# Que tal construir uma API Rest para controlar a aplicação de vacinas na população brasileira, utilizando Java, Spring Boot e JPA?

## *Rodrigo França desenvolvedor full-stack.*

## A ZUP é uma empresa que trabalha desenvolvendo produtos digitais que impactam diretamente no modelo de negócio dos seus clientes. Eles trabalham com pessoas incríveis imersas em uma cultura de inovação para gerar crescimento exponencial. Um de seus desenvolvedores, chamado Rodrigo França, recebeu uma missão de construir uma API Rest para controlar a aplicação de vacinas na população brasileira.

Pré-requisitos

É importante que você tenha um conhecimento básico nessas tecnologias:

* Java
* Protocolo HTTP
* Estilo Arquitetural REST
* SQL
* JPA
* Maven
* Arquitetura MVC
* Spring Framework

Essas são as ferramentas que precisam ser instaladas para você seguir com a construção da API Rest.

* JDK 8[1]
* MySql Workbench[2]
* Spring Tool Suite[3]
* Postman[4]

## Antes de iniciar a construção da nossa API Rest vamos primeiro entender os requisitos:

## Construir um cadastro de pessoas, com os seguintes dados: nome, e-mail, CPF e data de nascimento.

## Criar um cadastro de vacinas, com os seguintes dados: nome da vacina, como ela é aplicada (por exemplo, por injeção na parte superior do braço), quem deve tomar, quando é preciso tomar e quais os benefícios da vacina.

## Registrar quais pessoas tomaram certa dose de uma vacina e em que data ocorreu essa vacinação.

## Ao realizar o cadastro com sucesso é necessário retornar o status 201. No caso de erros de preenchimento, o status deve ser 400 – conforme especificação HTTP [5].

## Uma vez entendido os macro requisitos, vejamos quais os próximos passos para construção de nossa API Rest.

## Modelar o domínio de negócios e criar a base de dados;

## Carregar dados de vacina através de um script SQL;

## Fazer o *bootstrap* da aplicação e realizar o *Hello World*!

## Implementar classes de domínio.

## Adicionar a dependência do Spring Data JPA e as anotações da JPA.

## Adicionar as dependências do Spring Web e do H2 DB e realizar testes locais.

## Criando o primeiro endpoint Rest.

## Criando o primeiro Serviço.

## Criando o primeiro Repository.

## Salvando uma Pessoa no banco de dados.

## Passo 1: Modelar o domínio de negócios e criar a base de dados.

## Para essa atividade escolhi o MySql Workbench, uma ferramenta simples de usar e que além de modelar, também permite gerar o script SQL do modelo lógico e ainda possui uma interface gráfica para executar nossos scripts e testar nossas Querys.

## Modelagem feita, agora vamos gerar o script SQL.

## Agora é só executar o script e ver se o Schema e as tabelas foram criados com sucesso.

## Passo 2: Carregar dados de vacina através de um script SQL.

## Vamos criar nosso script para “popular” nossa tabela de vacina. Para isso basta fazer um *insert* simples como mostra a figura abaixo:

## 

## Passo 3: Fazer o bootstrap da aplicação e realizar o Hello World.

## Para essa etapa, vamos utilizar um gerador de projetos, essa é a forma mais simples e prática de se criar um projeto com Spring Boot. Após acessar o site do Spring Initializr[6], e preencher algumas informações, teremos algo parecido com a figura abaixo e um botão *generate*, que vai permitir salvar o projeto em seu computador.

## 

## Importe o projeto (salvo no seu computador) em sua IDE favorita, nós vamos utilizar o Spring Tool Suite (STS) uma IDE do ecossistema Spring, com ela temos alguns recursos, como o Boot Dashboard para iniciar, interromper e fazer o debug da aplicação de forma simplificada. Já com o Edit Starters vamos adicionar novas dependências de forma mais prática sem precisar mexer no arquivo pom. xml.

## Implemente o famoso *Hello World* , dentro do método *main* na classe principal do projeto, que fica no pacote br.com.zup e se chama ControleAplicacaoVacinaApplication.java.

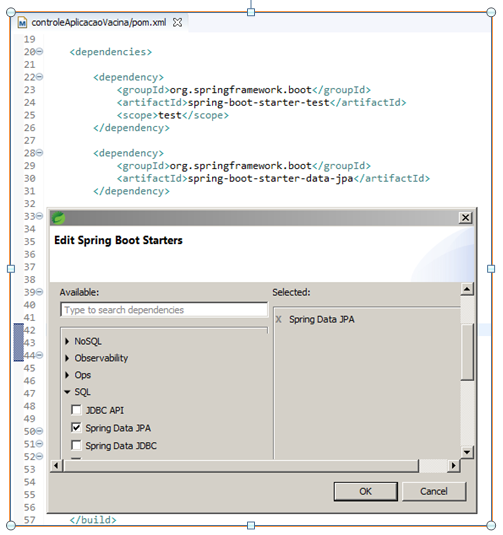
## Passo 4: Implementar classes de domínio.

## Conforme modelo de negócio teremos três (3) classes: Pessoa, Vacina e Vacinacao. Todas com atributos privados para garantir o encapsulamento, além dos métodos de acesso gets e sets, equals e hashCode.

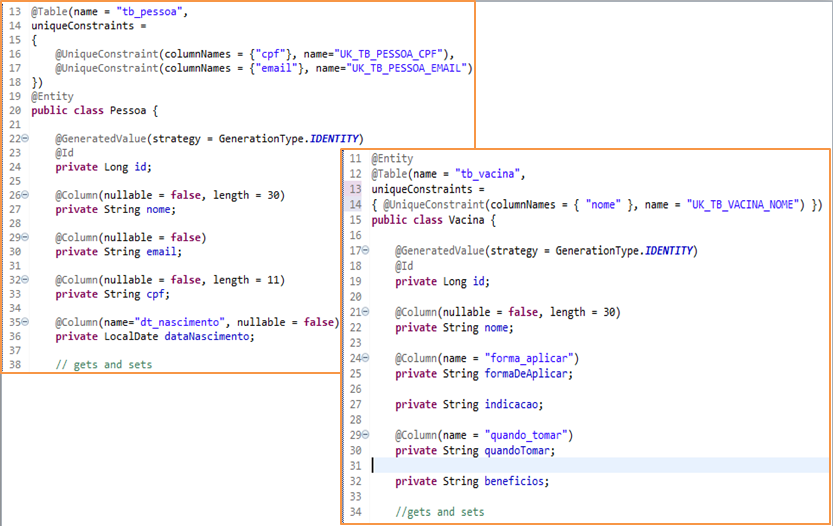
## 

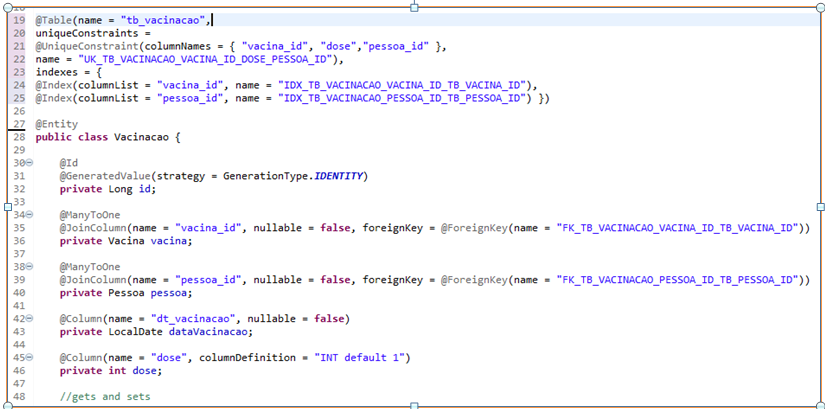
**Passo 5: Adicionar a dependência do Spring Data JPA e as anotações da JPA.**

1. Adicione a dependência pelo **Edit Starters** ou pelo arquivo **pom. xml**.   
   O **pom. xml** é um arquivo de configuração do **Maven**[7] aonde declaramos as dependências do nosso projeto dentre outras configurações.  
   Com essa dependência temos tudo que é necessário para um desenvolvimento com a **JPA** e o **Hibernate** bem mais simples e mais prático.

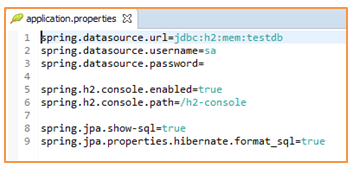


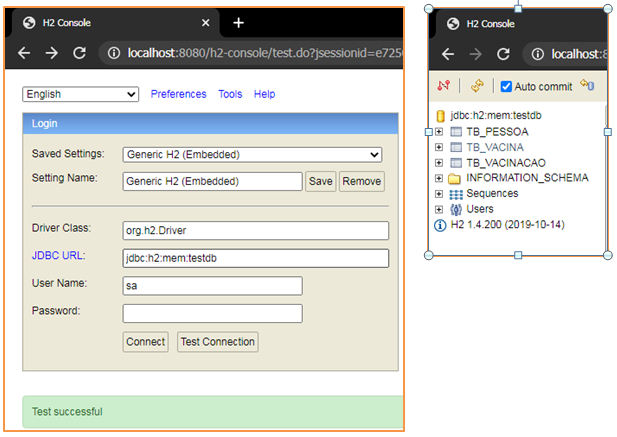
1. Adicione as anotações da **JPA**, são elas que vão permitir que nossas classes **Java** (mundo orientado a objetos) “conversem” com nossas tabelas do banco de dados (mundo relacional).





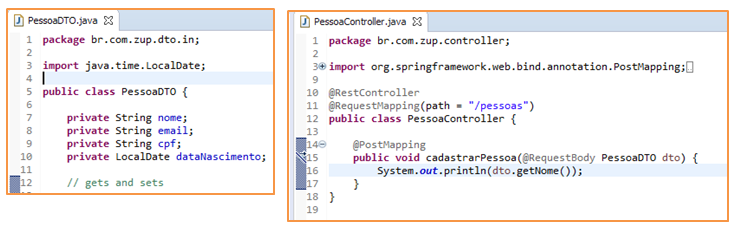
**Passo 6: Adicionar as dependências do Spring Web e do H2 DB e realizar testes locais.**

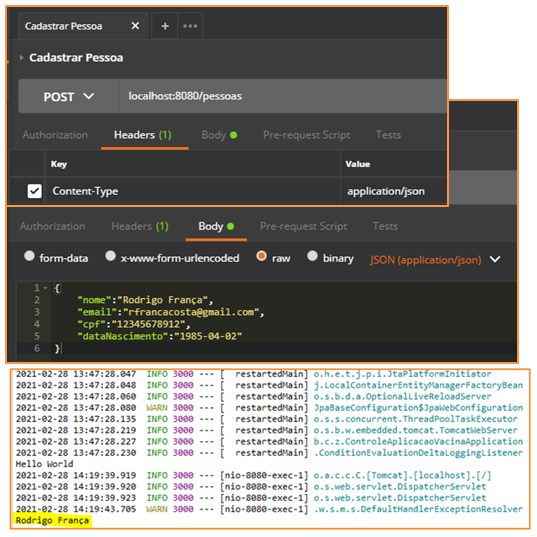
1. Adicione a dependência do **Spring Web** e com isso vamos poder criar nossos serviços web de forma bem simples, comparado à criação de um ***Web Service REST*** utilizando a especificação **JAX-RS** do **Java EE**, além de outros recursos do mundo Java para Web.
2. Adicione a dependência do **H2 DB** e com isso vamos ter um banco de dados carregado em memória, além de uma interface gráfica para executar nossas **querys**. Tudo isso via ***browser***, dentro do contexto da nossa aplicação, por isso a importância da dependência **Spring Web** também.
3. Configure o arquivo **application.properties** com as informações para conexão com o banco de dados **H2**.  
   
4. Teste a conexão com o banco através da **url** <http://localhost:8080/h2-console>.



**Passo 7: Criando o primeiro endpoint Rest.**

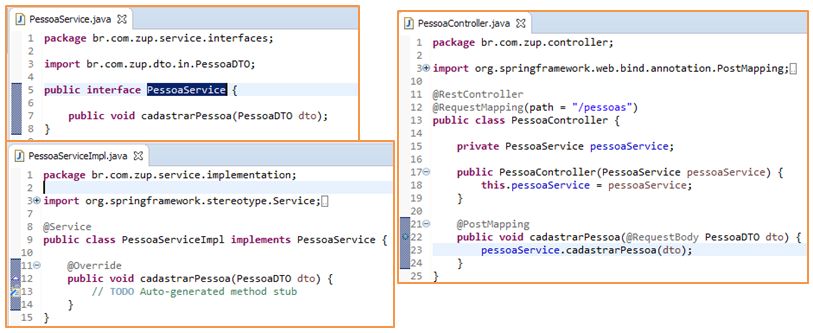
1. Implemente a classe **PessoaController** no pacote **br.com.zup.controller** que terá como responsabilidade de receber as requisições (request) **HTTP**. No mundo **Spring** essa classe é chamada de ***Controller***.
2. Adicione a anotação **@RestController**, e **@RequestMapping**, para a ultima , vamos informar um **path** (“/pessoas”) que vai indicar ao **Spring** qual é o “caminho“ para chegar nessa classe.
3. Definir um método chamado, **cadastrarPessoa**, ele vai receber como parâmetro uma **PessoaDTO**, e seu retorno será **void**.
4. Adicione a anotação **@PostMapping** que vai mapear esse método para requisições do tipo **POST**, e **@RequestBody** que vai indicar que o parâmetro do método deve vir no corpo da requisição, essa anotação é usada antes da declaração do parâmetro do método.
5. Implemente a classe **PessoaDTO** dentro do pacote **br.com.zup.dto.in** que vai ter todos os atributos de uma classe **Pessoa** com exceção do ***id***. Também os métodos **gets** e **sets**, **equals** e **hashCode**. É uma boa prática não expor nossas entidades de negócio (**Pessoa**).

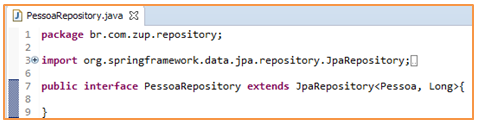


1. Realize o teste com o **Postman** faça uma chamada para:  
   **localhost: 8080/pessoas**, informe na aba **Headers** o **Content-Type** do tipo **application/json** e no corpo da requisição informe uma **PessoaDTO** no formato **JSON.**

**Passo 8: Criando o primeiro Serviço.**

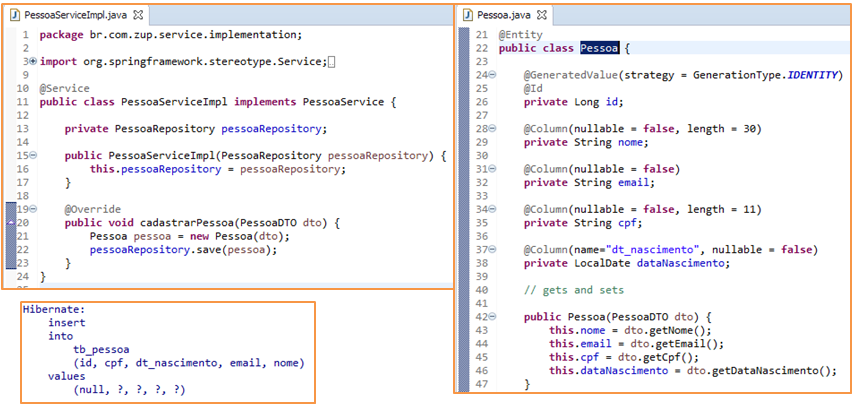
1. Crie uma **interface** **PessoaService** no pacote **br.com.zup.service.interfaces** defina um método que receba como parâmetro uma **PessoaDTO**.
2. Implemente a classe **PessoaServiceImpl**, dentro do pacote **br.com.zup.service.implementation**, ela vai implementar (***implements***) o comportamento da interface **PessoaService**, e com isso teremos mais flexibilidade, pois se surgir a necessidade de uma nova implementação (comportamento diferente) para essa **interface**, basta seguir o contrato (***interface***) que não vamos problema algum.
3. Adicione a anotação **@Service** para Indicar que a classe **PessoaServiceImpl** será um componente gerenciado pelo **Spring** e assim poder ser injetada em nosso ***Controller*** por exemplo.

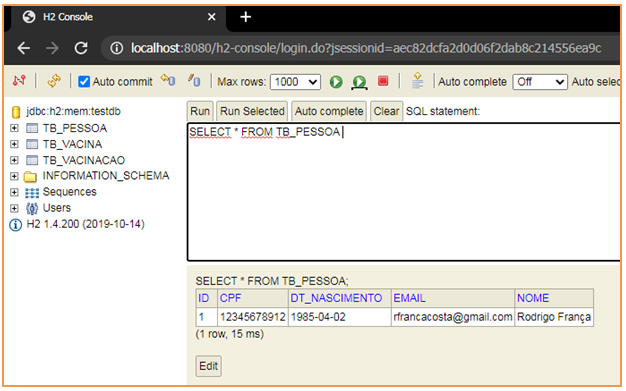
  
  
**Passo 9 : Criando o primeiro Repository.**

1. Crie uma **interface** chamada **PessoaRepository** no pacote **br.com.zup.repository** e faça uma **herança** (***extends*** ) da **interface** **JpaRepository** e fazendo uso do **Generics**, vamos informar as classes **Pessoa** e **Long**, para que os métodos dessa **interface** possam trabalhar em “cima” dessas duas classes.  
   

**Passo 10: Salvando uma Pessoa.**

1. Faça a injeção de dependência da **PessoaRepository** na classe **PessoaServiceImpl**, que o **Spring** vai instanciar um objeto que sabe implementar a **interface** **PessoaRepository.**
2. Implemente o método **cadastrarPessoa** da classe **PessoaServiceImpl** da seguinte forma, instancie uma **Pessoa** (**new**) passando como argumento para o construtor o dto (**PessoaDTO**) recebido do **Controller.** Com essa instancia de pessoa,vamos passa-la como argumento para o método **save,** da interface **PessoaRepository,** que foiherdado de **JpaRepository.**
3. Defina um construtor na classe **Pessoa** que receba como parâmetro uma **PessoaDTO** e em seu comportamento configure uma **Pessoa** baseada nesse dto pois ele tem todos os atributos que uma Pessoa precisa.
4. Faça a injeção de dependência de **PessoaService**  na classe **PessoaController**, que o **Spring** vai injetar uma instancia da classe **PessoaServiceImpl** .
5. Implemente o método **cadastrarPessoa** da classe **PessoaController** da seguinte forma, invoque o método **cadastrarPessoa** do objeto que está implementando a **interface** **PessoaService**, passando como argumento o dto (**PessoaDTO**) recebido no corpo da requisição **HTTP**;
6. Teste com o **Postman**.
7. Verifique se a pessoa foi salva no banco com sucesso.





Referências

1. “Java SE Development Kit 8 Downloads”:  
   <https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html>
2. “MySQL Workbench 8.0.23”:  
   <https://dev.mysql.com/downloads/workbench/>
3. “Spring Tools”:  
   https://spring.io/tools
4. “Download Postman”:  
   <https://www.postman.com/downloads/>
5. “Status Code and Reason Phrase”(em inglês):  
   <https://www.w3.org/Protocols/rfc2616/rfc2616-sec6.html#sec6.1.1>
6. “Spring initializr”:  
   <https://start.spring.io/>
7. “Downloading Apache Maven 3.6.3”:  
   <https://maven.apache.org/download.cgi#downloading-apache-maven-3-6-3>
8. “Cartilha de Vacinas Para quem quer mesmo saber das coisas”:  
   https://bvsms.saude.gov.br/bvs/publicacoes/cart\_vac.pdf

Mini-Bio do Autor

Rodrigo França é um profissional full-stack pleno, com conhecimento nas tecnologias (back-end) Java e especificação JEE (Servlet, JSF, CDI, EJB, JPA), Framework Bootstrap, PrimeFaces, Spring Framework (Spring Boot, Spring-Data). Servidores de aplicação/container (Weblogic, IBM WebSphere, WildFly e Apache TomCat). (front-end) Javascript Typscript, Angular2+, RxJS. HTML5, CSS, (Banco de dados) MySql, Postgree, SQLServer e Oracle. Sempre utilizando as melhores práticas e metodologias ágeis (cerimônias do SCRUM) e ferramentas colaborativas tais como Git, SVN, Slack e Trello.