquerda, em azul, podemos escolher tipo de projeto, a linguagem que será utilizada, a versão do Spring, os nomes de pastas e pacotes e a versão da linguagem escolhida mais acima.

Na parte de baixo, em laranja, temos a opção 'Generate'. Com isso um pacote zip com a estrutura do projeto será criado.

**Project**

☒ Maven Project ☐ Gradle Project

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 2.5.0 (SNAPSHOT) ☐ 2.5.0 (M3) ☐ 2.4.5 (SNAPSHOT) ☒ 2.4.4

☐ 2.3.10 (SNAPSHOT) ☐ 2.3.9

#### Project Metadata

**Group** teste.Orange

**Artifact** teste

**Name** testeOrangeTalents

**Description** API REST para Controle de Aplicação de Vacinas

**Package name** teste.Orange.teste

**Packaging** ☒ Jar ☐ War

**Java** ☐ 16 ☒ 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.

#### Spring Boot DevTools **DEVELOPER TOOLS**

Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

#### Spring Data JPA **SQL**

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

#### MySQL Driver **SQL**

MySQL JDBC and R2DBC driver.

#### Validation **I/O**

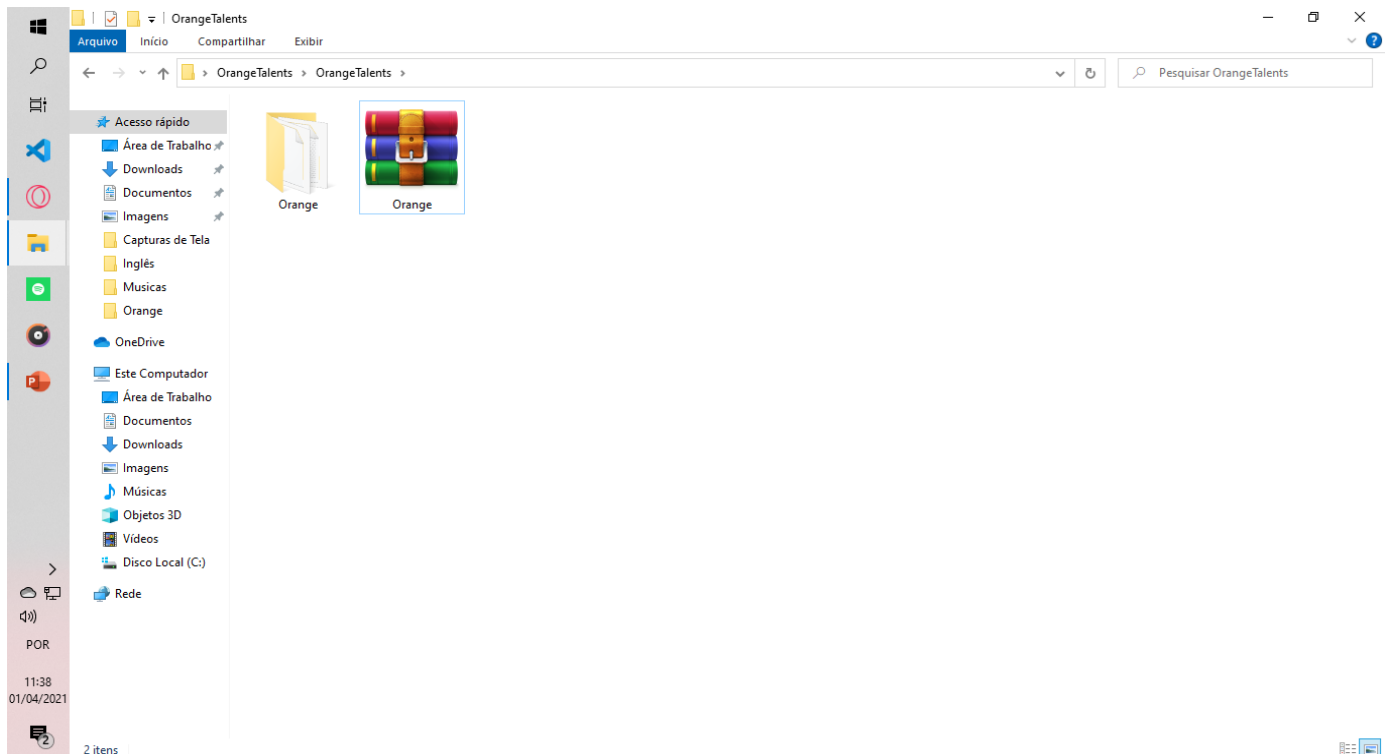
Bean Validation with Hibernate validator.

**GENERATE** CTRL + G

**EXPLORE** CTRL + SPACE

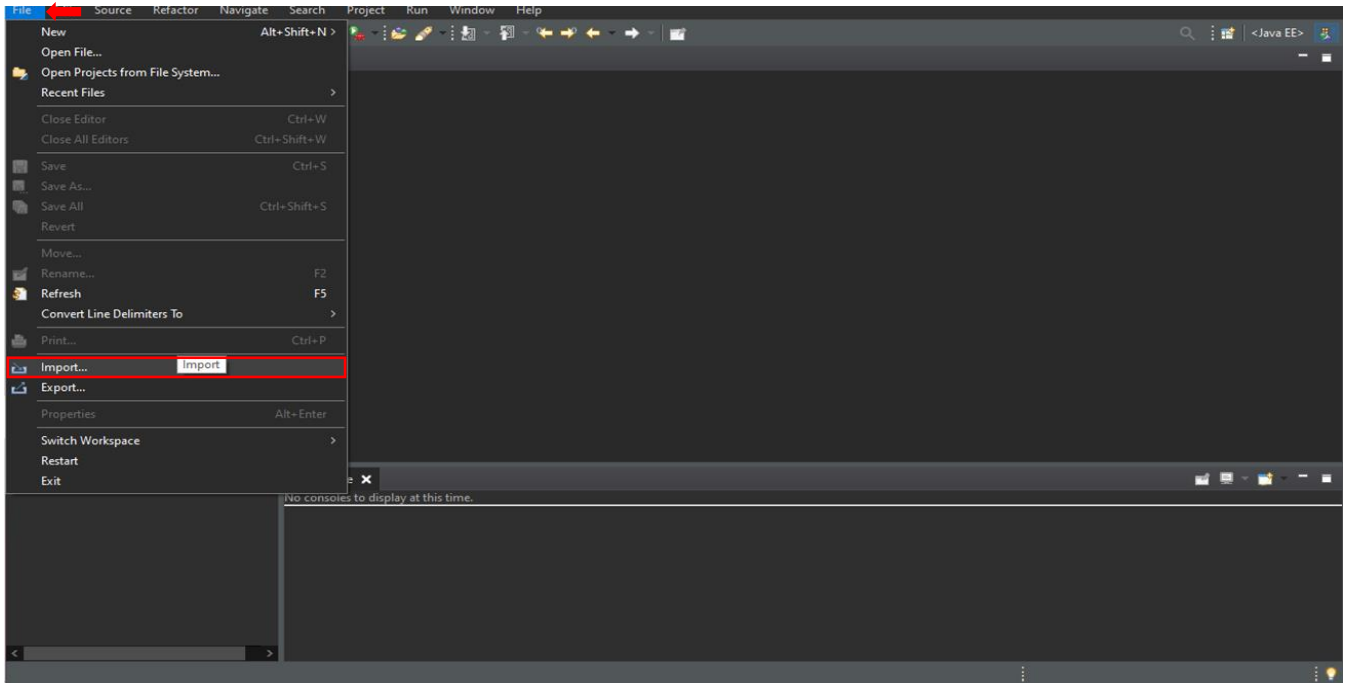
**SHARE...**

Zip gerado pelo Spring Initializr:

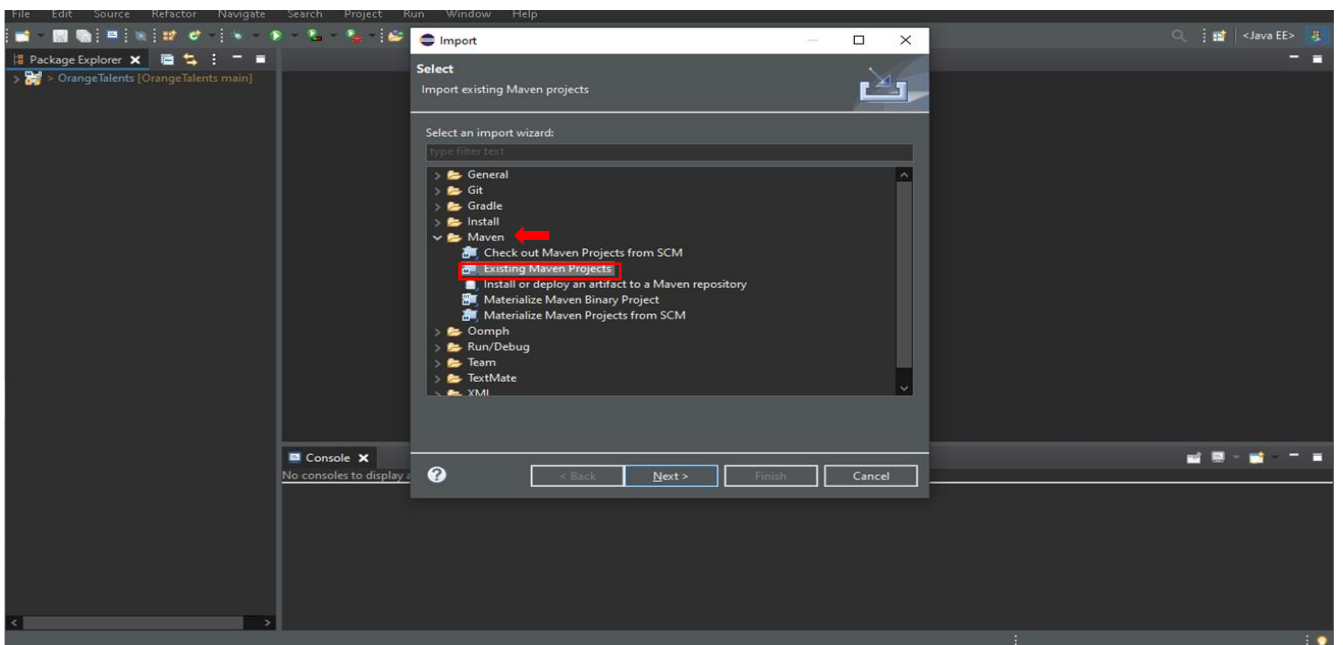


## Importando o projeto

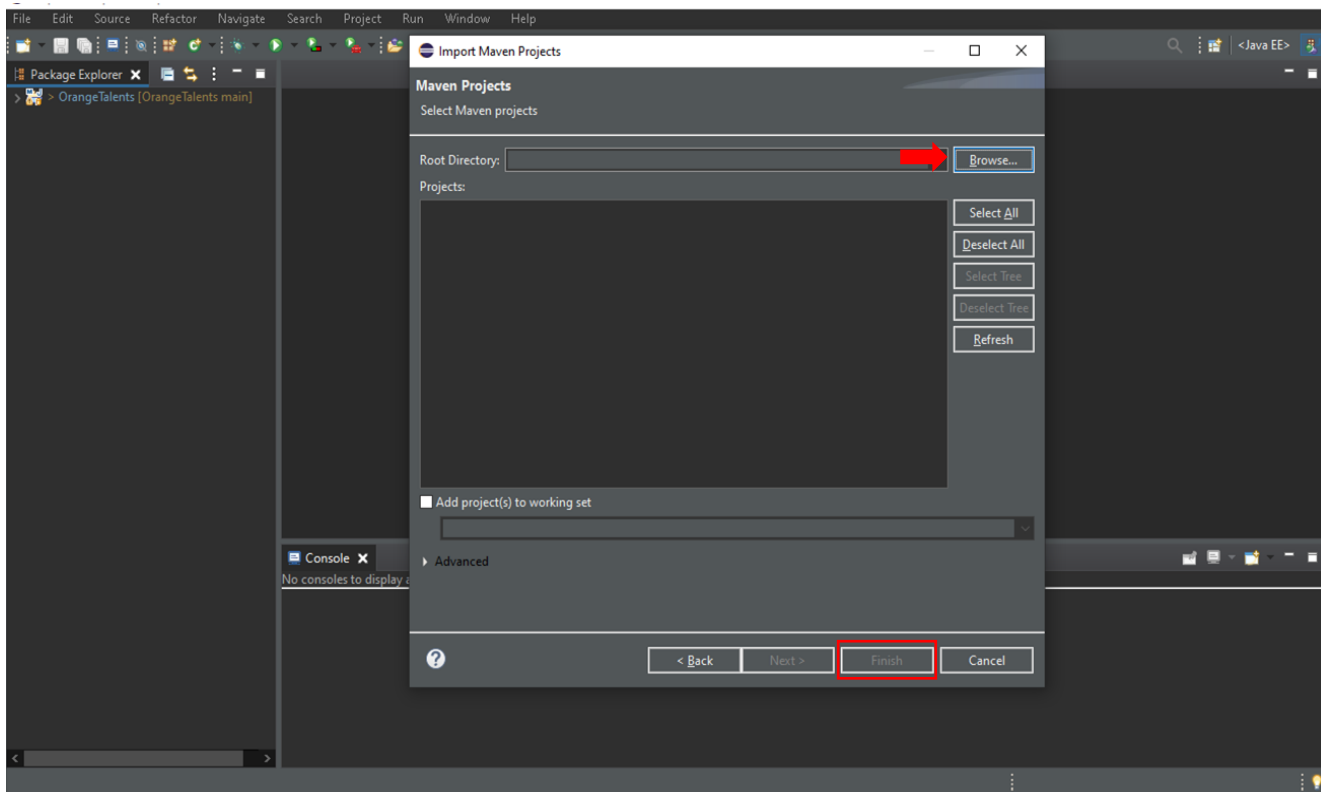
Após a criação a geração e a extração do projeto devemos abrir o Eclipse e importar o projeto seguindo estes passos: Clique em **FILE > IMPORT..**



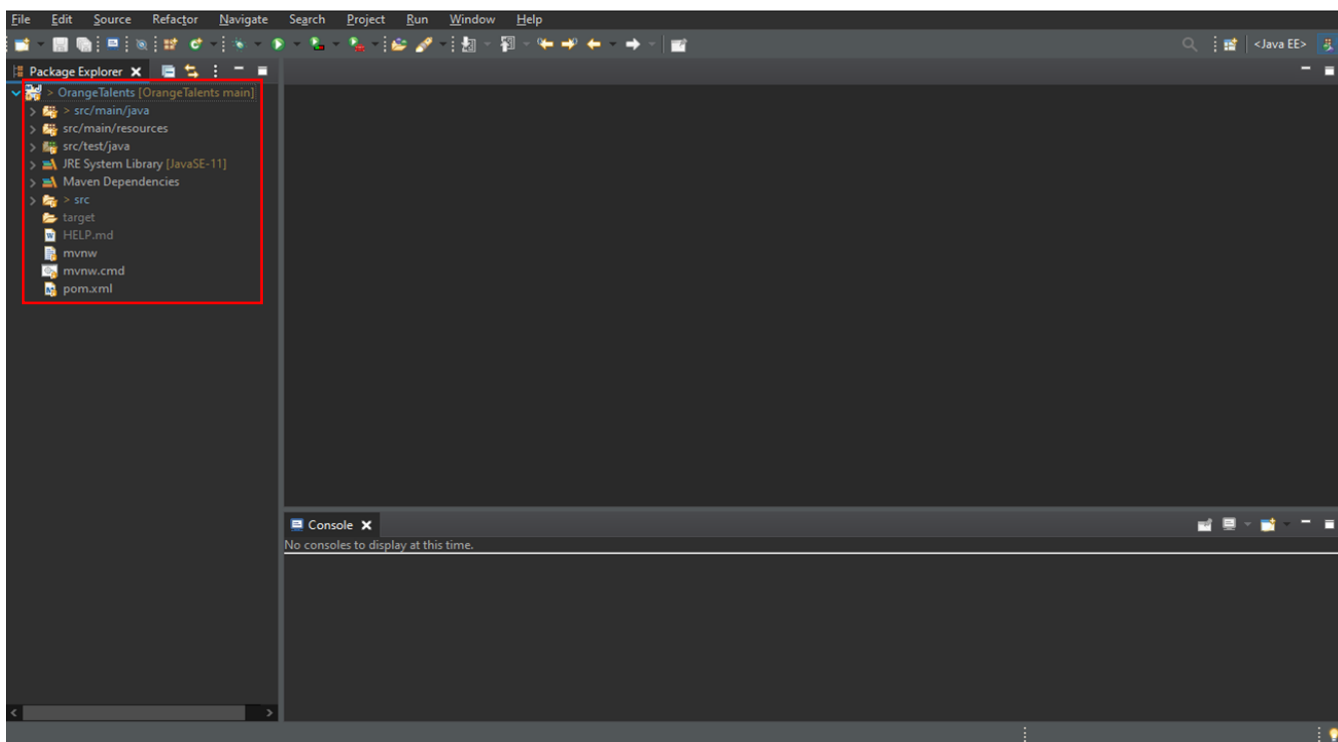
Irá abrir uma janela Clique em **MAVEN > EXISTING MAVEN PROJECTS**



Nesta janela que abriu clique em **BROWSE** escolha o arquivo extraído e clique abaixo em **FINISH**:



Assim que terminar o processo o Eclipse estará com o projeto igual este modelo abaixo:



## Configurando o projeto

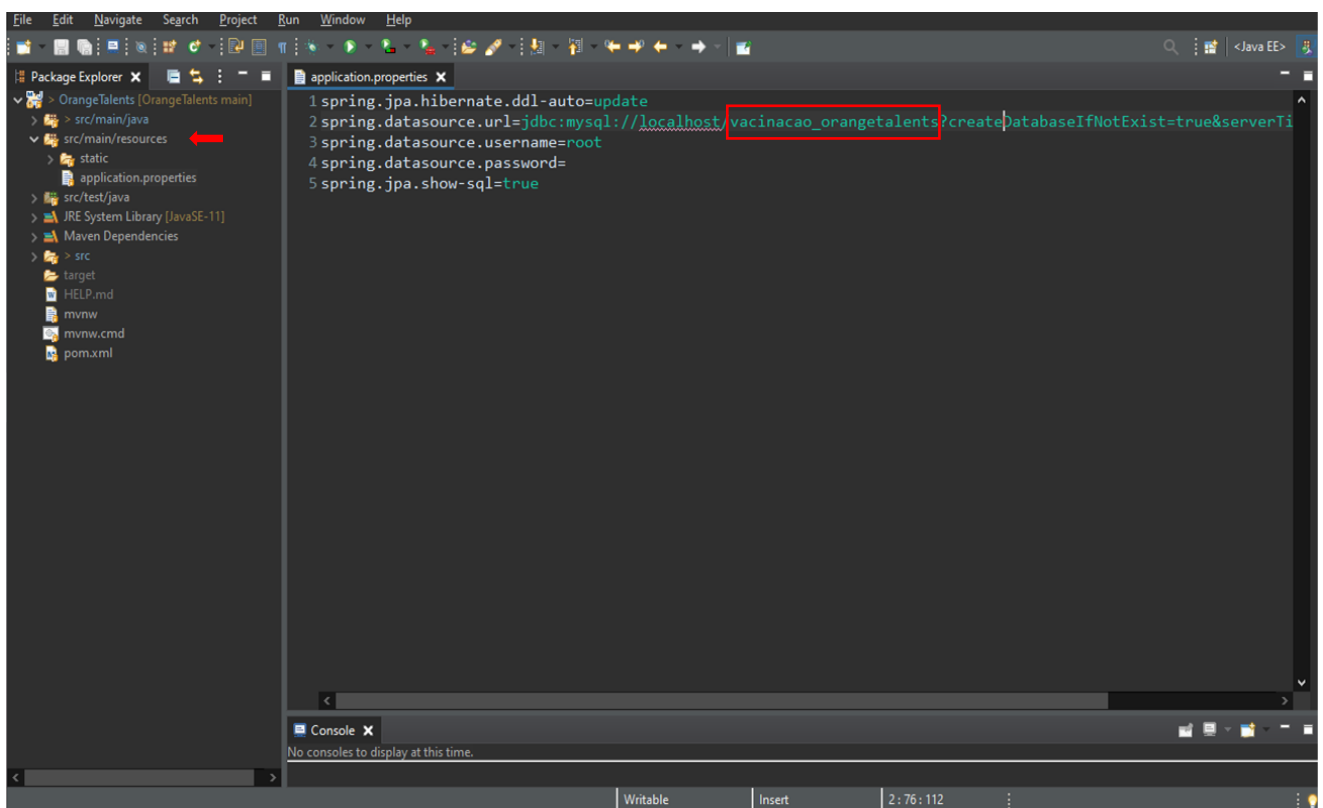
Agora vamos configurar o projeto através do `application.properties` que está localizado na pasta **src/main/resources** mostrado pela seta vermelha.

Na primeira linha estamos chamando o Hibernate que ativará a atualização automática ao salvarmos algum arquivo do código;

Na segunda linha estamos criando o banco de dados e dando o nome de `vacinacao_orangetalents`;

Na terceira e quarta linha estamos utilizando o usuário `root` e não estamos passando senha alguma para ele;

Na quinta linha é utilizada para mostrar no console o que o programa está fazendo em relação ao MySQL. Mostrará, por exemplo, se algo foi criado ou alterado:



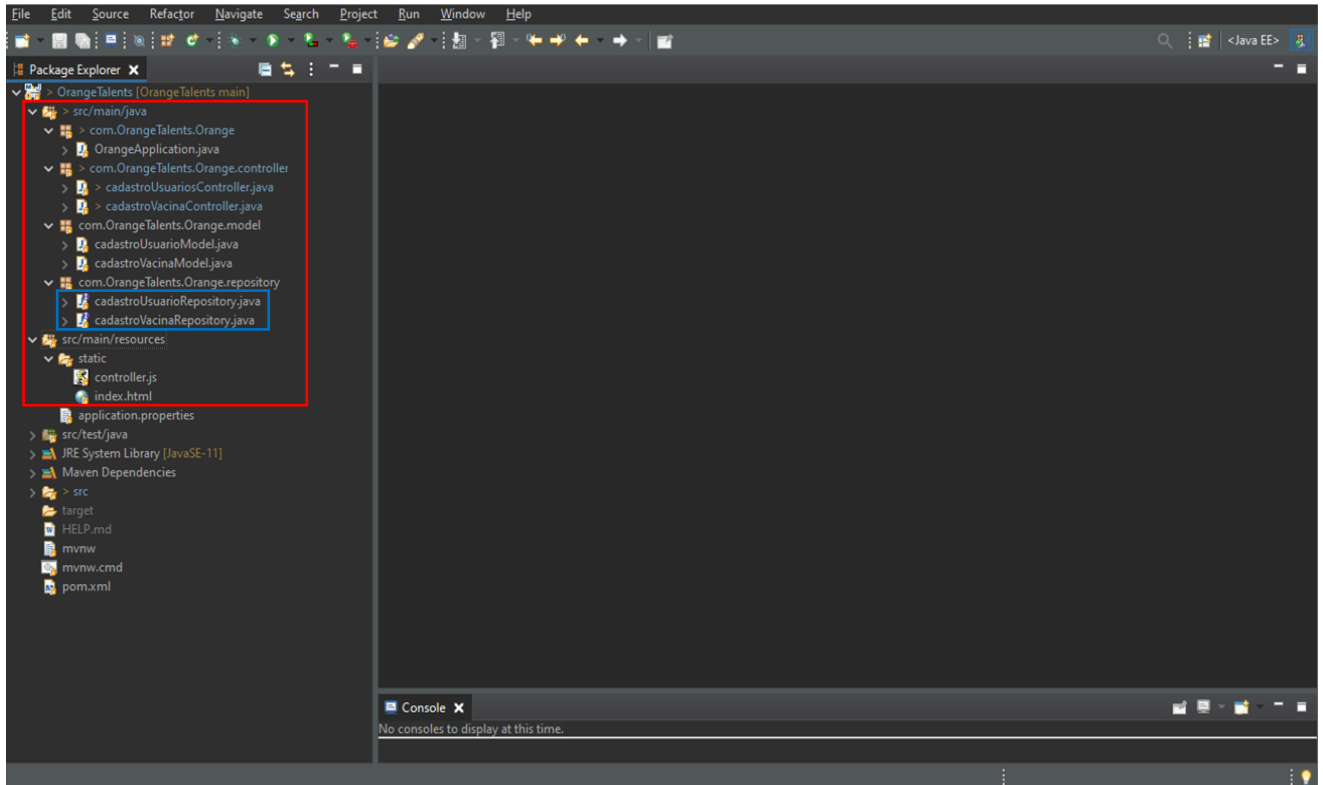
The screenshot shows an IDE window with the `application.properties` file open. The Package Explorer on the left shows the project structure with a red arrow pointing to the `src/main/resources` folder. The code in the editor is as follows:

```
1 spring.jpa.hibernate.ddl-auto=update
2 spring.datasource.url=jdbc:mysql://localhost/vacinacao_orangetalents?createDatabaseIfNotExist=true&serverTi
3 spring.datasource.username=root
4 spring.datasource.password=
5 spring.jpa.show-sql=true
```

The console at the bottom shows the message: "No consoles to display at this time."

## Criando as classes e interfaces que serão utilizadas

Neste projeto as classes e interfaces foram criadas com o modelo **Spring MVC (Model, View, Controller)**:



O modelo MVC é uma arquitetura baseada em HTTP criado para ter um padrão na criação de estruturas e para dividir a aplicação em três camadas, sendo uma camada voltada para a interação do usuário (**View**), uma camada para manipular os dados (**Model**), e outra camada para o controle (**Controller**).

Para a camada de **View** criaremos o index.html e o controller.js

Para a camada de **Model** criaremos o cadastroUsuarioModel, cadastroVacinaModel, cadastroUsuarioRepository e cadastroVacinaRepository. Vale ressaltar que os repositories (Destacados da cor azul) tem como função fazer os dados persistirem no banco de dados.

Para a camada de **Controller** criaremos o cadastroUsuarioController e o cadastroVacinaController.



## Mostrando todas as classes criadas para o projeto

Agora vamos mostrar todas as classes criadas neste projeto e falar um pouco sobre o conteúdo da classe:

### cadastroUsuarioModel(1/2)

```
cadastroUsuarioModel.java x
1 package com.OrangeTalents.Orange.model;
2
3 import javax.persistence.*;
4
5 @Entity
6 @Table(name="tb_cadastroUsuario")
7 public class cadastroUsuarioModel {
8
9     @Id
10    @GeneratedValue (strategy=GenerationType.IDENTITY)
11    private Long idUsuario;
12
13    @Column
14    @NotNull
15    private String nome;
16
17    @Column(unique=true)
18    @NotNull
19    @Email
20    private String email;
21
22    @Column(unique=true)
23    @NotNull
24    @Size(min = 14, max = 14)
25    @CPF
26    private String cpf;
27
28    @Column
29    @NotNull
30    @JsonFormat(pattern="yyyy-mm-dd")
31    private String nascimento;
32 }
```

Notações e seus propósitos:

**@Entity** – tem o propósito de dizer que a classe é uma entidade;

**@Table** – tem o propósito de dizer que a classe é uma tabela e logo em seguida demos um nome a ela;

**@Id** e **@GeneratedValue** – estas notações trabalham em conjunto onde o **@Id** tem o propósito de dizer que vira um código identificador que é único, e o **@GeneratedValue** tem o propósito de dizer para o banco de dados tomar conta da geração deste código único;

**@Column** e **@NotNull** – serve para dizer que vira uma coluna e que ela não pode ser nula;

**@Email** – é uma notação do Hibernate usado para ter alguns padrões para a inserção de dados. Neste caso precisa ter '@' e '.' no dado para ser aceito;

## cadastroUsuarioModel(2/2)

**@Column(unique=true)** – notação com a mesma função de um **@Column** porém com o padrão de um dado ser único sem poder repetir o dado novamente;

**@Size** – serve para ter um tamanho especificado, podendo colocar um mínimo aceitável e/ou um máximo aceitável;

**@CPF** – assim como o **@Email** esta notação é para informar um CPF que seja válido seguindo o padrão de CPF estabelecido pelo governo;

**@JsonFormat** – serve para deixar a data em um formato específico no caso desta aplicação ano-mês-dia(yyyy-mm-dd).

## cadastroVacinaModel(1/1)

```
cadastroVacinaModel.java X
1 package com.OrangeTalents.Orange.model;
2
3 import javax.persistence.*;
4
5
6
7
8 @Entity
9 @Table(name="tb_cadastroVacina")
10 public class cadastroVacinaModel {
11
12     @Id
13     @GeneratedValue (strategy=GenerationType.IDENTITY)
14     private Long idVacina;
15
16     @Column
17     @NotNull
18     private String nomeDaVacina;
19
20     @Column(unique=true)
21     @NotNull
22     private String emailUsuario;
23
24     @Column
25     @NotNull
26     @JsonFormat(pattern="yyyy-mm-dd")
27     private String dataAplicacao;
28 }
```

Nesta classe usamos o mesmo padrão do **cadastroUsuarioModel** única diferença foi que mudamos o nome da tabela (linha 9) para “tb\_cadastroVacina”

## cadastroUsuarioRepository(1/1)

```
cadastroUsuarioRepository.java X
1 package com.OrangeTalents.Orange.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7
8
9 public interface cadastroUsuarioRepository extends JpaRepository<cadastroUsuarioModel, Long>{
10
11     public Optional<cadastroUsuarioModel> findByEmail(String email);
12
13 }
14
```

Aqui nós temos o repository a função dele é fazer os dados persistirem no banco de dados. Isso quer dizer que ele é responsável por colocar os dados no banco de dados. Para configurar o repository para um banco de dados utilizamos a função `JpaRepository` e dentro dela devemos informar em qual entidade ele irá persistir e qual o tipo do id da entidade. Também é onde podemos colocar algum método que poderemos utilizar no controller. Neste repository criamos um método na linha 11 para procurar um e-mail.

## cadastroVacinaRepository(1/1)

```
cadastroVacinaRepository.java X
1 package com.OrangeTalents.Orange.repository;
2
3 import org.springframework.data.jpa.repository.JpaRepository;
4
5
6
7 public interface cadastroVacinaRepository extends JpaRepository<cadastroVacinaModel, Long>{
8
9 }

```

Já neste repository nos estendemos o `cadastroVacinaModel` e não criamos método algum nele. Utilizaremos apenas algum método já existente no `JpaRepository`.

## cadastroUsuariosController(1/1)

```
1 package com.OrangeTalents.Orange.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6 @RestController
7 @CrossOrigin(origins = "*", allowedHeaders = "*")
8 public class cadastroUsuariosController implements WebMvcConfigurer{
9
10     public void addViewControllers(ViewControllerRegistry index) {
11         index.addViewController("/").setViewName("forward:/index.html");
12     }
13
14     @Autowired
15     private cadastroUsuarioRepository repository;
16
17     @PostMapping("/cadastroUsuario")
18     public ResponseEntity<> cadastrarUsuario(@RequestBody cadastroUsuarioModel objetoCadastroUsuario) {
19         try {
20             repository.save(objetoCadastroUsuario);
21             return ResponseEntity.status(HttpStatus.CREATED).build();
22         } catch (Exception e) {
23             return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
24         }
25     }
26 }
```

Aqui nós temos o controller do **cadastroUsuariosModel** é aqui onde criaremos os métodos para fazer o cadastro de um novo usuário. Nesta classe temos algumas notações. A primeira é **@RestController**, a função dela é falar que a classe é um controller. Logo em seguida temos o **@CrossOrigin**, que serve para liberar o compartilhamento de recursos entre diferentes origens. Alguns computadores precisam desta liberação. Na linha 15 temos o primeiro método do controller que serve basicamente para redirecionar a página caso a url seja apenas uma "/". Caso seja apenas a "/", ele redireciona a página para o arquivo "index.html". Na próxima notação, **@Autowired**, nós estamos injetando o repository no programa. Com **@PostMapping** nós estamos criando um método Post (método para criação de dados) e criando um link para chamá-lo, no caso "/cadastroUsuario". Nesta função nós pegamos um dado que é um objeto com as mesmas propriedades do cadastroUsuarioModel e então tentamos salvar o objeto passado, chamando o método save (método padrão criado no JpaRepository). Caso dê certo retorna um CREATED(201), se errado retorna um BAD\_REQUEST(400).

## cadastroVacinaController(1/1)

```
cadastroVacinaController.java x
1 package com.OrangeTalents.Orange.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6 @RestController
7 @CrossOrigin(origins = "*", allowedHeaders = "*")
8 public class cadastroVacinaController {
9
10     @Autowired
11     private cadastroVacinaRepository repository;
12
13
14     @Autowired
15     private cadastroUsuarioRepository repositoryUsuario;
16
17
18     @PostMapping("/cadastroVacina")
19     public ResponseEntity<> cadastrarVacina(@RequestBody cadastroVacinaModel objetoCadastroVacina) {
20         try {
21             if(repositoryUsuario.findByEmail(objetoCadastroVacina.getEmailUsuario()).isPresent()) {
22                 repository.save(objetoCadastroVacina);
23                 return ResponseEntity.status(HttpStatus.CREATED).build();
24             }else{
25                 return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
26             }
27         }catch (Exception e){
28             return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
29         }
30     }
31 }
32 }
```

Aqui nós temos o controller de **cadastroVacinaModel**. Basicamente igual ao **cadastroUsuariosController**, a diferença é que aqui nós injetamos ambos repositories: o **cadastroVacinaRepository** e o **cadastroUsuarioRepository**. Injetamos o repository de usuário para utilizar aquele método que tínhamos criado para buscar um e-mail, pois para o cadastro de alguma aplicação de vacina temos que ter algum usuário que a tomou, senão os dados ficariam inseguros e não confiáveis! O conceito de toda a classe foi o mesmo, a diferença está na linha 24, que mudamos a rota do Post (para chamar o Post de vacina utilizamos o caminho “/cadastroVacina”); e na linha 27 onde chamamos o repository de usuário e chamamos o método `findByEmail`, e passamos o e-mail do objeto que estamos tentando inserir no banco de dados da aplicação da vacina. Se caso o e-mail usado para fazer a aplicação da vacina esteja presente no banco de dados de usuário, ele retorna um **CREATED(201)**. Caso não esteja presente ou caso aconteça algum outro erro ele retorna um **BAD\_REQUEST(400)** mesmo conceito do **cadastroUsuariosController**.



# Index(1/1)

```
index.html M X
C:\Users\kakeu\Desktop\OrangeTalents\OrangeTalents\Orange> src\main\resources\static> index.html > html > head > title

4 <head>
5   <meta charset="UTF-8">
6   <meta name="viewport" content="width=device-width, initial-scale=1.0">
7   <title>Cadastro Aplicação de Vacinas</title>
8   <link rel="stylesheet" type="text/css" href="//netdna.bootstrapcdn.com/bootstrap/3.0.3/css/bootstrap.min.css">
9   <link rel="stylesheet" type="text/css" href="//assets.locaweb.com.br/locastyle/2.0.6/stylesheets/locastyle.css">
10  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.9/angular.min.js"></script>
11  <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.0-beta1/dist/css/bootstrap.min.css" rel="stylesheet" integrity="sha384-g1jF6kkqN00v0+PMOP7azD0L8xtbF1caT9wJH08bD0ddVhyTfAAAsrekWp1" crossorigin="anonymous">
12  <script src="https://cdn.jsdelivr.net/npm/sweetalert2@10"></script>
13  <script src="controller.js"></script>
14 </head>
15
16 <body>
17   <div class="mx-2">
18     
19     
20     <hr>
21     <div ng-app="myApp" ng-controller="myCtrl" class="mb-2">
22       <div class="container">
23         <div class="row">
24           <div class="col-md-6">
25             <p class="fs-1">Cadastro Usuario</p>
26             <form>
27               Nome: <p><input type="text" class="form-control" size="40" ng-model="nome"
28                 placeholder="Digite seu nome completo" required></p>
29               CPF: <p><input type="text" class="form-control cpf-mask" size="14" ng-model="cpf"
30                 placeholder="Ex.: 000.000.000-00" required></p>
31               Email: <p><input type="text" class="form-control" size="50" ng-model="email"
32                 placeholder="example@example.com" required></p>
33               Data de Nascimento: <p><input type="date" class="form-control" ng-model="nascimento"
34                 required></p>
35               <button class="btn btn-primary w-100" ng-click="salvarUsuario()">Cadastrar Usuario</button>
36             </form>
37           </div>
38           <div class="col-md-6">
39             <p class="fs-1">Cadastro Vacinado</p>
40             <form>
41               Nome da Vacina: <p><input type="text" class="form-control" size="40" ng-model="nomeDaVacina"
42                 placeholder="Digite o nome da vacina" required></p>
43               Email: <p><input type="text" class="form-control" size="50" ng-model="emailUsuario"
44                 placeholder="example@example.com" required></p>
45               Data da Aplicação da Vacina: <p><input type="date" class="form-control"
46                 ng-model="dataAplicacao" required></p>
47               <button class="btn btn-primary w-100" ng-click="salvarVacinado()">Cadastrar Aplicação
48                 da Vacina</button>
49             </form>
50           </div>
51         </div>
52       </div>
53     </div>
54   </div>
```

Esta aqui é a index, onde teremos os formulários e tudo o que aparecerá no nosso site. Importei alguns frameworks para ajudar no processo da criação do nosso front, para ficar algo mais bonito de se ver! Entre elas tem uma para formatar o CPF: ele formata o CPF para entrar no banco de dados já formatado com a pontuação (por exemplo, 000.000.000-00. Outro é um pop-up quando cadastrarmos alguma informação nova ou caso dê algum erro na inserção dos dados!

## Controller(1/1)

```
JS controller.js X
C:\Users> Users > kekeu > Desktop > OrangeTalents > OrangeTalents > Orange > src > main > resources > static > JS controller.js > ...
1  var app = angular.module('myApp', []);
2
3  app.controller('myCtrl', function ($scope, $http) {
4
5      $scope.cadastro = new Object();
6
7      $scope.salvarUsuario = function () {
8          $http.post("http://localhost:8080/cadastroUsuario", {
9              'id': $scope.id,
10             'nome': $scope.nome,
11             'cpf': $scope.cpf,
12             'email': $scope.email,
13             'nascimento': $scope.nascimento
14         })
15         .then(resposta => {
16             Swal.fire('Ok !!', 'Usuario cadastrado com sucesso!!', 'success')
17         })
18         .catch(erro => {
19             Swal.fire('Erro !!', 'Erro 400!', 'error')
20         })
21     };
22
23     $scope.salvarVacinado = function () {
24         $http.post("http://localhost:8080/cadastroVacina", {
25             'idVacina': $scope.idVacina,
26             'nomeDaVacina': $scope.nomeDaVacina,
27             'emailUsuario': $scope.emailUsuario,
28             'dataAplicacao': $scope.dataAplicacao
29         })
30         .then(resposta => {
31             Swal.fire('Ok !!', 'Vacinação registrada com sucesso!!', 'success')
32         })
33         .catch(erro => {
34             Swal.fire('Erro !!', 'Erro 400!', 'error')
35         })
36     };
37
38 });
39 });
```

Este aqui é o Controller, onde teremos a conexão entre o controller.js, **cadastroUsuarioController** e o **cadastroVacinaController**, para cadastrarmos algum usuário ou uma vacina nova! Aqui basicamente estamos criando um objeto com o formulário da index e mandando para os posts que criamos através do link “/cadastroVacina” e “/cadastroUsuario”, e retornando uma mensagem de sucesso caso retorne um CREATED(201) ou de erro caso retorne um BAD\_REQUEST(400).

**Agora que vimos o programa completo vamos fazer o teste de mesa!**

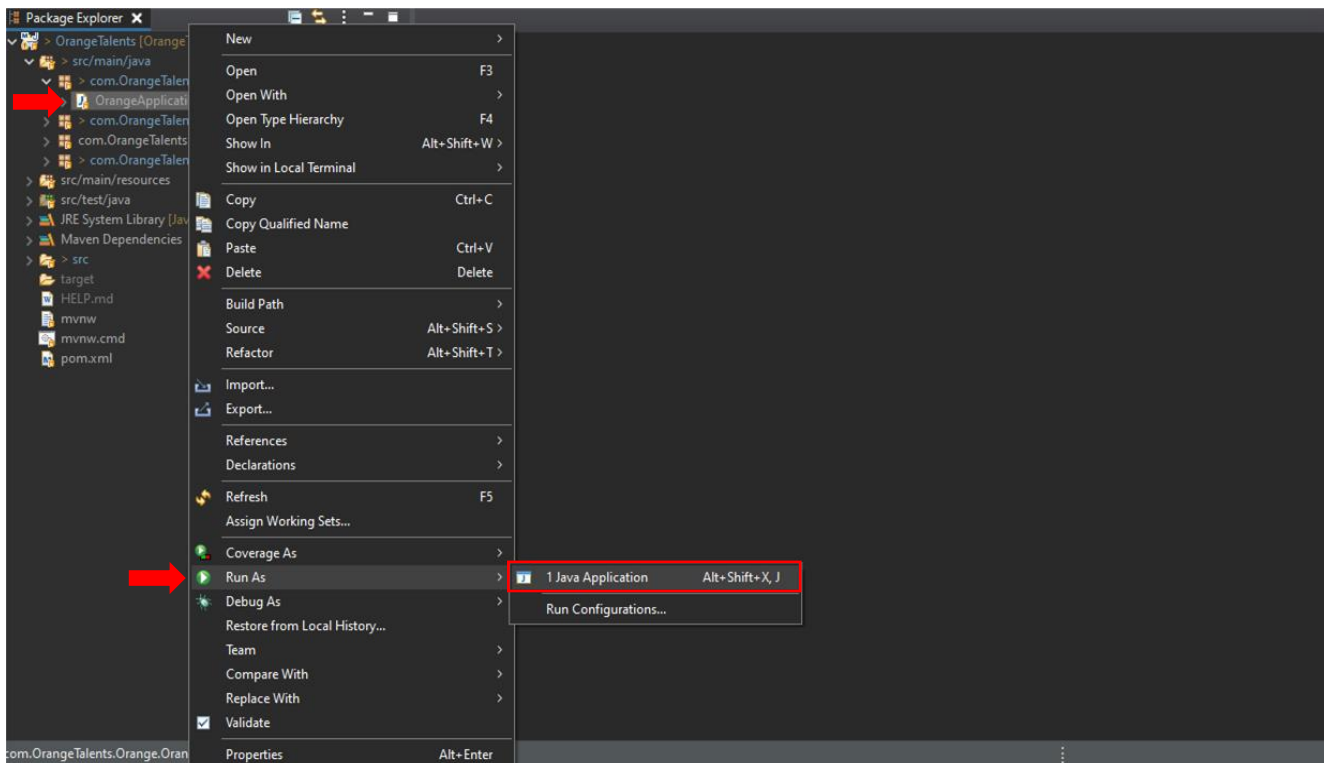
## Utilizando o XAMPP

Para fazer o teste de mesa teremos que usar o XAMPP para simular um servidor! Abra o XAMPP e ao entrar clique no **Start** do Apache e do MySQL, igual na foto:



## Utilizando o Eclipse

Após ligar o XAMPP, rode o projeto com o botão “Run As” no Eclipse, como na imagem abaixo:





## Console do Eclipse

Após rodar o programa o console do Eclipse deverá ter esta última linha (contornado em azul). Observação: Lembram no application.properties que nós configuramos para mostrar o que acontece no banco de dados? Então, esse é o resultado (contornado em vermelho):

```
Console
OrangeApplication (1) [Java Application] C:\Program Files\Java\jdk-14.0.2\bin\javaw.exe (2 de abr. de 2021 17:21:12)
2021-04-02 17:21:16.521 INFO 5828 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (h
2021-04-02 17:21:16.537 INFO 5828 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-04-02 17:21:16.537 INFO 5828 --- [ restartedMain] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/
2021-04-02 17:21:16.662 INFO 5828 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicat
2021-04-02 17:21:16.662 INFO 5828 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initializati
2021-04-02 17:21:16.848 INFO 5828 --- [ restartedMain] o.hibernate.jpa.internal.util.LogHelper : HHH000204: Processing PersistenceUnitInf
2021-04-02 17:21:16.908 INFO 5828 --- [ restartedMain] org.hibernate.Version : HHH000412: Hibernate ORM core version 5.
2021-04-02 17:21:17.126 INFO 5828 --- [ restartedMain] o.hibernate.annotations.common.Version : HCANN000001: Hibernate Commons Annotatio
2021-04-02 17:21:17.346 INFO 5828 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2021-04-02 17:21:17.611 INFO 5828 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2021-04-02 17:21:17.645 INFO 5828 --- [ restartedMain] org.hibernate.dialect.Dialect : HHH000400: Using dialect: org.hibernate.
Hibernate: create table tb_cadastro_usuario (id_usuario bigint not null auto_increment, cpf varchar(14) not null, email varchar(255) not nul
Hibernate: create table tb_cadastro_vacina (id_vacina bigint not null auto_increment, data_aplicacao varchar(255) not null, email_usuario va
Hibernate: alter table tb_cadastro_usuario drop index UK_506214dd4vm0wk9sr4sk8kxh4
Hibernate: alter table tb_cadastro_usuario add constraint UK_506214dd4vm0wk9sr4sk8kxh4 unique (cpf)
Hibernate: alter table tb_cadastro_usuario drop index UK_bmlke7m09iy56rmsmd71cf1jl
Hibernate: alter table tb_cadastro_usuario add constraint UK_bmlke7m09iy56rmsmd71cf1jl unique (email)
Hibernate: alter table tb_cadastro_vacina drop index UK_hpg33fjnr763xl60ha3l71ksq
Hibernate: alter table tb_cadastro_vacina add constraint UK_hpg33fjnr763xl60ha3l71ksq unique (email usuario)
2021-04-02 17:21:20.196 INFO 5828 --- [ restartedMain] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000490: Using JtaPlatform implementat
2021-04-02 17:21:20.208 INFO 5828 --- [ restartedMain] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for
2021-04-02 17:21:20.227 INFO 5828 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 357
2021-04-02 17:21:20.945 WARN 5828 --- [ restartedMain] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by de
2021-04-02 17:21:21.428 INFO 5828 --- [ restartedMain] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicatio
2021-04-02 17:21:21.588 INFO 5828 --- [ restartedMain] o.s.b.a.w.s.WelcomePageHandlerMapping : Adding welcome page: class path resource
2021-04-02 17:21:21.770 INFO 5828 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) w
2021-04-02 17:21:21.781 INFO 5828 --- [ restartedMain] c.o.Orange.OrangeApplication : Started OrangeApplication in 8.116 secon
```

Após iniciar o XAMPP e o Eclipse, entre no link <http://localhost:8080/> e você estará na index que tem esta cara:

Cadastro Aplicação de Vac: X +

localhost:8080

Orange TALENTS

### Cadastro Usuario

Nome:

CPF:

Email:

Data de Nascimento:

### Cadastro Vacinado

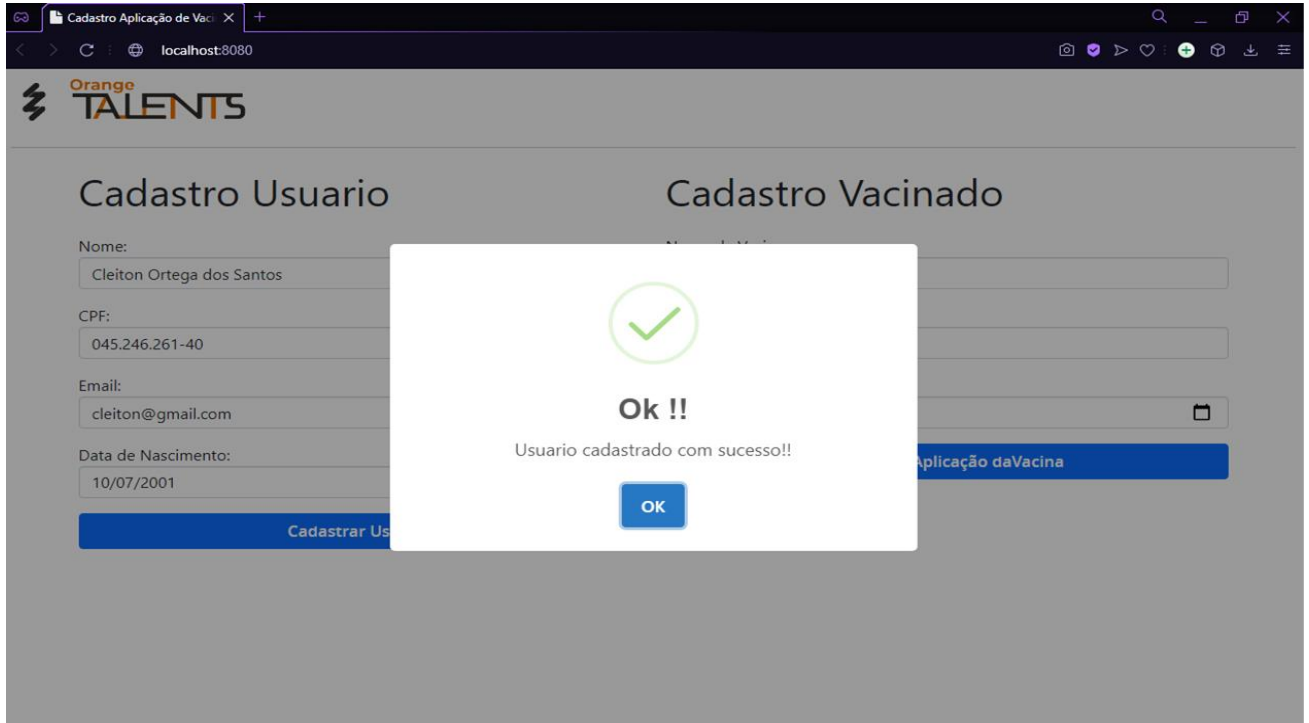
Nome da Vacina:

Email:

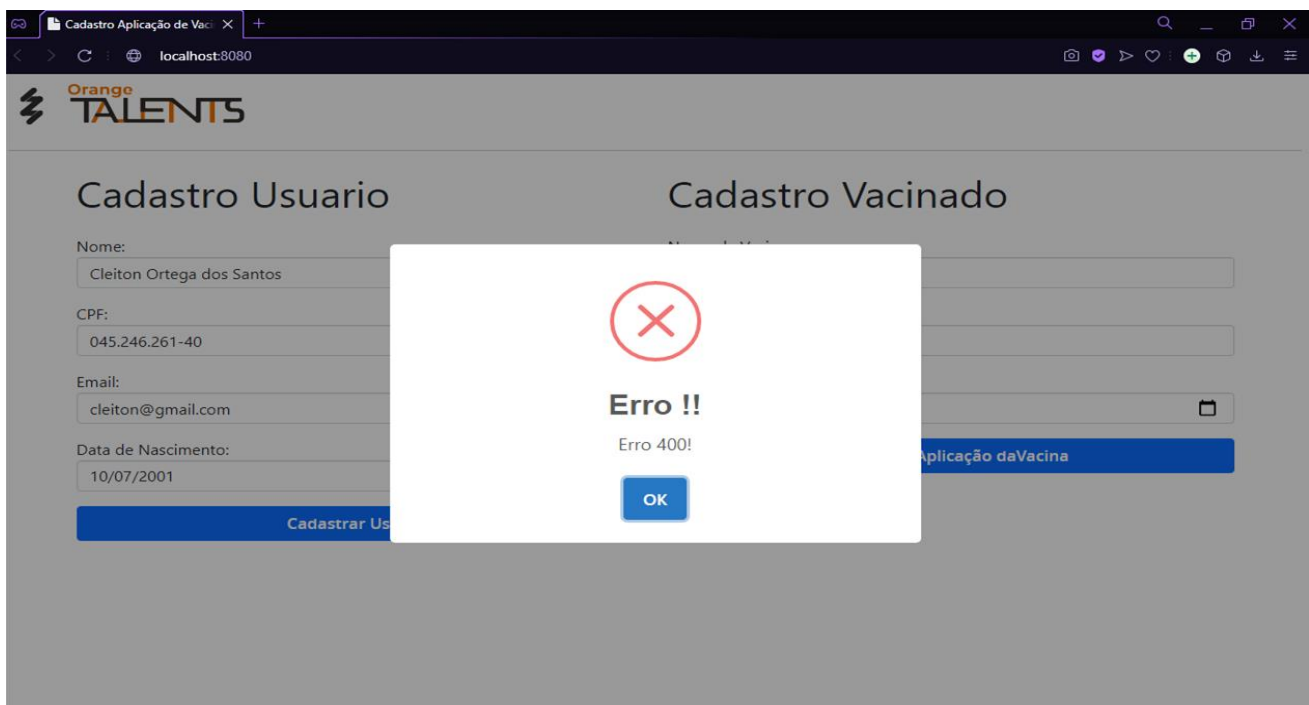
Data da Aplicação da Vacina:

## Criando os dados de usuário

Agora vou cadastrar um novo usuário para tomar a vacina logo em seguida!

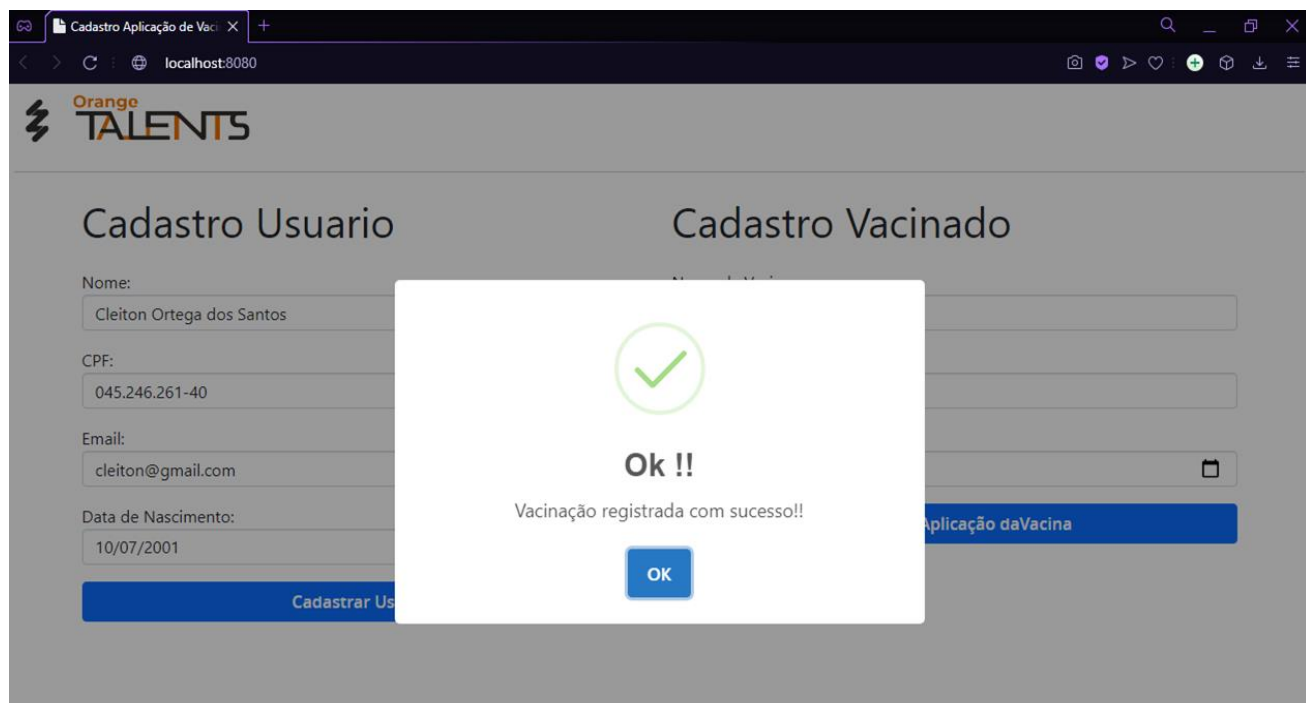


Usuário cadastrado, mas caso tente cadastrar de novo, com os mesmos dados, o resultado será diferente:

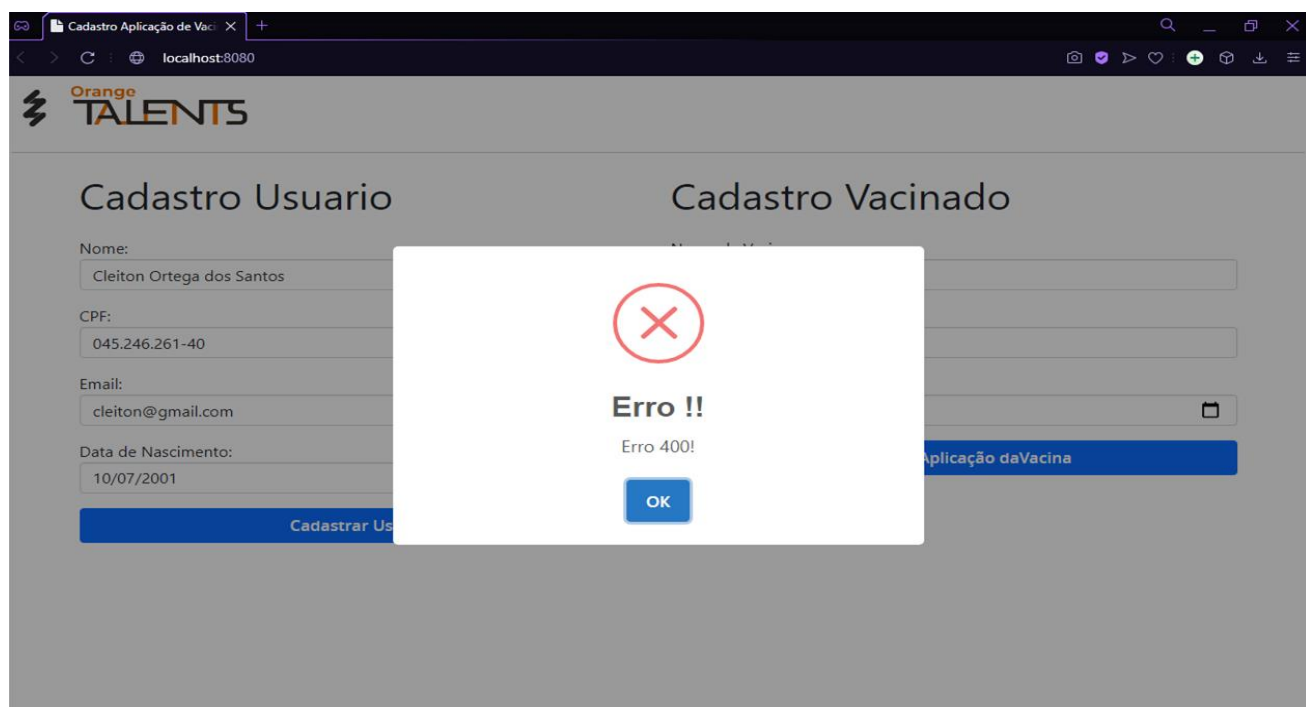


## Criando os dados de aplicação de vacina

Agora vou vacinar o usuário cadastrado:

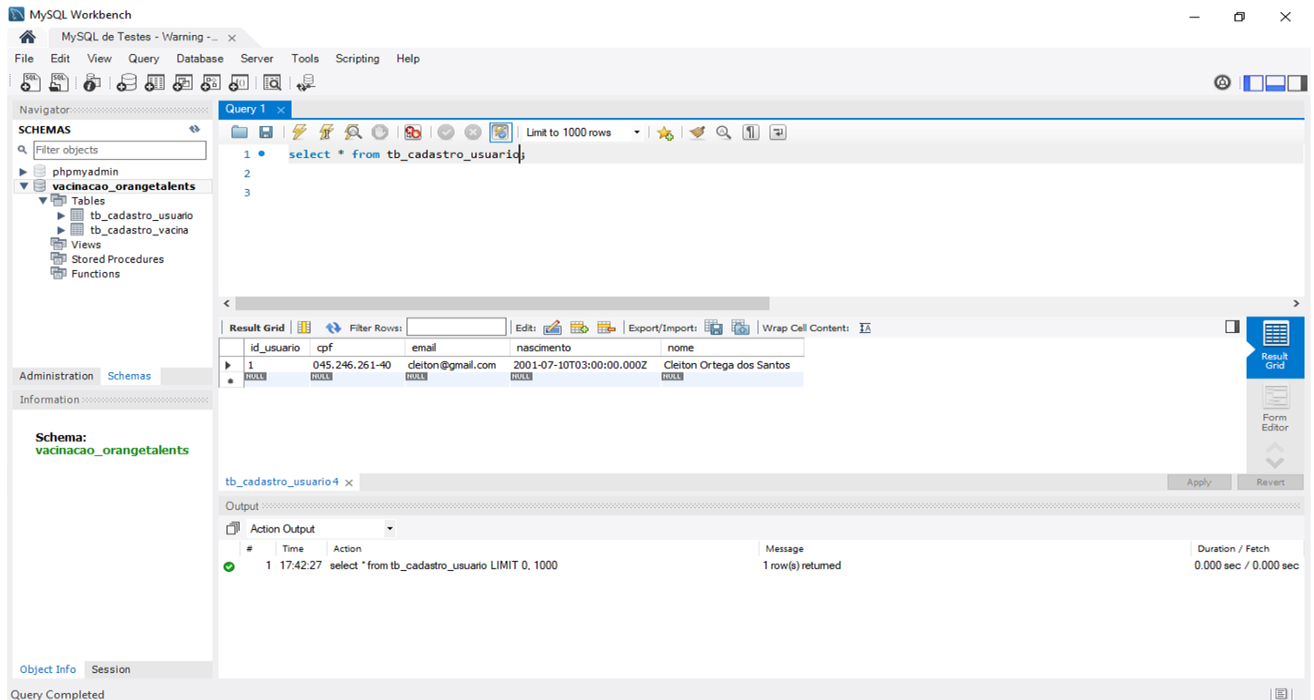


Usuário vacinado! Caso tente vacinar novamente o mesmo usuário ou tente vacinar alguém não cadastrado ele retornará o erro abaixo:



## Vendo os dados no MySQL

Agora para fechar com chave de ouro irei mostrar no MySQL os dados registrados! Primeiro o usuário e depois a aplicação da vacina:

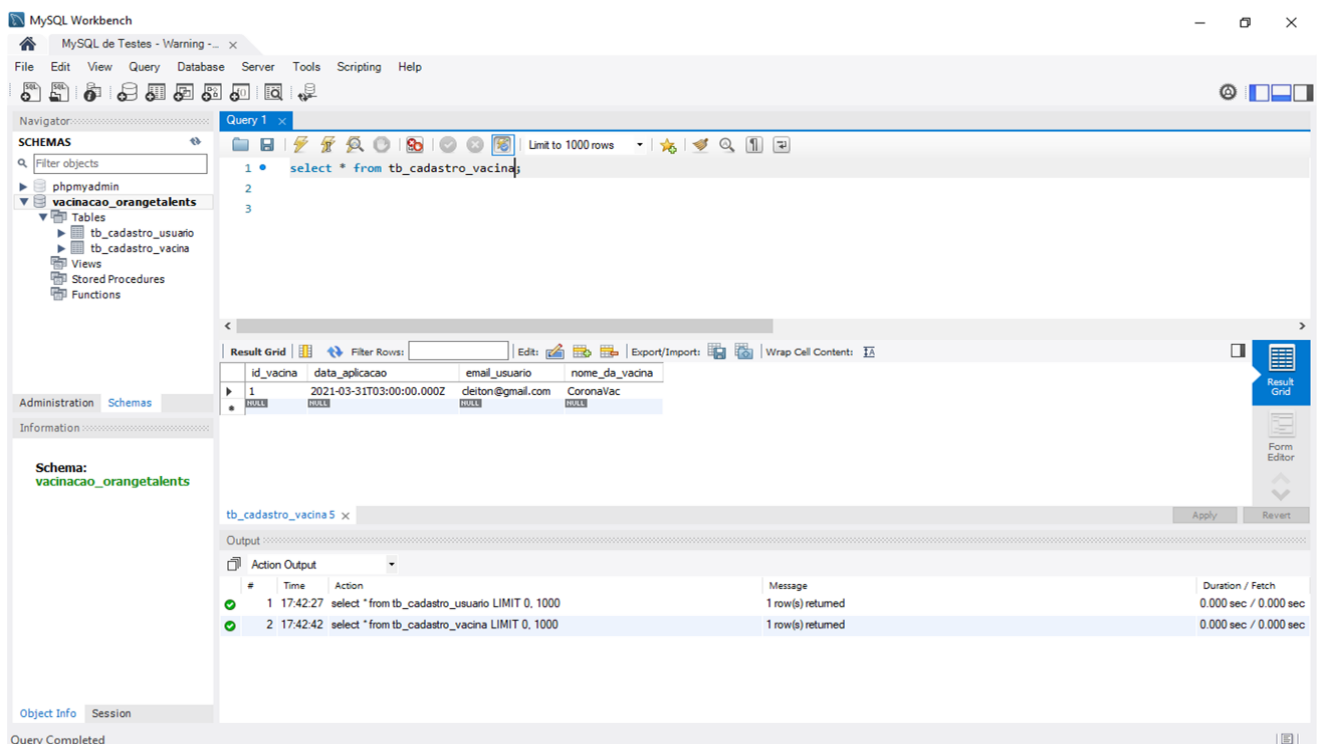


The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'vacinacao\_orangetalents' selected. The main query editor shows the query: `select * from tb_cadastro_usuario;`. The 'Result Grid' tab is active, displaying the following data:

	id_usuario	cpf	email	nascimento	nome
1	045,246.261-40	cleiton@gmail.com	2001-07-10T03:00:00.000Z	Cleiton Ortega dos Santos	

The 'Output' tab shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	17:42:27	select * from tb_cadastro_usuario LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec



The screenshot shows the MySQL Workbench interface. The left sidebar displays the 'SCHEMAS' tree with 'vacinacao\_orangetalents' selected. The main query editor shows the query: `select * from tb_cadastro_vacina;`. The 'Result Grid' tab is active, displaying the following data:

	id_vacina	data_aplicacao	email_usuario	nome_da_vacina
1	2021-03-31T03:00:00.000Z	cleiton@gmail.com	CoronaVac	

The 'Output' tab shows the execution details:

#	Time	Action	Message	Duration / Fetch
1	17:42:27	select * from tb_cadastro_usuario LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec
2	17:42:42	select * from tb_cadastro_vacina LIMIT 0, 1000	1 row(s) returned	0.000 sec / 0.000 sec

# Fim do desafio

Obrigado por verem minha API! Caso tenha alguma dúvida, podem me encontrar através do meu e-mail, LinkedIn ou GitHub. Deixarei eles logo abaixo.

Tenham um bom dia!



Email: [c.ortega200935@gmail.com](mailto:c.ortega200935@gmail.com)



LinkedIn: <https://www.linkedin.com/in/cleitonortegadev/>



GitHub: <https://github.com/CleitonOrtega>