

RELATÓRIO ENDPOINT CONTROLADOR DE VACINAS.

Otávio Augusto Marcelino Izidoro

INTRODUÇÃO:

Para criação deste endpoint usarei o spring Data JPA, spring Boot, Thymeleaf, spring Validation e spring Framework incorporado com spring MVC e dependências. Escolhi o spring boot por liberar e facilitar algumas aplicações que está sendo citadas abaixo para criação deste endpoint.

Spring Data JPA terá como facilidade pra mim armazenar os futuros cadastros criados neste endpoint que poderá ser salvo no meu banco de dados MySQL. Mas nesta primeira etapa que podemos chamar de teste estarei usando Banco de dados H2.

Spring Validation irá fazer a validação de todos os dados que será cadastrado no meu sistema, para que não seja salvo dados incorretos ao usuário tentar salvar os dados.

Thymeleaf vai auxiliar a criação de um template HTML junto com o desenvolvimento Java. Trazendo uma interação e comunicação Do HTML e JAVA, ou vice-versa. Vejo por outra forma que torna um pouco mais entendível a aplicação que está sendo criada, podendo mostrar alguns erros durante alguns testes antes de ser realmente lançado.

CONSTRUÇÃO:

Classes Criadas:

Cientes.java, CadastroController, CadastroApplication.java.

Interfaces:

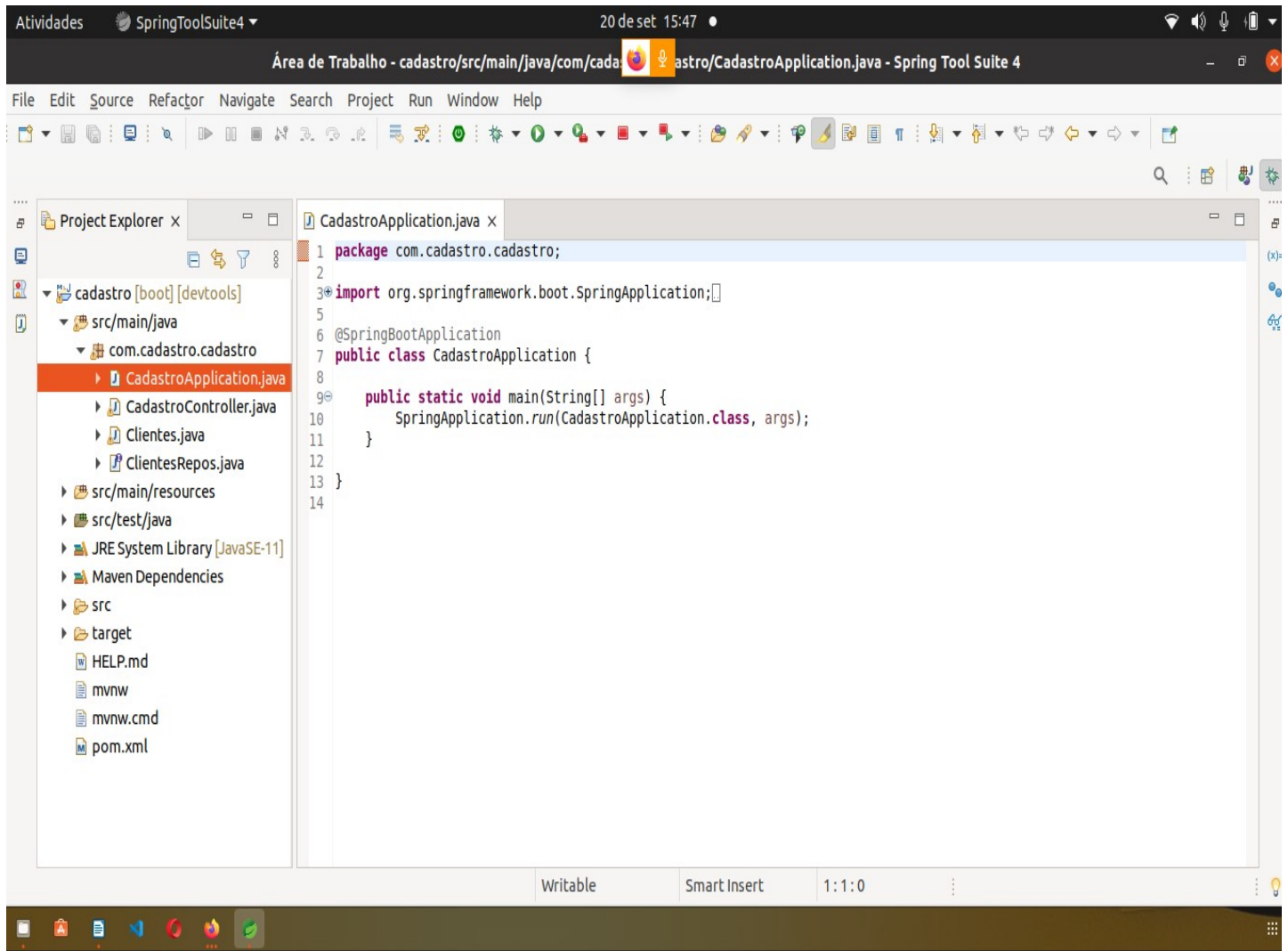
CientesRepos.java

Logo abaixo segue a explicação de como eu pensei em construir este endpoint e as classes citadas acima. Junto a explicação, contém imagem da IDE com os códigos implementados.

Irei usar para elaboração dos códigos a IDE STS e Postman para realizar a consulta e inserção dos dados.

Criei o projeto pela startSpring já selecionando todas as aplicações a serem usadas ao decorrer do projeto.

Primeira classe criada juntamente com o projeto e a classe `CadastroApplication.java`. Ela é o nosso método responsável por realizar a inicialização da aplicação e o servidor já integrado Apache Tomcat.



Segunda classe sendo criada e `CadastroController.java`. Nesta classe eu começo chamando a anotação `@Controller` que ira definir a estrutura da minha classe como uma Spring MVC. Logo eu faço um instanciamento da classe `ClientesRepos.java` e a defino com a anotação `@Autowired`.

A primeira função sendo criada dentro da classe e para listar todos os clientes já cadastrado. Nesta função eu deixei comentado como mostra na imagem abaixo. Anotação `@ResponseStatus(HttpStatus.BAD_REQUEST)` foi criada no primeiro momento seria para exibir status o código 400, mas fiquei na duvida se seria necessário exibir este código antes mesmo do cliente já ter preenchido algum campo ou poderia deixar ele dar o status 200 na exibição da lista ou jogar para minha segunda função abaixo.

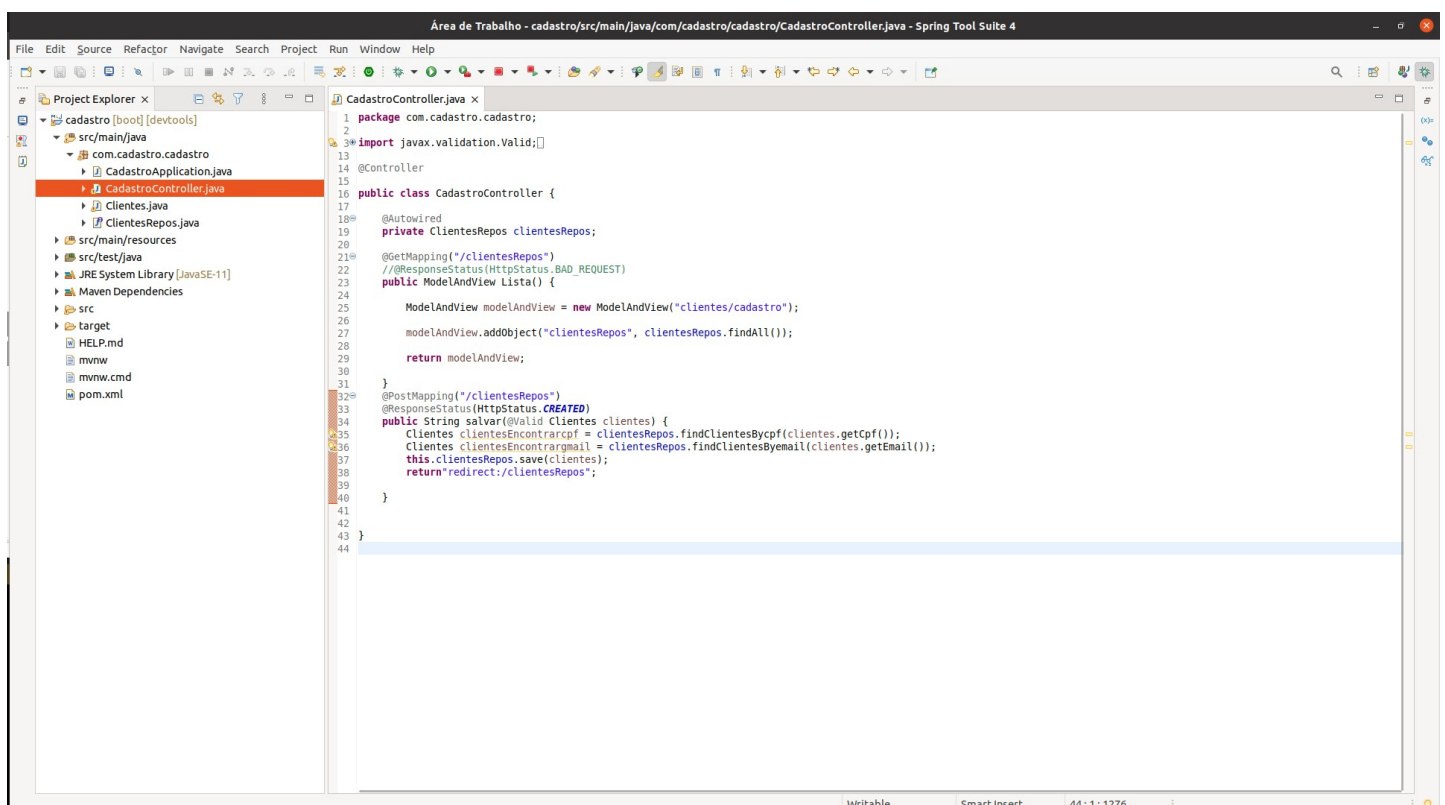
ModelAndView eu defino a minha pagina HTML onde eu possa ver a interface da listas dos clientes e possa fazer de forma entendível o meu `@PostMapping`.

Nesta segunda função eu começo utilizando a anotação `@PostMapping` que faz anotação dentro da minha classe `ClientesRepos.java`. Anotação `@ResponseStatus(HttpStatus.CREATED)` retorna o status 201 assim que todos os campos criados e validados pela anotação `@Valid` forem concluídos.

```
Clientes clientesEncontrarcpf=clientesRepos.findClientesBycpf(clientes.getCpf());  
Clientes clientesEncontrargmail=clientesRepos.findClientesByemail(clientes.getEmail());
```

Estas duas linhas de código mostrada esta sendo chamada da minha classe `ClientesRepos.java`. A função delas e verificar o e-mail e CPF dentro do nosso banco de dados e analisar se não ha nenhum desses dois dados já criados. Assim tornando eles únicos dentro de nosso sistema.

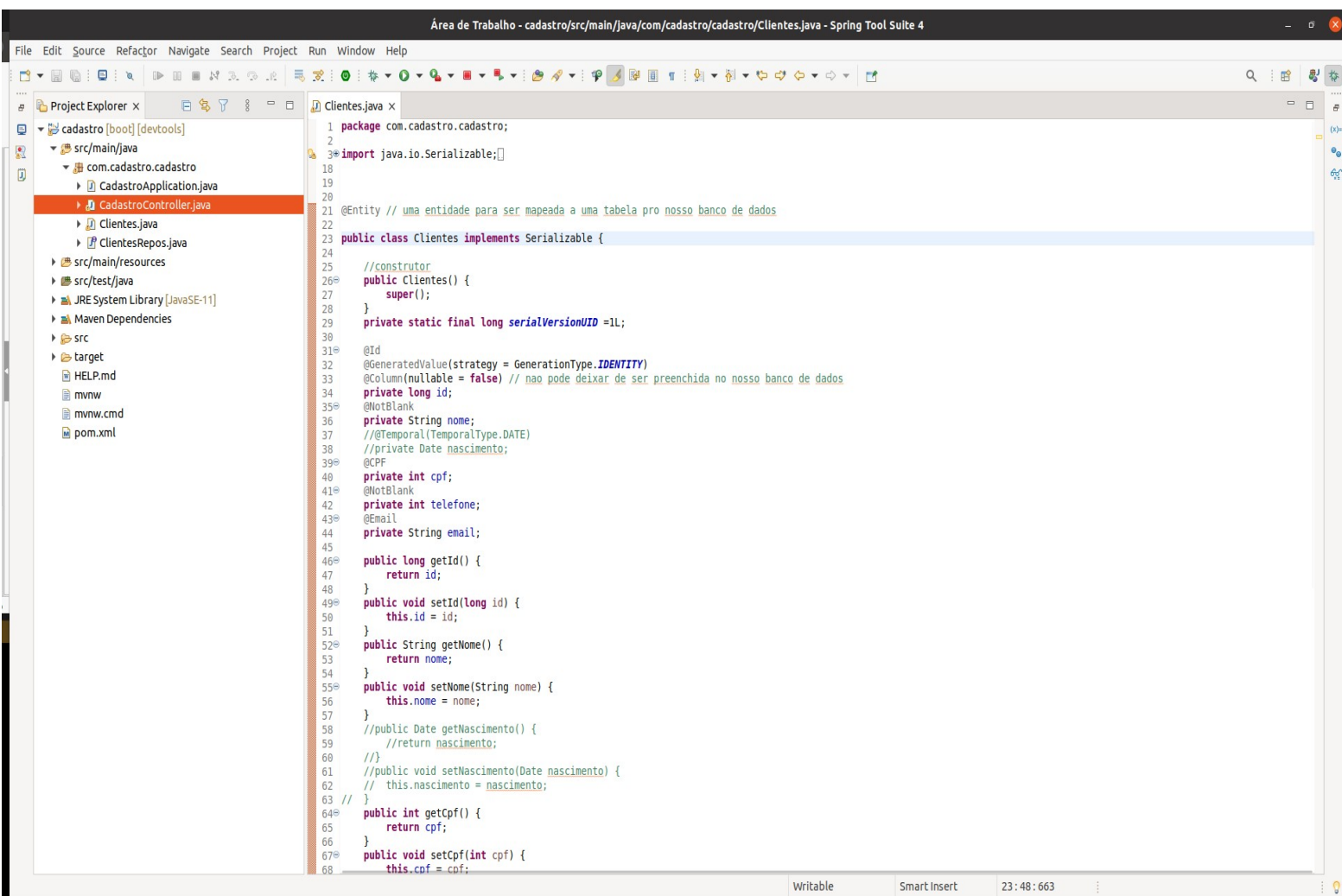
Return eu indico para que assim que a função salvar for criada ela retorna para a pagina de cadastro só que os campos em branco para um novo cadastro.



Eu começo definindo minha classe Clientes.java com a anotação @Entity para que ela seja uma entidade a ser mapeada no banco de dados que no meu caso estou utilizando o H2.

Logo abaixo eu defino os meus atributos. O atributo id estará gerando um numero identificador dentro do meu banco de dados. Os demais atributos são nome, CPF, data de nascimento, telefone, e-mail. Sobre eles eu coloquei anotações que fazem ligação com o @Valid colocado na minha função “salvar” da classe CadastroController.java para que aconteça validação dos dados inseridos.

Logo após gerei meus getters e setters e hashCode. Mas poderia ter economizado código utilizando a anotação Lombok que geraria automaticamente estes métodos.

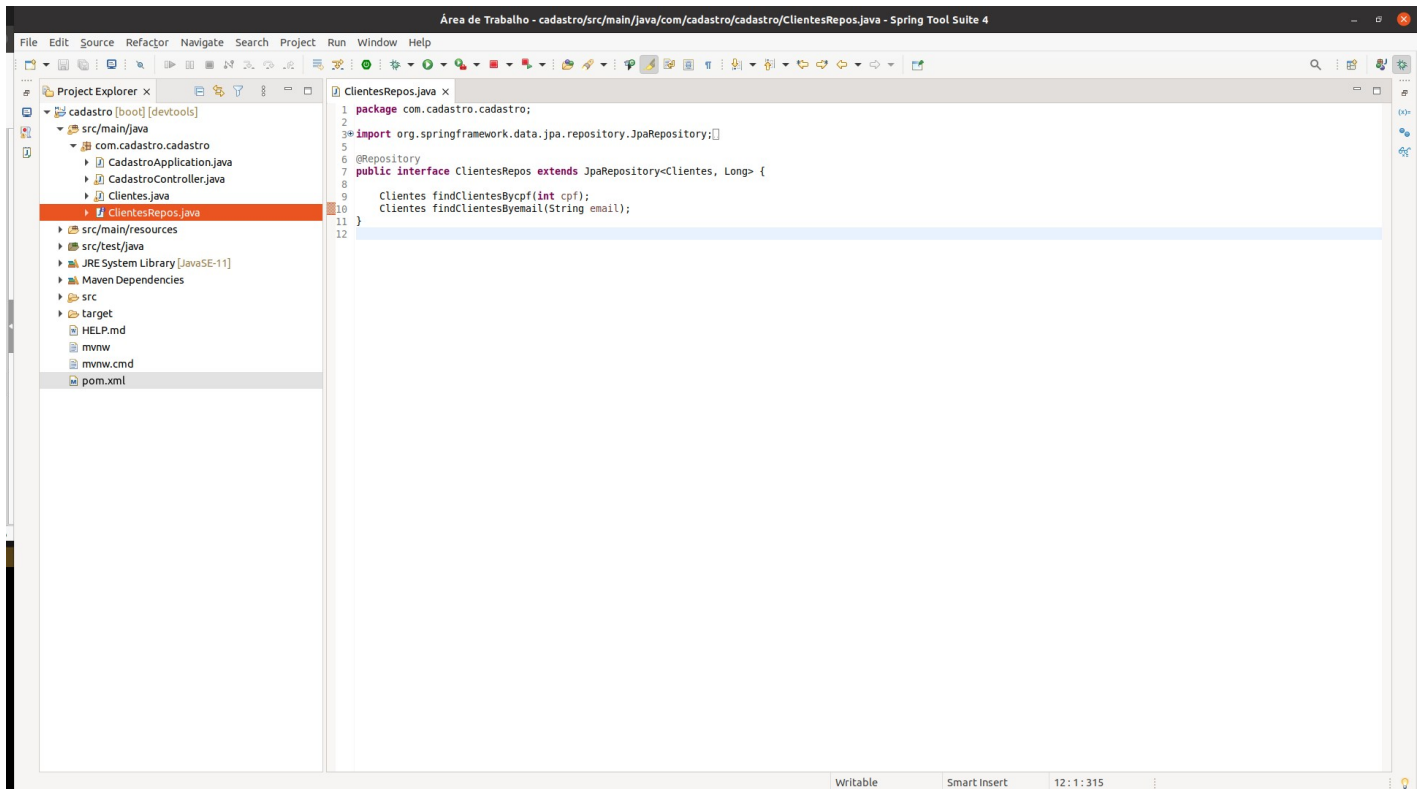


```
1 package com.cadastro.cadastro;
2
3 import java.io.Serializable;
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21 @Entity // uma entidade para ser mapeada a uma tabela pro nosso banco de dados
22
23 public class Clientes implements Serializable {
24
25     //construtor
26     public Clientes() {
27         super();
28     }
29
30     private static final long serialVersionUID = 1L;
31
32     @Id
33     @GeneratedValue(strategy = GenerationType.IDENTITY)
34     @Column(nullable = false) // nao pode deixar de ser preenchida no nosso banco de dados
35     private long id;
36
37     @NotBlank
38     private String nome;
39
40     @Temporal(TemporalType.DATE)
41     private Date nascimento;
42
43     @CPF
44     private int cpf;
45
46     @NotBlank
47     private int telefone;
48
49     @Email
50     private String email;
51
52     public long getId() {
53         return id;
54     }
55
56     public void setId(long id) {
57         this.id = id;
58     }
59
60     public String getNome() {
61         return nome;
62     }
63
64     public void setNome(String nome) {
65         this.nome = nome;
66     }
67
68     //public Date getNascimento() {
69     //    return nascimento;
70     //}
71
72     //public void setNascimento(Date nascimento) {
73     //    this.nascimento = nascimento;
74     //}
75
76     public int getCpf() {
77         return cpf;
78     }
79
80     public void setCpf(int cpf) {
81         this.cpf = cpf;
82     }
83 }
```

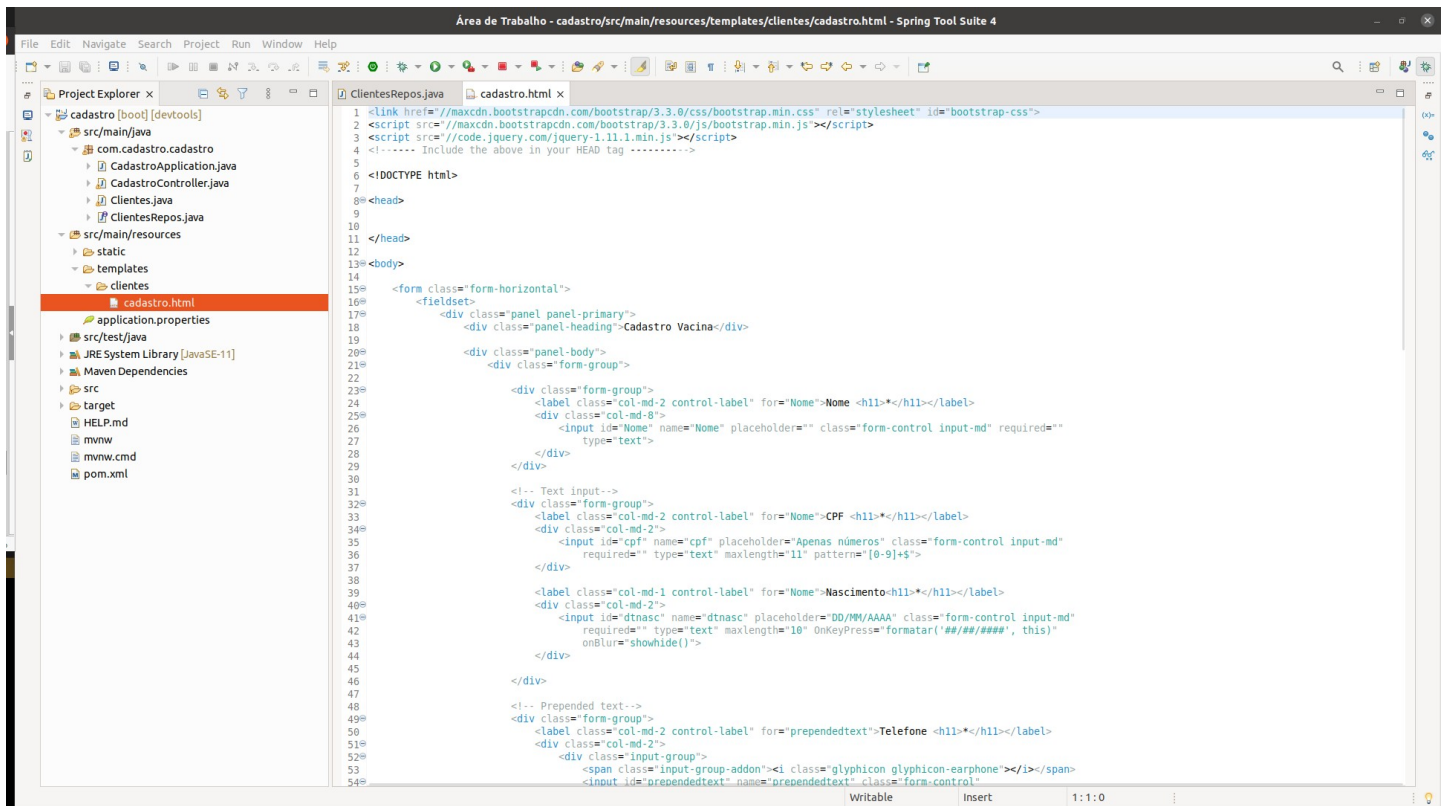
Nesta parte estou criando minha interface “Clientesrepos.java” ela que fara a ligação com o meu controller. Assim dando vida ao sistema para que ele possa listar e inserir clientes.

Clientes findClientesBycpf(int cpf);
Clientes findClientesByemail(String email);

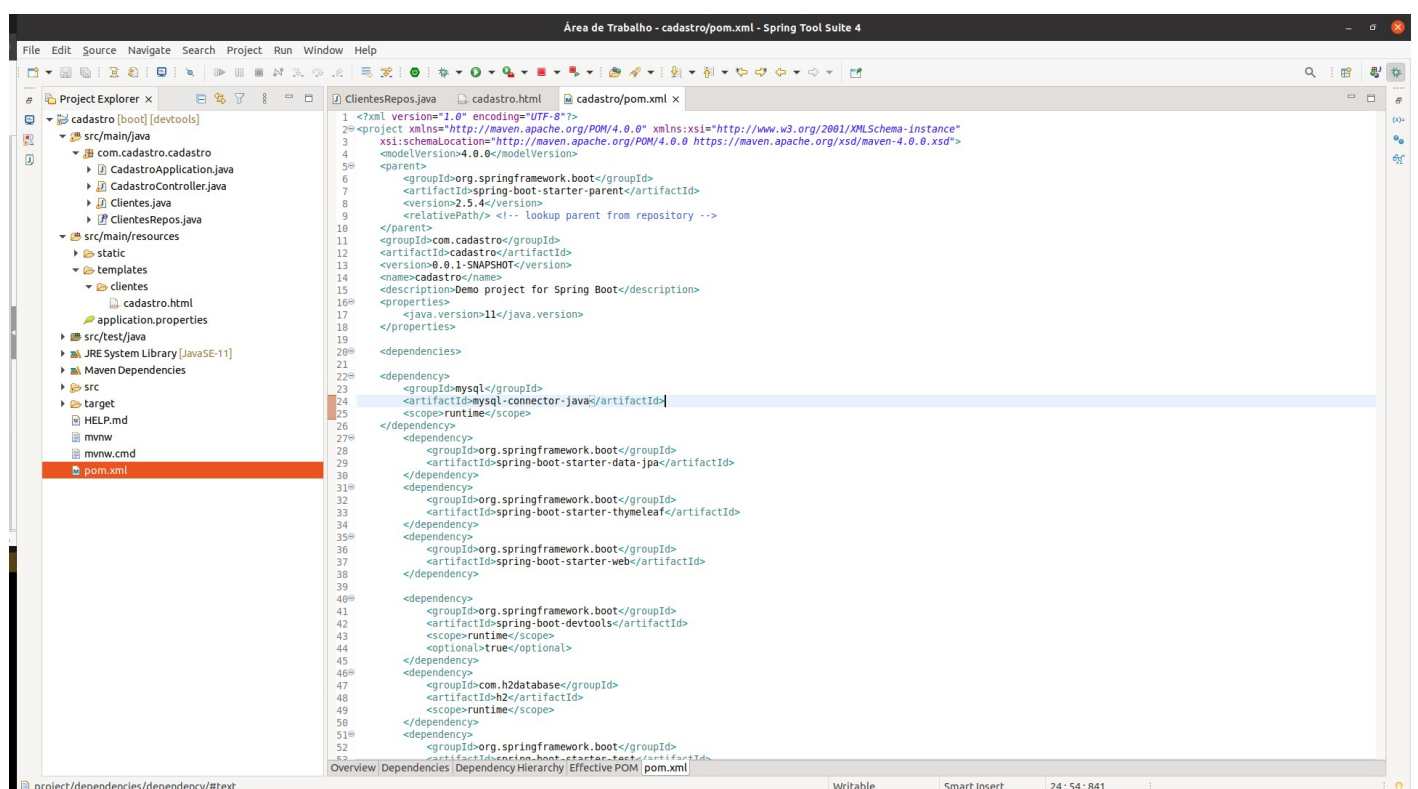
Estas duas linhas de códigos são uma função para verificar se os CPF e e-mail, são únicos. Esta função e chamada na minha classe “CadastrosController.java” onde e executada junto com a função “salvar” como já mencionada nas explicações acima.



Já na parte de web optei por usar um bootstrap, por poupar tempo em ter que implementar HTML e css. Dentro do HTML coloquei todos os atributos necessários que o cadastro pedia e ordenei que todos fossem campos obrigatórios.



Na imagem abaixo esta meu pom.xml criado onde inseri as dependências do meu JPA, Bootstrap, H@, SpringBoot e varias outras.



Produtividade no Desenvol...

Banco de dados H2 com...

localhost:8080/clientesRepos

+

localhost:8080/clientesRepos

Cadastro Vacina

Nome *

CPF *

Apenas números

Nascimento *

DD/MM/AAAA

Telefone *

XX XXXXX-XXXX

Email *

email@email.com

Cadastrar

Cancelar

Lista de Clientes vacina

Nome	Cpf	Data de Nascimento	Telefone	Email
Otávio	123.456.789-09	23/01/2001	35 11111-1111	o@gmail.com
Jose	123.456.789-09	23/01/2001	35 11111-1111	o@gmail.com
Maria	123.456.789-09	23/01/2001	35 11111-1111	o@gmail.com

CONSIDERAÇÕES

Este endpoint não esta 100% funcionando. Pois encontrei dificuldades na questão de fazer a ligação com os dados inseridos na minha HTML e jogá-los para meu java para que possa ser alocado no Banco de Dados. Estes dados colocados e apenas fictícios inseridos no próprio código da minha HTML.

Espero ter alcançado os objetivos propostos. Desde já informo que sou leigo e não havia ainda programado nesta parte de string. Espero ter tido resiliência a essa nova aprendizagem. Desde já agradeço a oportunidade!

