Índice

Assuntos: Desenvolvimento JAVA e Springboot.

Abertura Alunos:
Apresentação pitch
Carin Maleski e Marília Tostes.

Spring boot

Fullstack é quem trabalha com desenvolvimento frontend e backend.

API Rest

API significa Interface de Programação de Aplicações (do inglês, Application Programming Interface). É um conjunto de regras, protocolos e ferramentas que permite a comunicação entre diferentes softwares.

Uma API define como diferentes componentes de software devem interagir, permitindo que aplicativos, sistemas operacionais e diferentes serviços se comuniquem entre si. Ela fornece uma maneira para que um programa solicite serviços de outro, definindo claramente como essa interação deve ocorrer.

API Rest Spring é uma arquitetura que se usa para integrar o backend com o frontend, com alguns protocolos de comunicação como o http.

Rest (representation state transfer):

REST, ou Transferência de Estado Representacional (Representational State Transfer), é um estilo arquitetural para projetar sistemas de software baseados na internet, especialmente em APIs (Interface de Programação de Aplicações) web.

Em termos mais simples, o REST define um conjunto de princípios e diretrizes que orientam como os sistemas devem se comunicar pela internet. Ele se baseia nos seguintes conceitos principais:

1. **Recursos (Resources)**: Na arquitetura REST, tudo é considerado um recurso. Um recurso pode ser qualquer coisa na web que tenha uma identificação única (URL). Por exemplo, um usuário, uma postagem em um blog, uma foto, entre outros.
2. **Operações sobre Recursos (Operations on Resources)**: As operações (ou métodos) comuns usados em REST são o GET, POST, PUT, DELETE, entre outros. Cada operação tem um significado específico: GET para obter informações, POST para criar, PUT para atualizar e DELETE para excluir um recurso.
3. **Padrões de comunicação (Communication Standards)**: REST utiliza os padrões do protocolo HTTP (Hypertext Transfer Protocol) para comunicação. Cada operação sobre um recurso é mapeada para um método HTTP, e as respostas são retornadas com códigos de status HTTP para indicar o resultado da operação.
4. **Estado e Representação (State and Representation)**: O REST enfatiza a transferência de representações de estado entre cliente e servidor. Isso significa que, ao acessar um recurso, o servidor envia uma representação dos dados desse recurso (geralmente em formatos como JSON ou XML) para o cliente.

SOFEA

Service-oriented front-end architecture

A arquitetura orientada a serviços no front-end (SOFEA - Service-Oriented Front-End Architecture) é um modelo de arquitetura de software que separa a lógica de apresentação (front-end) da lógica de negócios e serviços (back-end).

Em uma arquitetura SOFEA, a comunicação entre o front-end e o back-end é frequentemente realizada por meio de serviços web, como APIs RESTful, para buscar e enviar dados. O formato de dados comumente utilizado para essa comunicação é o JSON (JavaScript Object Notation).

Em uma arquitetura SOFEA:

1. **Front-end (Cliente)**: Esta camada concentra-se na interface do usuário, interação com o usuário e exibição de dados. Geralmente, é implementada usando tecnologias como HTML, CSS, JavaScript e frameworks como Angular, React ou Vue.js. O front-end se comunica com o back-end por meio de serviços web.
2. **Serviços (Back-end)**: Nesta camada, estão localizados os serviços de negócios e a lógica de processamento. Eles são acessados pelo front-end por meio de serviços web, como APIs RESTful, para buscar e manipular dados. Os serviços são independentes da interface do usuário e podem ser reutilizados por diferentes aplicativos ou front-ends.

Principais pontos da SOFEA:

- **Separação de Responsabilidades**: Permite a separação clara entre a apresentação visual e a lógica de negócios, facilitando a manutenção e escalabilidade.
- **Reutilização de Serviços**: Os serviços no back-end podem ser reutilizados por várias interfaces de usuário (múltiplos front-ends), proporcionando consistência nos dados e na lógica de negócios.
- **Flexibilidade e Manutenção**: Ao dividir a aplicação em camadas distintas, é possível atualizar e modificar o front-end ou back-end independentemente um do outro.
- **Comunicação via Serviços Web**: O front-end se comunica com o back-end por meio de serviços web (como RESTful APIs) para buscar e enviar dados, tornando a comunicação entre as camadas mais flexível e eficiente.

JSON (JavaScript Object Notation) é um formato de dados leve e legível por humanos utilizado para representar informações estruturadas. Ele é comumente usado para a troca de dados entre um servidor e um cliente na web.

- Conjunto de métodos para poder representar/fazer exatamente as funcionalidades
- Envia as requisições através dos métodos HTTP → GET, POST, PUT, DELETE.
- O servidor recebe as requisições, processa, e retorna respostas, gerando uma saída

endpoint

Explicação 1:

Um "**endpoint**" pode ser resumido como um ponto de extremidade de comunicação em um sistema de software. Ele representa um ponto de entrada específico para interagir com um serviço ou uma API. É um URL específico (geralmente uma rota em uma API web) que pode ser acessado por um cliente para realizar uma ação ou obter informações de um serviço.

Explicação 2:

Imagine que você está em um restaurante e deseja fazer um pedido. O cardápio do restaurante é como a documentação de uma API, mostrando todas as opções disponíveis (os recursos). Agora, cada prato no cardápio seria como um endpoint.
Um endpoint é como uma instrução específica que você dá ao garçom para obter ou fazer algo específico. Por exemplo, se você pedir um hambúrguer, você está usando o endpoint "hambúrguer" para solicitar uma ação específica ao restaurante.
Na linguagem das APIs, um endpoint é uma ponte de comunicação específica que você acessa para obter ou enviar informações. Assim como você pede um prato específico no restaurante, você acessa um endpoint específico na API para realizar uma tarefa específica.

HTTP E HTTPS

HTTP: Hyper-text transfer protocol
HTTPS: Hyper-text transfer protocol secure

Protocolo padrão de transferência de texto, sendo responsável por transferir texto entre usuário e servidores.

Métodos HTTP

Os termos GET, PUT, POST e DELETE são métodos ou verbos HTTP usados para indicar a ação que um cliente deseja executar em um determinado recurso em uma aplicação web ou API RESTful.

1. **GET**: O método GET é utilizado para solicitar dados de um recurso específico a partir de um servidor. Ele é usado quando o cliente quer apenas obter informações e não está realizando nenhuma alteração nos dados. Por exemplo, ao acessar uma página da web, o navegador envia uma solicitação GET para buscar a página.
2. **PUT**: O método PUT é usado para atualizar ou substituir completamente um recurso no servidor. Ele envia os dados para um recurso específico no servidor, substituindo completamente o conteúdo existente por novos dados. Normalmente, é utilizado quando se quer fazer uma atualização completa de um recurso com dados novos, e a requisição PUT deve conter todos os dados do recurso.
3. **POST**: O método POST é usado para enviar dados para o servidor para criar um novo recurso. Ao contrário do PUT, o POST é usado para criar um novo recurso e o servidor normalmente gera o identificador (ID) para o novo recurso. É comumente utilizado ao enviar formulários online ou ao criar um novo registro em um banco de dados.
4. **DELETE**: O método DELETE é utilizado para remover um recurso específico no servidor. Ele solicita a exclusão do recurso indicado na URL da requisição. Por exemplo, ao excluir uma postagem de um blog, a requisição DELETE é usada para solicitar a remoção dessa postagem do servidor.

Status HTTP

- Códigos iniciados com 200: Mensagem de sucesso
    - 200 = OK (sucesso)
    - 201 = Created (criado com sucesso)
- Códigos 400: Erros gerados pelo cliente
    - 400 = Bad Request (falta de informações)
    - 401 = Unauthorized (informações inválidas)
- Códigos 500: Erros gerados pelo servidor
    - 500 = Internal Server Error (erro no servidor)
    - 501 = Not Implemented (não consegue atender sua solicitação)

MVC

MVC (model view controller), otimiza a velocidade entre as requisições de usuários e servidores.

Explicação 1:
Model: Modelo é responsável por gerenciar de como os dados podem se comportar.
Controller: Controladora, é responsável por intermediar as requisições pela view, com a camada de model, ela liga ambas as outras camadas.
View: É a visão, responsável por apresentar as informações de forma visual para o usuário, como mensagens, botões, etc.

Explicação 2:

Model: Modelo que tem a responsabilidade de fazer todo o controle/gerenciamento da forma que os dados podem se comportar, por meio de regras de negócios, lógica… basicamente é o modelo que temos para gerar uma saída em forma de dados

View: Visão responsável por apresentar informações de forma visual ao usuário. Dentro da view podemos aplicar recursos (mensagens, botões, telas)

- PS: A View vamos trabalhar no Frontend, mas todas as camadas trabalham em conjunto e constantemente.

Controller: Camada de controle responsável por intermediar as requisições enviadas pela View, com as respostas fornecidas pela Model. Faz todo o processamento de dados que o usuário informou e vai repassar para as outras camadas. Fica no meio (intermediário), recebendo requisições das views/usuários e repassando para a model através de processamento de informações para poder repassar para os usuários e parar as outras camadas

Maven

No Java, se você estiver usando Maven, adiciona as dependências no arquivo pom.xml. Se estiver usando Gradle, adiciona no arquivo build.gradle. Essas ferramentas gerenciam automaticamente o download e a inclusão das dependências no seu projeto.

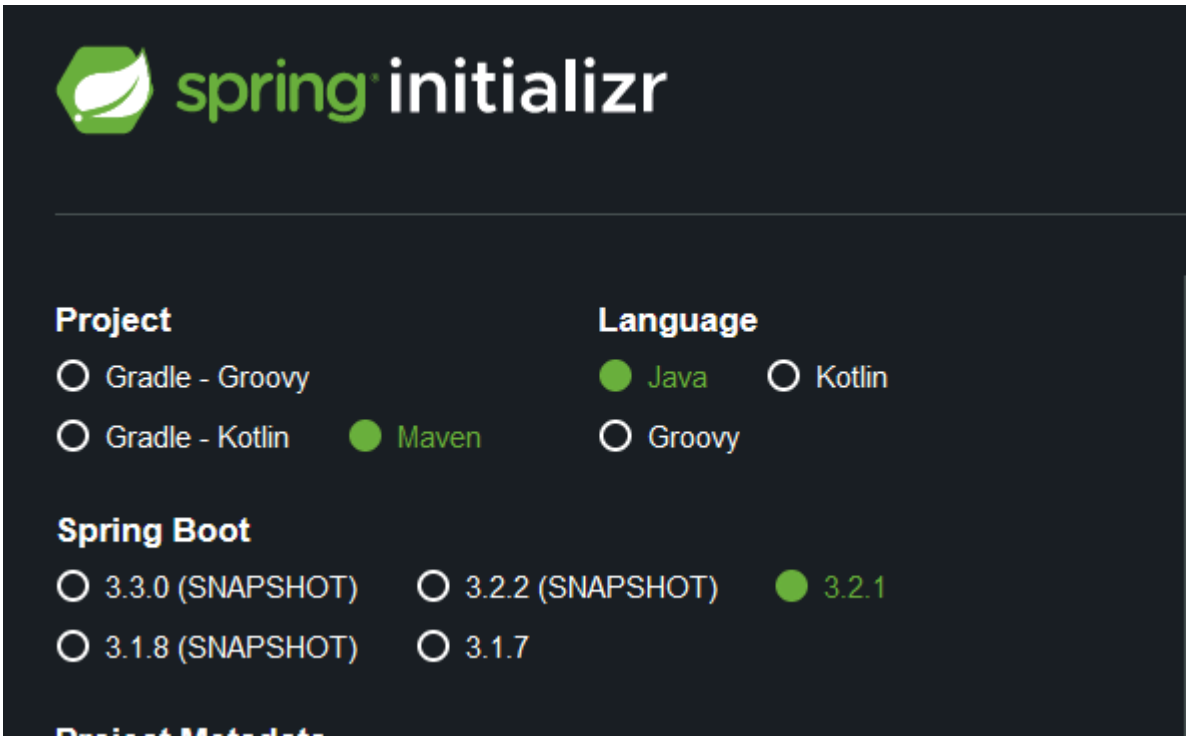
mavnrepository.com é um site útil para se ver se existem dependências/bibliotecas para diferentes projetos.

Explicação 2:

Usamos o Maven no Java porque é como um assistente eficiente que organiza automaticamente todas as "peças" necessárias para construir nosso projeto. Em vez de buscar manualmente bibliotecas e frameworks, o Maven faz isso por nós, simplificando o processo de desenvolvimento e reduzindo erros.

Spring initializr

Configuração inicial no Spring initializr



Em dependências geralmente se adicionam:

- Spring Web
- Spring Boot Dev Tools

Comandos e afins

Caminho para se importar um projeto maven para o STS

File > Import > Existing Maven Projects > Browse (selecionar diretório da pasta do projeto maven) > /pom.xml (marcar a opção) > Finish (aguardar a importação)

@RestController é uma anotação para se definir uma classe como controle.

Deve-se importar a biblioteca de RestController

@RequestMapping ("/*URL*") é uma anotação utilizada para se mapear as solicitações de uma classe controller, e ele busca no endereço o recurso.

@GetMapping é uma anotação que mapeia um Getter, é chamado quando se faz uma solicitação http get

Explicação 2:

O @GetMapping é como um cardápio onde você especifica um pedido usando um método específico. Em termos simples, ele mapeia solicitações HTTP GET para métodos em um @RestController. Assim como pedir uma pizza específica de um menu, você usa @GetMapping para indicar que um método específico deve ser executado quando uma solicitação GET é recebida

NomeApplication.java > Run as > 9.Spring Boot App

Caminho para rodar a aplicação

http://localhost:8080/hello-world

Exemplo de host local http. **localhost** específica que se está hosteando de forma local, seguido de **' : '** e a porta fornecida no Spring, nessa caso **8080**. **hello-world** representa o método especificado no Spring que você quer chamar.

Links e afins

Site para a criação de projetos Spring:

https://start.spring.io/

Site para verificação maven:

https://mvnrepository.com/

Cookbook, Spring, Spring README:

https://github.com/conteudoGeneration/cookbook_java_fullstack/blob/main/04_spring/README.md

Backend - Spring

1. Introdução ao Spring
2. Primeiros passos com Spring BOOT
3. Introdução ao JPA
4. Projeto Blog Pessoal - Projeto Spring
5. Projeto Blog Pessoal - Classe Postagem Model
6. Projeto Blog Pessoal - Interface Postagem Repository
7. Projeto Blog Pessoal - Classe Postagem Controller - Método Listar tudo
8. Projeto Blog Pessoal - Classe Postagem Controller - Método Buscar por id
9. Projeto Blog Pessoal - Classe Postagem Controller - Método Buscar por título
10. Projeto Blog Pessoal - Classe Postagem Controller - Método Cadastrar
11. Projeto Blog Pessoal - Classe Postagem Controller - Método Atualizar
12. Projeto Blog Pessoal - Classe Postagem Controller - Método Apagar
13. Projeto Blog Pessoal - Relacionamento entre Classes - Recurso Tema - Parte 01
14. Projeto Blog Pessoal - Relacionamento entre Classes - Recurso Tema - Parte 02
15. Introdução a Spring Security
16. Projeto Blog Pessoal - Spring Security - Ecossistema do Usuário - Parte 01
17. Projeto Blog Pessoal - Spring Security - Ecossistema da Segurança - Parte 01
18. Projeto Blog Pessoal - Spring Security - Ecossistema da Segurança - Parte 02
19. Projeto Blog Pessoal - Spring Security - Ecossistema da Segurança - Parte 03
20. Projeto Blog Pessoal - Spring Security - Ecossistema do Usuário - Parte 02
21. Introdução a Spring Testing + JUnit
22. Projeto Blog Pessoal - Teste Unitário - Configurando o ambiente de testes
23. Projeto Blog Pessoal - Teste Unitário - Testes na Camada Controller
24. Projeto Blog Pessoal - Documentação com SpringDoc
25. Projeto Blog Pessoal - Deploy no Render via Github

Dúvidas