



RUMO AO ONE STACK

A ODISSEIA JAVA COM SPRING BOOT E THYMELEAF

RODRIGO BARBOSA DE SOUSA

Sumário

1. Introdução – A Lenda do One Stack

- O que é o One Stack?
- Por que usar Java com Spring Boot e Thymeleaf?
- A jornada do dev como uma aventura.

2. Preparando o Navio – Ambiente de Desenvolvimento

- Instalando o JDK e Spring Tool Suite (ou VSCode/Eclipse)
- Criando o projeto com Spring Initializr
- Estrutura básica do projeto gerado

3. Primeiras Velas – Entendendo Spring Boot

- O que é o Spring Boot?
- Annotations essenciais: @SpringBootApplication, @Controller, @GetMapping

1. Mapeando Rota – Controllers e Navegação

- Criando uma rota básica
- Passando dados para a view
- Introdução ao uso de Model

2. A Ilha Thymeleaf – Views Dinâmicas

- Integrando o Thymeleaf ao projeto
- Criando sua primeira página .html
- Usando expressões como \${ } e th:*

Sumário

6. Mini Missão – Página com Nome da Tripulação

- Exemplo prático: página que exibe os nomes dos personagens
- Código Java simples + template Thymeleaf básico

7. Conclusão – Os Primeiros Passos no Mar Fullstack

- O que você conquistou até aqui
- Próximos mares: persistência de dados, serviços, APIs

INTRODUÇÃO

■

A LENDA DO ONE STACK

Desbrave os mares do desenvolvimento fullstack com Java, Spring Boot e Thymeleaf — e encontre o tesouro do código limpo, performático e escalável.



O que é o One Stack?

Em um vasto oceano digital onde linguagens, frameworks e tecnologias emergem como ilhas desconhecidas, existe uma stack lendária conhecida como One Stack — uma combinação poderosa de Java, Spring Boot e Thymeleaf.

Essa stack representa uma rota segura (mas não fácil) para os desenvolvedores que desejam criar aplicações web robustas, organizadas e com boas práticas desde o início. O One Stack é o seu navio. Java é o casco firme, Spring Boot é o motor que acelera a viagem, e Thymeleaf é a vela que dá direção à interface visual. Juntos, eles formam uma tripulação confiável para enfrentar os desafios do mundo web.

Por que usar Java com Spring Boot e Thymeleaf?

A stack que você está prestes a explorar não foi escolhida por acaso. Veja por que ela é tão valiosa:

- **Java** é uma linguagem madura, confiável e muito usada em aplicações empresariais. Seu ecossistema é vasto, estável e com forte suporte da comunidade.
- **Spring Boot** simplifica o desenvolvimento de aplicações Java, removendo a complexidade de configuração e acelerando o tempo até a primeira entrega.
- **Thymeleaf** é um mecanismo de templates moderno, que se integra naturalmente ao Spring Boot e facilita a criação de páginas HTML dinâmicas, com sintaxe intuitiva.

Essa combinação permite que você desenvolva projetos fullstack (backend + frontend no mesmo projeto), ideal para aprendizado rápido e portfólios iniciais.

A jornada do dev como uma aventura

Todo programador é, na essência, um aventureiro. Você começa como um novato curioso, sem saber bem para onde vai. Mas aos poucos, com cada linha de código escrita, cada erro enfrentado, cada funcionalidade entregue, você sobe de nível.

Neste eBook, você embarca como um aspirante a Rei dos Devs, navegando pelo mar do desenvolvimento fullstack. A cada capítulo, novas ilhas (conceitos) serão descobertas. E ao final da jornada, você terá sua própria aplicação funcional com Java e Spring — um tesouro que muitos buscam, mas poucos encontram de verdade.

Prepare seu ambiente, afivele seu chapéu de palha e vamos zarpar.

O One Stack te espera.

CAPÍTULO 2 - PREPARANDO O NAVIO: AMBIENTE DE DESENVOLVIMENTO

Antes de zarpar para sua jornada fullstack, você precisa garantir que seu navio — o ambiente de desenvolvimento — esteja pronto. Neste capítulo, você aprenderá a instalar as ferramentas essenciais no Ubuntu 22.04 ou Windows, criar um projeto com Spring Initializr e entender a estrutura inicial do projeto.

Instalando o JDK e a IDE

1. Instalando o JDK 17

- **Ubuntu**

Abra o terminal e execute:

```
1 sudo apt update  
2 sudo apt install openjdk-17-jdk
```

Verifique se foi instalado corretamente:

```
1 java -version
```

Saída esperada: openjdk version "17..."

- **Windows**

1. Acesse o site: <https://adoptium.net/pt-BR/temurin/releases?version=17&os=any&arch=any>
2. Baixe o instalador do Temurin JDK 17 para Windows.
3. Execute o instalador e siga os passos.
4. Verifique a instalação:
 - Abra o terminal (CMD ou PowerShell)
 - Digite: java -version

2. Escolha e instale sua IDE

Você pode usar qualquer IDE Java. Aqui estão 3 opções populares com instruções rápidas:

Spring Tool Suite (STS) – Recomendado para iniciantes com Spring

<https://spring.io/tools>

Baseado no Eclipse, vem com plugins prontos para Spring Boot.

- **Ubuntu:** Baixe a versão .tar.gz, extraia e execute STS ou crie um atalho.
- **Windows:** Baixe o .exe e instale normalmente.

Visual Studio Code (VSCode) – Leve e rápido

<https://code.visualstudio.com/>

Extensões recomendadas:

- Extension Pack for Java
- Spring Boot Extension Pack
- Java Debugger

Após instalar, abra o terminal do VSCode (Ctrl +) e teste com `java -version`.

IntelliJ IDEA (Community Edition) – Poderoso e inteligente

<https://www.jetbrains.com/idea/download/>

- Suporte completo ao Spring Boot (e plugins opcionais para Thymeleaf)
- Ótima navegação de código

Criando o Projeto com Spring Initializr

Com o navio pronto, é hora de montar os primeiros compartimentos!

Acesse: <https://start.spring.io/>

Preencha os campos assim:

- **Project:** Maven
- **Language:** Java
- **Spring Boot:** 3.x
- **Group:** com.onepiece.dev
- **Artifact:** rumoaoonestack
- **Name:** rumoaoonestack
- **Description:** Projeto Fullstack com Java, Spring Boot e Thymeleaf
- **Package name:** com.onepiece.dev.rumoaoonestack
- **Packaging:** Jar
- **Java:** 17

Dependências:

- **Spring Web**
- **Thymeleaf**

Clique em "**Generate**" para baixar o projeto.

Após baixar o `.zip`, extraia e abra na sua IDE.

Estrutura Básica do Projeto Gerado

Dentro do projeto, você encontrará algo assim:

```
rumoaoonestack/
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/onepiece/dev/rumoaoonestack/
│   │   │       └── RumoaoonestackApplication.java
│   │   └── resources/
│   │       ├── static/          ← arquivos estáticos (CSS, JS, imagens)
│   │       ├── templates/       ← páginas HTML Thymeleaf
│   │       └── application.properties ← configuração da aplicação
│   └── test/
└── pom.xml                         ← testes automatizados (JUnit)
                                         ← gerenciador de dependências Maven
```

Principais arquivos e pastas:

- `RumoaoonestackApplication.java`: ponto de entrada da aplicação (`@SpringBootApplication`)
- `templates/`: onde você colocará suas views em HTML com Thymeleaf
- `application.properties`: configurações como porta, banco de dados, etc.
- `pom.xml`: arquivo onde você define e gerencia as dependências do projeto

Com o ambiente devidamente configurado, você está pronto para levantar as velas com Spring Boot e navegar pelos primeiros endpoints.

No próximo capítulo, começamos a dar vida ao projeto com código real!

CAPÍTULO 3 - PRIMEIRAS VELAS - ENTENDENDO SPRING BOOT

Agora que já temos o ambiente preparado e o projeto criado, é hora de içar as primeiras velas e entender o que faz o navio navegar: o Spring Boot.

O que é o Spring Boot?

Spring Boot é um framework baseado no Spring que **facilita a criação de aplicações Java**, eliminando a necessidade de configurações extensas. Ele cuida de muita coisa para você, como:

- **Configuração automática** (auto-configuration)
- **Servidor embutido** (como Tomcat)
- **Empacotamento em um único .jar**
- **Foco em produtividade com menos "boilerplate"**

Exemplo: Em vez de configurar arquivos XML para cada componente, o Spring Boot permite que você defina tudo usando **anotações** diretamente no código.

Annotations Essenciais

`@SpringBootApplication`

É a anotação principal. Marca a classe de entrada da aplicação. Internamente, combina três anotações:

- `@Configuration`: indica que a classe pode conter definições de beans.
- `@EnableAutoConfiguration`: ativa a configuração automática.
- `@ComponentScan`: ativa a varredura de componentes no pacote base.

```
● ● ●  
1 package com.rodrigobsjava.mini_missao;  
2 // Faça os imports necessários  
3 @SpringBootApplication  
4 public class MiniMissaoApplication {  
5     public static void main(String[] args) {  
6         SpringApplication.run(MiniMissaoApplication.class, args);  
7     }  
8 }
```

@Controller

Usada para definir uma classe responsável por receber requisições web e retornar respostas (normalmente páginas ou dados).

```
● ● ●  
1 @Controller("/")  
2 public class HomeController {  
3     // Métodos para tratar requisições  
4 }
```

@GetMapping

Indica que um método deve responder a uma requisição HTTP do tipo GET.

```
● ● ●  
1 @GetMapping("/")  
2 public String home() {  
3     return "index";  
4 }
```

Exemplo Completo

```
● ● ●  
1 @SpringBootApplication  
2 public class MiniMissaoApplication {  
3  
4     public static void main(String[] args) {  
5         SpringApplication.run(MiniMissaoApplication.class, args);  
6     }  
7  
8 }  
9 @Controller  
10 public class HomeController {  
11     @GetMapping("/")  
12     public String home() {  
13         return "index";  
14     }  
15 }
```

Conclusão do Capítulo

Com essas anotações, você já consegue criar **sua primeira rota funcional** no Spring Boot! No próximo capítulo, vamos adicionar páginas HTML com **Thymeleaf** e deixar a aplicação com uma interface bonita e navegável.

CAPÍTULO 4 - MAPEANDO ROTA - CONTROLLERS E NAVEGAÇÃO

A brisa começa a soprar mais forte e o navio ganha velocidade. Nosso próximo desafio é dominar as rotas e a comunicação entre backend e frontend. É aqui que os verdadeiros capitães do One Stack aprendem a se comunicar com suas tripulações — ou melhor, com suas views.

Criando uma Rota Básica

No Spring Boot, toda requisição que chega precisa de um caminho: esse é o papel dos *controllers*. Vamos começar criando uma rota simples que responda quando alguém visitar nosso navio pela primeira vez.

```
...
1 @Controller
2 public class HomeController {
3     @GetMapping("/")
4     public String home() {
5         return "index";
6     }
7 }
```

@Controller: Diz ao Spring que esta classe é responsável por receber e responder requisições HTTP.

@GetMapping (" / "): Mapeia a URL (rota) raiz (/) para o método home () .

return "index": Diz qual view deve ser renderizada (ex: index.html) que estará na pasta resources/templates.

Passando Dados para a View

Uma rota sem informação é como um navio sem carga. Vamos enviar mensagens para nossa página HTML usando o objeto Model:

```
...
1 @GetMapping("/")
2 public String home(Model model) {
3     model.addAttribute("mensagem", "Bem-vindo ao navio One Stack!");
4     return "index"; // mensagem.html
5 }
```

No arquivo index.html (com Thymeleaf):

```
 1 <!DOCTYPE html>
 2 <html xmlns:th="http://www.thymeleaf.org">
 3 <head>
 4   <title>Home</title>
 5 </head>
 6 <body>
 7   <h1 th:text="${mensagem}">Mensagem padrão</h1>
 8 </body>
 9 </html>
10
```

Explicando:

- model.addAttribute("frase", ...): Enviamos o dado para a view com a chave "frase".
- th:text="\${frase}": O Thymeleaf insere esse conteúdo no HTML de forma dinâmica.

Introdução ao uso de Model

O objeto Model é como um baú de tesouros: nele colocamos tudo que queremos compartilhar entre o backend e o HTML. Com ele, enviamos dados para exibir mensagens, listas de produtos, informações do usuário e muito mais.

Com poucos comandos, já conseguimos:

- Criar um *Controller* simples.
- Mapear rotas usando @GetMapping.
- Renderizar uma página HTML com Thymeleaf.
- Passar dados do backend para o frontend com Model.

CAPÍTULO 5 - A ILHA THYMELEAF - VIEWS DINÂMICAS

Após navegar pelos mares dos Controllers e Models, você finalmente avista uma nova terra: a Ilha Thymeleaf. Aqui, as páginas HTML ganham vida com dados dinâmicos e expressões mágicas. Preparado para criar views poderosas com o Thymeleaf?

Integrando o Thymeleaf ao Projeto

Antes de navegar pela Ilha, precisamos trazer o barco (ou melhor, a dependência) certa.

No seu pom.xml, adicione:

```
...
1 <dependency>
2   <groupId>org.springframework.boot</groupId>
3   <artifactId>spring-boot-starter-thymeleaf</artifactId>
4 </dependency>
```

O Spring Boot já reconhece essa dependência e configura tudo por baixo dos panos. Mas se quiser personalizar (como mudar o caminho padrão dos templates), pode ajustar no application.properties:

```
...
1 spring.thymeleaf.prefix=classpath:/templates/
2 spring.thymeleaf.suffix=.html
3 spring.thymeleaf.cache=false
```

Dica de Navegador: coloque seus arquivos HTML na pasta src/main/resources/templates.

Uma rota sem informação é como um navio sem carga. Vamos enviar mensagens para nossa página HTML usando o objeto Model:

```
...
1 @GetMapping("/tripulacao")
2 public String tripulacao(Model model) {
3   List<String> tripulantes = List.of("Luffy", "Zoro", "Nami", "Usopp", "Sanji", "Chopper",
4   "Robin", "Franky", "Brook", "Jinbe");
5   model.addAttribute("tripulantes", tripulantes);
6   return "tripulacao";
7 }
```

Criando Sua Primeira Página .html

Agora que o Thymeleaf está a bordo, crie seu primeiro HTML:

src/main/resources/templates/home.html

• • •

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <title>Tripulação</title>
5 </head>
6 <body>
7     <h1>Tripulação do Chapéu de Palha</h1>
8
9     <ul>
10        <li th:each="membro : ${tripulacao}" th:text="${membro}">Nome do Membro</li>
11    </ul>
12 </body>
13 </html>
```

Essa página usará os dados enviados pelo backend para listar os membros da tripulação.

Usando Expressões \${ } e Atributos th:*

O \${ } em Thymeleaf permite acessar variáveis que o Controller envia para a view. Por exemplo:

• • •

```
1 model.addAttribute("tripulacao", List.of("Luffy", "Zoro", "Nami", "Usopp));
```

Dentro do HTML, th:text="\${membro}" irá substituir o conteúdo do elemento pelo valor da variável.

Alguns dos principais atributos Thymeleaf:

- th:text: substitui o texto interno de um elemento
- th:each: itera sobre listas
- th:if e th:unless: controle de exibição condicional
- th:href: manipula links dinamicamente
- th:value: usado em inputs e forms

Exemplo de uso prático com th:each e th:text:

```
1 <ul>
2   <li th:each="membro : ${tripulacao}" th:text="${membro}">
3     Nome do Membro
4   </li>
5 </ul>
6
```

Esse trecho cria uma lista com os nomes da tripulação, vindos do backend.

No próximo capítulo você vai criar uma página real que exibe os nomes da tripulação do seu navio pirata. Prepare-se para praticar o que aprendeu e mostrar quem é o verdadeiro capitão do projeto!

Conclusão do Capítulo

O coração do Thymeleaf está na sua capacidade de **misturar lógica com HTML sem perder a legibilidade**.

- \${ }: acessa dados do Model (como variáveis, listas, objetos).
- th:text: substitui o conteúdo de um elemento.
- th:href, th:src, th:if, th:each, etc.: permitem links dinâmicos, imagens, condicionais e laços de repetição.

Exemplo com lista:

```
1 @Controller
2 public class TripulacaoController {
3
4     @GetMapping("/tripulacao")
5     public String mostrarTripulacao(Model model) {
6         List<String> nomes = List.of("Luffy", "Zoro", "Nami", "Sanji", "Chopper");
7         model.addAttribute("tripulacao", nomes);
8         return "tripulacao";
9     }
10 }
```

tripulacao.html

```
1 <ul>
2     <li th:each="membro : ${tripulacao}" th:text="${membro}"></li>
3 </ul>
```

Prepare-se para explorar a rota da /tripulacao, onde você criará uma nova página usando Thymeleaf para exibir os nomes dos piratas que fazem parte da tripulação do One Stack!

CAPÍTULO 6 - MINI MISSÃO - PÁGINA COM NOME DA TRIPULAÇÃO

Chegou a hora de aplicar os conhecimentos da Ilha Thymeleaf para criar uma página dinâmica com os nomes dos membros da tripulação.

Objetivo da Missão

Criar uma página que mostre os nomes dos integrantes da tripulação, usando Java + Spring Boot no backend e Thymeleaf no frontend.

1. Criando a lista de nomes no backend

Vamos começar no nosso controller Java. Você vai criar uma lista com os nomes da tripulação e enviar esses dados para a view.

Controlador Java (`TripulacaoController.java`)

```
...
1 package com.exemplo.onestack.controller;
2
3 import org.springframework.stereotype.Controller;
4 import org.springframework.ui.Model;
5 import org.springframework.web.bind.annotation.GetMapping;
6
7 import java.util.List;
8
9 @Controller
10 public class TripulacaoController {
11
12     @GetMapping("/")
13     public String home(Model model) {
14         model.addAttribute("welcome", " Bem-vindo ao navio One Stack!");
15         return "index";
16     }
17
18     @GetMapping("/tripulacao")
19     public String tripulacao(Model model) {
20         List<String> tripulantes = List.of("Luffy", "Zoro", "Nami",
21             "Usopp", "Sanji", "Chopper", "Robin", "Franky", "Brook", "Jinbe");
22         model.addAttribute("tripulantes", tripulantes);
23         return "tripulacao";
24     }
25 }
```

Explicação:

- `@Controller`: Marca a classe como um controlador Spring MVC.
- `@GetMapping (" / ")`: Rota que representa a **página inicial**.
- `model.addAttribute (. . .)`: Enviamos dados para a **view** (no caso, para o Thymeleaf).
- A lista `tripulantes` representa nossa **tripulação fictícia**. Adicione nomes reais se desejar.

Página Inicial (index.html)

```
1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <meta charset="UTF-8">
5     <title>One Stack</title>
6 </head>
7 <body>
8     <h1 th:text="${welcome}"></h1>
9     <a href="/tripulacao" style="margin-top: 20px; display: inline-block;">
10        ⚓ Veja a nossa tripulação!
11    </a>
12 </body>
13 </html>
14
```

Explicação:

`th:text="${welcome}"`: Thymeleaf insere o conteúdo da variável `welcome` no HTML.

O link direciona para a página `/tripulacao`, com um emoji náutico `⚓` para manter a estética pirata.

Página da Tripulação (tripulacao.html)

```
● ● ●

1 <!DOCTYPE html>
2 <html xmlns:th="http://www.thymeleaf.org">
3 <head>
4     <meta charset="UTF-8">
5     <title>Tripulação</title>
6     <style>
7         body {
8             font-family: Arial, sans-serif;
9         }
10        .tripulante {
11            padding: 8px;
12            border-bottom: 1px solid #ccc;
13        }
14        a {
15            margin-top: 20px;
16            display: inline-block;
17        }
18    </style>
19 </head>
20 <body>
21     <h2>🕒 Nossa Tripulação</h2>
22     <div th:each="tripulante : ${tripulantes}">
23         <div class="tripulante" th:text="${tripulante}"></div>
24     </div>
25
26     <a href="/">➡ Voltar para o Navio</a>
27 </body>
28 </html>
```

Explicação:

- `th:each="tripulante : ${tripulacao}":` Thymeleaf percorre a lista tripulacao e exibe cada membro como um item da lista.
- `th:text="${tripulante}":` Insere o nome de cada tripulante no HTML.

Conclusão

Parabéns!

Você completou mais uma etapa dessa jornada insana pelo Grand Code Line! Com esta mini missão, demos vida à nossa aplicação com **listas dinâmicas**, unindo o **backend com Spring Boot** ao **frontend com Thymeleaf** de forma fluida, como um navio deslizando pelo mar calmo

Agora você viu como:

- Preparar uma **tripulação (lista)** no backend,
- Enviar essa informação à view com o **Model do Spring**,
- E navegar por essa lista com o poderoso `th:each` do Thymeleaf.

Esse é o verdadeiro espírito One Stack: **dominar cada parte do navio**, do casco (backend) às velas (frontend). E o melhor: você está começando a entender como **componentes da tripulação (dados)** podem ser organizados e renderizados de forma **clara e reaproveitável!**

No próximo capítulo, a tripulação One Stack ancora para refletir sobre os primeiros tesouros conquistados. Com o mapa já parcialmente desenhado, vamos revisar as rotas traçadas, entender os ventos que nos impulsionaram — e preparar o navio para navegar em mares mais profundos da stack Java + Thymeleaf.

CONCLUSÃO - OS PRIMEIROS PASSOS NO MAR FULLSTACK

Após içar as velas e enfrentar os primeiros ventos, você completou uma etapa fundamental da sua jornada como desenvolvedor One Stack. Agora conhece os conceitos base do HTML, Thymeleaf e Java com Spring Boot — e começou a entender como eles se conectam dentro de um mesmo navio. naufragar nos primeiros bugs.

Você não apenas criou páginas dinâmicas, mas aprendeu a estruturar um projeto com boas práticas, entender o papel de controladores, templates e modelos. Isso é mais do que código — é navegação consciente.

O que você conquistou até aqui:

Fundamentos sólidos de HTML e Thymeleaf.

- Criação de páginas dinâmicas e interativas.
- Configuração inicial do Spring Boot com integração ao Thymeleaf.
- Compreensão das rotas (controllers) e da comunicação entre camadas.

Esses são os alicerces do seu navio. Agora você tem o que é preciso para seguir em frente sem naufragar nos primeiros bugs.

Esses são os alicerces do seu navio. Agora você tem o que é preciso para seguir em frente sem naufragar nos primeiros bugs.

Próximos Mares

Na próxima etapa da jornada One Stack, vamos deixar o front ancorado por um tempo e mergulhar fundo nas águas da persistência de dados, serviços e APIs. É hora de:

- Criar e modelar entidades no banco de dados,
- Trabalhar com Spring Data JPA para persistência,
- Desenvolver serviços RESTful com boas práticas,
- E preparar o navio para se comunicar com o mundo por meio de APIs.

O horizonte está aberto — e você já tem o mapa na mão. Avante, Capitão!

A GRADECIMENTO

Obrigado por ler até aqui!

Obrigado, Tripulante!

A tripulação da One Stack está comemorando: você completou a primeira grande travessia rumo ao domínio do desenvolvimento Full Stack!

Esse ebook foi criado por IA, e diagramador por humano.

O passo a passo se encontra no meu Github.

<https://github.com/rodrigobsjava/Rumo-ao-One-Stack-A-Odisseia-Java-com-Thymeleaf-e-Postgres>

[https://github.com/rodrigobsjava/Rumo-ao-
One-Stack-A-Odisseia-Java-com-Thymeleaf-e-
Postgres](https://github.com/rodrigobsjava/Rumo-ao-One-Stack-A-Odisseia-Java-com-Thymeleaf-e-Postgres)