DESENVOLVIMENTO WEB BACK-END

AULA 3

Profª Luciane Yanase Hirabara Kanashiro

CONVERSA INICIAL

Nesta etapa, veremos o desenvolvimento Java *web* com Spring MVC. Ao adentrarmos o universo do desenvolvimento *web* com Spring MVC, deparamo-nos com diversos conceitos fundamentais. Esses conceitos estão dispostos em cinco temas, brevemente descritos a seguir.

A camada de visão (tema 1) representa a camada responsável por apresentar os dados ao usuário, sendo frequentemente associada a páginas HTML. Estas últimas, escritas em HTML (tema 3) puro ou em conjunto com o Thymeleaf (tema 2), formam a estrutura visual de nossas aplicações. O Thymeleaf, por sua vez, é um motor de *template* que simplifica a criação de páginas dinâmicas, permitindo a incorporação de expressões e lógica de servidor diretamente nos *templates* HTML. *Front controller* (tema 4) é um padrão de *design* comumente utilizado em aplicações *web* para centralizar a gestão das requisições do cliente. Em vez de cada requisição ser tratada por um *script* separado, o *front controller* recebe todas as requisições e decide como lidar com elas, geralmente roteando-as para o controlador apropriado. Por fim, a camada de controle (Tema 5) atua como o ponto central, recebendo requisições do cliente, coordenando a lógica de negócios e preparando os dados para a visão, contribuindo para a estruturação coesa e eficiente de nossas aplicações *web*.

TEMA 1 – CAMADA DE VISÃO

Nesta etapa, vamos rever as características da camada de visão (*view*). Já vimos muitas características dessa camada, mas agora as veremos com mais detalhes, no contexto do Spring MVC (Model-View-Controller), *framework* do ecossistema Spring.

Figura 1 – Spring

Fonte: Spring, 2024d.

A camada de visão é a camada responsável pela interface gráfica e interação com o usuário em um modelo MVC. A camada de visão no Spring MVC

tem algumas características específicas que a diferenciam e que são fundamentais para o desenvolvimento eficiente de interfaces de usuário. Uma das principais características é a integração com JSP (JavaServer Pages) e Thymeleaf. O Spring MVC oferece suporte para diferentes tecnologias de *template* para a camada de visão. Como mencionado, JSP e Thymeleaf são algumas das opções suportadas. Isso proporciona flexibilidade na escolha da tecnologia de visualização, dependendo das necessidades do projeto.

Spring MVC define as interfaces **ViewResolver** e **View**, que permitem renderizar modelos em um navegador sem vincular você a uma tecnologia de visualização específica. **ViewResolver** fornece um mapeamento entre nomes de visualizações e visualizações reais. A **View** aborda a preparação de dados antes de passar para uma tecnologia de visualização específica.

Além das características citadas, temos ainda como principais características da camada de visão do Spring as seguintes:

- **ViewResolver**: já citada anteriormente, mas colocada aqui para dar ênfase a essa que é uma das características-chave do Spring MVC. O Spring MVC utiliza um mecanismo de **resolução de vistas** (*view resolver* em inglês) para encontrar e renderizar as páginas corretamente. Ele facilita a associação entre as visões lógicas definidas no controlador e os recursos reais que representam essas visões.
- **Suporte a internacionalização**: o Spring MVC oferece suporte integrado para internacionalização (i18n) e localização (l10n). Isso facilita a criação de aplicativos que podem fornecer conteúdo em diferentes idiomas e regiões.
- **Objeto Model**: a camada de visão tem acesso ao objeto Model, que contém dados que devem ser exibidos na interface do usuário. Esse objeto é preenchido pelo controlador e disponibilizado para a camada de visão para renderização.
- **Expressões Spring EL**: a camada de visão pode beneficiar-se do uso de expressões Spring EL (Expression Language) para acessar dados no modelo e tomar decisões de exibição com base nesses dados.
- **Suporte a temas e padrões de *design***: o Spring MVC permite a definição de temas e o uso de folhas de estilo para melhorar o aspecto visual das páginas. Isso facilita a aplicação consistente de padrões de *design.*

- **Suporte a *upload* de arquivos**: o Spring MVC facilita o *upload* de arquivos, permitindo que a camada de visão trate eficientemente a entrada do usuário para envio de arquivos.
- **Manuseio de exceções**: o Spring MVC oferece maneiras de lidar com exceções durante a renderização da visão, permitindo, por exemplo, a configuração de páginas de erro personalizadas.
- **Integração com outras tecnologias do ecossistema Spring**: a camada de visão no Spring MVC é facilmente integrada com outras tecnologias do ecossistema Spring, como Spring Boot, Spring Security e Spring Data.

Essas características fazem do Spring MVC uma escolha popular para o desenvolvimento de aplicações *web* em Java, proporcionando um ambiente flexível e robusto para a implementação da camada de visão em aplicações *web*.

TEMA 2 – THYMELEAF

O Thymeleaf é um *software* de código aberto criado em 2011 por Daniel Fernández. Atualmente é desenvolvido e mantido por um grupo de pessoas "incríveis", de acordo com a página do próprio desenvolvedor. Ele não é feito nem apoiado por nenhuma empresa de *software* (ou qualquer outro tipo de empresa), sendo oferecido de forma totalmente gratuita.

O Thymeleaf é uma ferramenta popular para o desenvolvimento de aplicações *web* Java, especialmente em conjunto com o *framework* Spring.

O *software* destaca-se por sua sintaxe amigável ao desenvolvedor, o que facilita a criação de páginas HTML, pois utiliza uma sintaxe que se assemelha ao HTML puro, e a linguagem pode ser incorporada em páginas HTML. O Thymeleaf é, ainda, de fácil compreensão para *designers* e desenvolvedores.

Figura 2 – O icônico emblema do Thymeleaf

Fonte: Thymeleaf, 2024b.

Segundo o *site* do Thymeleaf:

> Thymeleaf é um mecanismo de modelo Java moderno do lado do servidor para ambientes *web* e autônomos. O principal objetivo do Thymeleaf é trazer modelos elegantes e naturais para o seu fluxo de trabalho de desenvolvimento — HTML que podem ser exibidos corretamente em navegadores e funcionar como protótipos estáticos, permitindo uma colaboração mais forte nas equipes de desenvolvimento.
> Com módulos para Spring Framework, uma série de integrações com suas ferramentas favoritas e a capacidade de conectar suas próprias funcionalidades, o Thymeleaf é ideal para o desenvolvimento *web* JVM HTML5 moderno – embora possa fazer muito mais. (Thymeleaf, 2024b, tradução nossa)

Listamos a seguir as principais características e funcionalidades do Thymeleaf.

- **Sintaxe simples**: o Thymeleaf usa uma sintaxe simples e natural, que se assemelha ao HTML puro, tornando os *templates* fáceis de ler e escrever. Essa abordagem facilita a colaboração entre desenvolvedores e *designers*.
- **Suporte a expressões**: o Thymeleaf suporta expressões em seu código, permitindo a manipulação de variáveis, iteração sobre coleções, condicionais, dentre outras operações. As expressões são incorporadas diretamente no HTML e são avaliadas pelo Thymeleaf durante a renderização.
- **Integração com o Spring**: é comumente utilizado em conjunto com o *framework* Spring MVC para o desenvolvimento de aplicações *web* em Java. O Spring fornece suporte nativo para o Thymeleaf, tornando a configuração e o uso conjuntos simples e eficientes.
- **Suporte a internacionalização**: o Thymeleaf facilita a internacionalização de aplicações, permitindo a inclusão de mensagens localizadas diretamente nos *templates*.
- **Processamento no lado do servidor**: diferentemente de alguns motores de *template* que realizam processamento no lado do cliente (navegador), o Thymeleaf é processado no lado do servidor. Isso significa que as páginas são geradas no servidor antes de serem enviadas para o navegador, proporcionando melhor desempenho e suporte a SEO.
- **Facilidade de integração com recursos estáticos**: o Thymeleaf facilita a referência e a integração com recursos estáticos, como folhas de estilo

e *scripts* JavaScript, simplificando o desenvolvimento de interfaces de usuário ricas.

- **Modularidade e reutilização**: permite a criação de fragmentos de *templates* (como cabeçalhos e rodapés) que podem ser reutilizados em várias páginas, promovendo a modularidade e facilitando a manutenção.
- **Extensibilidade**: o Thymeleaf é extensível, o que significa que é possível criar e utilizar atributos ou dialetos personalizados para atender a requisitos específicos.
- **Suporte a testes**: facilita a realização de testes, permitindo a verificação visual dos *templates* e a validação de expressões.

Cabe ressaltar que Thymeleaf é um *template engine* (mecanismo de *template* ou mecanismo de modelo) e não um *framework*. Um *template engine* permite que os desenvolvedores combinem dados dinâmicos (como variáveis, listas e objetos) com marcação estática (como HTML, XML ou texto) para produzir conteúdo personalizado para cada requisição.

TEMA 3 – HTML

Páginas HTML (HyperText Markup Language) são documentos utilizados na construção de páginas *web*. Elas contêm marcações, chamadas de *tags*, que definem a estrutura e o conteúdo da página, como texto, imagens, *links*, formulários, entre outros elementos.

Figura 3 – Ícone do HTML 5

HTML5 representa a versão mais atualizada da HTML, introduzindo novos recursos de marcação, como suporte a multimídia, e incluindo novas *tags* e elementos, bem como o desenvolvimento de novas APIs. Além disso, uma das inovações proeminentes do HTML5 é sua capacidade de incorporar conteúdo de áudio e vídeo diretamente no navegador.

As páginas HTML são interpretadas pelos navegadores da *web*, que as renderizam visualmente para os usuários. No modelo MVC as páginas HTML ficam na camada de visão.

A seguir temos um exemplo de código de página HTML.

```
<!DOCTYPE HTML>
<HTML lang="pt-br">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
    <title>Minha Página HTML</title>
</head>
<body>
    <header>
        <h1>Minha Página HTML</h1>
    </header>
    <div>
        <p>Olá, mundo! Esta é uma página HTML simples.</p>
    </div>
</body>
</HTML>
```

As páginas HTML com Thymeleaf são documentos HTML que incorporam expressões Thymeleaf e são processados no lado do servidor, permitindo a criação de páginas *web* dinâmicas e a integração eficiente com lógica de servidor em aplicações Java.

A utilização de páginas HTML com Thymeleaf envolve vários aspectos, descritos a seguir.

- **Integração com Spring MVC**: o Thymeleaf é frequentemente usado em conjunto com o *framework* Spring MVC. Em uma aplicação Spring MVC, as páginas HTML do Thymeleaf são colocadas no diretório de recursos (geralmente em “src/main/resources/templates”) e tratadas como *templates* que podem ser processados pelo Thymeleaf durante a renderização.
- **Sintaxe Thymeleaf**: o Thymeleaf utiliza uma sintaxe especial para incorporar expressões e lógica de servidor nos *templates* HTML. Essa sintaxe é reconhecida pelo Thymeleaf durante o processamento no lado do servidor.

Exemplo de expressão Thymeleaf incorporada em uma página HTML:

```
<p> Olá, <span th:text="${usuario.nome}">Visitante</span> !
</p>
```

Nesse exemplo, a expressão ${usuario.nome} é avaliada no lado do servidor e o resultado é inserido na página HTML.

- **Expressões e atributos Thymeleaf**: além de incorporar dados dinâmicos nas páginas HTML, o Thymeleaf suporta uma variedade de expressões e atributos especiais que facilitam a manipulação de dados, o controle de fluxo e outras operações lógicas.
- **Processamento no lado do servidor**: as páginas HTML com Thymeleaf são processadas no lado do servidor antes de serem enviadas ao navegador. Isso permite a execução de lógica de servidor e a manipulação de dados antes da renderização final.

Ferramentas para desenvolvimento: existem ferramentas e *plugins* que oferecem suporte ao desenvolvimento com Thymeleaf em ambientes de desenvolvimento integrado (IDEs – *integrated development environments*), proporcionando destaque de sintaxe, autocomplemento e outras funcionalidades úteis.

- **Facilidade de integração com outras tecnologias**: o Thymeleaf é facilmente integrado com outras tecnologias *web*, como CSS (Cascade Style Sheet), JavaScript e *frameworks* JavaScript, proporcionando uma abordagem aberta para o desenvolvimento *web*.

Veremos a seguir um exemplo de página HTML com Thymeleaf. Para este exemplo vamos considerar também uma classe Java, chamada Usuario, com dois atributos (*nome e e-mail*) do tipo String. Observe que a classe Java em questão é um POJO (Plain Old Java Object), uma classe Java simples. Os POJOs foram discutidos em etapas anteriores. Os métodos de captura e configuração (*getters* e *setters*), assim como os métodos construtores, não foram implementados, pois não são códigos relevantes para o contexto deste exemplo.

Classe Java Usuario

```
public class Usuario {
    private String nome;
    private String email;
```

```
    // getters e setters
}
```

Logo abaixo temos a implementação da página HTML com Thymeleaf. O código em questão refere-se a um formulário HTML para editar ou adicionar um novo usuário.

Especial atenção deve ser dada ao conteúdo da *tag* HTML que associa o **namespace Thymeleaf** à *tag* HTML. Um **namespace** é uma forma de organizar elementos em um sistema, geralmente usado em programação para evitar conflitos de nomes e permitir a modularidade e o reúso de código.

Em contextos relacionados à *web*, **namespaces** são usados para evitar conflitos de nomeação em elementos e atributos. Isso permite o uso de expressões Thymeleaf dentro do documento HTML, proporcionando uma integração eficaz entre o HTML e o Thymeleaf.

```
<HTML xmlns:th="http://www.thymeleaf.org">
```

O atributo `xmlns:th` define um namespace com o nome `th` que aponta para a URL `http://www.thymeleaf.org`, permitindo o uso de atributos e expressões específicos do Thymeleaf dentro do documento HTML.

Vejamos a seguir a codificação do formulário HTML com Thymeleaf.

```
<!DOCTYPE HTML>
<HTML xmlns:th="http://www.thymeleaf.org">
<head>
    <title>Formulário de Usuário</title>
</head>
<body>

    <h2>Formulário de Usuário</h2>

    <form th:object="${usuario}" th:action="@{/salvar}"
method="post">
        <label for="nome">Nome:</label>
        <input type="text" id="nome" th:field="*{nome}" />

        <br/>

        <label for="email">Email:</label>
      <input type="text" id="email" th:field="*{email}" />

        <br/>
```

```
        <button type="submit">Salvar</button>
    </form>

</body>
</HTML>
```

O Quadro 1 resume a explicação dos comandos utilizados no exemplo fornecido nessa codificação.

Quadro 1 – Comandos do Thymeleaf e significados

Comando	Significado
`th:object="${usuario}"`	Define o objeto de dados (Usuario) associado ao formulário.
`th:action="@{/salvar}"`	Especifica a URL para onde o formulário será enviado ao ser submetido. Isso normalmente corresponderia a um controlador Spring.
`th:field="*{nome}"`	Vincula o campo de entrada ao atributo *nome* do objeto Usuario.
`th:field="*{email}"`	Vincula o campo de entrada ao atributo *email* do objeto Usuario.
`type="submit"`	Indica que este é um botão de envio do formulário.

Ao processar esse formulário no lado do servidor com o Spring, você normalmente teria um controlador que manipula a submissão do formulário e executa a lógica necessária.

Além disso, é importante configurar o Spring e o Thymeleaf corretamente em sua aplicação para que o Thymeleaf seja processado. É importante ainda certificar-se de ter as dependências corretas e as configurações adequadas no arquivo application.properties.

Como nosso foco não é o *front-end* da aplicação, não aprofundaremos em detalhes a implementação com HTML e Thymeleaf, mas, caso deseje aprofundar-se na sintaxe, você pode consultar a página oficial do Thymeleaf (disponível em: <https://www.thymeleaf.org/>; acesso em: 4 abr. 2024).

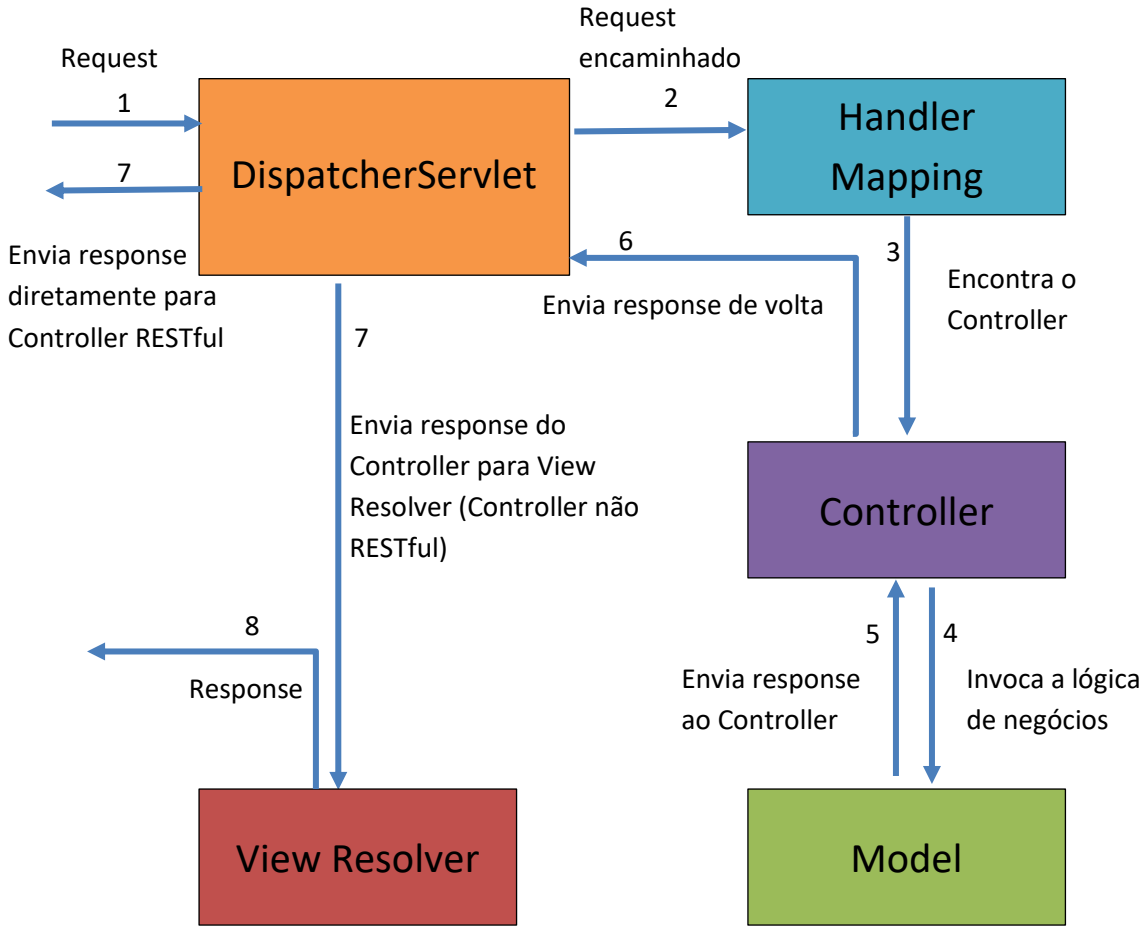
TEMA 4 – *FRONT CONTROLLER*

Vamos começar este tópico retrocedendo um pouco e examinando o conceito de *front controller* na arquitetura típica do Spring MVC.

Front controller é um **padrão de *design* arquitetural** em que um componente centralizado, como o DispatcherServlet no Spring MVC, é responsável por receber todas as requisições HTTP em uma aplicação *web* e encaminhá-las para os *controllers* apropriados com base em suas configurações de mapeamento. Ele atua como o primeiro ponto de contato para todas as requisições, decidindo como cada requisição será tratada pela aplicação.

A Figura 4 apresenta um diagrama rápido para o fluxo de alto nível no Spring MVC. Nesse diagrama há cinco retângulos, nos quais temos as representações de **DispatcherServlet**, **Handler Mapping, Controller, Model** e o **View Resolver**.

Figura 4 – Diagrama de fluxo – Spring MVC



Fonte: Elaborada por Kanashiro, 2024, com base em Baeldung 2024.

No diagrama as setas representam o fluxo da requisição (*request*) e os números fazem referência à ordem em que o fluxo acontece. É importante frisar ainda que a sequência representada pelo número 7 pode seguir um dos dois caminhos (para o *controller* RESTful ou para View Resolver). Por essa razão existem dois fluxos representados pelo número 7.

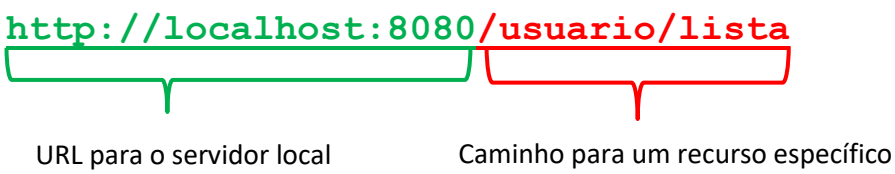
No Quadro 2 estão listados os fluxos e as respectivas sequências, apresentadas no diagrama de fluxo simplificado do Spring MVC (Figura 1).

Quadro 2 – Sequência do fluxo

Sequência	Fluxo
1	É feita uma *request* ao **DispatcherServlet**.
2	*Request* encaminhada ao **Handler Mapping**.
3	Encontra o **Controller** mapeado (ou seja, encontra o *controller* específico).
4	Invoca a lógica de negócios (no **Model**).
5	Envia *response* ao **Controller**.
6	O **Controller** envia *response* de volta ao **DispatcherServlet**.
7	O **DispatcherServlet** envia *response* para um **Controller RESTful.**
7	Envia *response* do *controller* para **View Resolver** (Controller não RESTful).
8	O **View Resolver** envia *response* para um ambiente externo.

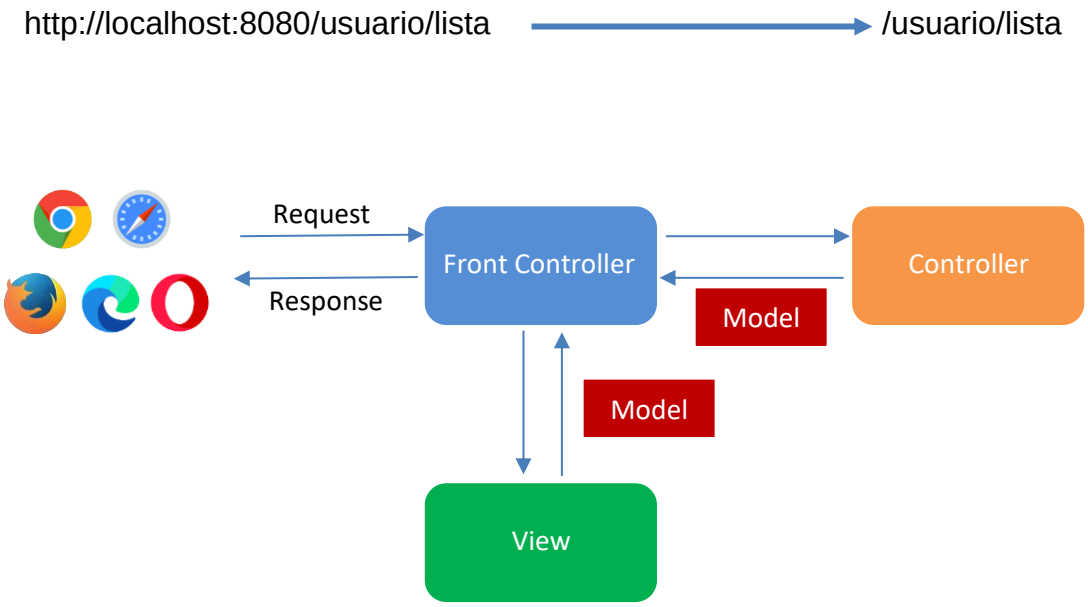
Olhando para trás e resgatando o que aprendemos anteriormente, temos a ilustração do funcionamento do Spring MVC, para relembrarmos onde o *front controller* atua, funcionando como um DispatcherServlet, ou seja, recebendo todas as requisições HTTP e encaminhando-as para os *controllers* apropriados com base em suas configurações de mapeamento.

No esquema a seguir a requisição feita pelo navegador é representada pela URL.



Para a URL apresentada, o que importa para o *front controller* é o que há depois da / (barra) – no caso, usuario/lista –, que nos levará ao *controller* do usuario. Desse modo, o **Controller** vai saber para qual *controller* mandar a solicitação

Figura 5 – Esquema de funcionamento do Spring MVC



Como pudemos perceber, o Spring MVC é projetado em torno desse padrão *front controller*, onde um *servlet* central, o DispatcherServlet, fornece um algoritmo compartilhado para processamento de solicitações, enquanto o trabalho real é executado por componentes delegados configuráveis.

O DispatcherServlet, como qualquer *servlet*, precisa ser declarado e mapeado de acordo com a especificação do *servlet* usando-se configuração Java. No entanto, ao se utilizar Spring Boot, não é necessário declarar explicitamente o DispatcherServlet no arquivo de configuração, como em uma aplicação Spring MVC tradicional. O Spring Boot fornece configuração automática para o DispatcherServlet e outras configurações comuns, o que simplifica bastante o desenvolvimento de aplicações *web*. O Spring Boot foi abordado em etapas anteriores.

TEMA 5 – CAMADA DE CONTROLE

No contexto do Spring MVC, uma camada de controle, ou simplesmente um *controlador* (ou *controller*), é um componente responsável por receber as requisições do cliente, processá-las e coordenar a resposta.

Os controladores são uma parte crucial do padrão arquitetural MVC, que separa a lógica de apresentação (View) e a lógica de negócios (Model) por meio de uma camada intermediária de controle.

Listamos a seguir as principais funções de uma camada de controle no Spring MVC.

- **Receber requisições**: os controladores são responsáveis por receber as requisições HTTP vindas do cliente. Isso inclui a interpretação dos parâmetros da requisição, como dados do formulário, parâmetros de URL, cabeçalhos e outros.
- **Chamar serviços e lógica de negócios**: os controladores podem interagir com classes de serviço (ou serviços) que contêm a lógica de negócios da aplicação. Isso permite que a lógica de apresentação (controlador) permaneça desacoplada da lógica de negócios, seguindo o princípio de separação de responsabilidades.
- **Preparar dados para a View**: uma das tarefas principais de um controlador é preparar os dados necessários para a renderização pela View. Ele consulta o modelo (Model) ou outros serviços para obter dados e os disponibiliza para a camada de visualização.
- **Selecionar a View adequada**: com base na lógica de negócios processada e nos dados preparados, o controlador seleciona a View adequada para exibição. O Spring MVC utiliza um mecanismo de resolução de vistas para determinar qual View será usada.
- **Responder ao cliente**: após a execução da lógica de negócios e a preparação dos dados, o controlador é responsável por enviar a resposta adequada de volta ao cliente. Isso pode incluir a renderização de uma página HTML, o redirecionamento para outra URL ou até mesmo a entrega de dados em formato JSON para aplicações RESTful. O formato de dados JSON (JavaScript Object Notation) e o RESTful serão abordados posteriormente.

Na implementação do código a seguir temos um exemplo básico de uma camada de controle no Spring MVC. Observe que foram utilizadas várias **anotações** (metadados usados para fornecer instruções ao compilador, ao tempo de execução ou a outras ferramentas sobre como o código deve ser processado ou tratado.

Classe ExemploController

```
@Controller
@RequestMapping("/exemplo")
public class ExemploController {
```

```
    @Autowired
    private ExemploService exemploService;

    @GetMapping("/pagina")
    public String exibirPagina(Model model) {
        List<Exemplo> exemplos =
exemploService.obterTodosExemplos();
        model.addAttribute("exemplos", exemplos);
        return "pagina";
    }

    @PostMapping("/salvar")
    public String salvarExemplo(@ModelAttribute("exemplo")
Exemplo exemplo) {
        exemploService.salvarExemplo(exemplo);
        return "redirect:/exemplo/pagina";
    }
}
```

O Quadro 3 apresenta as anotações utilizadas nesse exemplo de camada de controle e uma breve explicação de cada uma delas.

Quadro 3 – Anotações

Anotação	Explicação
@Controller	Indica que a classe é um controlador do Spring MVC.
@RequestMapping	Especifica o mapeamento de URL base para todas as requisições tratadas por esse controlador, ou seja, define o mapeamento de requisições para esse controlador. No exemplo em questão, todos os métodos do controlador que estão dentro da classe anotada com **@RequestMapping("/exemplo")** responderão a requisições que começam com **/exemplo.**
@GetMapping	Anotação específica para manipular requisições GET. No exemplo em questão, significa que quando uma requisição GET é feita para o URL /pagina, o método correspondente será chamado para lidar com essa requisição.
@PostMapping	Anotação específica para manipular requisições POST. No caso do exemplo, a anotação

	@PostMapping("/salvar") indica que esse método será chamado quando uma requisição POST for feita para o **URL /exemplo/salvar**.
@ModelAttribute	Anotação utilizada para passar dados do controlador para a View. Nesse método, a anotação **@ModelAttribute("exemplo")** é usada para vincular os dados enviados pelo formulário HTML a um objeto **Exemplo**. Quando um formulário HTML é submetido via método POST, os dados são enviados ao servidor no corpo da requisição. No exemplo em questão, a anotação **@ModelAttribute("exemplo")** indica que o Spring deve tentar preencher um objeto **Exemplo** com os dados do formulário.
@Autowired	Responsável pela injeção de dependência automática. No caso, é utilizado para injetar uma instância de ExemploService, que poderia ser uma classe de serviço contendo a lógica de negócios relacionada a exemplos.

Essa estrutura básica representada no código de exemplo demonstra como um controlador Spring MVC interage com a lógica de negócios, prepara dados para a View e responde às requisições do cliente.

FINALIZANDO

A compreensão e a aplicação eficaz dos conceitos fundamentais discutidos, como a camada de visão, responsável por apresentar informações de forma acessível, as páginas HTML, que estruturam visualmente nossas aplicações, e o poderoso *template engine* Thymeleaf, que torna dinâmica a criação dessas páginas, são essenciais para o sucesso no desenvolvimento *web* com Spring MVC. *Front controller* é um padrão de *design* poderoso para aplicações *web* que centraliza o gerenciamento das requisições do cliente, enquanto a camada de controle atua como orquestradora, garantindo uma interação fluida entre a lógica de negócios e a apresentação. Ao incorporar esses elementos de maneira coesa, construímos aplicações robustas e eficientes que atendem tanto aos requisitos de usabilidade quanto à integridade dos dados, proporcionando uma experiência aprimorada aos usuários finais.

REFERÊNCIAS

BAELDUNG. **Guia rápido para controladores Spring**. Disponível em: <https://www-baeldung-com.translate.goog/spring-controllers?_x_tr_sl=en&_x_tr_tl=pt&_x_tr_hl=pt-BR&_x_tr_pto=wapp>. Acesso em: 4 abr. 2024.

DEITEL, P. J.; DEITEL, H. M. **Java**: como programar. 10. ed. São Paulo: Pearson, 2017.

GAMMA, E.; HELM, R.; JOHNSON, R. **Padrões de projetos**: soluções reutilizáveis de *software* orientado a objetos. Porto Alegre: Bookman, 2007.

ORACLE. **Building Web Components – Chapter 3**: Design Patterns and Frameworks. Disponível em: <https://docs.oracle.com/cd/E19276-01/817-2334/03_design_issues.HTML>. Acesso em: 4 abr. 2024.

SPRING. **Annotation Interface Transactional**. Disponível em: <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.HTML>. Acesso em: 4 abr. 2024a.

_____. **Serving Web Content with Spring MVC**. Disponível em: <https://spring.io/guides/gs/serving-web-content/>. Acesso em: 4 abr. 2024b.

_____. **View Resolution**. Disponível em: <https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-servlet/viewresolver.HTML>. Acesso em: 4 abr. 2024c.

_____. **Spring Trademark Guidelines**. Disponível em: <https://spring.io/trademarks>. Acesso em: 4 abr. 2024d.

THYMELEAF. **Tutorial**: Thymeleaf + Spring. Disponível em: <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.HTML#the-concept>. Acesso em: 4 abr. 2024a.

_____. **Thymeleaf**. Disponível em: < https://www.thymeleaf.org>. Acesso em: 4 abr. 2024b.