

Cap 02

Padrões Arquiteturais

Model View Controller

MVC: Model View Controller

SPA: Single Page Application

MOM: Message Oriented Middleware

SOA: Service Oriented Architecture

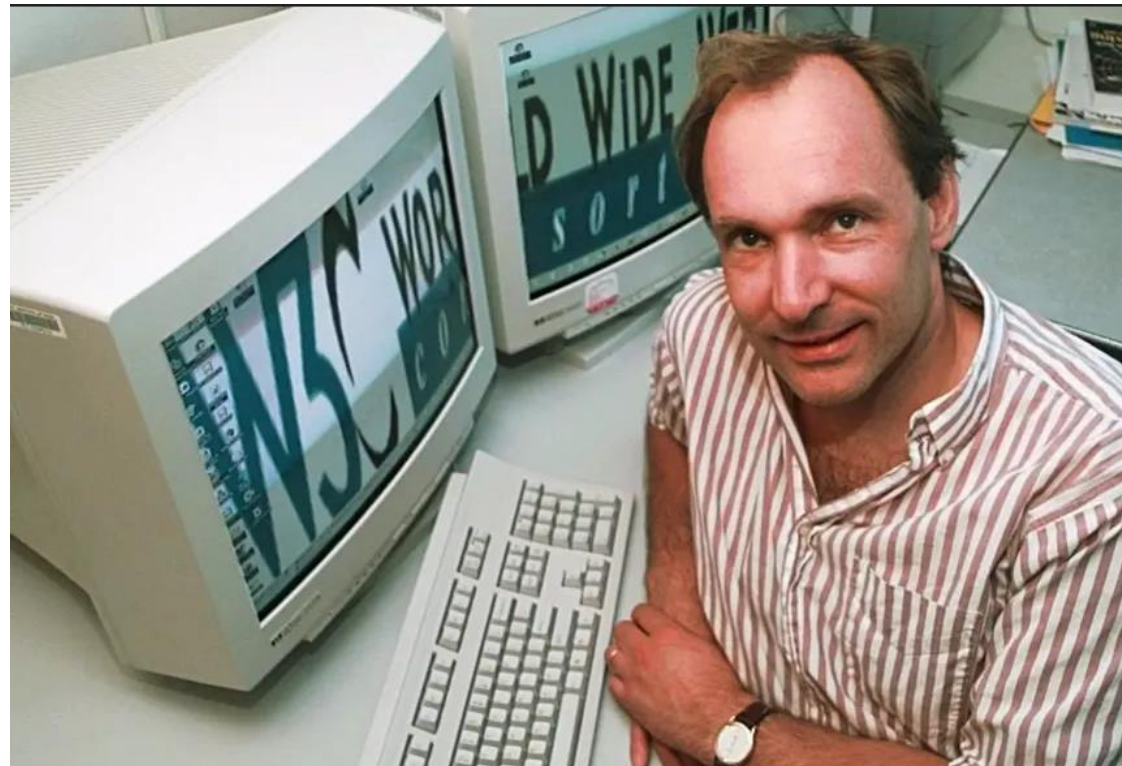
MVC: Model View Controller

SOA: Service Oriented Architecture

MOM: Message Oriented Middleware

SPA: Single Page Application

- **A World Wide Web – WWW ou simplesmente Web**
 - A **Internet** foi idealizada para troca de informação a nível global
 - A princípio, estas informações seriam apenas **textos**
 - Em 1992, o cientista Tim Berners-Lee, criou a WWW com a proposta de ir além de textos



MVC: Model View Controller

- A WWW conseguiu ser **viabilizada** por outras tecnologias:

HTTP

Protocolo da Internet baseado em hyper-textos

HTML

Linguagem de marcação estrutural de hyper-textos

CSS

Linguagem de estilização para o HTML

Servidor Web

Software *server-side* para prover arquivos HTML

Navegador Web

Software *client-side* para interpretar arquivo HTML

Arquitetura da WWW

Cliente

Servidor

Arquitetura da WWW

Cliente

Servidor

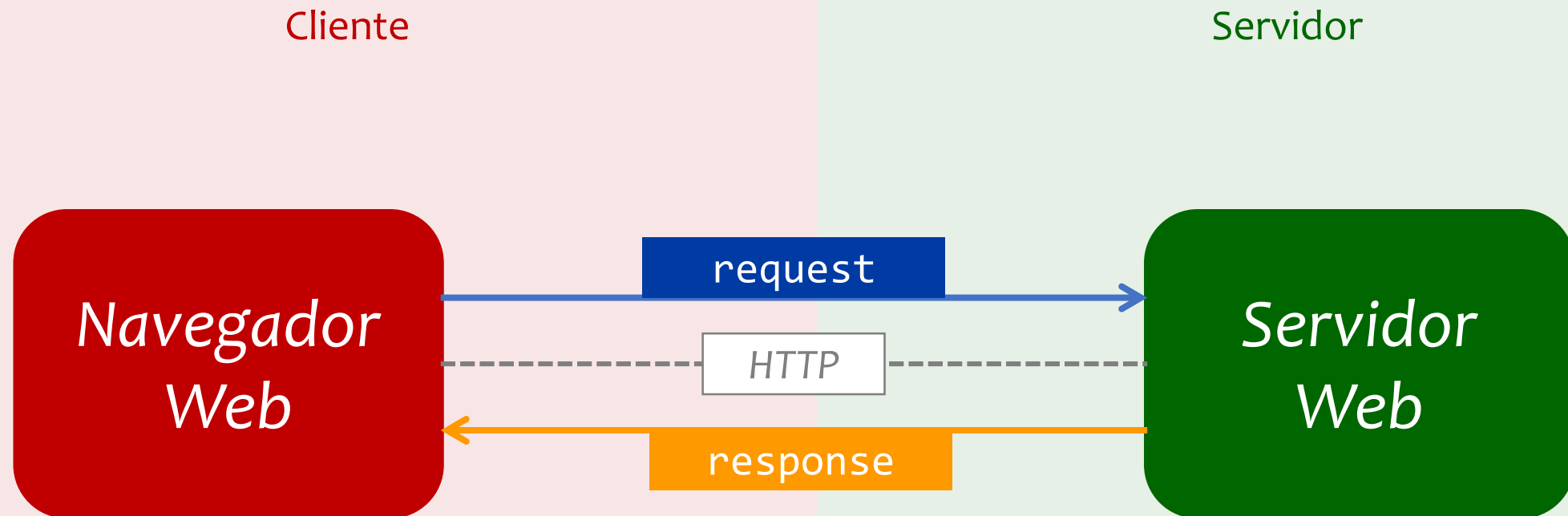
*Navegador
Web*

HTTP

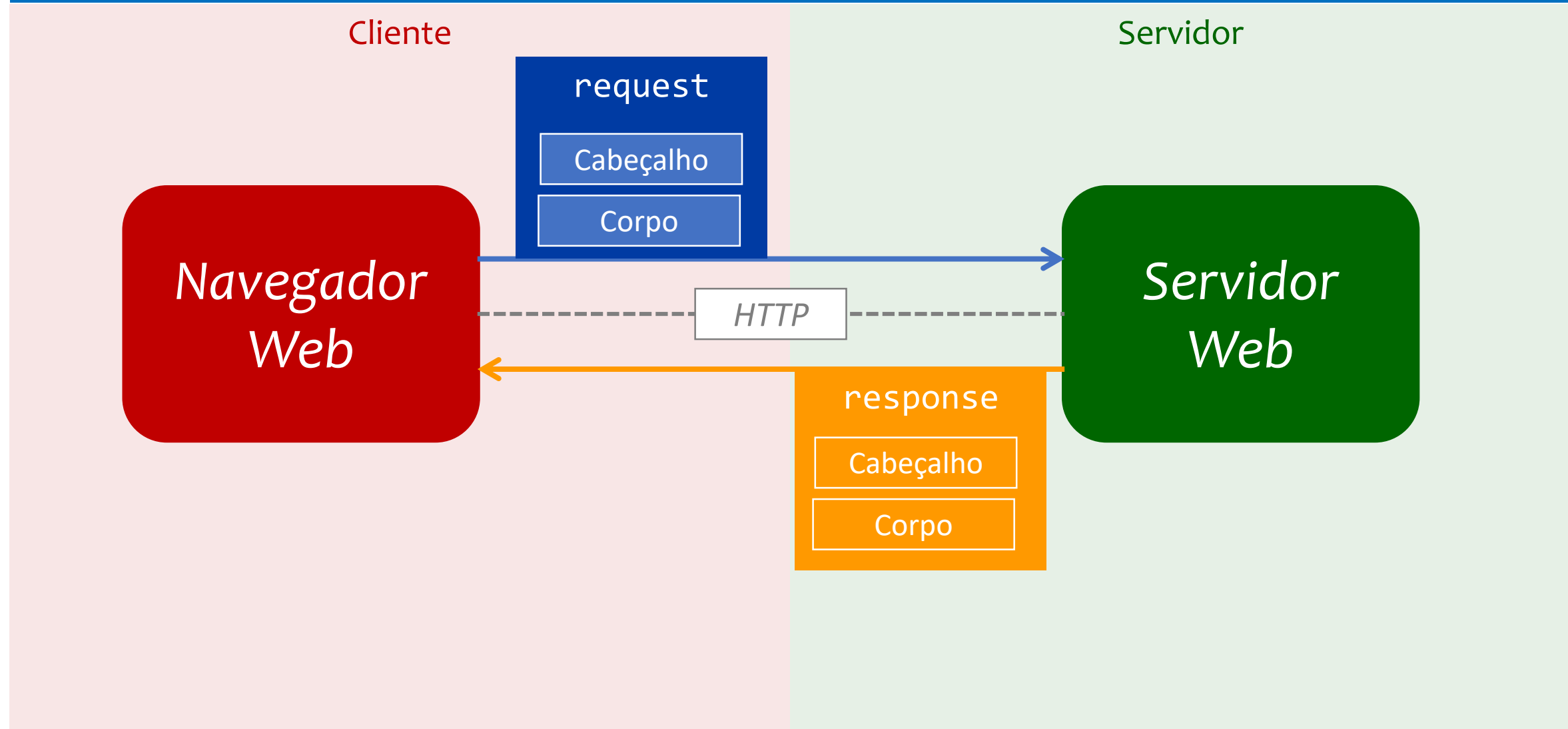
*Servidor
Web*



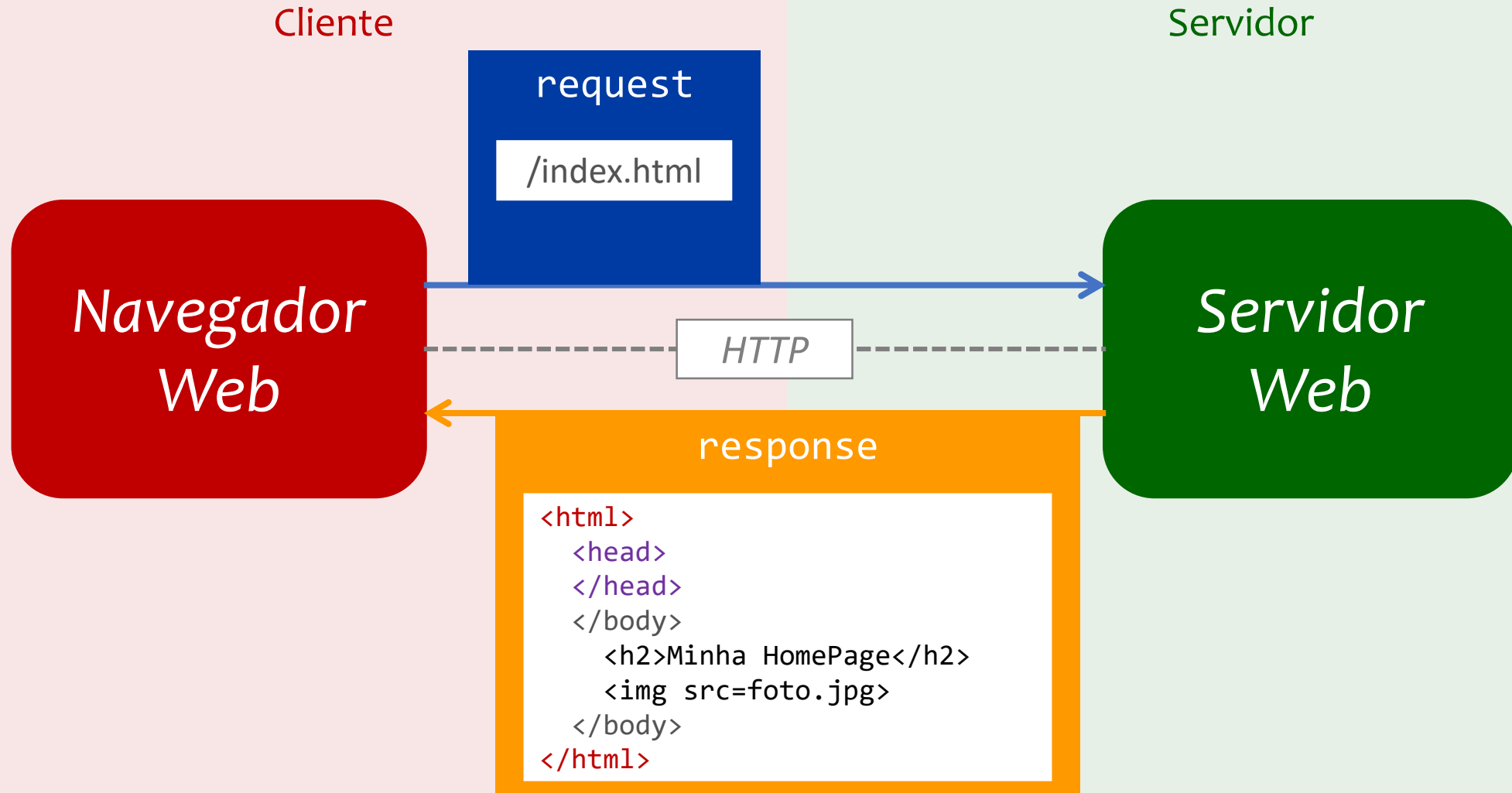
Arquitetura da WWW



Arquitetura da WWW



Arquitetura da WWW



Exemplos de Navegadores Web (Browsers)

*Navegador
Web*



Exemplos de Servidores Web



NGINX



*Servidor
Web*

Exercício:
1) Servidor Web

Exercício: 1) Servidor Web

Vamos configurar o **servidor web Tomcat**:

- a) Abrir o Eclipse
- b) Fechar todos os projetos do workspace

Exercício: 1) Servidor Web

- c) Acessar start.spring.io
- d) Preencher o formulário segundo a tabela abaixo:

Project	Maven Project
Language	Java
Spring Boot	(versão estável)

Project Metadata:	Group	br.inatel.labs
	Artifact	Padrao_MVC
	Name	Padrao_MVC
	Description	Aplicação MVC
	Package name	br.inatel.labs.padraomvc
	Packaging	Jar
	Java	17

Dependencies
Spring Web
Spring Boot Dev Tools
Thymeleaf

Exercício: 1) Servidor Web

Dica:

Antes de importar, desabilitar **Preferences > Maven > Download Artifact Sources**

e) Clicar em **Generate**

f) Descompactar zip para a pasta do repositório do Eclipse

g) No Eclipse, importar como projeto Maven:

File > Import

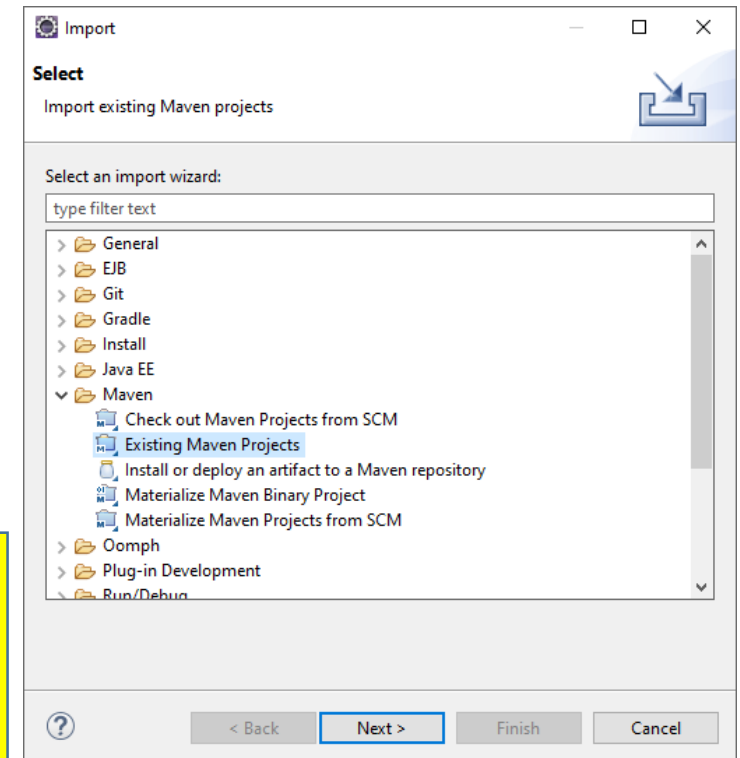
* Expandir **Maven > Existing Maven Projects**

* Selecionar a pasta descompactada

* Clicar **Finish**

Importante:

Aguardar o build do projeto




Exercício: 1) Servidor Web

h) Subir a aplicação:

- Na aba Project Explorer, expandir o pacote principal
- Na classe **PadraoMvcApplication**:
 - Clicar com botão direito > **Run As** > **Java Application**

Exercício: 1) Servidor Web

h) Na aba Console, é possível ver que o Tomcat subiu na porta 8080



```

Console X
PadraoMvcApplication [Java Application] C:\Program Files\Java\jdk-17\bin\javaw.exe (29 de out. de 2023 15:47:32) [pid: 1912]

:: Spring Boot :: (v3.1.5)

2023-10-29T15:47:35.410-03:00 INFO 1912 --- [ restartedMain] b.i.labs.padraomvc.PadraoMvcApplication : Starting PadraoMvcApplication using Java 17.0.6 with PID 19
2023-10-29T15:47:35.412-03:00 INFO 1912 --- [ restartedMain] b.i.labs.padraomvc.PadraoMvcApplication : No active profile set, falling back to 1 default profile: "
2023-10-29T15:47:35.474-03:00 INFO 1912 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.adc
2023-10-29T15:47:35.474-03:00 INFO 1912 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'lc
2023-10-29T15:47:36.488-03:00 INFO 1912 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2023-10-29T15:47:36.502-03:00 INFO 1912 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-10-29T15:47:36.503-03:00 INFO 1912 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.15]
2023-10-29T15:47:36.579-03:00 INFO 1912 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-10-29T15:47:36.581-03:00 INFO 1912 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 116
2023-10-29T15:47:36.963-03:00 INFO 1912 --- [ restartedMain] o.s.b.d.a.OptionalLiveReloadServer : LiveReload server is running on port 35729
2023-10-29T15:47:36.995-03:00 INFO 1912 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path '/'
2023-10-29T15:47:37.005-03:00 INFO 1912 --- [ restartedMain] b.i.labs.padraomvc.PadraoMvcApplication : Started PadraoMvcApplication in 1.964 seconds (process runn
  
```

Conclusão:

- ❖ O Spring Boot é um framework para escrever aplicações onde o servidor **Tomcat vem embutido**
- ❖ É a maneira mais simples de configurar um **servidor web**

A Web tem evoluído bastante e tal evolução pode ser classificada em:

- **Web 1.0:**
 - HomePages
 - Aplicações web
- **Web 2.0:**
 - Rede sociais
- **Web 3.0:**
 - Semântica
 - IoT
 - Metaverso

Web 1.0

- A possibilidade de criar páginas HTML e torná-las públicas encantou as pessoas
- As conhecidas **Homepages** se popularizaram
- As pessoas criaram suas **homepages** segundo suas áreas de interesse: bandas de músicas, culinárias, poesias, diários, ...
- Estas **homepages** eram **conteúdo estático**
 - Para atualizá-las, precisavam **editar manualmente os arquivos HTML**

Exercícios: 2) Homepage

a) Criar arquivo minha-homepage.html

- Em **src/main/resources > static**, clique da direita: **New > File**

minha-homepage.html

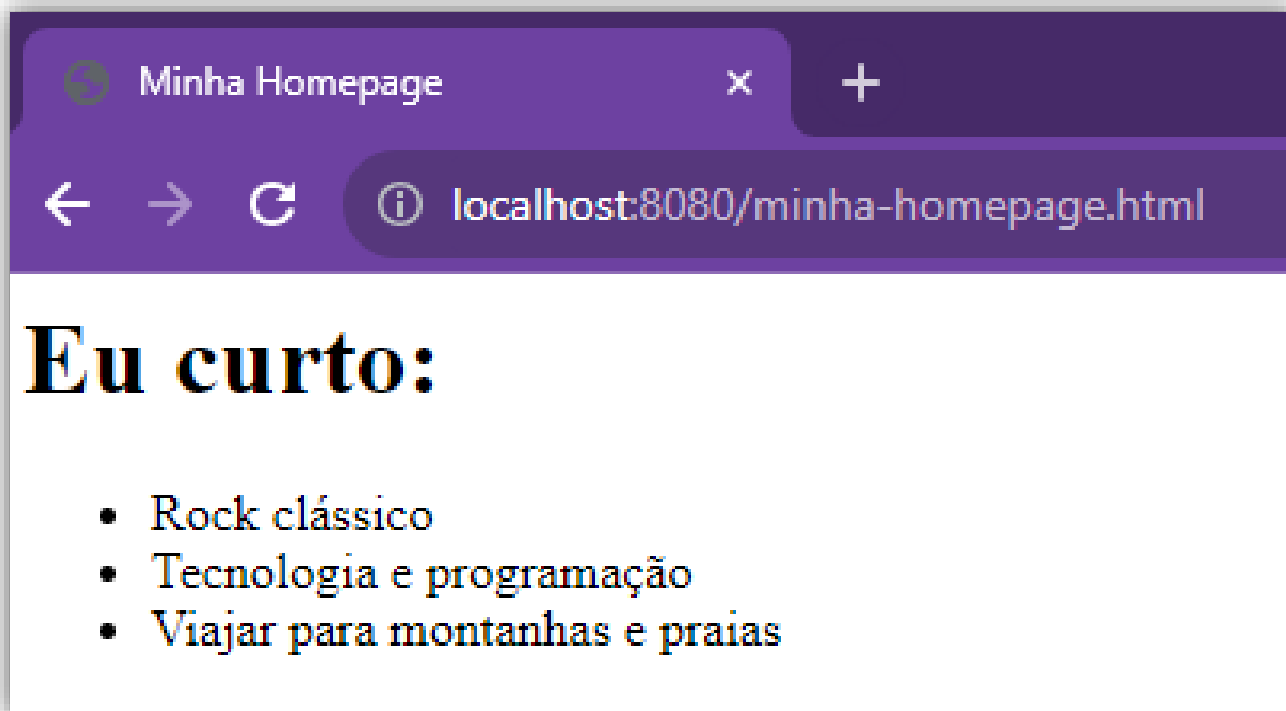
```
<html>
  <head>
    <meta charset="UTF-8">
    <title>Minha Homepage</title>
  </head>
  <body>
    <h1>Eu curto:</h1>
    <ul>
      <li>Rock clássico</li>
      <li>Tecnologia e programação</li>
      <li>Viajar para montanhas e praias</li>
    </ul>
  </body>
</html>
```

Exercícios: 2) Homepage

c) Assegure o Spring Boot esteja no ar

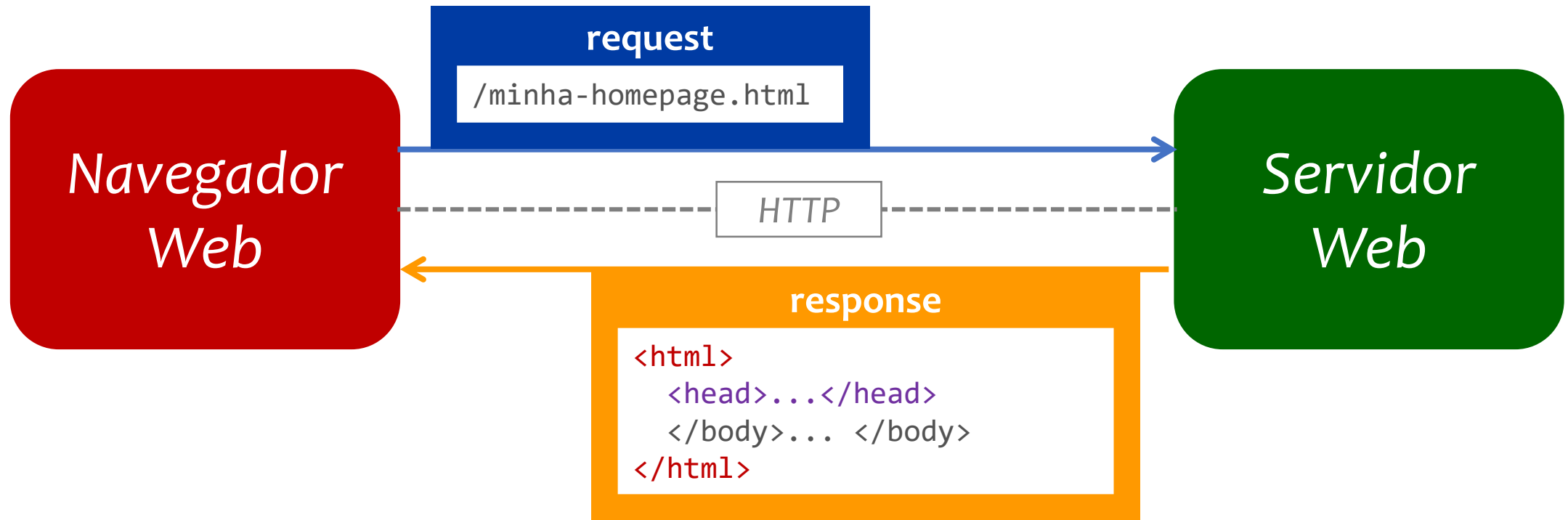
- Abrir o navegador e acessar:

`http://localhost:8080/minha-homepage.html`



Exercícios: 2) Homepage

A idéia deste exercício é mostrar como é uma pagina estática rodando dentro de um servidor Web



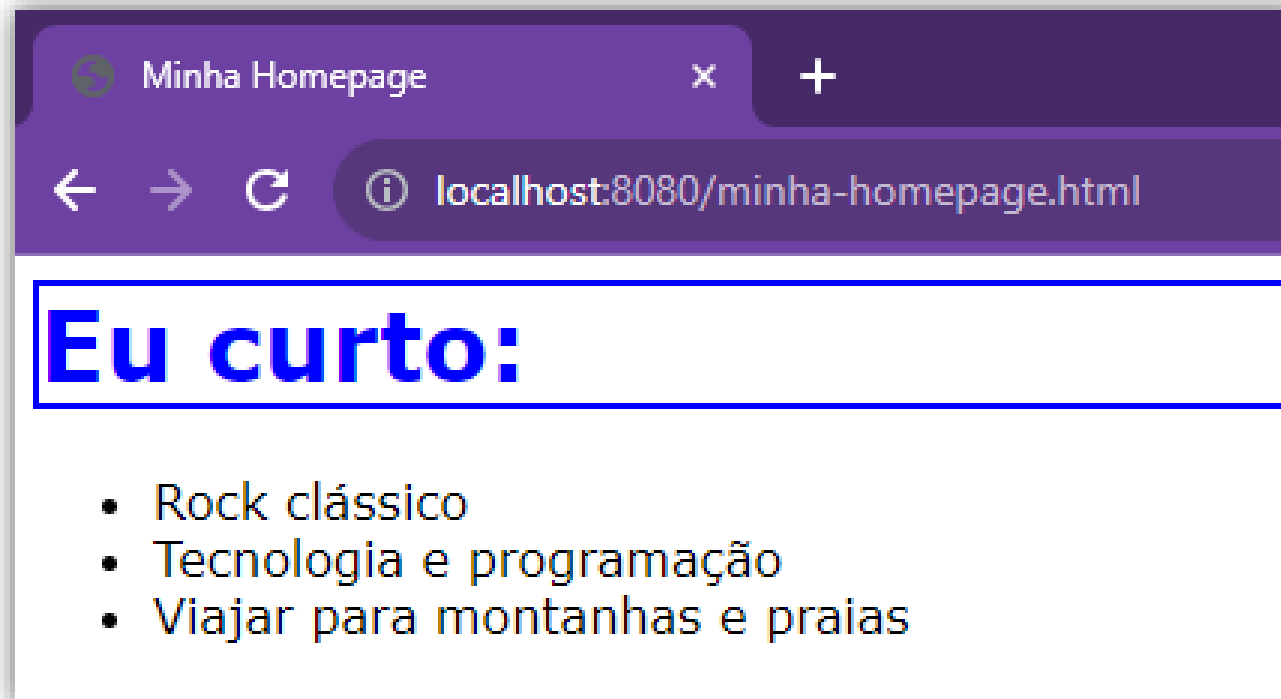
Opcional:

Para entender um pouco de CSS, vamos aplicar alguns estilos simples.

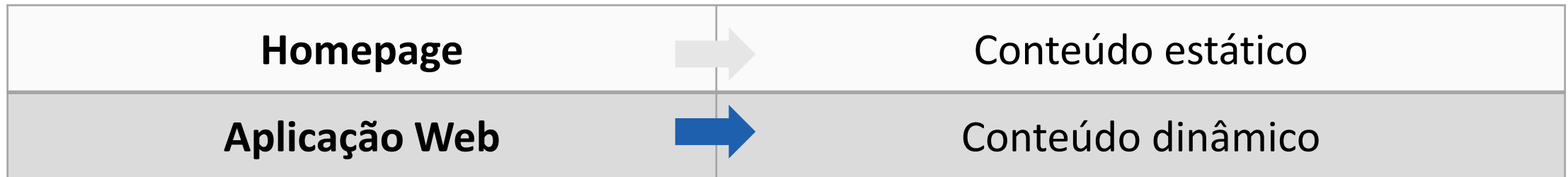
- Dentro de <head>, declarar a tag <style>

```
<style>
  body {
    font-family: verdana;
  }
  h1 {
    color: blue;
    border: 2px solid blue;
  }
</style>
```

Como resultado, temos uma página mais “estilosa”



- O potencial da Web para o desenvolvimento de sistemas corporativos logo despertou interesse das diretorias de TI das empresas
- Mas conteúdo estático não atendia as demandas corporativas
- Empresas de tecnologia começaram então a criar soluções para tornar possível a geração de conteúdo dinâmicos
- Surgiram então as **Aplicações Web**



- Entende-se por **conteúdo dinâmico** a capacidade de exibir diferentes informações conforme o usuário logado, contexto, filtros de pesquisa, ...
- Muitas empresas criaram soluções para dar suporte a conteúdos dinâmicos:
 - *Microsoft*: **ASP**
 - *Adobe*: **FLEX**
 - **PHP**
 - *Sun Microsystems*: **Java Servlets**

Java Servlets

```
@WebServlet(urlPatterns="/saldo-bancario")
public class SaldoBancarioServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        //1.busca saldo atual
        String usuario = getUsuarioLogado();
        Double saldoBancarioAtual = getSaldo(usuario);

        //2.renderiza resposta
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"UTF-8\">");
        out.println("<title>Saldo bancário</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Seu saldo é</h1>");
        out.println(saldoBancarioAtual);
        out.println("</body>");
        out.println("</html>");
    }
}
```

Java Servlets

Lógica de
negócio

Lógica de
apresentação

```
@WebServlet(urlPatterns="/saldo-bancario")
public class SaldoBancarioServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {

        //1.busca saldo atual
        String usuario = getUsuarioLogado();
        Double saldoBancarioAtual = getSaldo(usuario);

        //2.renderiza resposta
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<meta charset=\"UTF-8\">");
        out.println("<title>Saldo bancário</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Seu saldo é</h1>");
        out.println( saldoBancarioAtual );
        out.println("</body>");
        out.println("</html>");

    }
```

Tudo na mesma
classe...

..quebra da
responsabilidade única

Conteúdo dinâmico é **positivo** mas precisa ter:

- ✓ Organização
- ✓ Princípios de desenvolvimento
- ✓ Arquitetura

Lógica de apresentação

Lógica de negócio

MVC: Model View Controller

Lógica de apresentação

View

Lógica de negócio

Model

MVC: Model View Controller

Orquestrador

Controller

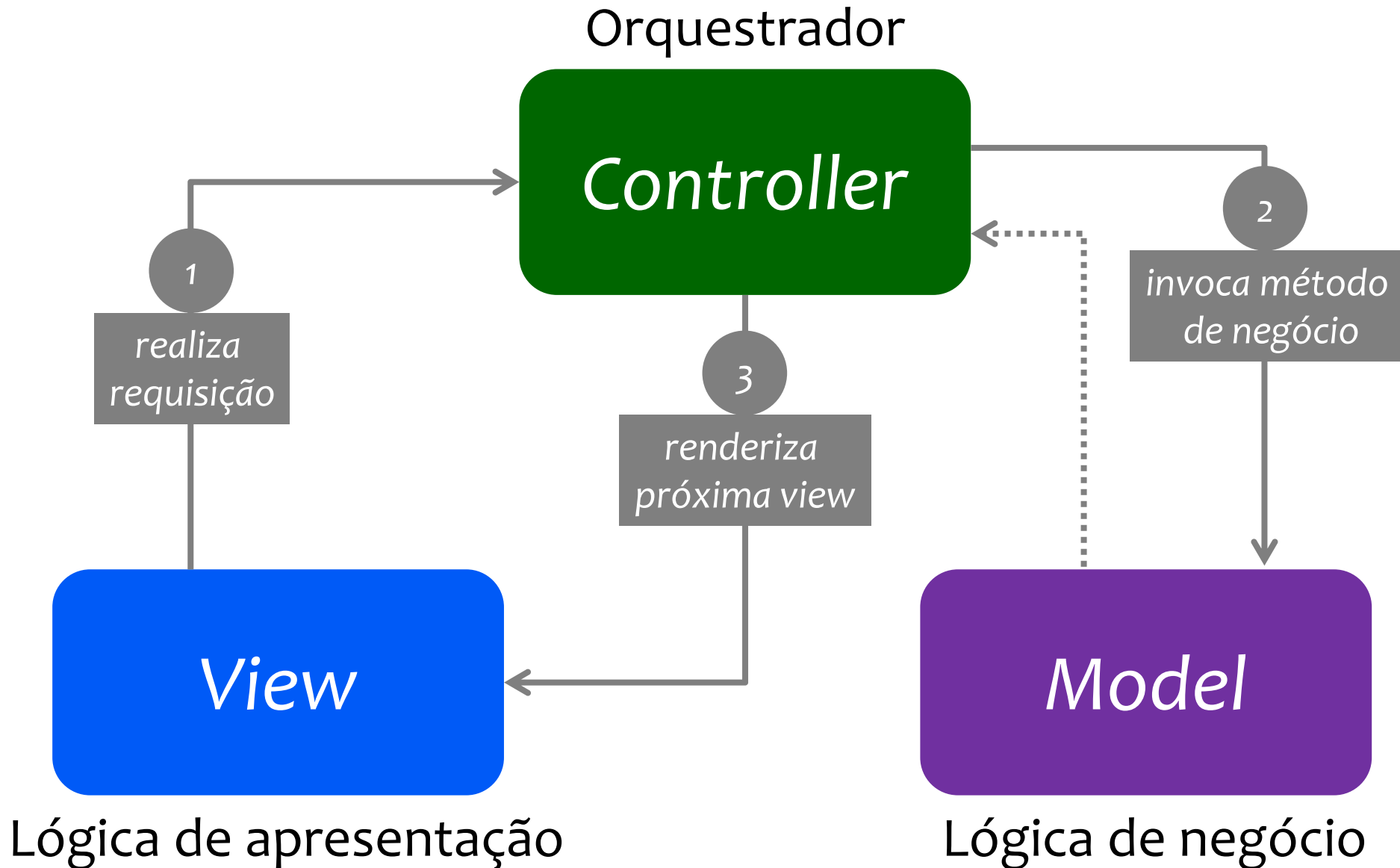
View

Lógica de apresentação

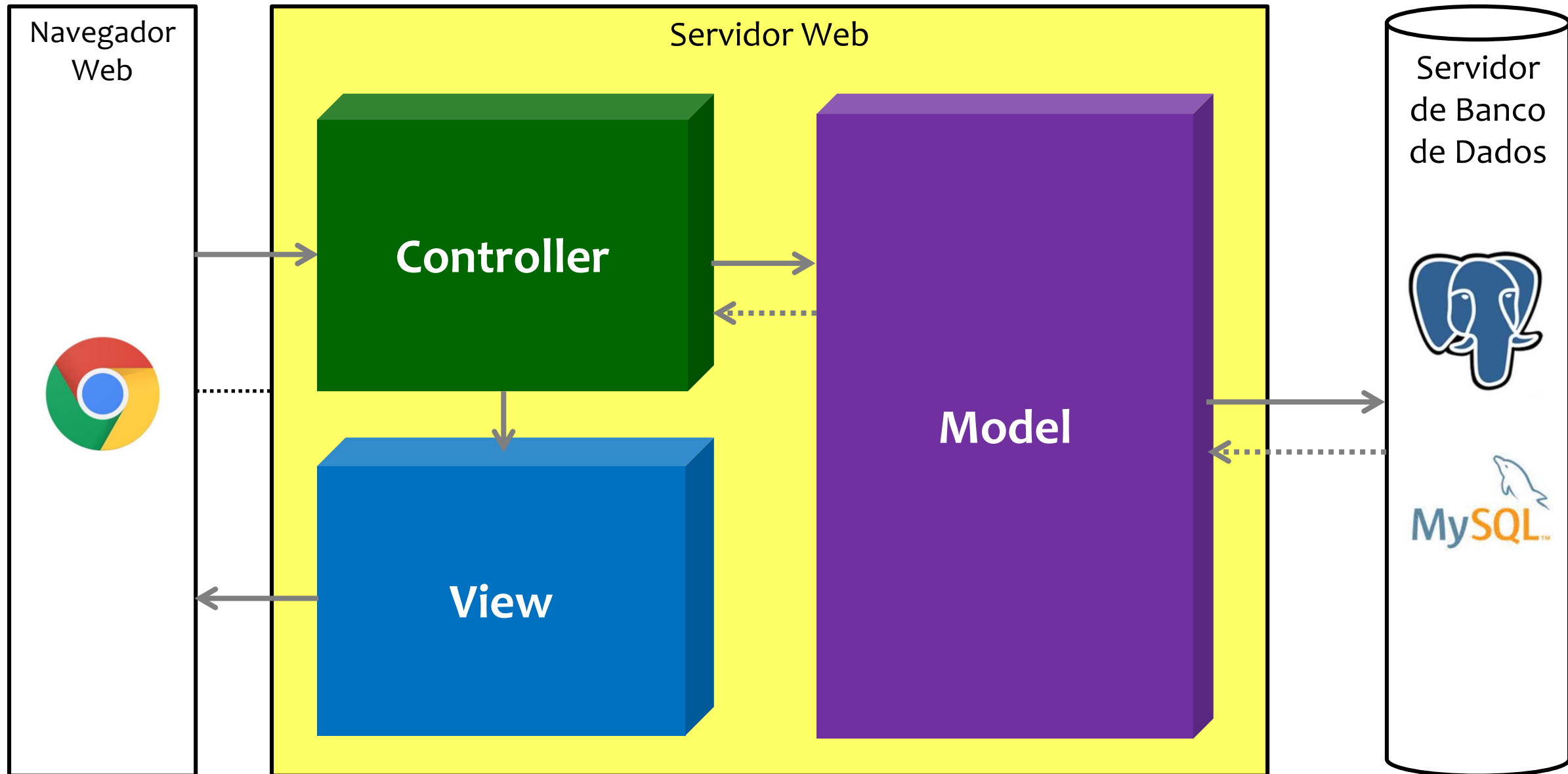
Model

Lógica de negócio

MVC: Model View Controller



MVC: Model View Controller

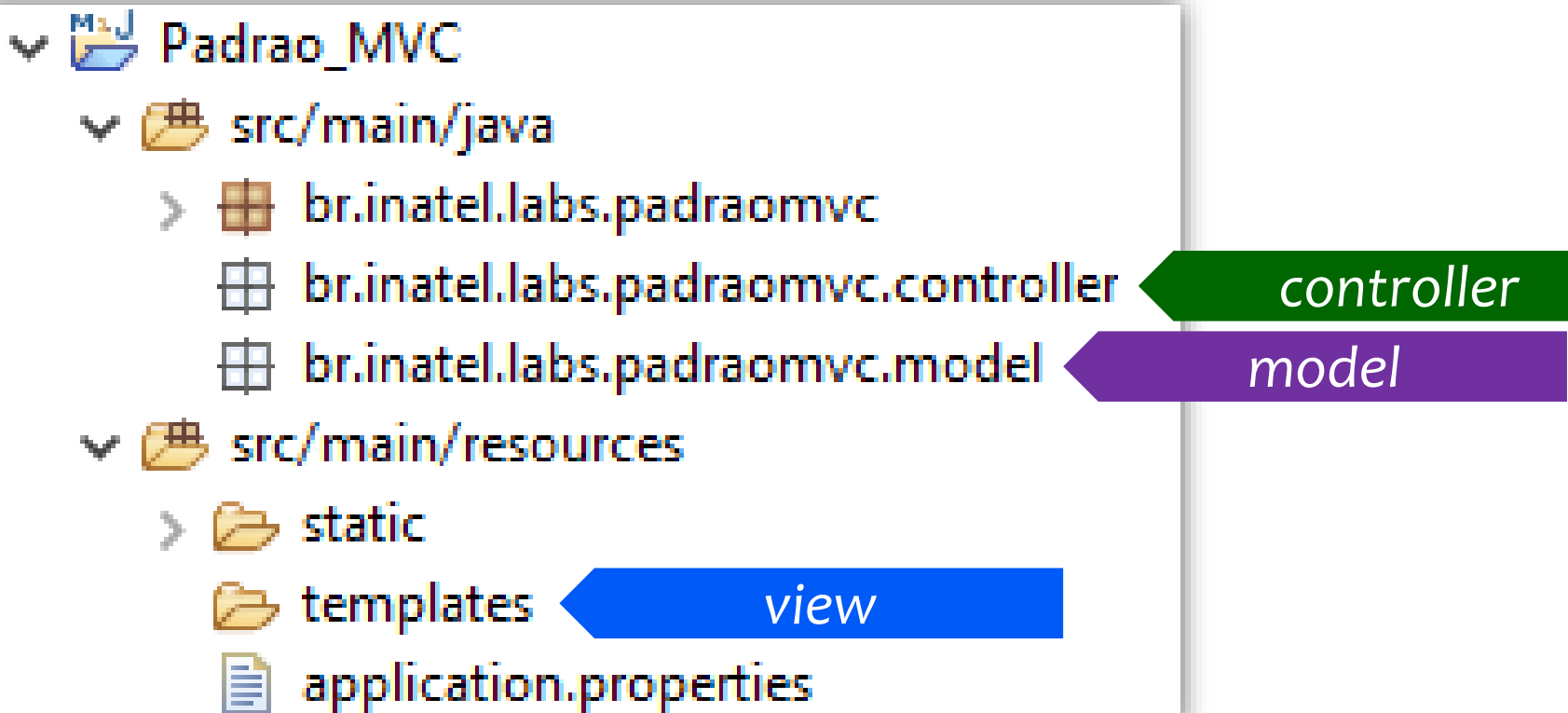


Exercícios: 3) MVC

Exercícios: 3) MVC

a) Criar os sub-pacotes **controller** e **model**

- A estrutura interna do projeto ficará assim:



b) Criando a primeira view:

- Clique da direita na pasta **templates** > new > File : hello-mvc.html

hello-mvc.html

```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Olá mundo. Mas quem é você?
  </body>
</html>
```

c) Criando o primeiro **Controller**:

- No sub-pacote controller, criar a classe: HelloMvcController

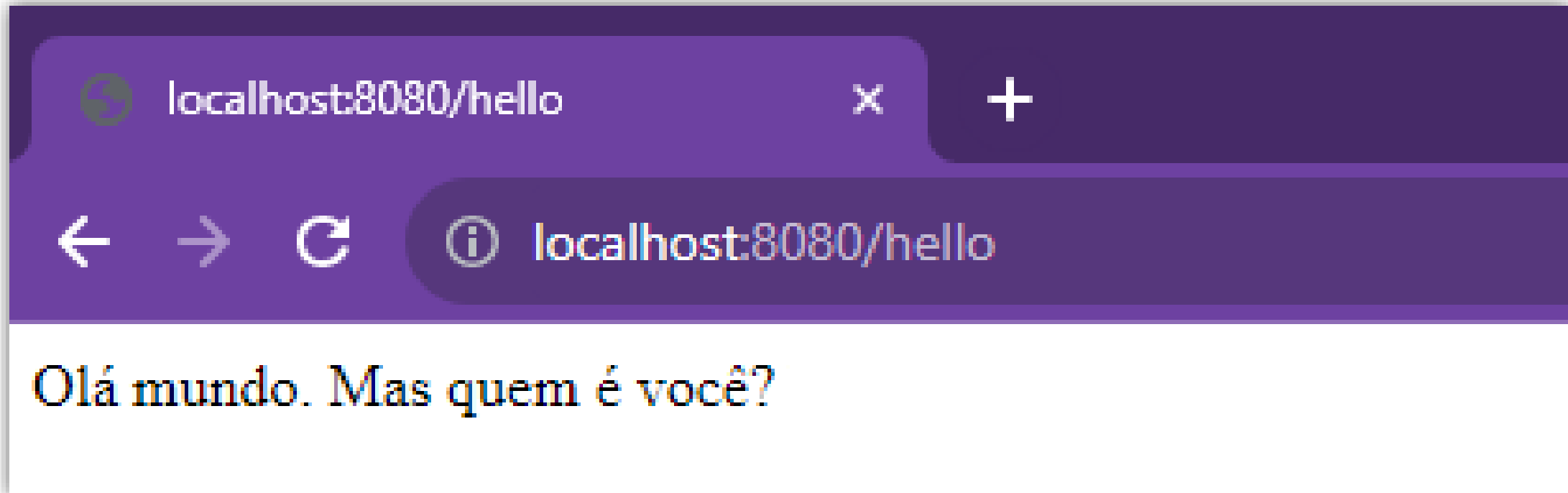
```
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
public class HelloMvcController {

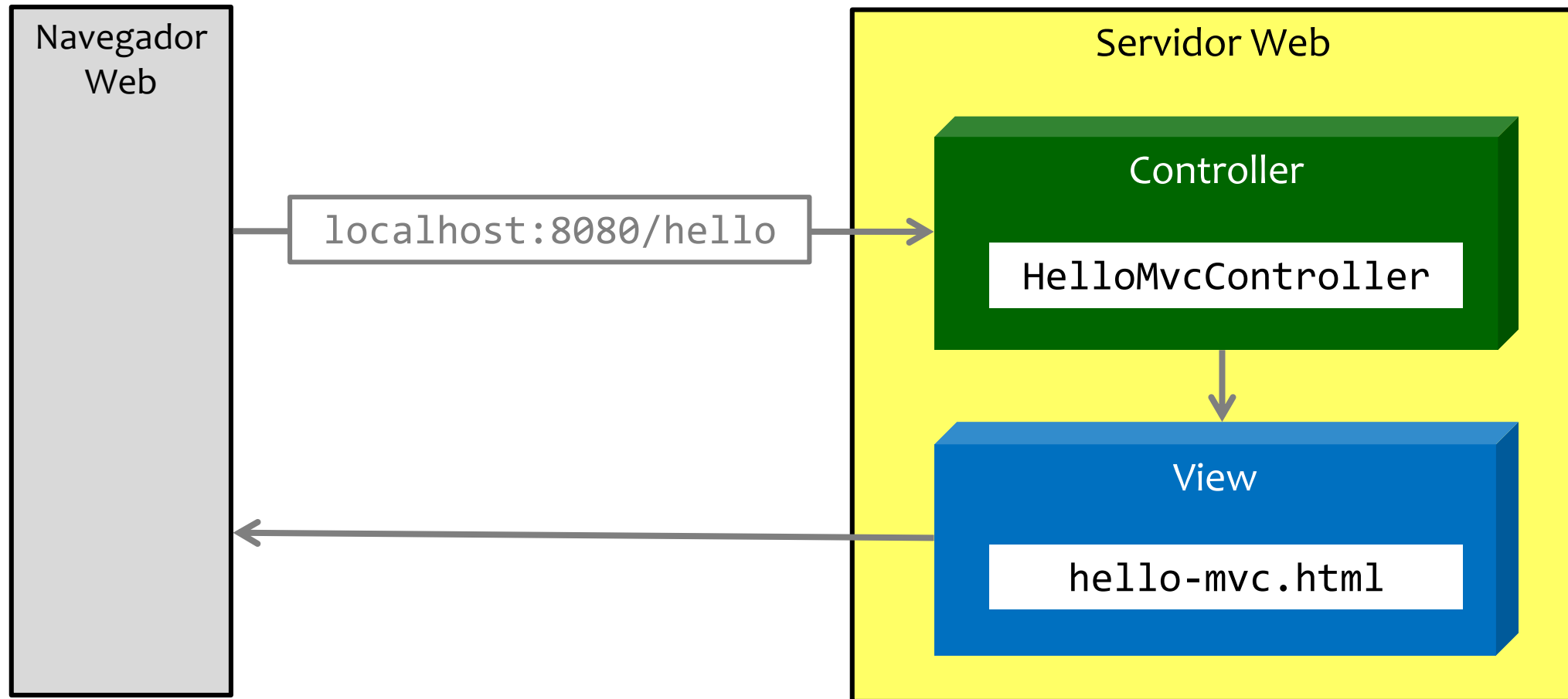
    @GetMapping("/hello")
    public String getHello() {
        return "hello-mvc"; // Não precisa colocar a extensão '.html'
        // O que acontece aqui?
        // Será buscado na pasta 'templates' o arquivo o 'hello-mvc.html',
        // cujo conteúdo será o corpo da resposta
    }
}
```


d)No navegador:

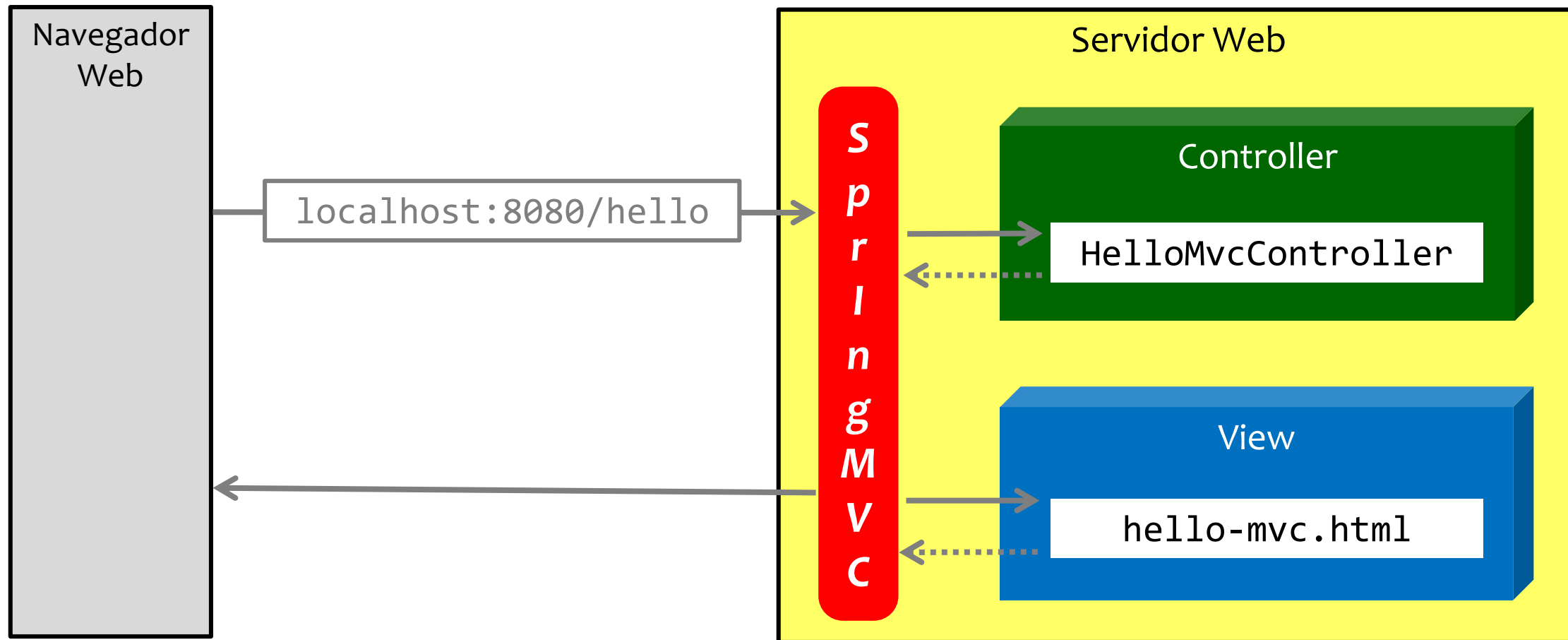
- acessar localhost:8080/hello



O que parece que aconteceu...



O que realmente aconteceu



Funcionou, mas e o conteúdo dinâmico?

Existe uma maneira de passar atributos para ser renderizados na VIEW:

1) No método do controller, declarar da seguinte maneira:

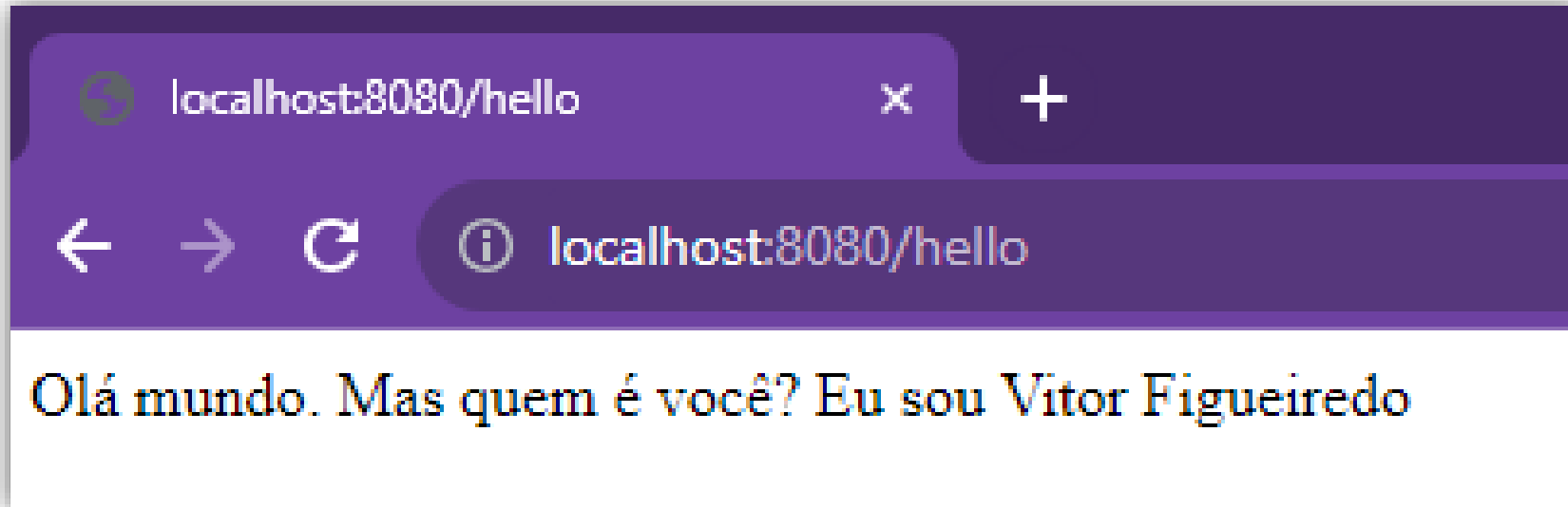
```
@GetMapping("/hello")  
public String getHello(Model model) {  
  
    String nomeUsuario = "Vitor Figueiredo";  
    model.addAttribute("usuario", nomeUsuario );  
    return "hello-mvc"; // Não precisa colocar a extensão '.html'  
}
```

2) Na VIEW, é possível acessar este atributo através diretivas do **Thymeleaf** e de **Expression Language** `${...}`

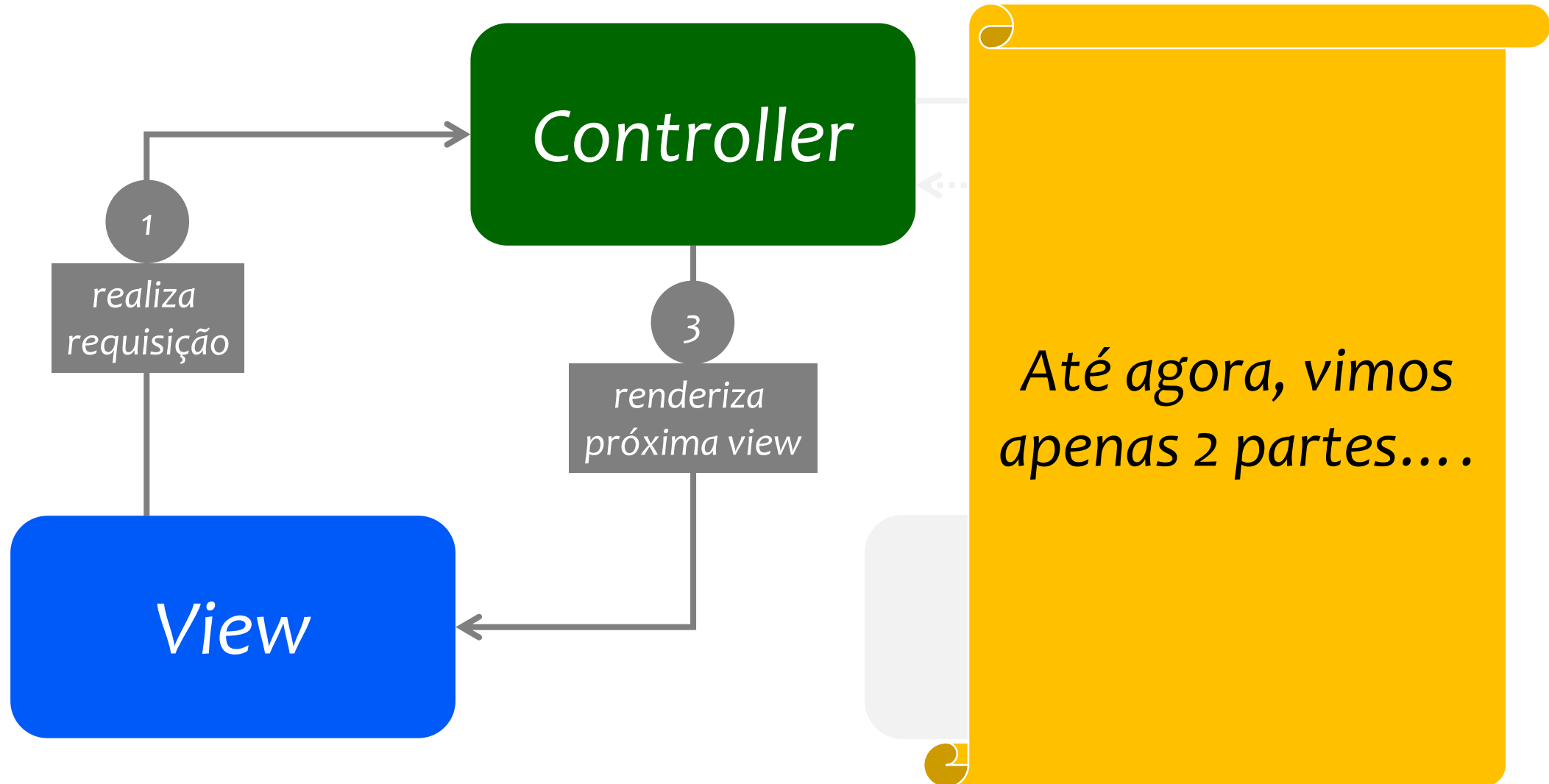
hello-mvc.html

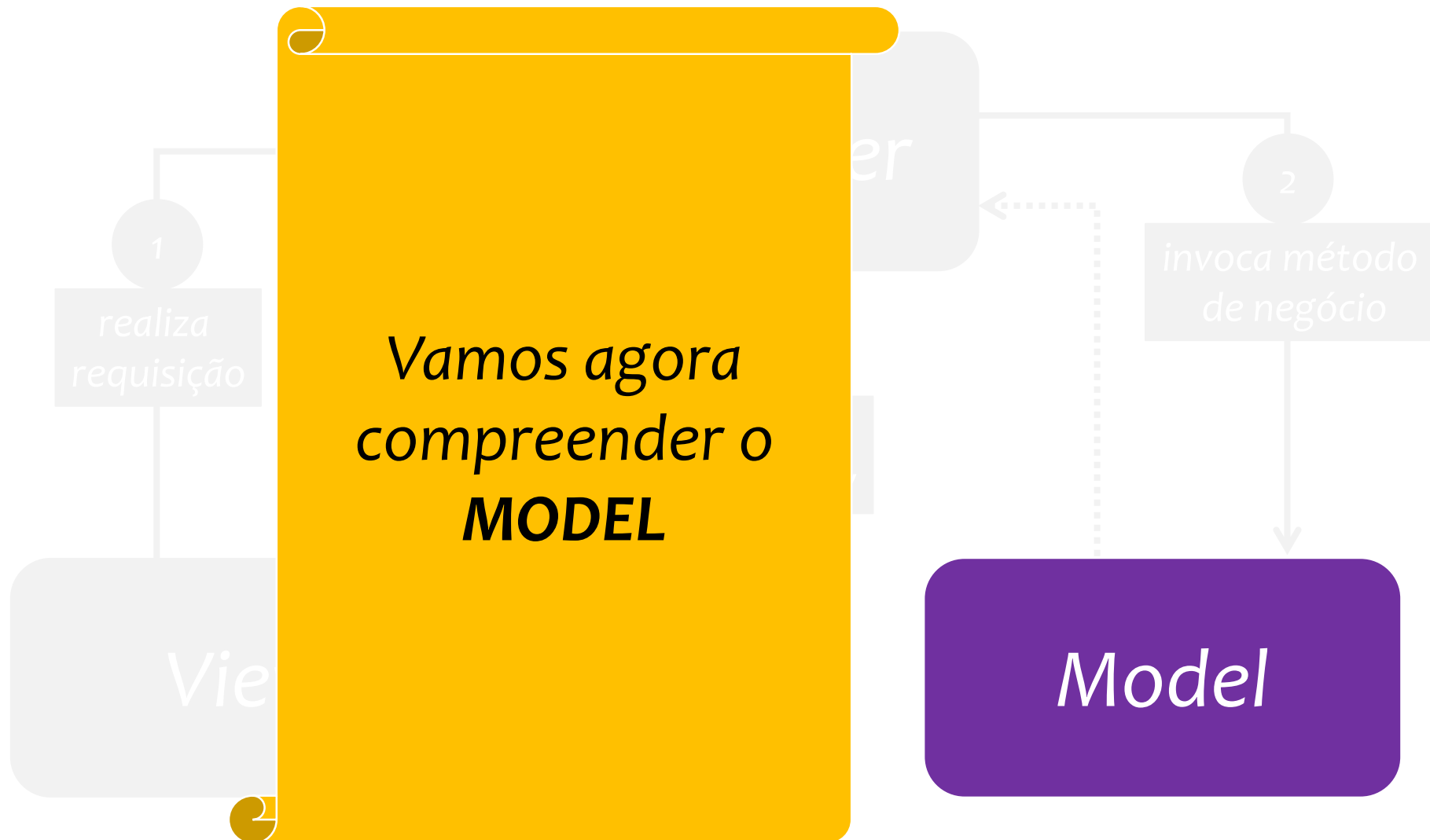
```
<html>
  <head>
    <meta charset="UTF-8">
  </head>
  <body>
    Olá mundo. Mas quem é você?
    Eu sou <span th:text="${usuario}"></span>
  </body>
</html>
```

3) Como resultado, teremos:



**Aqui, é um conteúdo 'hard-code'.
Mas poderia ser um valor vindo do banco de dados, ou de um arquivo, ...**





Model é sub-dividido

Model

Service

Entity

Regras de negócio

Model

Service

Classes de dados

Entity

Exemplo: Produtos

Controller

ProdutoController

View

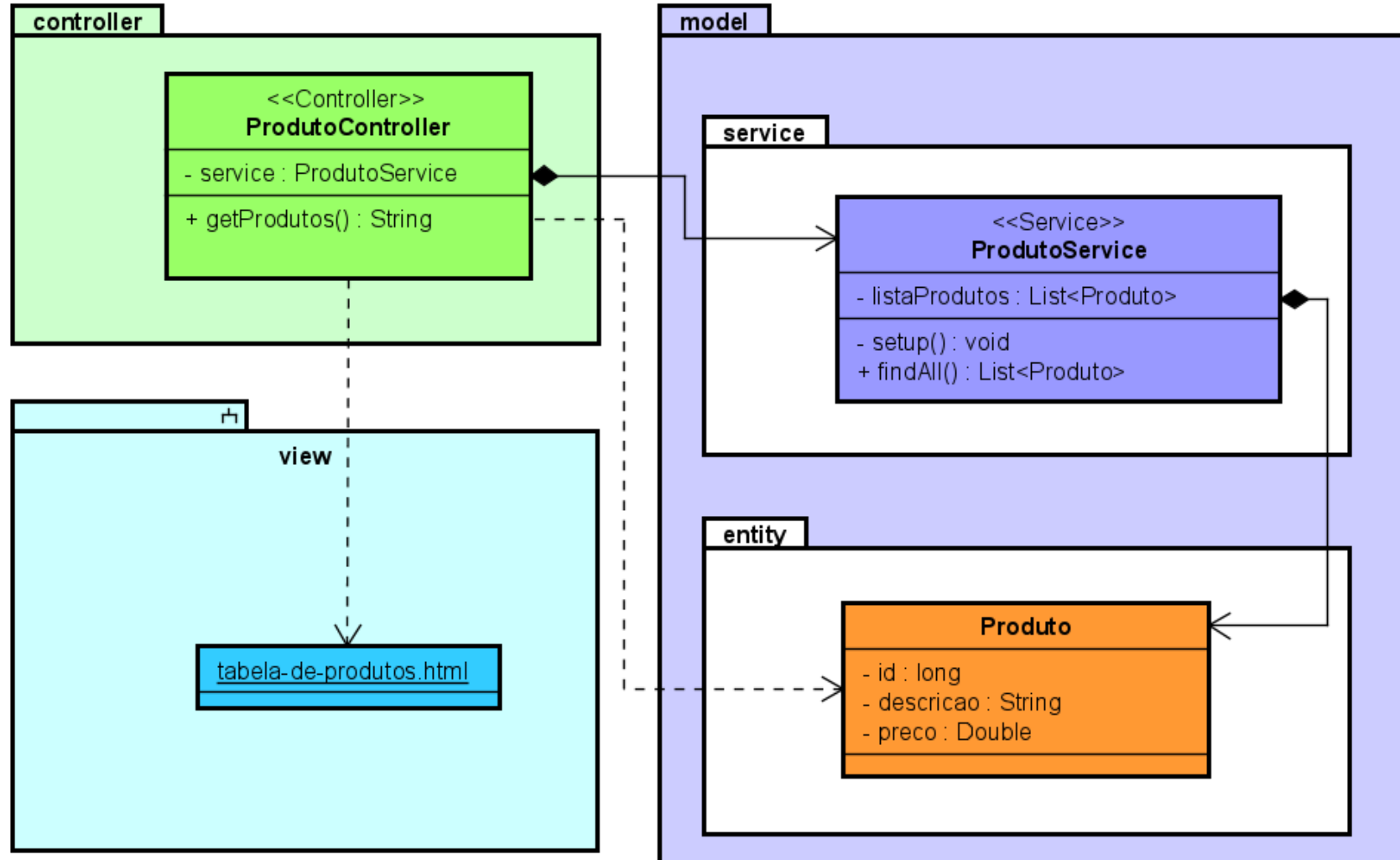
tabela-de-produtos.html

Model

ProdutoService

Produto

Exemplo: Produtos



a) Criar: `model.entity.Produto`

```
public class Produto {  
  
    private Long id;  
  
    private String descricao;  
  
    private Double preco;  
  
    //gerar:  
    // construtor completo  
    // construtor padrão  
    // getters  
    // equals() e hashCode()  
}
```

b) Criar: `model.service.ProdutoService`

```
@Service
public class ProdutoService {

    private List<Produto> listaProdutos = new ArrayList<Produto>();

    @PostConstruct
    private void setup() {
        Produto p1 = new Produto(1L, "Furadeira", 300.00);
        Produto p2 = new Produto(2L, "Lixadeira", 200.00);
        Produto p3 = new Produto(3L, "Serra circular", 500.00);
        listaProdutos.add( p1 );
        listaProdutos.add( p2 );
        listaProdutos.add( p3 );
    }

    public List<Produto> findAll() {
        return this.listaProdutos;
    }

}
```

c) Criar: controller. ProdutoController

```
@Controller
public class ProdutoController {

    @Autowired
    private ProdutoService service;

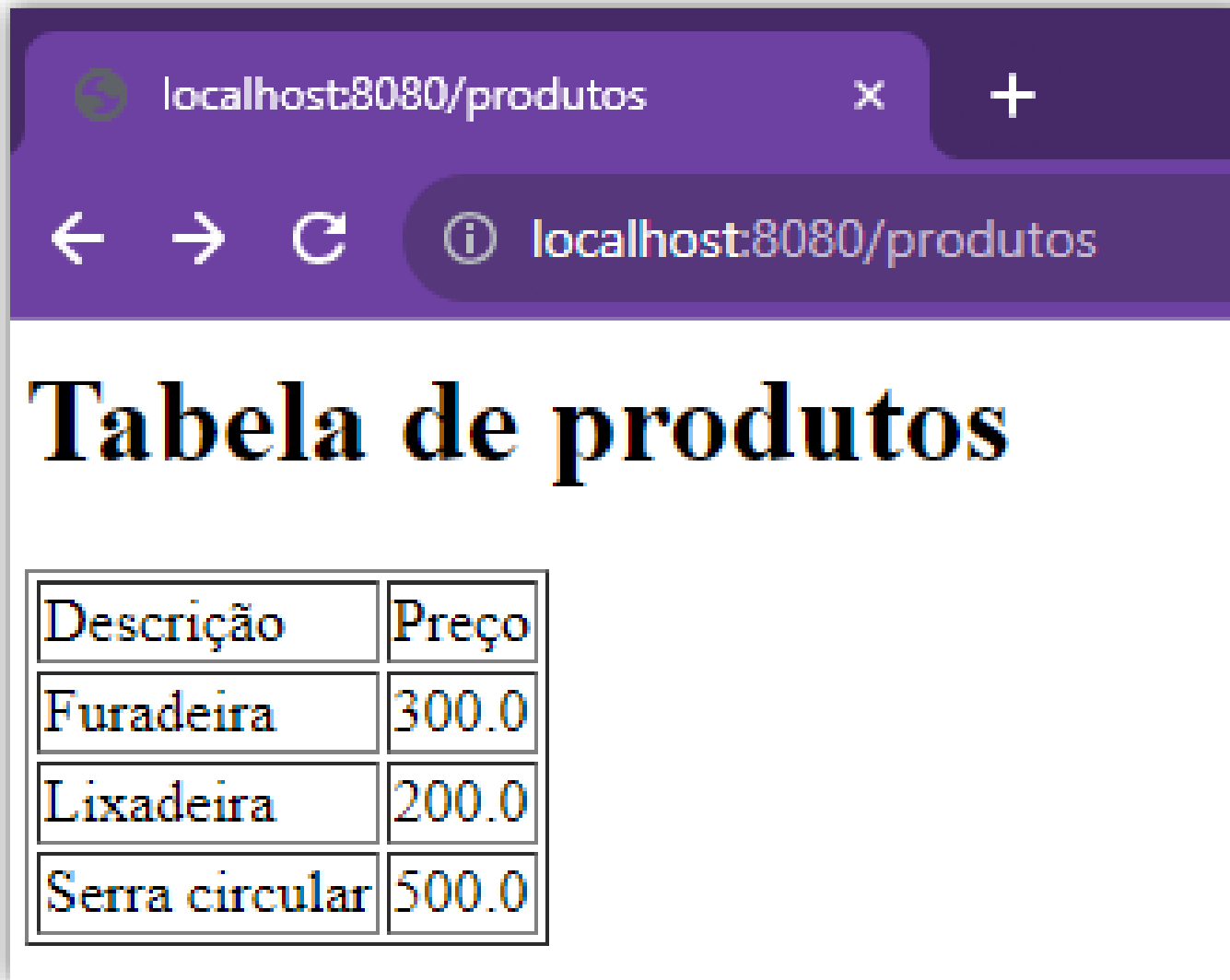
    @GetMapping("/produtos")
    public String getProdutos(Model model) {
        List<Produto> listaProdutos = service.findAll();
        model.addAttribute("listaProdutos", listaProdutos);
        return "tabela-de-produtos";
    }
}
```


d) Criar em **templates**: tabela-de-produtos.html

tabela-de-produtos.html

```
<body>
  <h1>Tabela de produtos</h1>
  <table border="1">
    <tr>
      <td>Descrição</td>
      <td>Preço</td>
    </tr>
    <tr th:each="pvar:${listaProdutos}">
      <td th:text="${pvar.descricao}"></td>
      <td th:text="${pvar.preco}"></td>
    </tr>
  </table>
</body>
```

e) Abrir o navegador e acessar: **localhost:8080/produtos**



localhost:8080/produtos

← → ↻ ⓘ localhost:8080/produtos

Tabela de produtos

Descrição	Preço
Furadeira	300.0
Lixadeira	200.0
Serra circular	500.0

DESAFIO: 4) Produtos

Adicionar a coluna 'id' na tabela de produtos

Arquitetura usada pelo Django

- **Model:** Mapeamento do banco de dados para o projeto;
- **Template:** Páginas para visualização de dados. Normalmente, é aqui que fica o HTML que será renderizado nos navegadores;
- **View:** Lógica de negócio. É aqui que determinamos o que irá acontecer em nosso projeto.

