

Guia De Desenvolvimento
Spring MVC
Thymeleaf
Spring Boot
Git
2023
Parte 3

Sumário

Introdução.....	3
Montando o ambiente	3
Possíveis problemas ao montar o ambiente.....	9
1 – Não carregou as dependências do projeto	9
Causa 1: Você, no seu workspace, não importou o arquivo config.epf	9
Solução 1: Importar o arquivo config.epf	9
Causa 2: Concorrência ao acessar o repositório corporativo local.....	11
Solução Causa 2: Pedir para o Maven tentar baixar as dependências explicitamente	11
2 – Erro no arquivo pom.xml	12
Causa: Repositório corporativo local não tem determinada versão de uma dependência	12
Solução: Tentar alterar a versão da biblioteca para uma versão mais antiga	13
Parte 3.....	14
Sobre Listas	14
Listando os produtos.....	14
Notação Thymeleaf.....	16
Executando <i>redirect</i>	19
Apresentando uma mensagem de sucesso no <i>redirect</i>	21

Introdução

Neste documento teremos o passo-a-passo para a construção de uma aplicação web utilizando as ferramentas Spring MVC, Thymeleaf, Spring Boot e Git.

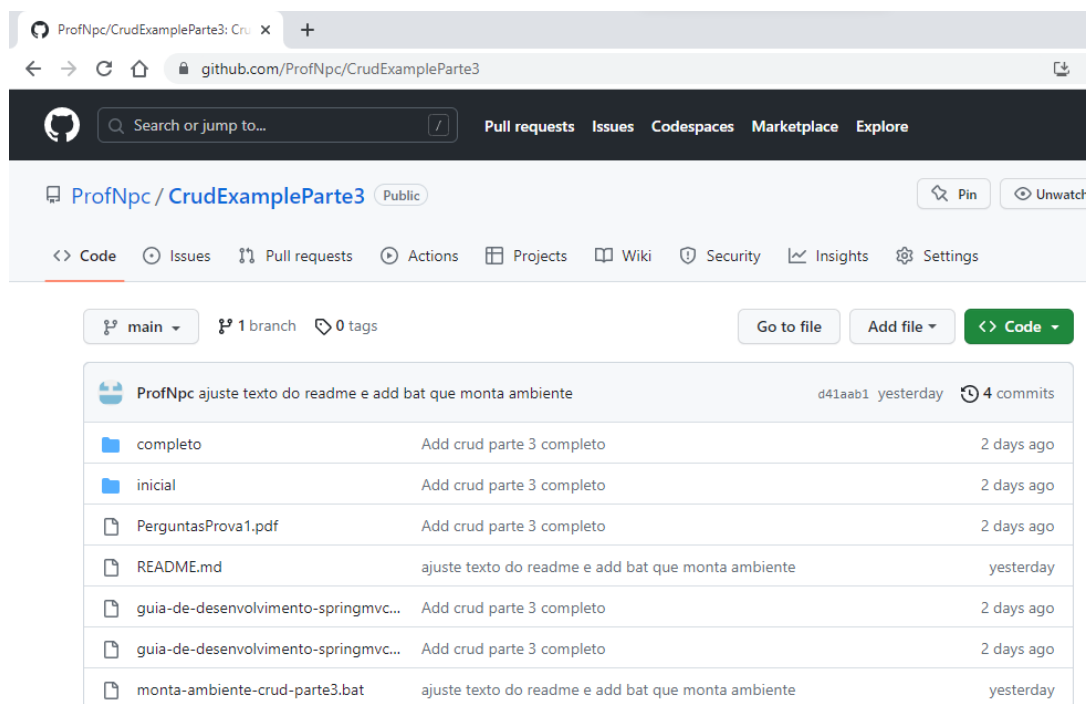
Este guia se destina a dar suporte as aulas de LIP1 do segundo módulo do curso de informática noturno do ITB Belval, portanto, algumas explicações mais detalhadas serão omitidas pois serão apresentadas pelo professor durante a aula.

Esta é a terceira parte do guia e para acompanhá-la é necessário obter o projeto desenvolvido na parte 1 em <https://github.com/ProfNpc/CrudExampleParte3/tree/main/inicial>.

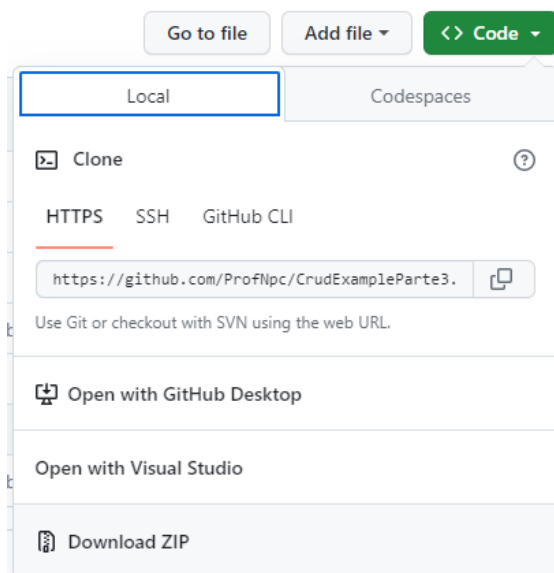
Montando o ambiente

Caso você não tenha concluído as partes 1 e 2 deste guia, é possível montar o ambiente necessário para poder seguir este guia da seguinte maneira:

1 – Acesse a url <https://github.com/ProfNpc/CrudExampleParte3>



2 – Clique no botão “Code” >> “Download ZIP”



Observação: Em algumas máquinas da escola o “Download ZIP” não funciona no navegador Chrome.

Nesse caso, abra a página em outro navegador, Edge ou Firefox, e tente fazer o download novamente.

ProfNpc/CrudExampleParte3: Cru x +

github.com/ProfNpc/CrudExampleParte3

Search or jump to... Pull requests Issues Codespaces Marketplace Explore

ProfNpc / CrudExampleParte3 Public

Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main 1 branch 0 tags

Go to file Add file <> Code

Local Codespaces

Clone ?

HTTPS SSH GitHub CLI

https://github.com/ProfNpc/CrudExampleParte3.

Use Git or checkout with SVN using the web URL.

Open with GitHub Desktop

Open with Visual Studio

Download ZIP

ProfNpc ajuste texto do readme e add bat que monta ambiente

completo Add crud parte 3 completo

inicial Add crud parte 3 completo

PerguntasProva1.pdf Add crud parte 3 completo

README.md ajuste texto do readme e add k

guia-de-desenvolvimento-springmvc... Add crud parte 3 completo

guia-de-desenvolvimento-springmvc... Add crud parte 3 completo

monta-ambiente-crud-parte3.bat ajuste texto do readme e add k

https://github.com/ProfNpc/CrudExampleParte3/archive/refs/heads/main.zip

CrudExampleParte....zip

3 – Acesse a pasta “Downloads” e descompacte o arquivo *.zip baixado.

Downloads

Novo

Classifica

Documentos

Meu Drive

Músicas

Nome

Hoje

CrudExampleParte3-main.zip

Abrir

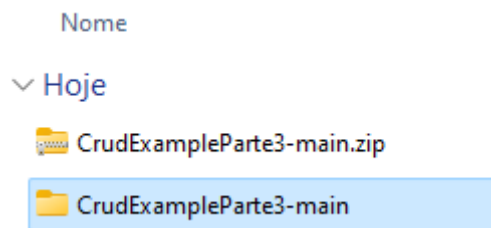
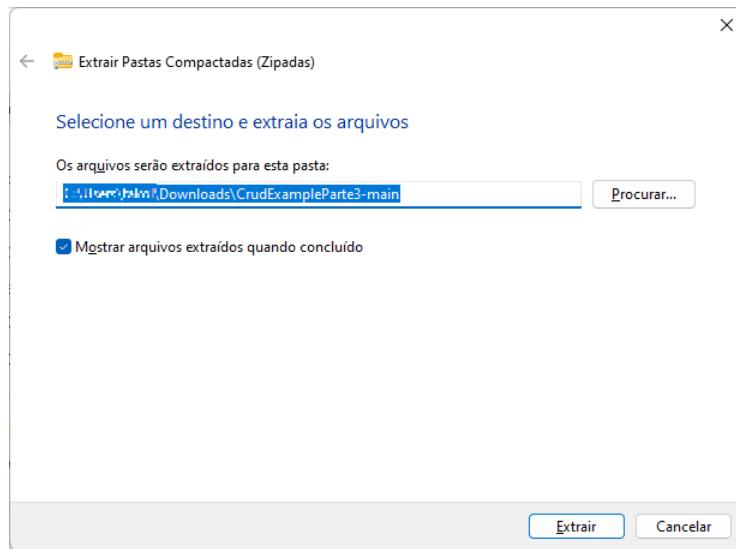
Abrir com

Abrir em nova guia

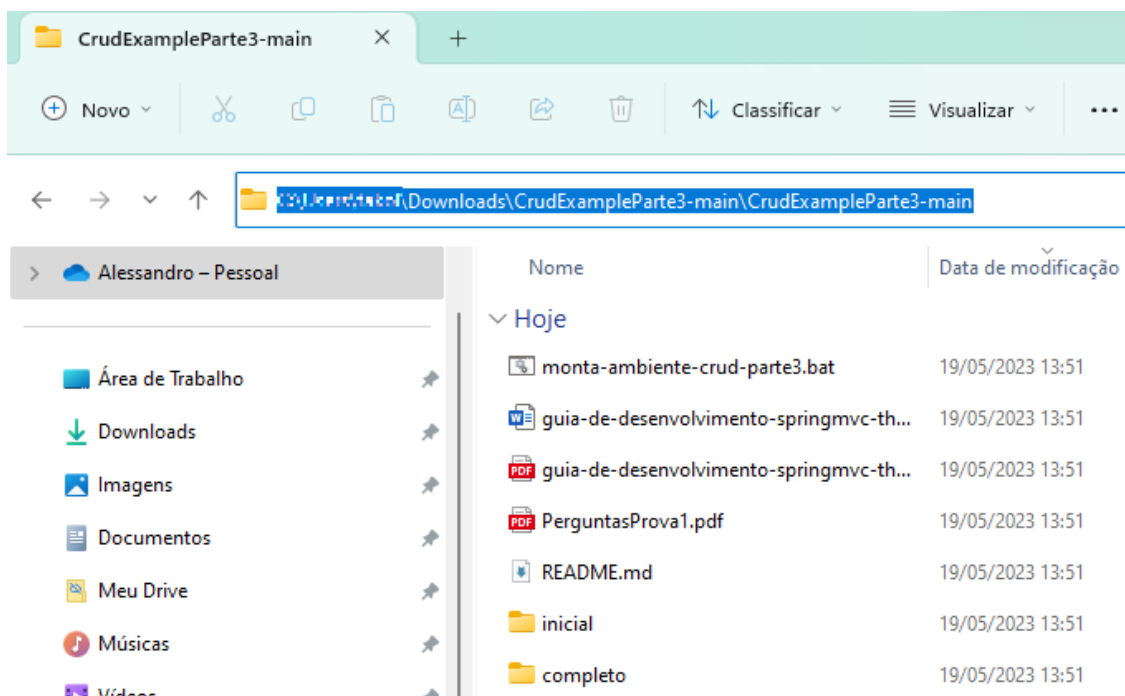
Abrir em nova janela

Extrair Tudo...

Fixar no Acesso rápido



4 – Acesse o diretório “..\Downloads\CrudExampleParte3-main\CrudExampleParte3-main” gerado

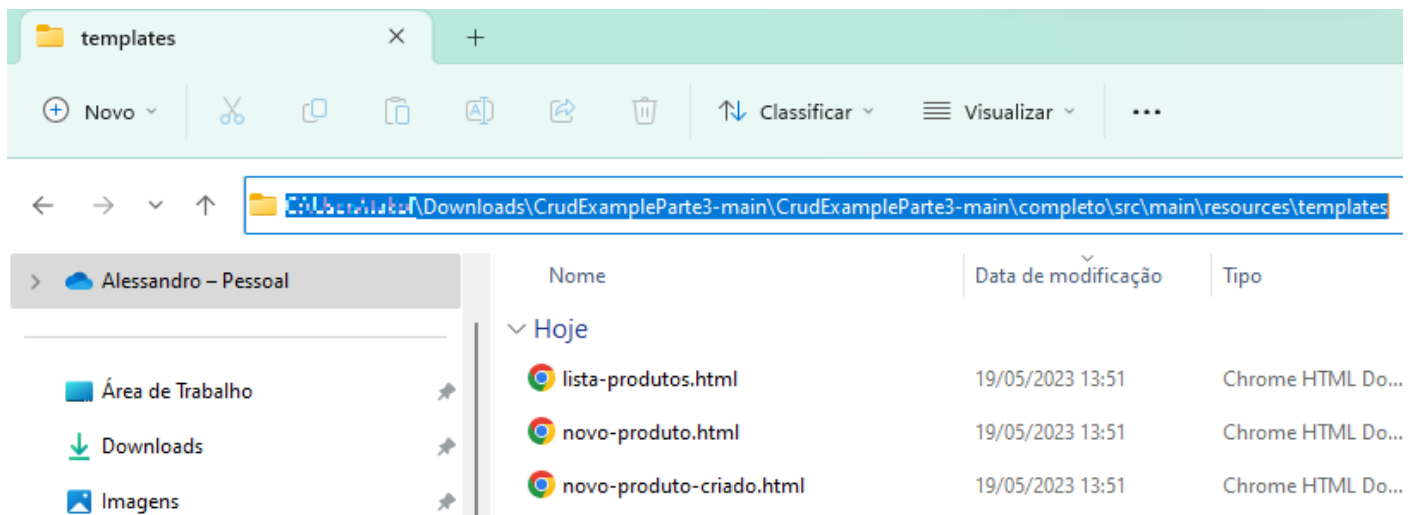


Neste diretório nós encontramos, entre outras coisas:

- Uma cópia deste guia
- Um diretório “inicial” com o projeto no ponto onde este guia se inicia
- Um diretório “completo” com o projeto no ponto onde este guia termina

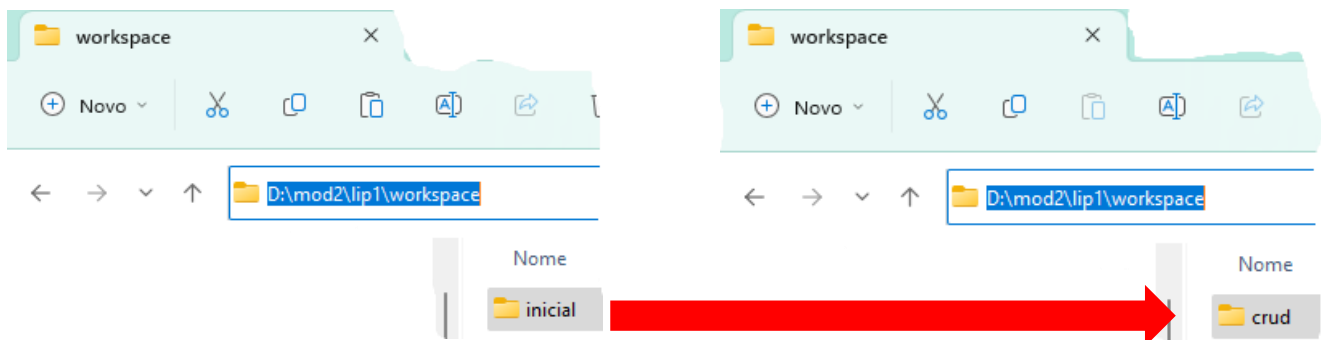
O diretório “inicial” deverá ser copiado para dentro de seu diretório workspace e deverá ser renomeado para “crud”.

Para que você não precise gastar tempo digitando o das páginas html, você pode acessar o diretório “completo”, mais especificamente “..\completo\src\main\resources\templates” e copiar os arquivos mencionados neste guia.

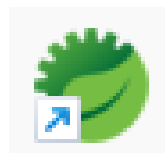


5 – Copie o diretório “..\Downloads\CrudExampleParte3-main\CrudExampleParte3-main\inicial” para dentro de seu workspace e renomeie o diretório “inicial” para “crud” depois de copiá-lo para dentro de workspace.

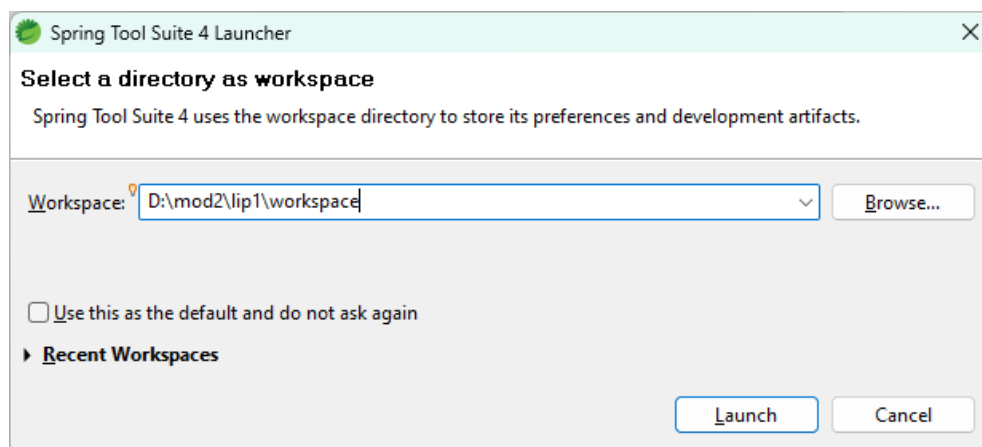
Atenção: caso você já tenha um diretório “crud” em seu workspace uma opção é renomear o diretório “inicial” para “crud3” por exemplo. Nesse caso, todas as vezes que o guia fizer menção ao diretório/projeto “crud” saiba que no seu caso o nome será esse outro.



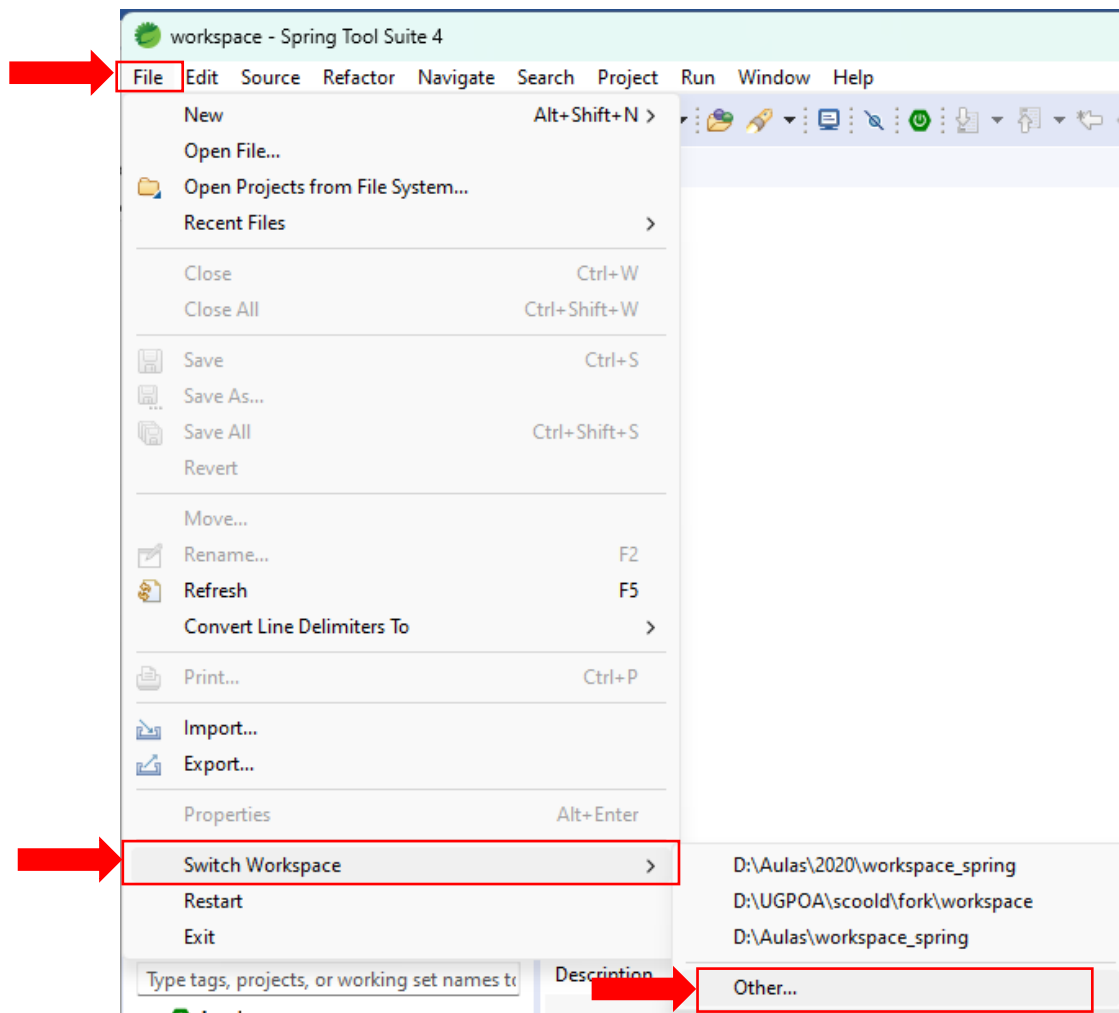
6 – Abra o Spring Tool Suite, deve haver um atalho como o abaixo na área de trabalho



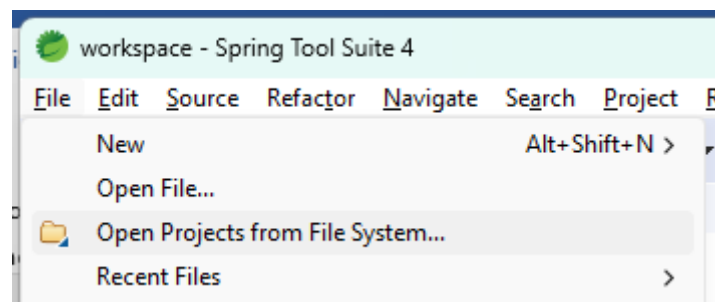
7 – Provavelmente aparecerá a tela onde informamos o caminho de nosso workspace, nesse caso, preencha o caminho do workspace que você vem usando.



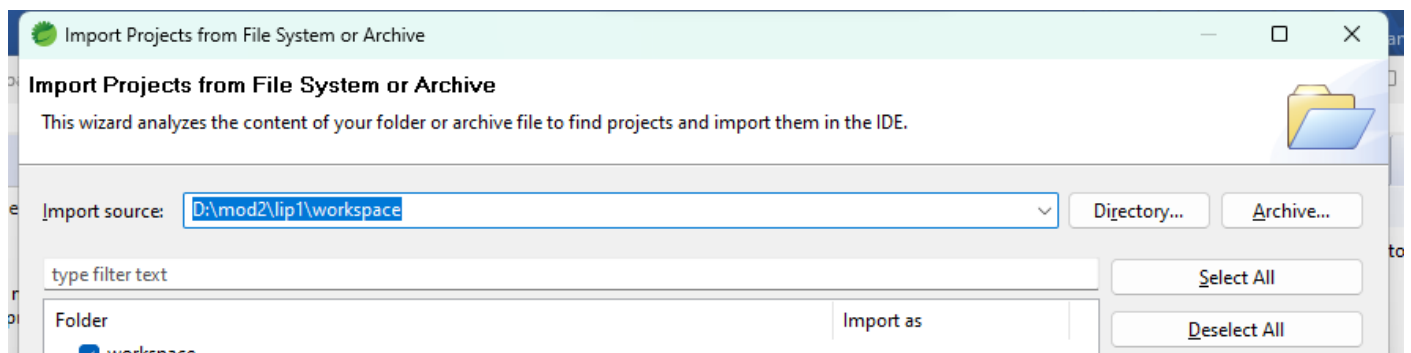
Atenção: Caso não apareça a tela mostrada acima e a ide abra diretamente, você deve se certificar que seu ambiente esteja apontando para o seu workspace. Para fazer isso, clique em **Menu File >> Switch Workspace >> Other** e na tela de seleção do workspace informe o caminho correto do seu workspace.



8 – Uma vez que o Spring Tool Suite, que chamaremos de STS daqui para frente, estive aberto e nos certificamos que ele está apontando para o nosso workspace utilizado costumeiramente, precisamos importar o projeto que está dentro do diretório **workspace** que é representado pelo diretório **crud**. Clique em no **Menu File >> Open Projects from File System...**

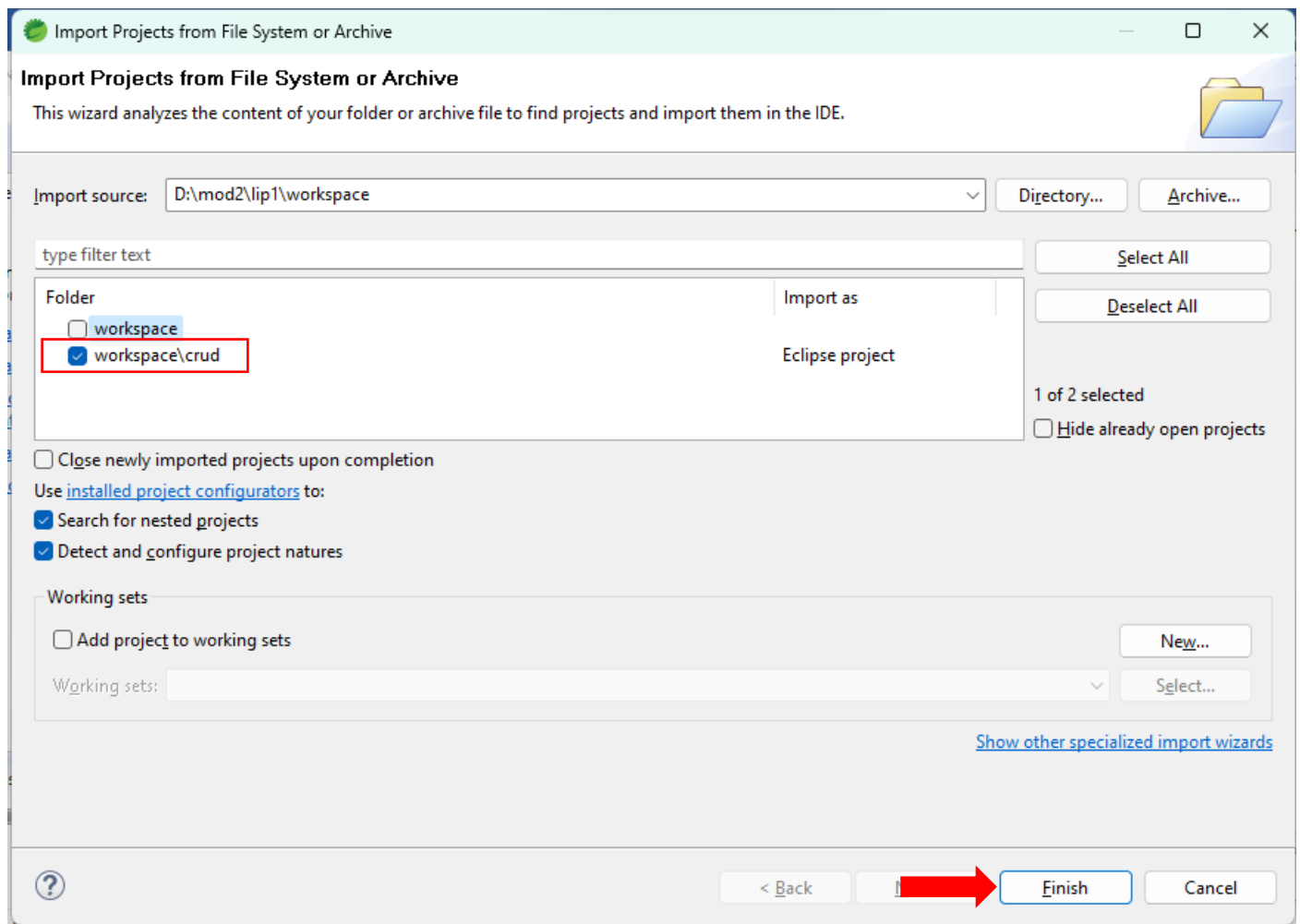


9 – Na tela seguinte, no campo “Input source” preencha com o caminho do seu workspace.

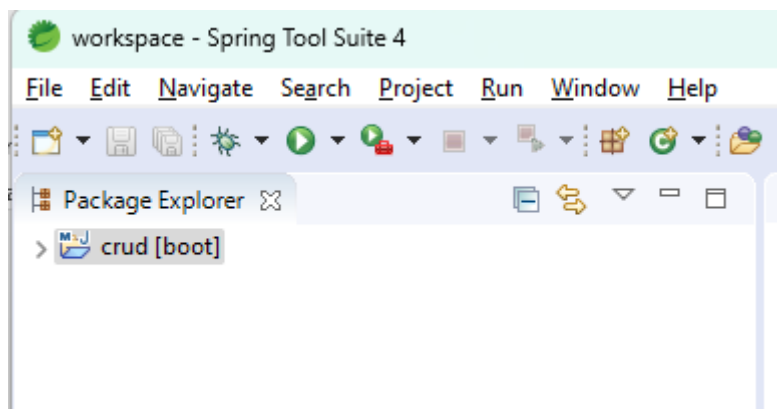


Assim que você terminar de digitar, o STS vai carregar algumas pastas.

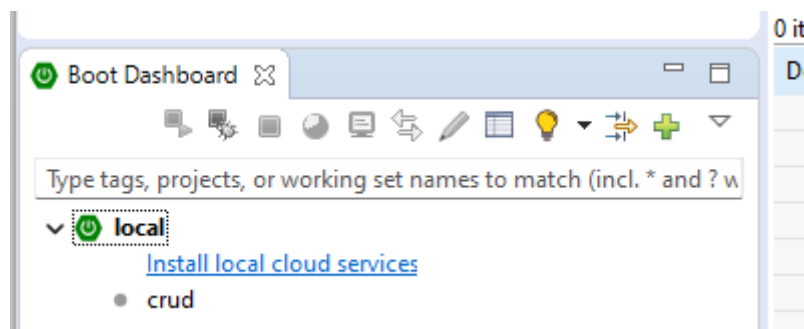
10 – Deixe selecionada apenas a pasta “workspace\crud” e clique em “Finish”.

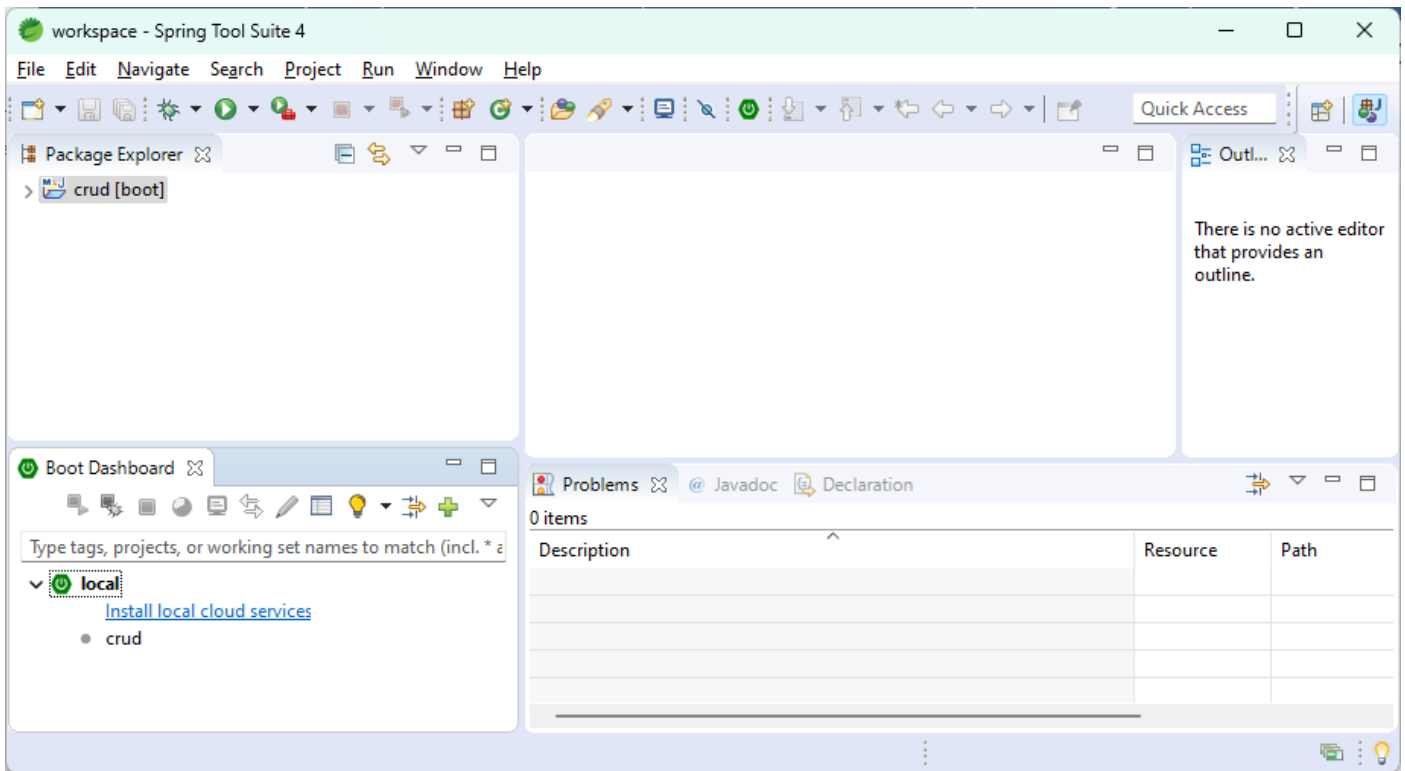


Nesse momento o projeto aparecerá na janela “Package Explorer”



E, em “Boot Dashboard”, nosso projeto deverá aparecer “dentro” de “local”





Possíveis problemas ao montar o ambiente

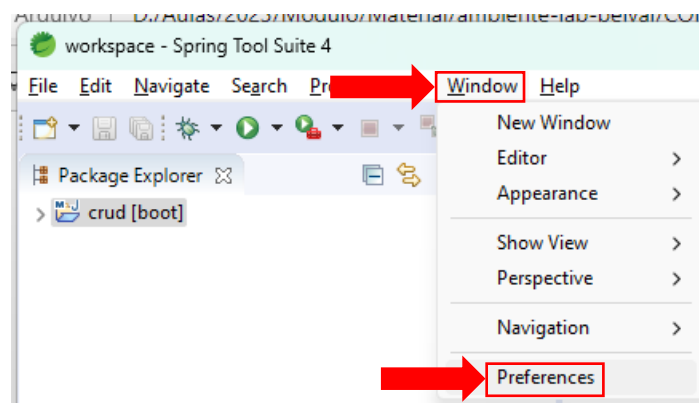
1 – Não carregou as dependências do projeto

Causa 1: Você, no seu workspace, não importou o arquivo [config.epf](#)

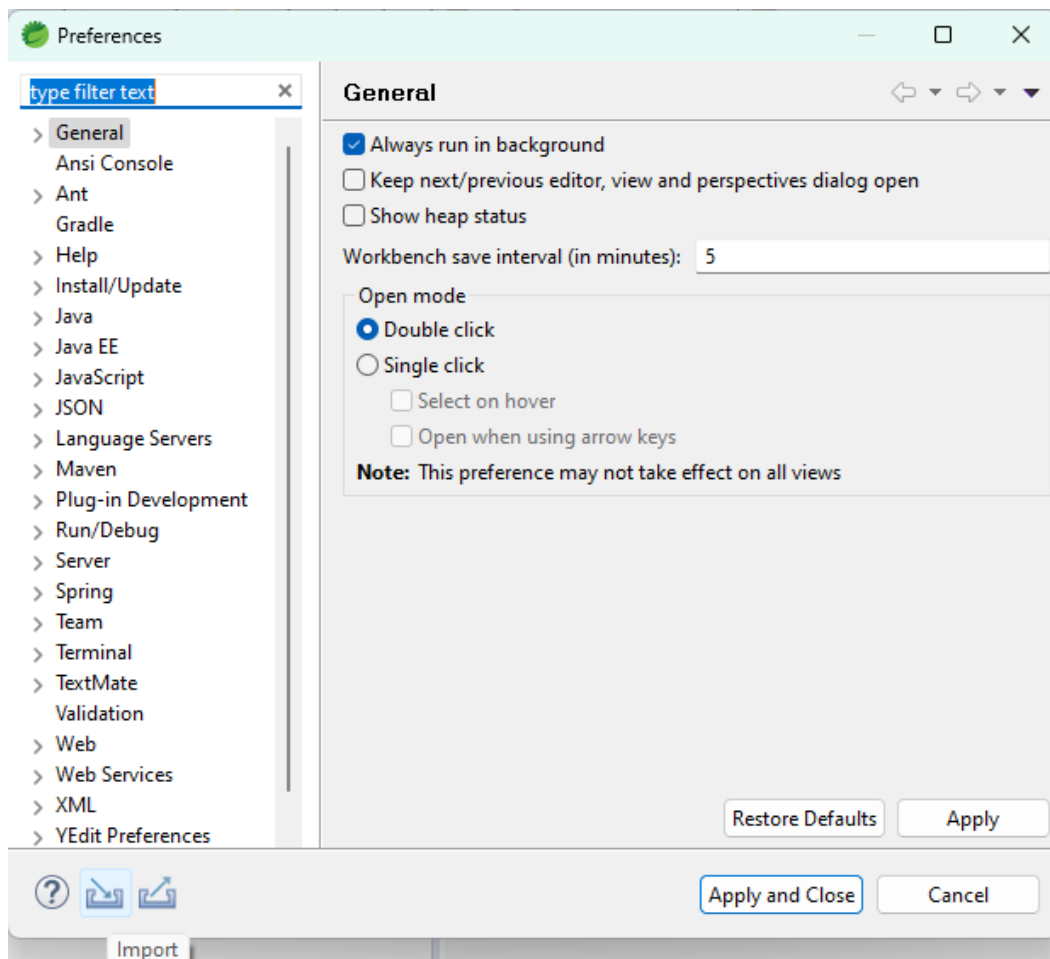
Uma causa desse problema pode ser porque as máquinas dos nossos laboratórios não conseguem sair diretamente para a internet para buscar as dependências, tarefa esta realizada pelo Maven – gerenciador de dependências. Elas precisam buscar essas dependências em nosso repositório corporativo, repositório este localizado em nossa rede local.

Solução 1: Importar o arquivo [config.epf](#)

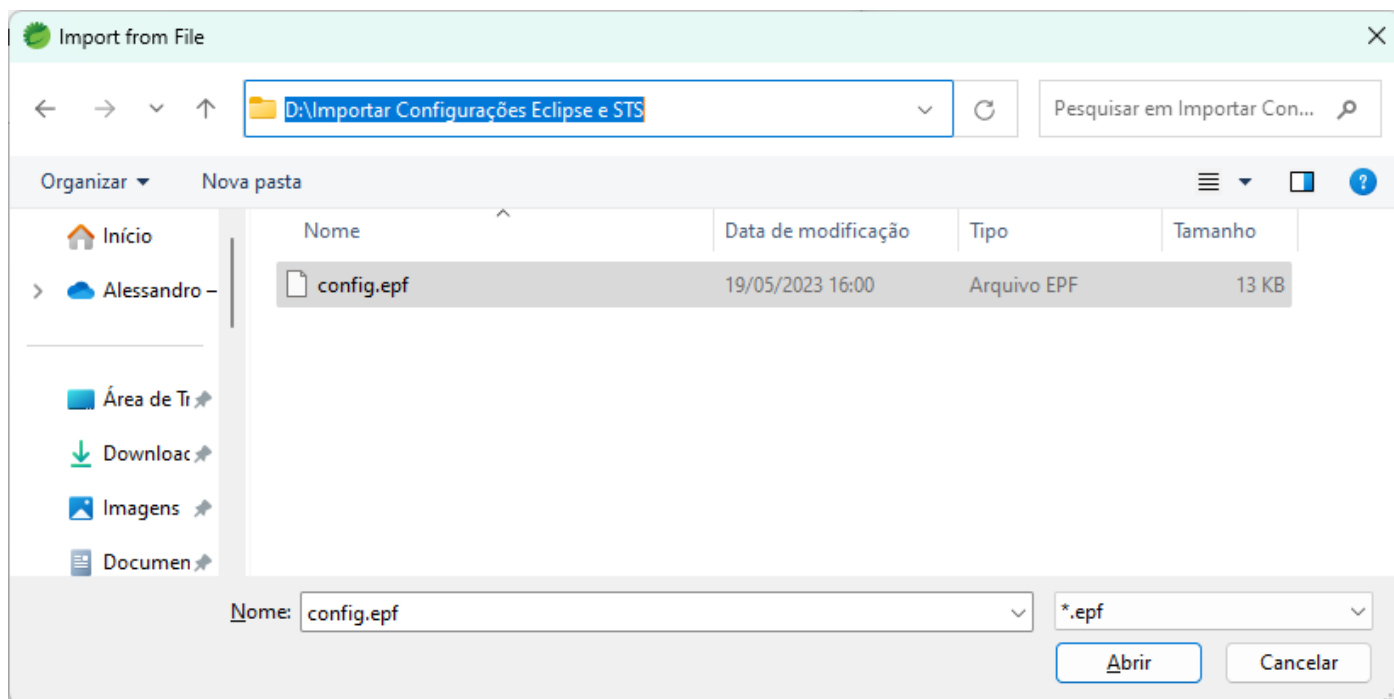
a) No STS, clique em **Window >> Preferences**



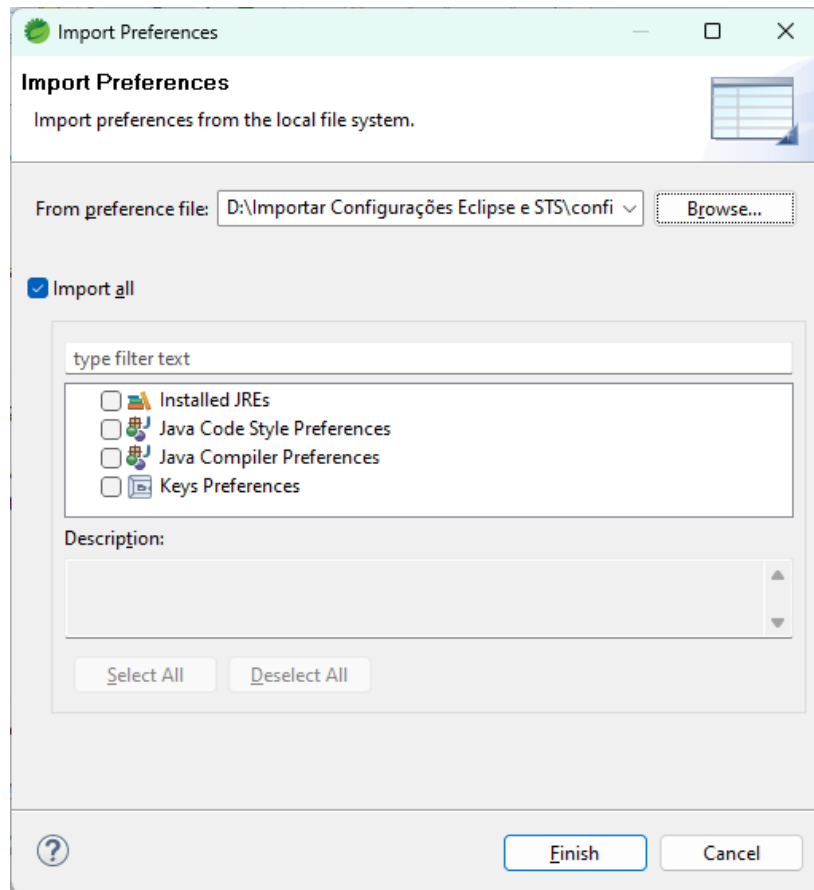
b) Na tela seguinte, clique no botão “**Import**” no canto inferior esquerdo da tela.



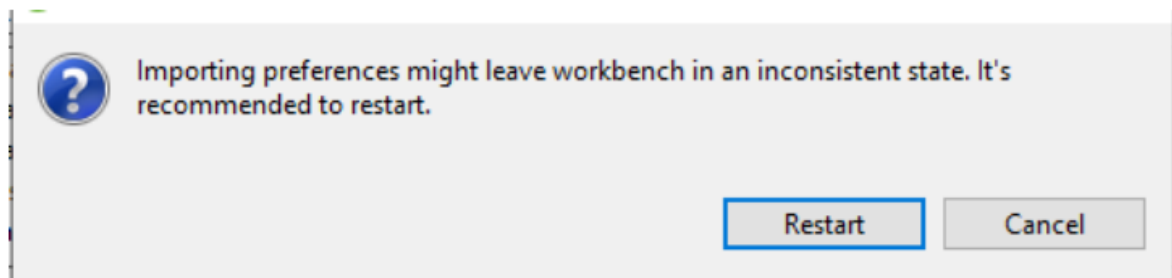
- c) Na tela seguinte, selecione o arquivo config.epf localizado no diretório **D:\Importar Configurações Eclipse e STS** presente em todas as máquinas dos laboratórios e clique em “Abrir”.



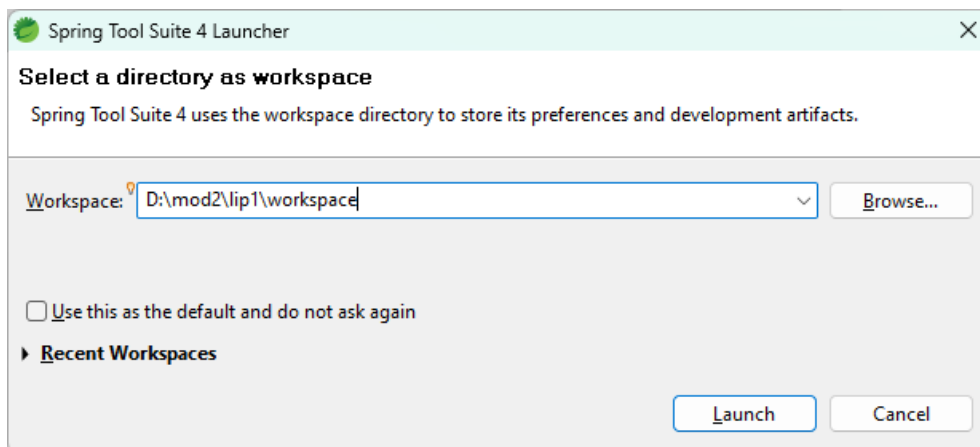
- d) Na tela seguinte, deixe a opção “Import all” marcada e clique em “Finish”.



e) Clique em “Restart”



f) Informe novamente o caminho do seu workspace e clique em “Launch”.

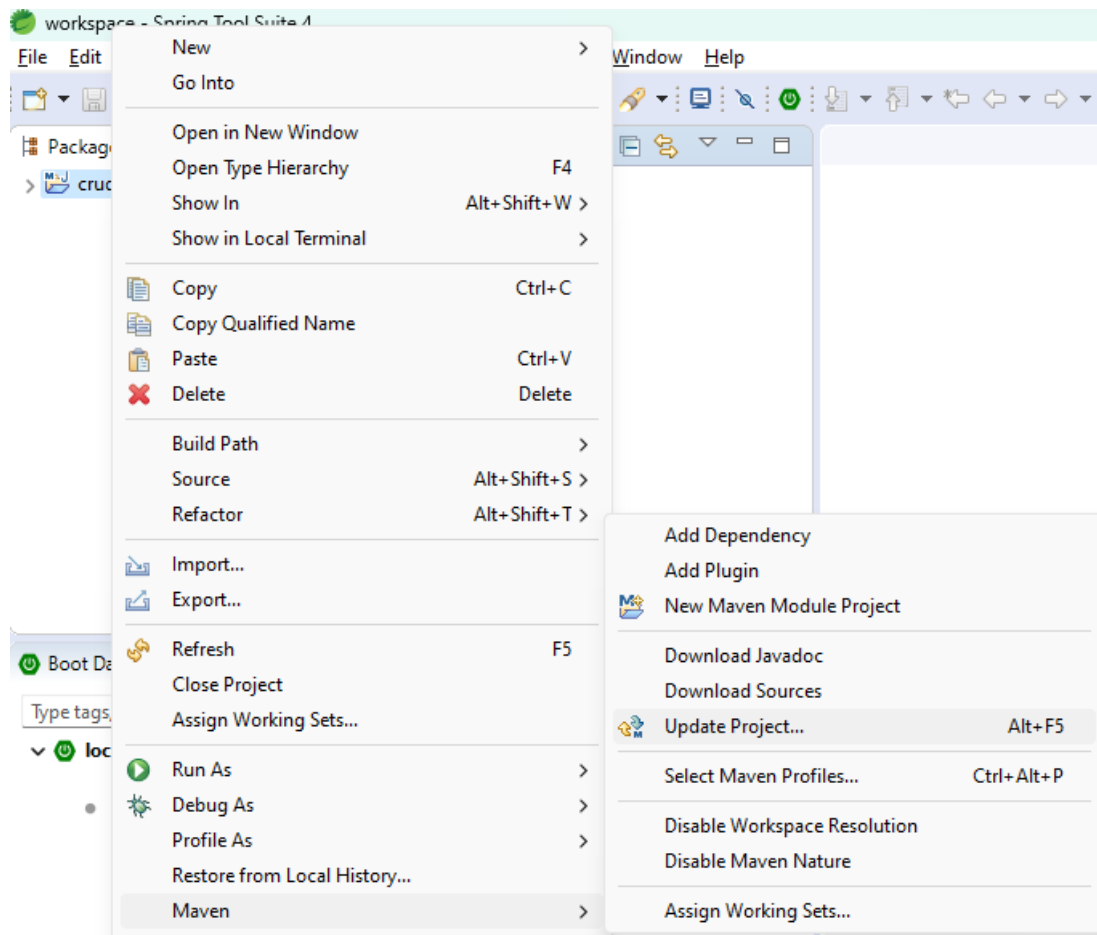


Causa 2: Concorrência ao acessar o repositório corporativo local

Outra causa para não conseguir baixar as dependências do projeto é que no momento que você tentou fazer isso outras pessoas estavam tentando fazer a mesma coisa e isso provocou um gargalo no repositório corporativo local.

Solução Causa 2: Pedir para o Maven tentar baixar as dependências explicitamente

a) Clique na **raiz do projeto** com o botão direito >> **Maven** >> **Update Project...**

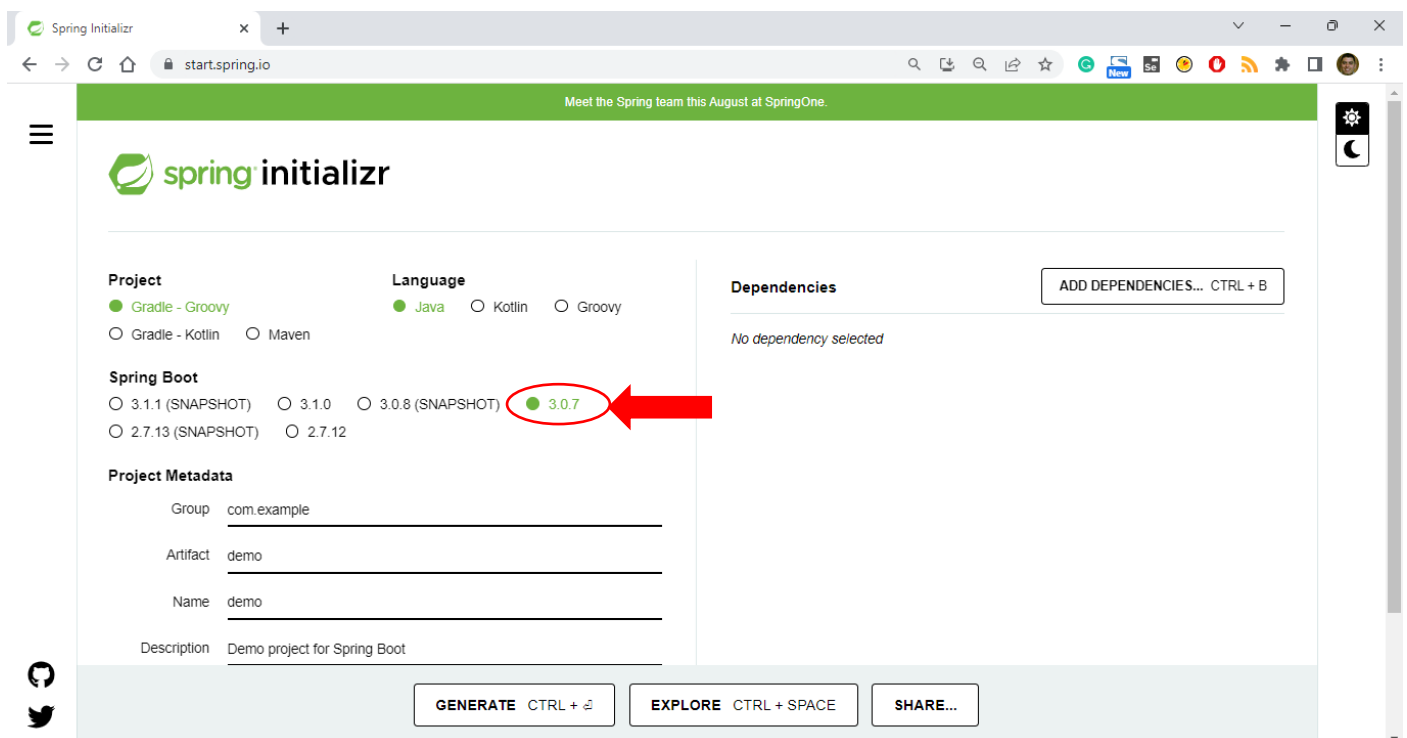


2 – Erro no arquivo pom.xml

Causa: Repositório corporativo local não tem determinada versão de uma dependência

Isso ocorre quando, em nosso arquivo pom.xml, possuímos a dependência de uma biblioteca que, por algum motivo, não está disponível em nosso repositório corporativo local.

Temos observado isso com relação à versão do spring boot. A última versão estável do spring boot é a versão 3.0.7.

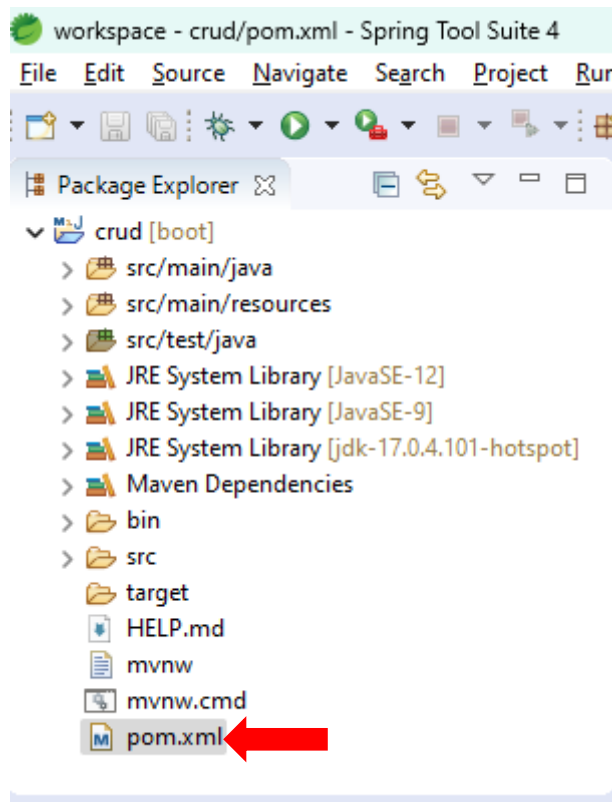


Por algum motivo, a última versão do spring boot disponível em nosso repositório corporativo local é a 3.0.5. Nesse caso o Maven pede uma versão e o repositório não consegue atender a solicitação.

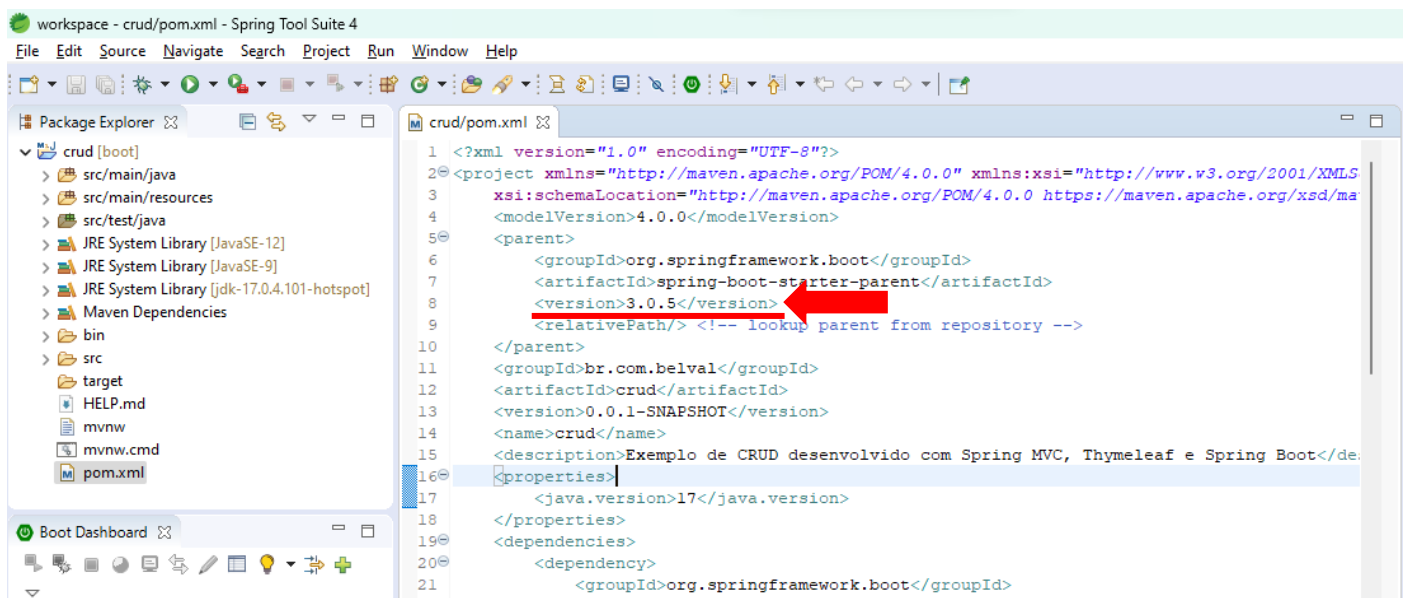
Solução: Tentar alterar a versão da biblioteca para uma versão mais antiga

No momento, uma possível solução viável é voltar a versão do spring boot para a versão 3.0.5. Como as principais bibliotecas que utilizamos não sofreram tantas mudanças entre essas versões, essa medida ainda é uma solução viável.

- a) Abra o arquivo pom.xml localizado na raiz do projeto



- b) Dentro do arquivo, procure pelo nó <parent>, normalmente abaixo do nó <modelVersion> e altere o conteúdo do sub-nó <version> para 3.0.5. No futuro, caso esse erro ocorra, talvez seja preciso testar com diferentes versões até encontrar a versão mais adequada.



- c) Talvez seja necessário executar clicar na **raiz do projeto com o botão direito >> Maven >> Update Project...** para que os efeitos sejam obtidos.

Parte 3

Nessa terceira parte de nosso guia continuaremos a implementar as ações do CRUD. No sentido de recuperar informações (**Recouver**), criaremos uma tela que nos permita ver a lista dos produtos inseridos. Para tanto, se faz necessário concluir a ação de criação (**Create**) de forma que o produto cujos dados foram preenchidos no formulário (desenvolvido na parte 2 desse guia) e enviados seja armazenado em algum lugar onde nossa aplicação possa acessar no momento que ela for solicitada a apresentar a lista dos produtos.

Numa primeira versão, armazenaremos os produtos em memória criando um atributo na classe **ProdutoController** do tipo **List<Produto>**.

Sobre Listas

Classes que implementam a interface **List** possuem a capacidade de armazenar conjuntos ordenados de objetos, isto é, o contrato da interface **List** pressupõem que haja uma ordem inerente aos objetos mantidos pela por um **List**. Essa característica se reflete no fato de que cada objeto armazenado possui um índice associado e pode ser recuperado através desse índice.

Listando os produtos

1 – Adicione o atributo **listaProdutos** do tipo **List<Produto>** à classe **ProdutoController** (Veja a seguir)

2 – Adicione o atributo **proxId** do tipo **int** à classe **ProdutoController**

```
@Controller
public class ProdutoController {

    private static List<Produto> listaProdutos = new ArrayList<Produto>();
    private static int proxId = 1;
```

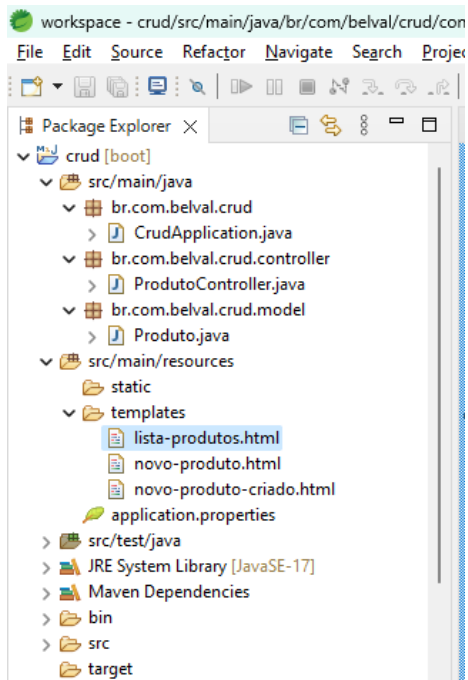
3 – Altere o método **novo(Produto novo)** e adicione as linhas abaixo

```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
    modelAndView.addObject("novoProduto", produto);
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}
```

4 – Adicione o método **list()** e associe ele à url **"/produto/list"**

```
@GetMapping("/produto/list")
public ModelAndView list() {
    ModelAndView modelAndView = new ModelAndView("lista-produtos");
    modelAndView.addObject("produtos", listaProdutos);
    return modelAndView;
}
```

5 – Adicione o arquivo **lista-produtos.html** aos templates



```
<!DOCTYPE html>
<html>
  <head>
    <title>Lista Produtos</title>
    <meta charset="utf-8">
  </head>
  <body>
    <script th:inline="javascript">
      /**/
        var msg = /*[[${msg}]]*/ null;
        if(msg != null) {
          alert(msg);
        }
      /*]]&gt;*/
    &lt;/script&gt;
    &lt;h1&gt;Lista Produtos&lt;/h1&gt;

    &lt;table border="1"&gt;
      &lt;tr&gt;
        &lt;th&gt;ID&lt;/th&gt;
        &lt;th&gt;Nome&lt;/th&gt;
        &lt;th&gt;&lt;/th&gt;
        &lt;th&gt;&lt;/th&gt;
      &lt;/tr&gt;
      &lt;tr th:each="prod : ${produtos}"&gt;
        &lt;td&gt;
          &lt;span th:text="${prod.id}"&gt;&lt;/span&gt;
        &lt;/td&gt;
        &lt;td&gt;
          &lt;span th:text="${prod.nome}"&gt;&lt;/span&gt;
        &lt;/td&gt;
        &lt;td&gt;
          &lt;a th:href="@{/produto/{id} (id=${prod.id}) }"&gt;Detalhe&lt;/a&gt;
        &lt;/td&gt;
        &lt;td&gt;
          &lt;a th:href="@{/produto/{id}/edit (id=${prod.id}) }"&gt;Editar&lt;/a&gt;
        &lt;/td&gt;
      &lt;/tr&gt;
    &lt;/table&gt;
    &lt;a th:href="@{/produto/novo}"&gt;Novo&lt;/a&gt;
  &lt;/body&gt;
&lt;/html&gt;</pre></div>
```

Notação Thymeleaf

Observe na segunda **tag <tr>** de **lista-produtos.html** o atributo **th:each**

```
<tr th:each="prod : ${produtos}">
```

Essa é uma anotação do thymeleaf que vai produzir um nó **<tr>** (uma linha da tabela) “para cada” elemento presente na coleção “**\${produtos}**” (*each* do inglês quer dizer “cada um”).

Observe que, cada elemento da coleção será armazenado em uma variável chamada “**prod**” e é através dessa variável que acessamos os atributos do elemento da **<tr>** “atual”.

Por exemplo, na primeira coluna (tag **<td>**) de cada **<tr>** apresentamos o conteúdo do atributo “**id**” do **Produto “atual”** que está armazenado em “**prod**” dentro de uma tag ****.

```
<td>
  <span th:text="${prod.id}"></span>
</td>
```

O mesmo acontece na segunda coluna (tag **<td>**) da tabela que é preenchida com o valor do atributo “**nome**” do **Produto “corrente”**.

```
<td>
  <span th:text="${prod.nome}"></span>
</td>
```

Já na terceira coluna temos uma situação diferente, ela é preenchida com um *hiperlink* marcado pela tag **<a>**, cujo texto apresentado para o usuário vai ser sempre o mesmo, “**Detalhe**”, mas cuja *url*, presente no atributo **th:href**, a ser requisitada quando o hiperlink for clicado será diferente para cada uma das linhas, pois essa *url* será “montada” utilizando o valor do atributo **id** do Produto daquela linha.

```
<td>
  <a th:href="@{/produto/{id} (id=${prod.id}) }">Detalhe</a>
</td>
```

Observe que a *url* é a parte

```
/produto/{id}
```

Onde “**{id}**” é uma variável definida na mesma linha, logo depois, a partir do valor do atributo “**id**” do objeto presente em “**prod**”.

```
(id=${prod.id})
```

Caso o produto possua o valor “**1**” para “**id**” a *url* gerada será “**/produto/1**”, caso seja “**2**”, a *url* muda para “**/produto/2**”. Essas *url*’s serão usadas para vermos os detalhes de cada produto.

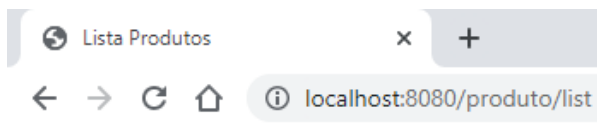
Na última coluna da tabela, ocorre algo semelhante

```
<td>
  <a th:href="@{/produto/{id}/edit (id=${prod.id}) }">Editar</a>
</td>
```

Mas as *url*’s produzidas seguirão a estrutura de “**/produto/1/edit**” e servirão para acessarmos uma página com um formulário parecido com o utilizado para inserir um novo produto com o diferencial de que os campos já virão preenchidos para podermos editar seus valores quando isso for necessário.

Por fim temos, depois da tabela, um hiperlink que levará para o formulário de criação de novos produtos gerado por

```
<a th:href="@{/produto/novo}">Novo</a>
```

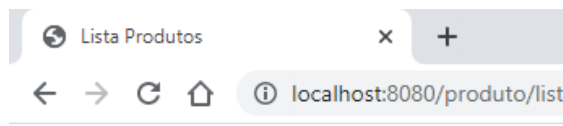



Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar
2	Manteiga	Detalhe	Editar

[Novo](#)

localhost:8080/produto/1

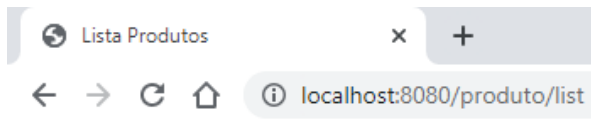


Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar
2	Manteiga	Detalhe	Editar

[Novo](#)

localhost:8080/produto/1/edit

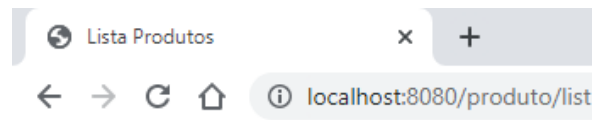


Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar
2	Manteiga	Detalhe	Editar

[Novo](#)

localhost:8080/produto/2

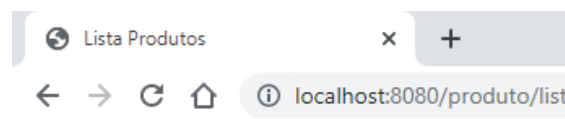


Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar
2	Manteiga	Detalhe	Editar

[Novo](#)

localhost:8080/produto/2/edit



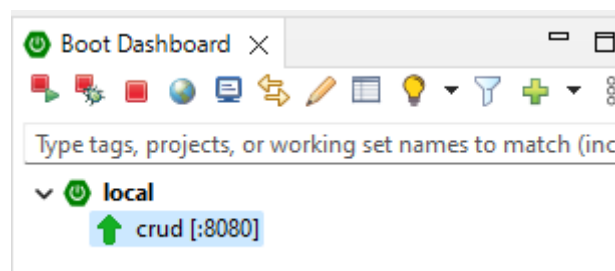
Lista Produtos

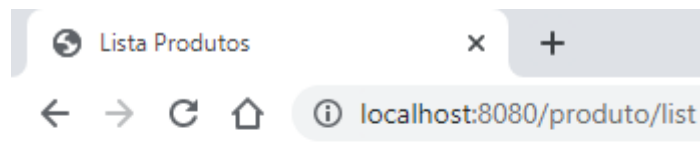
ID	Nome		
1	Bala	Detalhe	Editar
2	Manteiga	Detalhe	Editar

[Novo](#)

localhost:8080/produto/novo

6 – Execute a aplicação e acesse a url <http://localhost:8080/produto/list>

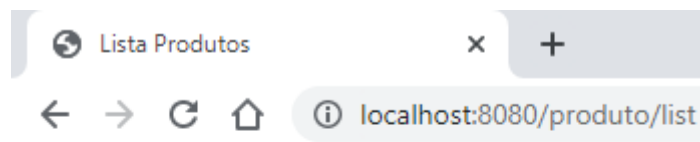




Lista Produtos

ID	Nome
	Novo

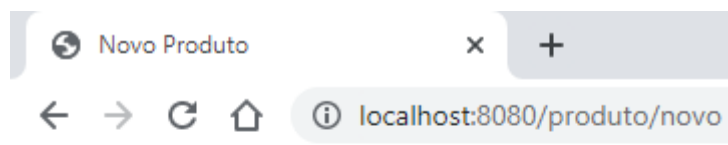
7 – Clique no link “Novo”



Lista Produtos

ID	Nome
	Novo

8 – Preencha os campos do formulário e clique em “Enviar”

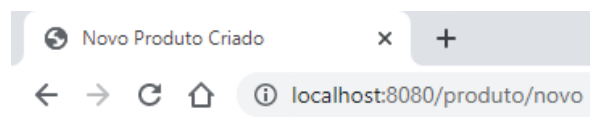


Novo Produto

Nome:

Descrição:

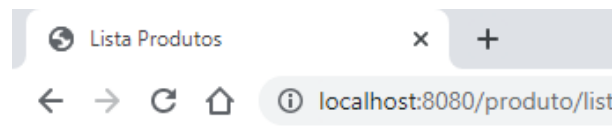
Preço:



Novo produto criado!

ID:	1
Nome:	Bala
Descrição:	Bala soft
Preço:	2.5

9 – Acesse novamente a url <http://localhost:8080/produto/list>



Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar

[Novo](#)

Executando *redirect*

Em certas circunstâncias, precisamos que o processamento de uma url dispare, as vezes de forma condicional, a execução de outro fluxo.

No nosso caso, para que a navegação de nossa aplicação seja mais fluida, ao concluir com sucesso a inserção de um novo produto, seria conveniente se apresentássemos na sequência a listagem dos produtos, que dá acesso às outras ações do CRUD.



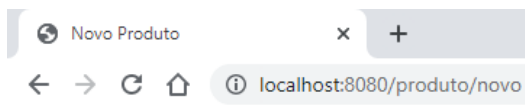
Nós poderíamos, simplesmente, alterar a página apresentada pelo método `novo(Produto produto)` para apresentar a lista ao final do processamento.

```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
    modelAndView.addObject("novoProduto", produto);
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}

@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("lista-produtos");
    modelAndView.addObject("produtos", listaProdutos);
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}
```

Atenção: Não faça essa alteração!!!

Se fizermos isso, ao cadastrarmos um novo produto, será apresentada a lista dos produtos incluindo o novo produto cadastrado.

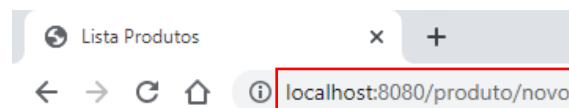


Novo Produto

Nome:

Descrição:

Preço:



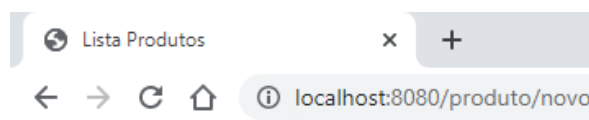
Lista Produtos

ID	Nome		
1	Bala	Detalhe	Editar

[Novo](#)

O problema dessa abordagem é que a url apresentada no browser enquanto é apresentada a lista dos produtos não é a url correspondente à lista dos produtos.

Se o usuário tentar atualizar a página nesse momento utilizando o atalho F5 ou mesmo clicando na barra de endereço e pressionando a tecla <ENTER> o que vai acontecer é que o último *submit* do formulário será reenviado provocando a tentativas de inserção dos mesmos dados.

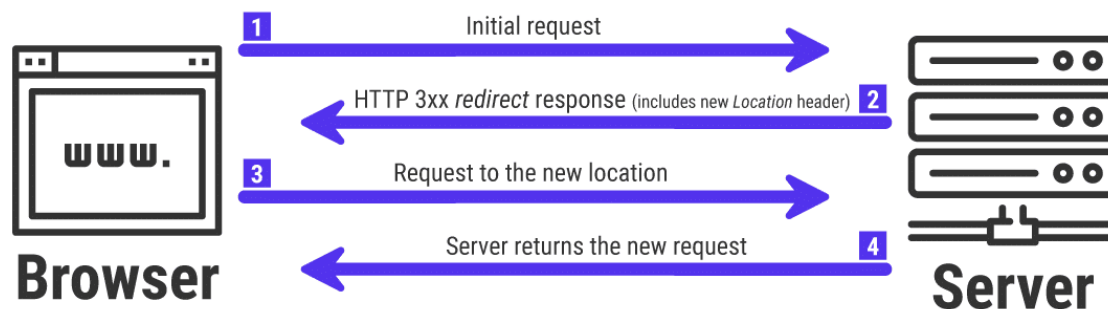


Lista Produtos

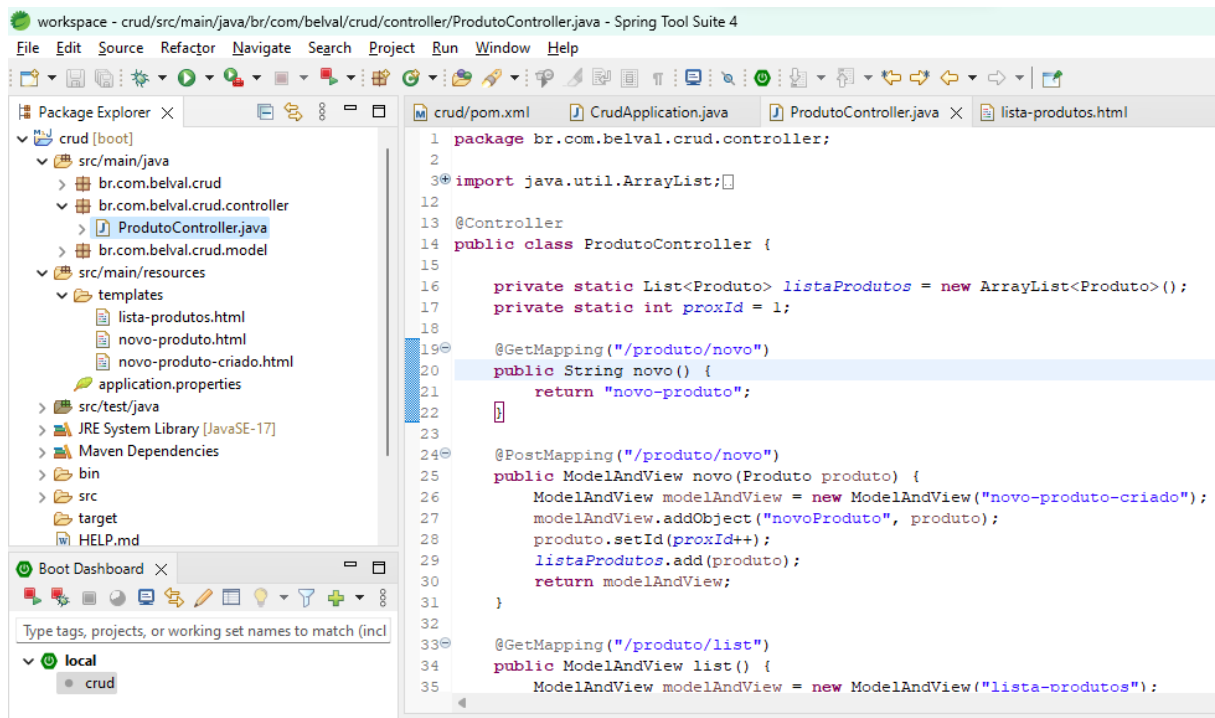
ID	Nome		
1	Bala	Detalhe	Editar
2	Bala	Detalhe	Editar
3	Bala	Detalhe	Editar
4	Bala	Detalhe	Editar
5	Bala	Detalhe	Editar

[Novo](#)

Para evitarmos esse comportamento, usaremos uma técnica chamada *redirecionamento de url*. Este é um recurso definido no próprio protocolo HTTP que define códigos de resposta especiais (3xx) para que a aplicação rodando no servidor HTTP possa indicar para o cliente (browser) que ele deve fazer uma nova requisição para uma url específica.



1 – Abra a classe ProdutoController



2 – No método **novo(Produto novo)**, anotado com **@PostMapping("/produto/novo")**, adicione **"redirect:/produto/list"** como o parâmetro passado no construtor de **ModelAndView**.

```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("novo-produto-criado");
    modelAndView.addObject("novoProduto", produto);
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}

@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("redirect:/produto/list");
    modelAndView.addObject("novoProduto", produto);
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}
```

3 – Como não serão apresentados os dados do produto recém inserido, podemos remover a linha

```
modelAndView.addObject("novoProduto", produto);
```

E ficaremos com o método alterado como abaixo

```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto) {
    ModelAndView modelAndView = new ModelAndView("redirect:/produto/list");
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}
```

Apresentando uma mensagem de sucesso no *redirect*

Para que o usuário ainda receba algum *feedback* de que a ação de inserção de produto foi executada com sucesso, seria interessante que chegasse à página de listagem de produtos a informação de que um produto acaba de ser

adicionado. Para isso, vincularemos ao *redirect* uma variável de sessão “*msg*” com um texto que informará o usuário esse fato. Faremos isso de forma que quando a tela de listagem de produtos for chamada diretamente essa variável “*msg*” esteja vazia e a listagem será apresentada normalmente.

1 – Adicione um parâmetro **RedirectAttributes redirectAttributes** ao método **novo(Produto produto)**

```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto, RedirectAttributes redirectAttributes) {
```

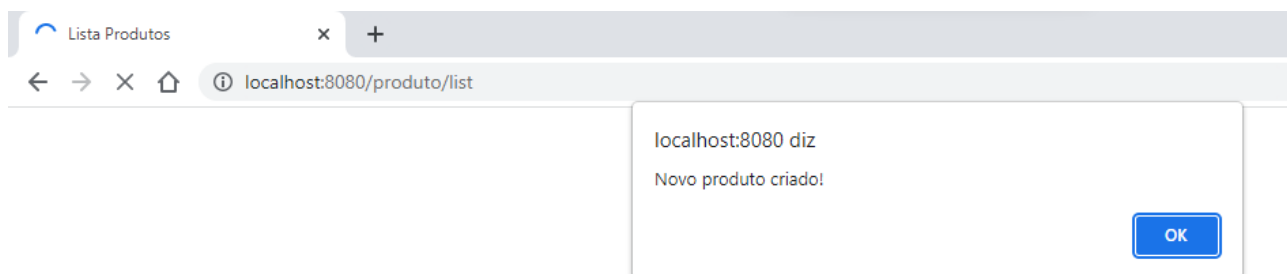
2 – Dentro do método, adicione um atributo “*msg*” com o valor “**Novo produto criado!**” através do método **addFlashAttribute()** do parâmetro **redirectAttributes**. Veja abaixo como fica o método alterado:

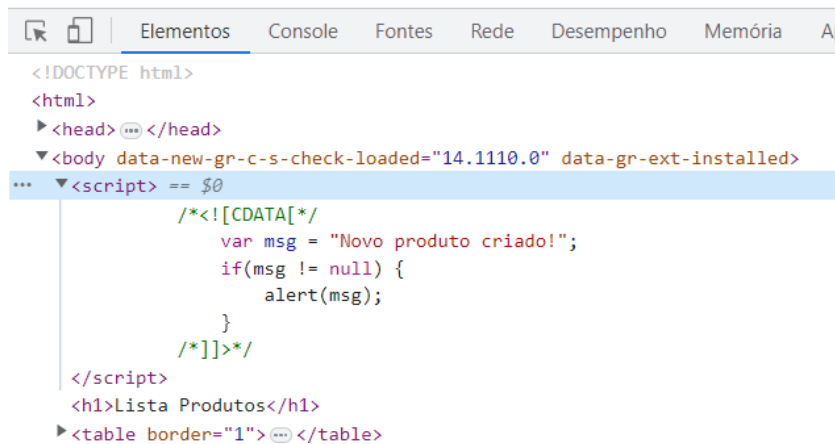
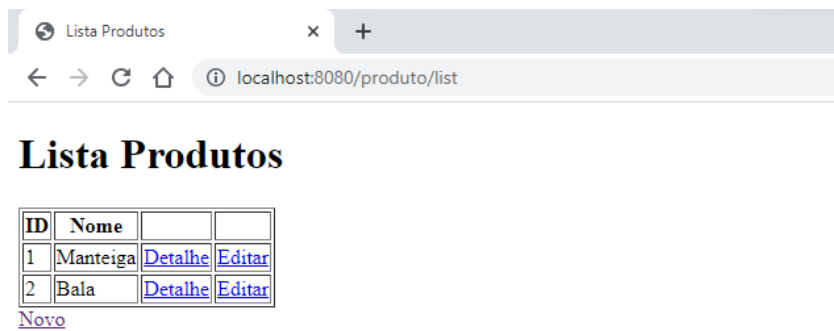
```
@PostMapping("/produto/novo")
public ModelAndView novo(Produto produto, RedirectAttributes redirectAttributes) {
    ModelAndView modelAndView = new ModelAndView("redirect:/produto/list");
    redirectAttributes.addFlashAttribute("msg", "Novo produto criado!");
    produto.setId(proxId++);
    listaProdutos.add(produto);
    return modelAndView;
}
```

Observação: o método **addFlashAttribute()** adiciona uma variável de sessão que só permanece na sessão até que o método para o qual foi feito o redirecionamento seja executado. Na sequência, a variável de sessão “*msg*” é removida da sessão e qualquer tentativa de recuperar seu valor retornará **null**.

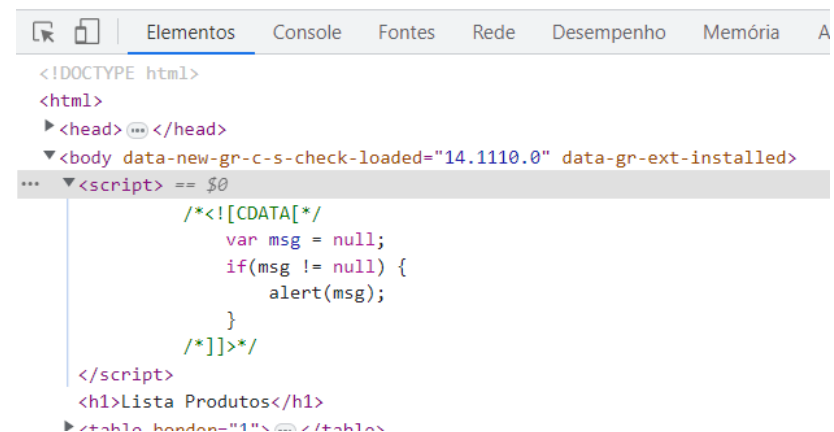
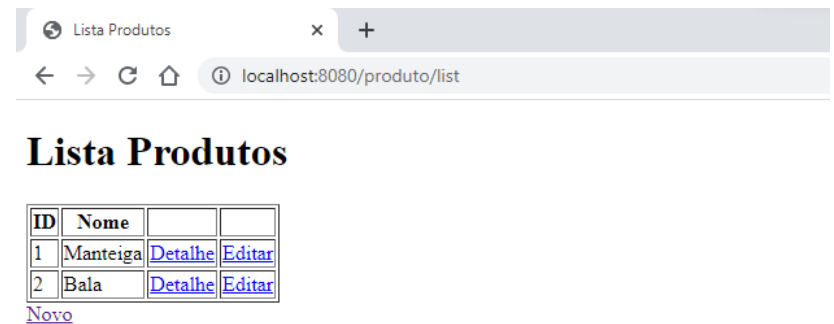
3 – Na página **lista-produtos.html** devemos incluir um trecho de código Javascript que verifica se o valor embutido na página da variável de sessão “*msg*” é ou não **null**. No caso de “*msg*” ser diferente de **null** então esse trecho de código deverá apresentar um **alert()** com o conteúdo de “*msg*”.

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4     <title>Lista Produtos</title>
5     <meta charset="utf-8">
6 </head>
7 <body>
8     <script th:inline="javascript">
9         /*<![CDATA[*/
10         var msg = /*[[${msg}]]*/ null;
11         if(msg != null) {
12             alert(msg);
13         }
14         /*]]>*/
15     </script>
16
17     <h1>Lista Produtos</h1>
18
19     <table border="1">
```





“/produto/list” quando chamado via *redirect*



“/produto/list” quando chamado diretamente

4 – Por fim, abra o **Git Bash** e execute o para adicionar as alterações feitas ao repositório.