Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
Unidade Acadêmica de Informática
Curso Superior de Tecnologia em Sistemas para Internet
Disciplina: Programação para a Web II
Professor: Frederico Costa Guedes Pereira

Prática-02: Maven e VSCode

por Frederico C. G. Pereira © Copyright 2022

Objetivos

- Criar um projeto web no Maven
- Compilar e executar uma aplicação web com Spring Boot
- Utilizar biblioteca externa

ATENÇÃO: Não copie e cole os trechos de código, DIGITE-OS VOCÊ MESMO! Isto faz uma gigantesca diferença para o aprendizado e é uma recomendação sempre encontrada em livros de programação. Sempre após digitar, certifique-se de que entendeu o que você fez, do contrário a prática é apenas um exercício de digitação. Siga corretamente os passos neste roteiro. Obedeça às versões indicadas do que deve ser baixado e não baixe alternativas mais recente porque pode quebrar seu código. Faça estas experiências depois de completar a prática.

Certifique-se de ter a versão Java 17 ou 21 e o VSCode com todas as extensões necessárias instalados em sua máquina. Veja primeiro exercício.

Antes de achar que o que você fez produziu um erro inesperado, leia a próxima questão ou próximo item da lista para saber se não era isso mesmo que deveria acontecer. Alguns exercícios são para provocar um erro inicial que será corrigido logo a seguir. Caso tenha sido erro mesmo, procure o professor para ajudá-lo. Alguns passos são meras explicações do passo anterior e não pedem para o aluno realizar nada no código.

Utilizaremos nesta disciplina a versão 17 ou 21 do JDK.

1. Parte 1 – Criar aplicação web que gere um PDF dinamicamente

1) Crie e configure um projeto Maven com Spring no VSCode.
   a) Crie uma pasta chamada **workspace** em alguma pasta do seu sistema de arquivos. Esta pasta será a raíz de todos os projetos Spring da disciplina.
   b) No VSCode, tecle Ctrl+P e digite **Java Project** no campo de entrada.
   c) Escolha a opção **Spring Boot**. (Obs: depois deste passo, caso faça uma escolha incorreta, use a seta de voltar ← e escolha opção indicada neste roteiro).
   d) Selecione a opção **Maven Project**.
   e) Selecione a versão **3.4.4** do Spring Boot
   f) Selecione **Java** como linguagem de programação.

      A partir de agora você informará as coordenadas Maven do seu projeto.
   g) Primeiro informe: **br.edu.ifpb.pweb2** (em geral, o Group Id do Maven é o DNS ao contrário da sua empresa ou instituição).
   h) Agora informe o ArtifactId: **genpdf**.
   i) Especifique o tipo de empacotamento do seu projeto, selecione **Jar**.
   j) Selecione a versão **17 ou 21** para Java.

Agora você escolherá os *starters* do *Spring Boot*, que gerarão as dependências no pom.xml do seu projeto Maven.

k) Digite **Spring Web** ou apenas **Web** e selecione a opção "Spring Web (Buildwebm uncluding RESTful...)"

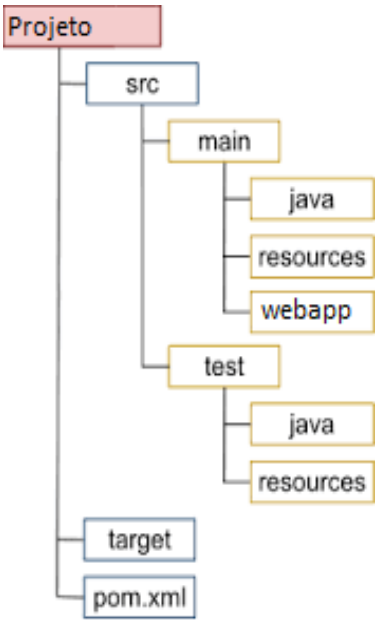
l) Digite **Devtools** e selecione a opção Spring Boot DevTools. Este starter fará seu swervidor reiniciar a aplicação sempre que a modificar, evitando assim que você mesmo faça o reinício ou o redeployment da aplicação. Uma mão na roda!

m) Tecle enter para encerrar a escolha de *starters*.

n) O VSCode perguntará em que pasta deseja criar o projeto, escolha a pasta **workspace** e tecle <ENTER> para confirmar.

o) O projeto Maven em Java com uso do framework Spring Boot deve ter sido criado no seu VS Code.

Um projeto Maven completo possui a seguinte estrutura padrão de pastas:



- **src/main/java** contém o código fonte Java;
- **src/main/resorces** contém os recursos da aplicação usados pelo código (arquivo persistence.xml, por exemplo). O *Spring Boot* usa esta pasta para colocar os recursos estáticos (arquivos de folhas de estilo, JS, imagens, etc) numa subpasta chamada ***static*** e os templates de páginas dinâmicas (usaremos o Thymeleaf) na subpasta ***templates***.
- **src/main/webapp** é a raiz da aplicação web, porém o Spring Boot não utiliza esta pasta. O VSCode não a cria no projeto.
- **src/test/java** e **src/test/resources** possuem o mesmo propósito que as main, porém para testes automatizados (se houver).
- **target** é a pasta onde o Maven/Eclipse põe o código Java compilado e onde ele monta a aplicação web completa para implantar no container. Em geral, não usamos muito esta pasta.
- **pom.xml** é o arquivo de configuração básico do Maven. Onde se estabelecem os metadados, as dependências e plugins do projeto.

2) Abra o arquivo pom.xml e observe as dependências que foram geradas a partir dos starters escolhidos.

3) Procure pelo GroupId e pelo ArtifactId do seu projeto no pom.xml.

4) Podemos adicionar novas dependências ao nosso projeto. No exemplo que implementaremos, vamos escrever uma aplicação Spring que gera um arquivo PDF dinamicamente. Para tanto, dependemos de uma biblioteca de Java chamada IText. Digite no navegador **mvn repository**[1] e vá para a primeira opção mostrada pelo navegador:

   a) Digite **itext** na busca e procure pela resposta com.itextpdf » itextpdf. Selecione a versão 5.5.13.2, que não é a mais recente, e cole o fragmento XML da dependência dentro de <dependencies> do pom.xml.

[1] Este é um dos inúmeros repositórios Maven de bibliotecas Java.

b) O VSCode pode solicitar se você deseja recompilar todo o projeto. Se isto acontecer, confirme.

5) Pronto, seu projeto está criado e configurado para rodar uma aplicação web Spring.

Criando conteúdo para a aplicação web

6) Vá até a pasta src/main/java e observe o pacote **br.edu.ufpb.pweb2.genpdf** que foi criado pelo assistente do VSCode.

7) Há uma classe de aplicação (com um método main) de nome **GenpdfApplication** dentro deste pacote. Esta classe está anotada com a anotação @SpringBootApplication.

   a) Acima do método main vê-se as opções **Run|Debug**[2]

   b) A opção Run sobe um container web Apache Tomcat embutido e põe a aplicação para executar dentro deste container.

   c) A opção Debug faz o mesmo, porém permite que você ponha *breakpoints* para depurar o código quando estiver procurando por bugs durante seu desenvolvimento. Para pôr um *breakpoint*, passe o cursor sobre o número da linha e clique no círculo vermelho que surje à esquerda.

8) Clique em Run na classe de aplicação e observe o terminal do VSCode subir a aplicação. Por enquanto não há nada de útil que ela possa fazer, mas podemos tentar acessar a aplicação num navegador digitando: **http://localhost:8080/**:

9) Como não temos nenhum recurso padrão (página index.html) ou um controlador Spring (classe Java que atende a uma requisição HTTP) que gere conteúdo dinamicamente, então o Spring gera esta página de erro equivalente a um HTTP 404 (Not Found).

10) Crie uma classe controlador Spring no seu projeto para gerar um documento PDF usando o iText:

   a) Clique com o botão direito do mouse na pasta src/main/java/br/edu/ifpb/pweb2/genpdf.

   b) Escolha **New File** e digite **controller/GeneratePdfController.java**.

   c) Um subpacote **controller** deve ter sido criado abaixo de genpdf e, dentro dele, uma classe chamada GeneratePdfController.java. A diretiva ***package*** da classe deve ter sido gerada com o nome do pacote.

[2] Você pode, a partir de qualquer classe aberta no editor do VSCode, executar ou depurar a classe de aplicação do projeto teclando Ctrl+F5 (para **Run**) ou F5 (para **Debug**). O arquivo aberto tem que ser uma classe Java do projeto, qualquer uma.

d) Caso o VSCode não tenha criado a classe como acima, apague a pasta **controller**, digite Ctrl+Shift+P e use a opção **clean java language server**. Clique em **Restart and delete**. Isto limpará o workspace Java das extensões do VSCode.

e) Repita os passos a) a c) acima.

11) Copie o código a seguir no arquivo GeneratePdfController.java (código copiável):

```
@Controller
@RequestMapping("/home")
public class GeneratePdfController {


    @RequestMapping("/genpdf")
    public ResponseEntity<byte[]> generatePdf() {
        Document document = new Document();
        ResponseEntity<byte[]> response = null;
        try {
           // Começa a criar o documento PDF usando a biblioteca iText
            ByteArrayOutputStream bos = new ByteArrayOutputStream();
            PdfWriter.getInstance(document, bos);
            document.open();
            Font titleFont = FontFactory.getFont(FontFactory.HELVETICA, 16, Font.BOLDITALIC);
            Font paragraphFont = FontFactory.getFont(FontFactory.HELVETICA, 12, Font.NORMAL);
            Chunk chunk = new Chunk("Título do Parágrafo", titleFont);
            Chapter chapter = new Chapter(new Paragraph(chunk), 1);
            chapter.setNumberDepth(0);
            chapter.add(new Paragraph("Texto do parágrafo... ", paragraphFont));
            document.add(chapter);
            document.close();


           // Começa a gerar a resposta HTTP para um tipo de arquivo diferente de HTML
            HttpHeaders headers = new HttpHeaders();
            headers.setContentType(MediaType.APPLICATION_PDF);
            String filename = "meupdf.pdf";
            headers.setContentDispositionFormData(filename, filename);
            headers.setCacheControl("must-revalidate, post-check=0, pre-check=0");
            response = new ResponseEntity<>(bos.toByteArray(), headers, HttpStatus.OK);
        } catch (DocumentException e) {
            System.out.println("Erro ao gerar PDF " + e.getMessage());
        }
```

```
        return response;
    }
}
```

A notação @Controller é do Spring e informa ao framework que esta classe é um controlador. Um controlador é um componente do Spring capaz de atender requisições HTTP. Seus métodos podem atender a diferentes URLs.

A anotação @RequestMapping informa que URL deve ser usada para ativar a classe e os métodos controladores.

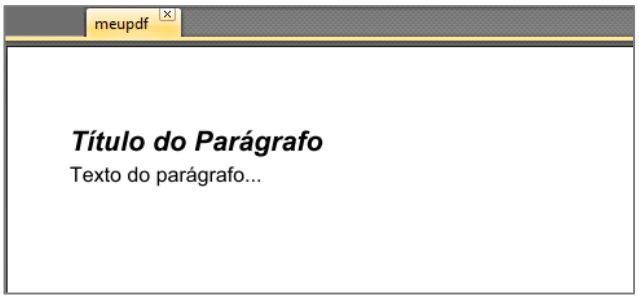
12) O editor do VSCode deve mostrar vários erros em classes, constantes etc desconhecidas. A classe Document do iText deve ser importada, por exemplo. Clique sobre a primeira ocorrência de Document e tecle "Ctrl + .", o VSCode gera o import ou pede para você escolher uma das opções, caso haja conflito de nomes (mais de uma classe com mesmo nome em pacotes diferentes.

13) Caso esteja com dificuldades com os imports, este são os corretos:

```
import com.itextpdf.text.Chapter;
import com.itextpdf.text.Chunk;
import com.itextpdf.text.Document;
import com.itextpdf.text.DocumentException;
import com.itextpdf.text.Font;
import com.itextpdf.text.FontFactory;
import com.itextpdf.text.Paragraph;
import com.itextpdf.text.pdf.PdfWriter;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
```

14) Digite um título e um texto e clique no botão "Gerar PDF". A aplicação deverá solicitar uma pasta para salvar um arquivo chamado meupdf.pdf. Selecione a pasta e grave-o. Abra o arquivo com um editor de PDF. O resultado deve ser este da figura a seguir:



O código parte da prática pode ser obtido em:
https://gitlab.com/fredguedespereira/genpdf

2. Parte 2 - Criando, instalando e usando uma lib Java

Nesta parte da prática você vai empacotar e instalar um projeto de uma biblioteca Java (arquivo .jar) no seu repositório Maven. O projeto já está implementado e é muito simples. Siga o roteiro.

Clone o projeto do repositório git:

1) Copie a seguinte URL do projeto no GitLab: https://gitlab.com/fredguedespereira/meusutilitarios.git
2) No VSCode, abra a paleta de comandos com F1 ou Ctrl+Shift+P
3) Digite **git clone** e, ao surgir as opções, selecione a primeira (ou apenas tecle ENTER quando ela surgir)
4) Copie a URL acima do passo (1)
5) O VSCode agora pede uma pasta para baixar o projeto. Indique uma pasta com permissão de escrita no seu computador.
6) O VS Code pode perguntar se você deseja abrir o repositório clonado. Clique em "Abrir".
7) Você deve estar com o projeto clonado! Verifique se consegue identificar a pasta **src/** e o arquivo **pom.xml**. Abra este último e identifique os metadados, as dependências e os plugins (seção build).

Compile e empacote o projeto:

1) Abra um terminal de comandos no VSCode em que você está com o projeto aberto: tecle Ctrl+Shift+` ou então selecione nos menus Terminal > New Terminal
2) Digite os comandos do Maven para compilar seu código:

   mvn clean compile

   a) Os comandos do ciclo de vida do projeto podem ser encadeados, o comando clean apaga a pasta target (que ainda não existe) para gerar novo "build" do zero.

   b) O comando compile pede a compilação de todas as classes

   c) Depois deste comando, deve aparecer uma pasta **target** no seu projeto. Explore-a. Na pasta classes encontram-se as classes Java do projeto compiladas.

3) O próximo comando gera o "entregável" do projeto. Como se trata de uma lib Java, ele gera um arquivo .jar como resultado (em target também). Comande:

   mvn package

   a) Este comando aproveita tudo que foi feito no anterior e dá mais um passo. No anterior compilamos as classes do projeto, neste geramos um jar para empacotar tudo.

   b) Se tivéssemos comandado mvn clean package, ele apagaria target, compilaria e depois empacotaria o projeto.

c) O cliclo de vida de um projeto Maven passa pelas seguintes fases (nem todos os projetos precisam passar por todas as fases):

clean → validate → compile → test → package → install → deploy

d) Este passo é apenas explicação! Quando você pede uma fase avançada, ele passa por todas as anteriores (menos a clean, que precisa ser comandada sempre que se quer desprezar tudo que já foi feito). Ou seja, ao comandar **mvn clean compile** ele também executou a **validate** sem você perceber (não havia nenhuma validação a ser feita, portanto nada foi executado).

e) Este passo é apenas explicação! Quando você comanda **mvn compile**, e é sua primeira compilação, ele vai compilar todas as entidades Java. Se você altera apenas uma linha de uma classe e comanda **mvn compile** novamente, ele só compila aquela classe alterada. Se quiser compilar tudo do zero precisa comandar **mvn clean compile**

4) O último comando deve ter gerado um arquivo **meusutilitarios-1.0.0.jar** na pasta **target.**
5) Este arquivo é apenas um ZIP com extensão JAR. Você pode abri-lo com o Winrar ou algum zipador instalado na sua máquina para vê-lo por dentro! (Não é essencial!)

Instale o projeto no seu repositório local Maven

1) Para isso, comande: **mvn clean install**
2) O Maven apaga o target, recompila, empacota e instala o jar no seu repositório local. Não sei se temos acesso a este repositório nos labs do IFPB, mas na sua casa você consegue ver a lib instalada acessando o seu repositóri a partir da sua pasta home no Windows (geralmente c:\Windows\usuarios\<seu usuário>\.m2\repository, no Linux vá para sua pasta home com **cd ~** e procure a partir daí). A partir daí, navegue nas pastas br/edu/ifpb/pweb2/util e dentro dela está uma pasta da lib meusutilitarios e, dentro dela, uma pasta para cada versão instalada.

Use a lib gerada em outro projeto

1) Crie um projeto Java no VSCode que use Maven:
   a) Tecle F1, digite **create java project** e ENTER
   b) Selecione **Maven create from archetype**
   c) Selecione No Archetype
   d) Digite **br.edu.ifpb.app** para o Group Id
   e) Digite **usautilitatios** para o Artifact Id
   f) Escolha a pasta **workspace** (cuidado para não selecionar a pasta do projeto anterior, escolha a pasta pai dos seus projetos)
   g) O VSCode pergunta se você quer abrir, mas acho que devido a um bug, ele não abre o projeto, mas a pasta pai dele. Apenas clique no X do painel de pergunta.
   h) Abra seu projeto **usautilitarios** numa nova janela do VSCode.
2) Com o projeto **usautilitarios** aberto no VSCode, edite o pom.xml gerado.

Explore a pasta src/main/java. Note que ele já cria uma classe principal com um método Main. Vamos aproveitá-la para usar nossa lib de formatação de CPFs, CNPJs, e validações que instalamos no nosso repositório local. Siga para o próximo passo.

3) Abra a classe Main.java e modifique o método **main** para o seguinte:

```
public static void main(String[] args) {
    String cpf = "12345678900";
    String cpfFormatado = MascarasUtil.formatarCPF(cpf);
    System.out.println("CPF formatado = " + cpfFormatado);
}
```

Note que há um erro de compilação, a classe MascarasUtil não foi encontrada. Tente forçar o import com Shift+Alt+O. Nada muda!

4) Precisamos informar ao Maven que temos dependência da lib meustutilitarios-1.0.0.jar. Edite o pom.xml para indicar esta dependência. Adicione, logo após a tag <version> o seguinte trecho:

```
<dependencies>
    <dependency>
        <groupId>br.edu.ifpb.pweb2.util</groupId>
        <artifactId>meusutilitarios</artifactId>
        <version>1.0.0</version>
    </dependency>
</dependencies>
```

5) Se o VSCode perguntar se deseja "rebuildar" (ou algo parecido) o projeto, selecione Yes (ou Always).

6) Volte à classe Main e tente gerar o import novamente com Shift+Alt+O ou então, clicando com o botão direito sobre o nome da classe MascarasUtil, depois clicando na pequena lâmpada que aparece à esquerda e escolhendo o comando Import adequado.

7) Execute o main. Deve ter mostrado aquele CPF no formato padrão.

8) Adicione o seguinte trecho ao main:

```
String cnpj = "12332422200011";
String cnpjFormatado = MascarasUtil.formatarCNPJ(cnpj);
System.out.println("CNPJ formatado = " + cnpjFormatado);
```

9) Execute novamente a classe.

10) Agora vamos usar outra classe da nossa lib, a classe de validação. Adicione ao final do main (você pode precisar gerar o import da classe ValidadorUtil ou o VSCOde pode gerar automaticamente ao digitar):

```
boolean valido = ValidadorUtil.validarCPF(cpfFormatado);
System.out.println(String.format("O cpf %s é válido? %s", cpfFormatado, valido));
```

11) Execute novamente a classe.

12) Digite um CPF válido (apenas dígitos) na linha que declara e inicializa a variável **cpf** no **main.** Pode ser o seu. Se não souber, use este: "47105001003"

13) Execute novamente o **main**.

14) Explore o autocompletar quando você digita os nomes das classes (ValidadorUtil e MascarasUtil) e depois ".". Há outros métodos nelas. Tente usar o método demo().

15) Fim da prática.