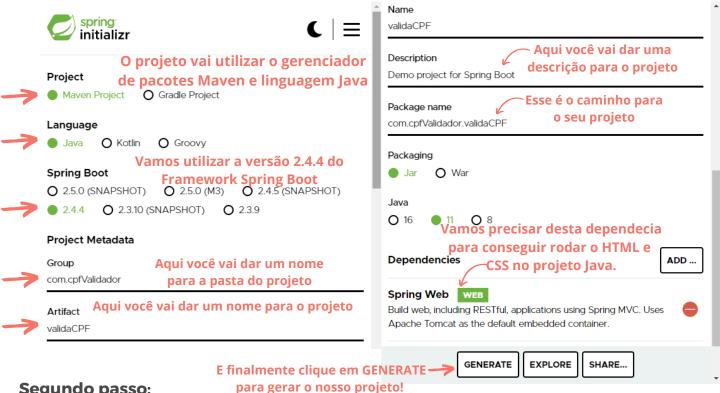




Criando o projeto

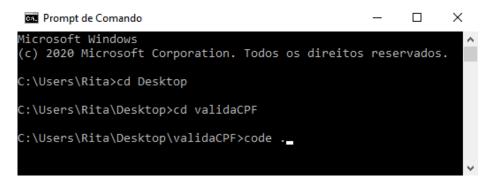
Primeiro passo:

Para esse projeto vamos precisar instalar o Java e um editor de código (no meu caso vou utilizar o Visual Studio Code). No Code é preciso baixar extensões pra rodar o Java (JDK, Java Extension Pack e Maven for Java) Depois de fazer as instalações, vamos criar um projeto do tipo Maven no site https://start.spring.io/:



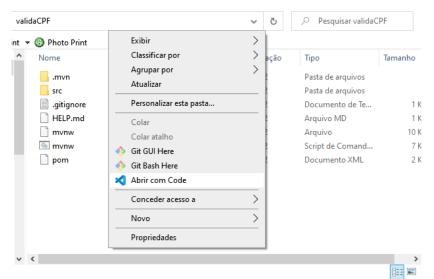
Segundo passo:

Vamos abrir no Editor de código o nosso projeto, existem 3 maneiras de fazer isso:



- 1 No Prompt de Comando ou terminal do seu sistema.
- Digite cd + o local de destino para onde você deseja ir (no meu caso extrai o projeto na minha área de trabalho, ou seja, Desktop)

Entrei no Desktop e depois entrei na pasta do projeto que se chama validaCPF com o comando cd. Após isso digite (code .) que o terminal vai abrir o editor de código já na pasta que você deseja.



2 - Na Pasta do projeto.

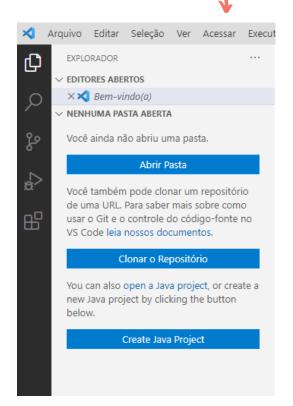
Abra a pasta do projeto e clique com o botão direito do mouse na parte branca da pasta.

Após isso clique em Abrir com Code.

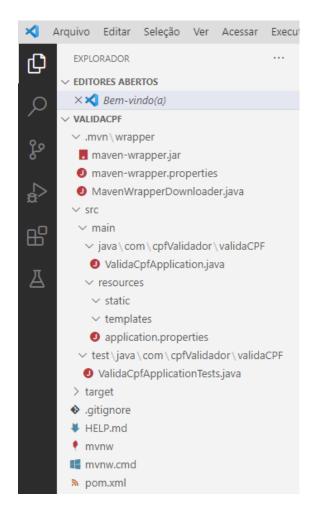
3 - Direto no Editor de código.

Abra o Editor de código e na tela principal você poderá clicar na opção Abrir Pasta.

Selecione a pasta que quiser importar e o editor irá abrir sua pasta.

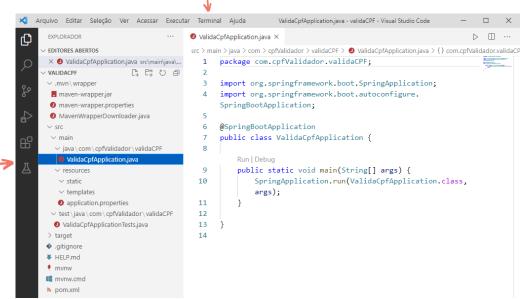


O seu projeto irá vir neste formato:

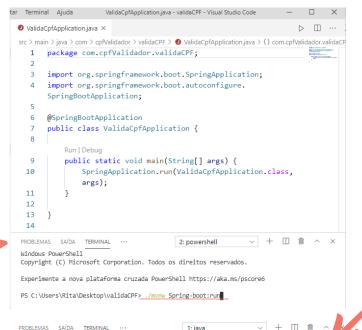


Terceiro passo:

Vamos 'startar' o Java para ver se está tudo OK com as configurações!



Selecionar a pasta src → java → com → cpfValidador → validaCPF → arquivo java. Esse é o arquivo: 'ValidaCpfApplication.java'. Vamos abrir o terminal para rodar o Spring Boot.



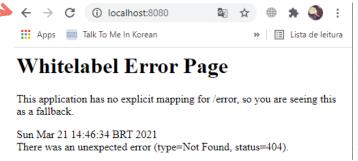
Depois de abrir o terminal, para você 'startar' uma aplicação você usa o comando:

./mvnw spring-boot:run - Para powershell mvnw.cmd sping-boot:run - Para cmd Ps: o meu sistema é Windows.

Essa é a mensagem que aparecerá quando você 'startar' o seu arquivo Java no terminal.

E quando você acessar a porta localhost:8080 no seu navegador, irá aparecer o erro 404(o protocoloco HTTP não conseguiu encontrar o File), porém esse erro indica que você conseguiu r'startar' e o Java já esta rodando!







Configurando o projeto

Para utilizarmos o HTML5 e CSS em projeto Java, precisamos fazer alguns ajustes. Para começar vamos criar uma pasta dentro da pasta java. Esse é o caminho: src → main → java → com → cpfValidador → validaCPF. Criei uma pasta chamada controllers e dentro desta pasta criei um arquivo Java chamado HomeController.java (lembrando que arquivo java sempre começa com letra maiúscula)

EXPLORADOR

VEDITORES ABERTOS

MomeControllers,java src\main\java\c...

VALIDACPF

mun

src

main

java\com\cpfValidador\validaCPF

controllers

HomeControllers,java

ValidaCpfApplication,java

> resources

> test

> target • .gitignore

₩ HELP.md

mvnw.cmd

mx.moq 🙈

• mvnw

```
src > main > java > con cpfValidador > validaCPF > controllers > ① HomeContr
      package com.cpfValidador.validaCPF.
      controllers;
      import org.springframework.stereotype.
      Controller:
      import org.springframework.web.bind.
      annotation.GetMapping;
  5
      @Controller
  7
      public class HomeControllers {
  8
  9
         @GetMapping("/")
 10
         public String index() {
             return "home/index";
 11
 12
 13
 14
```

arquivo vamos colocar uma classe Spring Boot chamada @Controller. Tudo estiver dentro desta classe o Controller vai direcionar para view (front-end) para obtermos a resposta que seria o HTML. lavaScript.

Ps: não pode esquecer de importar esses arquivos como mostra a linha 3 e 4.

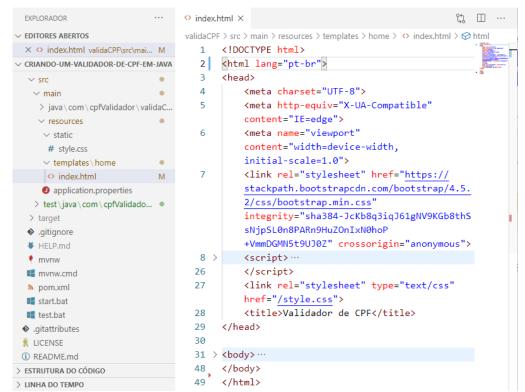
Dentro da classe @Controller teremos o @GetMapping que é um método que vai retornar um dado de um template. Ele vai mandar para uma rota chamada "home" que vai retornar uma String do arquivo HTML "index" que ainda iremos criar.

Após isso vamos adicionar um template web chamado **Thymeleaf** como dependência no arquivo Maven "pom.xml". Essa dependência é uma biblioteca que vai renderizar o arquivo HomeController.java em um arquivo HTML e CSS no navegador.



Criando um arquivo HTML e CSS

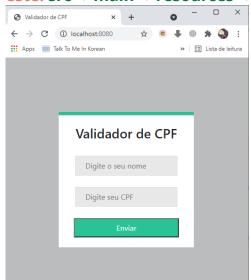
Agora já podemos rodar um arquivo HTML e CSS com Java no navegador. Então podemos criar o arquivo HTML.



Criei uma pasta chamada
"home" dentro da pasta
"templates" e depois criei
um arquivo "index.html"
dentro de home (a rota que
foi mencionada
anteriormente). Então esse
é o caminho: src → main →
resources → tempates →
home → index.html

No index utilizei o link de bootstrap que é um site que quando referenciado ou baixado dá uma formatação simples para o arquivo.

Mesmo assim utilizei um arquivo "style.css" para fazer formações mais complexas. Esse arquivo css foi criado na pasta 'static' e referenciado no aquivo "index.html". O caminho é este: src → main → resources → static → style.css



O meu resultado foi esse, criei um formulário com campo para Nome e CPF.

Para visualizar na porta localhost:8080 temos que **'startar'** o Java de novo. Toda alteração que você tiver que fazer no arquivo HTML, terá que reiniciar o Java. Você pode fazer isso fechando o terminal existente e abrindo de novo, e assim colocar o comando (./mvnw springboot:run.)

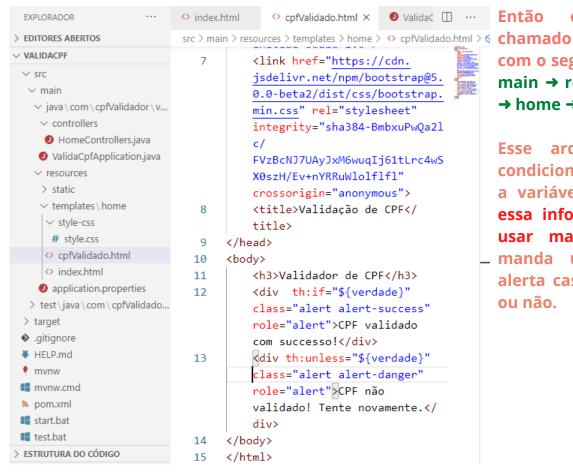
PS: pode dar erro se você tiver com 2 terminais abertos, olhe antes de 'startar'!

Você pode visualizar o meu código do arquivo HTML neste link: https://github.com/VeronicaBertozzi/Criando-um-Validador-de-CPF-em-Java/tree/main/validaCPF/src/main/resources/templates/home



Projeto Validador de CPF

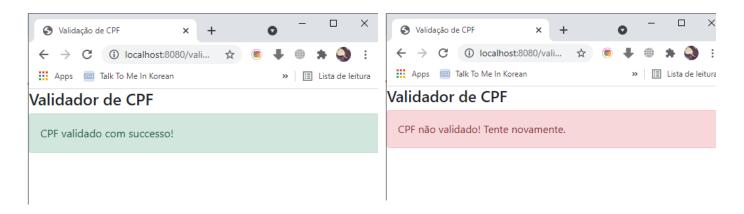
Agora temos que criar o validador de CPF em si. Vamos criar um arquivo HTML para mostrar pro usuário se o CPF dele foi validado ou não.



Então criei um arquivo chamado "cpfValidado.html" com o seguinte caminho: src → main → resources → tempates → home → cpfValidado.html.

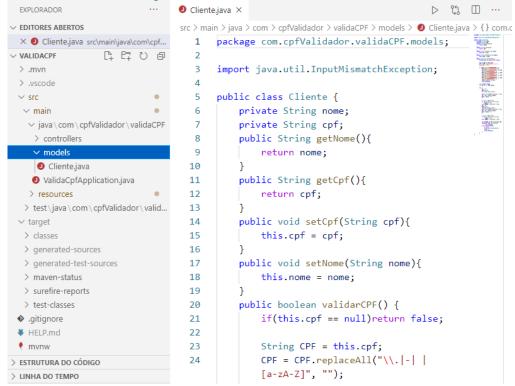
Esse arquivo contém uma condicional de Thymeleaf com a variável "verdade" (guarda essa informação, pois iremos usar mais pra frente) que manda uma mensagem de alerta caso o CPF foi validado ou não.

As mensagens ficarão assim:



Após isso vamos criar uma pasta chamada "models" dentro da pasta java. Dentro dessa pasta "models" vamos criar um arquivo Java chamado "Cliente.java" onde vamos colocar o código para validação do CPF. O caminho: src → main → java → com → cpfValidador → validaCPF →

models → Cliente.java



Nesse arquivo vamos criar classe Cliente e dentro vamos colocar atributos privados que são os 'private', eles carregam variáveis as 'nome' e 'cpf'. O Code vai sugerir para inserir o 'get' e o 'set' de cada variável, que tornam o acesso a essas variáveis públicas, então esses accessors mudam as propriedades privadas em públicas que vamos utilizar no TDD (Desenvolvimento orientado a teste).

Embaixo criei uma propriedade chamada 'validarCPF' com valor boolean(retorna true ou false) e dentro inseri um código de um validação de CPF que peguei no site do DevMedia, segue o site: https://www.devmedia.com.br/validando-o-cpf-em-uma-aplicacao-java/22097.

Após isso vamos voltar para o arquivo **HomeController.java** e adicionar mais um @**GetMapping** à classe @**Controller.**

```
⊳ ৸ 🗆 …
HomeControllers.iava ×
src > main > java > com > cpfValidador > validaCPF > controllers > ① HomeControllers.java > { } com.cpfValidac
  package com.cpfValidador.validaCPF.controllers;
       import com.cpfValidador.validaCPF.models.Cliente;
       import org.springframework.stereotype.Controller;
   6
       import org.springframework.ui.Model;
       import org.springframework.web.bind.annotation.GetMapping;
  8
  9
       @Controller
 10
       public class HomeControllers {
 11
 12
           @GetMapping("/")
 13
           public String index() {
               return "home/index";
 14
 15
 16
 17
           @GetMapping("/validaCpf")
           public String validaCpf(Cliente cliente, Model model){
 18
 19
               boolean verdade = cliente.validarCPF();
               model.addAttribute("verdade", verdade);
 20
 21
               return "home/cpfValidado";
 22
 23
       }
```

Para podermos usar o arquivo Cliente no arquivo do HomeController.java e permitir que o @GetMapping acesse o que foi digitado pelo usuário que são propriedades definidas no arquivo Cliente, no caso cpf e nome, devemos importar o caminho da pasta Cliente.

O @GetMapping vai pegar essas informações novamente do arquivo 'index.html' pois indiquei que o elemento action da tag form seria enviado para "/validaCpf".

Dentro de @GetMapping criei uma propriedade String validaCpf que tem como parâmetro Cliente e Model.

```
17  @GetMapping("/validaCpf")
18  public String validaCpf(Cliente cliente, Model model){
19  boolean verdade = cliente.validarCPF();
20  model.addAttribute("verdade", verdade);
21  return "home/cpfValidado";
22  }
```

- Colocando como parâmetro o arquivo Cliente, vai ser criado uma instancia de cliente(seria o cliente sem a letra maiúscula) que já vai 'setar' os parâmetros passados pelo usuário no navegador.
- Depois na linha 19 criei a variável verdade do tipo boolean que vai receber o valor da instancia cliente com a função 'validarCPF'(que é a propriedade que contem o código de validação), ou seja, a variável verdade pega o objeto 'validarCPF' já preenchido pela instancia cliente e avalia se é verdade ou não.
- Na linha 18 fiz uma injeção de dependência incluindo o Model como parâmetro de validaCpf, que vai passar os dados que estão no Controller para a View do servidor. Com isso consigo adicionar na linha 20 o model.addAttribute que vai atribuir ao atributo "verdade" a variável verdade.
- Desse modo o thymeleaf vai encontrar no template cpfValidado.html a variável 'verdade' e vai retornar a mensagem de validação ou não do cpf dependendo da dinâmica do código de validação dentro da função 'validarCpf' no arquivo Cliente.java.

PS: não esqueça de adicionar a dependência do Model.

Quando terminar as alterações o site criado vai conseguir pegar as informações passadas, pelo o usuário, processa-las e retornar a mensagem de alerta caso o CPF foi validado ou não.



Test Driven Development - TDD

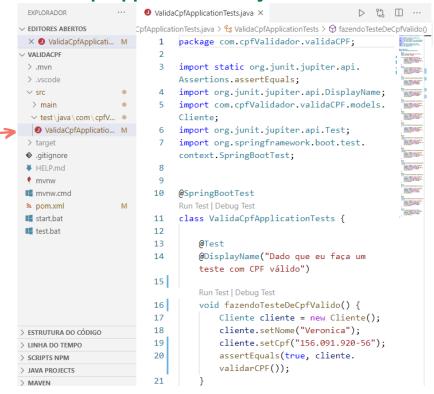
Dentro da pasta test existe um arquivo Java que permite realizar teste de validação e verificação. o arquivo vem com esse formato:

```
package com.cpfValidador.validaCPF;
2
3
    import org.junit.jupiter.api.Test;
    import org.springframework.boot.test.
    context.SpringBootTest;
5
    @SpringBootTest
6
    class ValidaCpfApplicationTests {
7
8
9
        @Test
0
        void contextLoads() {
1
2
3
4
```

Neste arquivo vamos construir casos de teste, que vamos colocar situações hipotéticas, com valor de entrada, restrição para sua execução e um resultado ou comportamento que esperamos dentro do @Test.

O caminho do arquivo: src → test → java → com → cpfValidador → validaCPF →

ValidarCpfApplicationTests.java



PS: Não se esqueça de fazer a importação dos arquivos utilizados antes do @SpringBootTest

Então dentro do @Test coloquei o @DisplayName que fornece um nome personalizado para a classe de teste. Inseri uma função fazendoTesteDeCpfValido(), e na linha 17 chamei a propriedade cliente do arquivo Cliente.java e criei uma nova instancia que vai clonar o objeto de Cliente para que possa validar o Cpf independente do modelo encapsulado no arquivo Cliente.

Na linha 18 vamos chamar o setNome da instancia cliente passando como parâmetro um possível nome(no caso coloquei o meu) e na linha 19 vamos chamar o setCpf passando como parâmetro um possível cpf valido(no caso coloquei um de um site de gerador de cpf).

O assertEquals serve para validar o teste então ele vem assim: assertEquals(resultadoEsperado, resultadoObtido);

No caso anterior espero que o teste seja um sucesso quando um usuário colocar um CPF válido e o valor retornado seja true.

No campo resultadoObtido chamei a função cliente.validarCPF() para processar os parâmetros que passei no setName e setCpf e avaliar se o CPF é de fato válido ou não.

```
23
          @Test
                                                        B. 10.765-
          Run Test | Debug Test
                                                        B 200-
          void fazendoTesteDeCpfInvalido() {
24
                                                        LINE.
25
              Cliente cliente = new Cliente();
              cliente.setNome("Veronica");
                                                       R. TAK
26
                                                       550
              cliente.setCpf("156.091.920-57");
27
              assertEquals(false, cliente.
28
              validarCPF());
29
```

Nesse outro caso espero que o teste seja um sucesso quando um usuário colocar um CPF invalido e o valor retornado seja false.

Então, para realizar o teste em si depois de escrever os casos de teste, devemos escrever no terminal o comando: (./mvnw test) e pressionar o enter, assim ele vai rodar o teste:

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\Users\Rita\Documents\MeuProjetos\Criando-um-Validador-de-CPF-em-Java\validaCPF> ./mvnw test
```

E se todos os testes funcionarem, essa é a mensagem que vai aparecer:

Caso queira ver todos os casos de testes que realizei acesse: https://github.com/VeronicaB ertozzi/Criando-um-Validador-de-CPF-em-Java/blob/main/validaCPF/src/test/java/com/cpfValidador/v alidaCPF/ValidaCpfApplication

Tests.java