Desenvolvimento de Aplicações para WEB

Aula: Desenvolvimento de Aplicações Full Stack Java

Prof. Anderson Augusto Bosing

O Spring MVC

O Spring MVC é um framework que ajuda no desenvolvimento de aplicações web. Com ele nós conseguimos construir aplicações web robustas e flexíveis.

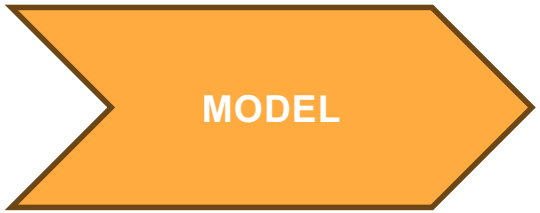
Ele já tem todas as funcionalidades que precisamos para atender as requisições HTTP, delegar responsabilidades de processamento de dados para outros componentes e preparar a resposta que precisa ser dada. É uma excelente implementação do padrão MVC.

Spring MVC

Nesta aula, vamos conhecer alguns conceitos do padrão model, view, controller o qual é implementado pelo Spring MVC.

O padrão MVC tem como característica principal dividir a aplicação em 3 componentes com responsabilidades distintas.

Spring MVC



Define o modelo ou domínio da aplicação(Classes Modelo, Classes de Entidade, Classes de Regras de Negócios(@Service), Classes de Persistências de Dados(@Repository)).

Integração com usuário, Representa a entrada e saída de dados.

Componente Intermediário, Recebe as requisições do usuário, Interage com o modelo em busca da respostas a ser retornada ao usuário.

Por que utilizar o Spring MVC ?

- Poderoso e Moderno

A partir dele podemos desenvolver desde simples até robustas aplicações WEB.

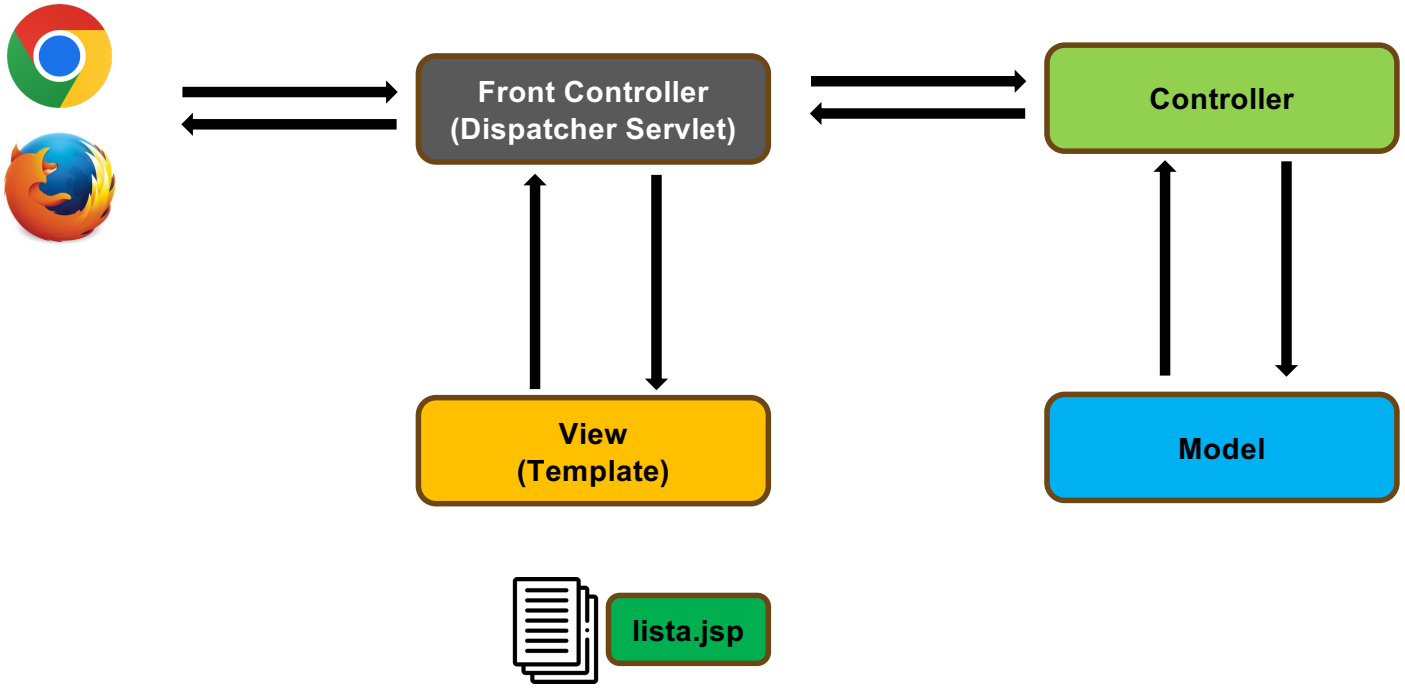
Com uso dos mais atuais recursos da linguagem Java.

Integração com diversos templates WEB(JSP, JSTL, JS, THYMELEAF).

Conversores de Dados entre UI e Modelo.

Validações back-end(Bean Validation).

Como o Spring MVC nos fornece o Padrão MVC?



Action Based vs Component Based (Spring MVC vs JSF)

Frameworks Component Based

Frameworks Component Based mantém sincronia entre os estados dos componentes da view e do seu modelo de dados no lado do servidor.

Quando o usuário interage com a tela, as alterações realizadas são, em um dado momento, refletidas no modelo que fica no servidor.

No JSF, por exemplo, a "tela" é gerada por um facelet, que nada mais é que um XML que define quais componentes serão exibidos para o usuário e associam os valores desses componentes a um objeto (Java Bean) que fica no servidor. Esses componentes são então renderizados em HTML e, quando o usuário executa uma ação, o JSF atualiza os objetos no servidor.

Action Based vs Component Based (Spring MVC vs JSF)

Em frameworks component based, a view é responsável por mapear valores para os beans e para o modelo. A imagem acima ilustra a ordem de chamadas:

1. O usuário executa uma ação no sistema
2. O front controller do framework atualiza os componentes da view com o estado atual
3. O método do Managed Bean é chamado (usando JSF como exemplo), podendo executar alguma regra de negócio com os novos valores
4. Finalmente, o modelo do sistema é atualizado.

Action Based vs Component Based (Spring MVC vs JSF)

Frameworks Action Based

Já os frameworks Action Based não mantém necessariamente esse vínculo entre os estados do servidor e do cliente.

Isso não quer dizer que o desenvolvedor não possa armazenar estado no servidor, por exemplo, na sessão do usuário, mas que o vínculo entre o modelo e a view não é tão acoplado como no modelo Component Based.

Um framework Action Based geralmente irá receber diretamente requisições HTTP. Isso torna o modelo action based mais flexível, já que o desenvolvedor pode optar por qualquer tipo de view que gere uma requisição HTTP compatível

Action Based vs Component Based (Spring MVC vs JSF)

Frameworks Action Based

O resumo dos passos da execução é:

1. **O usuário executa uma ação no sistema**
2. **O front controller do framework direciona a requisição e os parâmetros para um método do controller**
3. **O controller lê os parâmetros necessários e executa regras de negócio que atualizam o modelo**
4. **O controller "devolve" uma view para o usuário**

Action Based vs Component Based (Spring MVC vs JSF)

Podemos dizer que os frameworks component based são mais centrados nas views (com seus componentes que mapeiam o modelo e os dados do usuário), enquanto os action based são mais centrados nos controllers (que recebem parâmetros via request).

Exemplos:

Action based(VRaptor, Struts 2, WebWork, Grails/GSP, Play, Spring Web MVC)

Component based(Wicket, JSF, GWT, Vaadin)

O 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 também 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.

Características do Thymeleaf

Antes de mais nada a principal funcionalidade de um template engine é permitir que linguagens de programação possam ser incorporadas em páginas HTML. de tal forma, uma template engine permite que os programadores possam utilizar estruturas de condição, estruturas de repetição, herança e diversos outros recursos presentes apenas nas linguagens de programação em páginas HTML.

Analogamente o Thymeleaf não é diferente, ele permite que desenvolvedores incorporem código Java em páginas HTML e também utilizem as principais características da linguagem em seus templates.

Dentre diversas características, podemos citar as principais:

- Permite o uso de estruturas de condição e repetição em páginas HTML;
- Com o Thymeleaf é possível utilizar herança de layouts, garantindo uma estrutura com um maior reaproveitamento de código;
- Permite exibir o conteúdo de variáveis Java em páginas HTML;
- Sistema de fragmentos de templates, dentre outros.

Como o Thymeleaf funciona?

Basicamente, quando criamos um template com Thymeleaf e incorporamos código Java nas páginas HTML, a própria ferramenta traduz o código Java e incorpora à página HTML, já que o Browser não consegue exibir código diferente do HTML.

Abaixo temos um exemplo de código escrito com o Thymeleaf:

```
<ul>
  <li th:each="user : ${users}">
    <a
      th:href="/user/{username} (username=${user.username})"
      th:text="${user.firstname} + ' ' + ${user.lastname}"
    ></a>
  </li>
</ul>
```

Documentação Thymeleaf

https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html

Forward vs Redirect

Ao programarmos para Web, é comum a necessidade de redirecionar nosso usuário para uma página diferente. Existem duas formas comuns de se realizar o redirecionamento: Forward e Redirect

Forward vs Redirect

Quando estamos programando uma aplicação Java Web com Spring MVC, temos como padrão de retorno de cada Request uma String que será tratada como Forward. Ou seja, quando temos um código que, simplesmente, retorna um formulário para preencher dados de uma pessoa:

```
@RequestMapping(method = RequestMethod.GET, value = "novaPessoa")
public String novaPessoa(Pessoa pessoa) {
        return "pessoa/formulario";
}
```

Forward vs Redirect

Quando estamos programando uma aplicação Java Web com Spring MVC, temos como padrão de retorno de cada Request uma String que será tratada como Forward. Ou seja, quando temos um código que, simplesmente, retorna um formulário para preencher dados de uma pessoa:

```
@RequestMapping(method = RequestMethod.GET, value = "novaPessoa")
public String novaPessoa(Pessoa pessoa) {
        return "pessoa/formulario";
}
```

Forward vs Redirect

Estamos utilizando redirecionamento Forward, pois este é o padrão do framework. E, podemos utilizar deste padrão para qualquer Request se quisermos.

Como, por exemplo, um Request que adicione uma pessoa no banco de dados:

```
@RequestMapping(method = RequestMethod.POST, value = adicionaPessoa“)
public String adicionaPessoa(Pessoa pessoa) {
        pessoaDao.adiciona(pessoa);
        return “pessoa/listaPessoas“;
}
```

Forward vs Redirect

O Redirect é conhecido justamente por realizar o redirecionamento no lado do cliente. Ou seja, nosso navegador encaminhará para outra URL.

Com isso, temos um pequeno aumento no tempo de processamento da página pois são realizadas duas requisições no navegador.

```
@RequestMapping(method = RequestMethod.GET, value = "adicionado")
public String adicionado(Pessoa pessoa) {
        return "pessoa/adicionado";
}
@RequestMapping(method = RequestMethod.POST, value = "adicionaPessoa")
public String adicionaPessoa(Pessoa pessoa) {
        pessoaDao.adiciona(pessoa);
       return "redirect:adicionado";
}
```

Anotações do Spring MVC

@Controller

Transforma a classe em um bean do tipo controller do MVC.

```java
@Controller
@RequestMapping("/computador")
public class ComputadorController {

    @GetMapping
    public String getAllComputadores(ModelMap modelMap) {
        List<Computador> computadores = new ArrayList<>();
        modelMap.addAttribute( attributeName: "computadores", computadores);

        return "lista";
    }

}
```

Objetos de Resposta do Spring MVC

ModelMap

Objeto usado para enviar dados a página como resposta da solicitação.

Trabalha como uma resposta do tipo forward.

```
@Controller
@RequestMapping("/computador")
public class ComputadorController {

    @GetMapping
    public String getAllComputadores(ModelMap modelMap) {
        List<Computador> computadores = new ArrayList<>();
        modelMap.addAttribute( attributeName: "computadores", computadores);

        return "lista";
    }

}
```

Objetos de Resposta do Spring MVC

ModelAndView

Objeto usado para enviar dados a página como resposta de uma solicitação.

```java
@Controller
@RequestMapping("/computador")
public class ComputadorController {

    @GetMapping
    public ModelAndView getAllComputadores() {

        List<Computador> computadores = new ArrayList<>();

        ModelAndView modelAndView = new ModelAndView( viewName: "lista");
        modelAndView.addObject( attributeName: "computadores", computadores);

        return modelAndView;

    }

}
```

Objetos de Resposta do Spring MVC

Redirect

É uma operação usada para redirecionar a resposta de uma solicitação para outra solicitação.

Mas existe um problema no redirect, tanto o ModelMap quanto o ModelAndView só funcionam para forward.

```java
@Controller
@RequestMapping("/computador")
public class ComputadorController {

    @GetMapping
    public ModelAndView getAllComputadores() {

        List<Computador> computadores = new ArrayList<>();

        ModelAndView modelAndView = new ModelAndView( viewName: "lista");
        modelAndView.addObject( attributeName: "computadores", computadores);

        return modelAndView;

    }

    @PostMapping
    public String save(Computador computador) {
        //*****************************************
        //Código que salva o computador na base de dados
        //*****************************************

        return "redirect:/computador";

    }

}
```

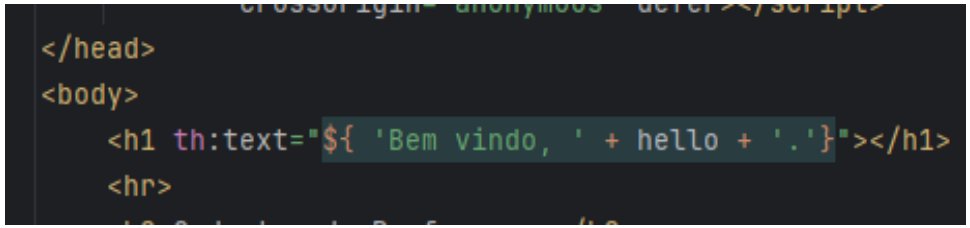
Objetos de Resposta do Spring MVC

Podemos fazer a passagem de atributos desejados para a página redirecionada através do objeto RedirectAttributes.

```java
@Controller
@RequestMapping("/computador")
public class ComputadorController {

    @GetMapping
    public ModelAndView getAllComputadores(ModelMap model) {

        List<Computador> computadores = new ArrayList<>();

        ModelAndView modelAndView = new ModelAndView( viewName: "lista");
        modelAndView.addObject( attributeName: "computadores", computadores);

        return modelAndView;

    }

    @PostMapping
    public String save(Computador computador, RedirectAttributes attributes) {
        //*****************************************
        //Código que salva o computador na base de dados
        //*****************************************
        attributes.addFlashAttribute( attributeName: "mesangem",
                attributeValue: "Computador Inserido com sucesso");

        return "redirect:/computador";

    }

}
```

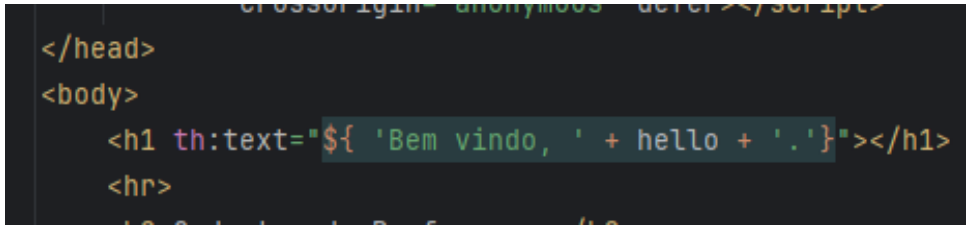
Thymeleaf Tags e Atributos

Todas as tags e atributos do thymeleaf iniciam por padrão com **th:**



Thymeleaf Tags e Atributos

Todas as tags e atributos do thymeleaf iniciam por padrão com **th:**



Sintaxe de expressão padrão

${...} : Variable expressions.

***{...} : Selection expressions.**

#{...} : Message (i18n) expressions.

@{...} : Link (URL) expressions.

${...} : Variable expressions.

Ajuda a injetar dados do controlador no arquivo de modelo . Ele expõe atributos do modelo à visualização da web.

A sintaxe da expressão variável combina um cifrão e chaves. Nosso nome de variável reside entre chaves:

${...}

Vamos injetar nossos dados do Dino no arquivo de modelo:

```
<span th:text="${dinos.id}"></span>
<span th:text="${dinos.name}"></span>
```

*{...} : Selection expressions.

A expressão de seleção opera em um objeto previamente escolhido . Isso nos ajuda a selecionar o filho do objeto escolhido.

A sintaxe da expressão de seleção é uma combinação de asterisco e chaves. Nosso objeto filho reside dentro das chaves:

*{...}

Vamos selecionar o id e o nome da nossa instância do Dino e injetá-lo em nosso arquivo de modelo:

```
<div th:object="${dinos}">
    <p th:text="*{id}">
    <p th:text="*{name}">
</div>
```

#{...} : Message (i18n) expressions.

Essa expressão ajuda a trazer texto externalizado para nosso arquivo de modelo . Também é chamado de externalização de texto.

A sintaxe da expressão da mensagem é uma combinação de hash e chaves. Nossa chave reside entre chaves:

#{...}

Por exemplo, vamos supor que queremos exibir uma mensagem específica em todas as páginas do nosso aplicativo web Dino. Podemos colocar a mensagem em um arquivo messages.properties :

welcome.message=welcome to Dino world.

```
<h2 th:text="#{welcome.message}"></h2>
```

@{...} : Link (URL) expressions.

As expressões de link são essenciais na construção de URL. Esta expressão é vinculada ao URL especificado .

A sintaxe da expressão do link combina o sinal “arroba” e chaves. Nosso link reside dentro das chaves:

@{...}

URLs podem ser absolutos ou relativos. Ao usar uma expressão de link com URLs absolutos, ela se vincula à URL completa começando com “ http(s) “:

```
<a th:href="@{http://www.unipar.br}"> Unipar </a>
```

Criando nosso Projeto

Starters necessários:

spring-boot-starter-web

spring-boot-devtools

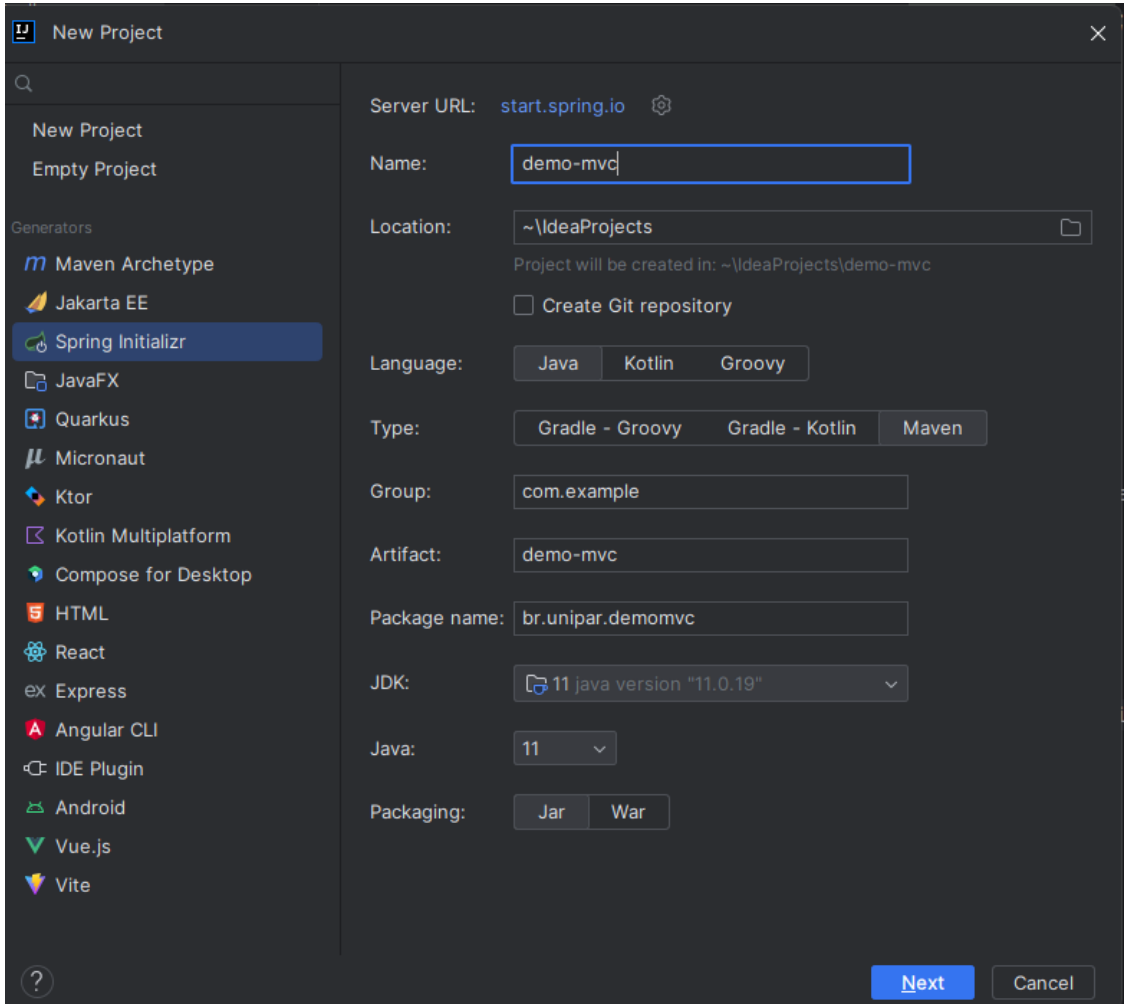
spring-boot-starter-thymeleaf

E se desejarem conectar no banco de dados:

spring-boot-starter-data-jpa

Postgresql

Versão do Spring: 2.7.16



Nosso Primeiro Controller

```java
package br.unipar.demomvc;

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


@Controller
@RequestMapping("/")
public class HomeController {

    @GetMapping
    public String hello() {
        return "hello";
    }

}
```

Nosso Primeiro Controller

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/css/bootstrap.min.css"
          rel="stylesheet"
          integrity="sha384-T3c6CoIi6uLrA9TneNEoa7RxnatzjcDSCmG1MXxSR1GAsXEV/Dwwykc2MPK8M2HN"
          crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.2/dist/js/bootstrap.bundle.min.js"
            integrity="sha384-C6RzsynM9kWDrMNeT87bh95OGNyZPhcTNXj1NW7RuBCsyN/o0jlpcV8Qyq46cDfL"
            crossorigin="anonymous" defer></script>
</head>
<body>
    <h1>Olá Mundo</h1>
</body>
</html>
```

Para ativar o autocomplete de atributos do thymeleaf no html ative o seguinte:

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

E como eu passo um valor/variável para uma view ?

```
<h1 th:text="${ 'Bem vindo, ' + user + '.'}"></h1>
```

```
@Controller
@RequestMapping("/")
public class HomeController {

    @GetMapping
    public String hello(Model model) {
        model.addAttribute( attributeName: "user",  attributeValue: "Anderson");
        return "hello";
    }

}
```

Tópicos

Olá Mundo com Spring MVC e Thymeleaf.

Criando nossa primeira aplicação.

Criando um CRUD.

Trabalho Bimestral.

Trabalho Bimestral (Valor = 3,0 pontos / em duplas)

Criaçao de uma aplicação de gestão financeira pessoal utilizando spring mvc e thymeleaf.

Cadastro de Categorias de Receitas Ou Despesas.

Cadastrar os registros de Receitas e Despesas do usuário.

Na home page da aplicação deve-se trazer um painel com os totalizadores de: Receita, Despesa, Total Liquido, Grid com a listagem dos últimos 10 lançamentos.

Tela de consulta de lançamentos com filtros de data, e Receitas ou despesas.

Bibliografia Base

https://developer.mozilla.org/pt-BR/docs/Web/HTTP

https://www.thymeleaf.org/documentation.html

Perguntas?

Conclusão