

Swagger

Desenvolvimento de Sistemas II



Introdução

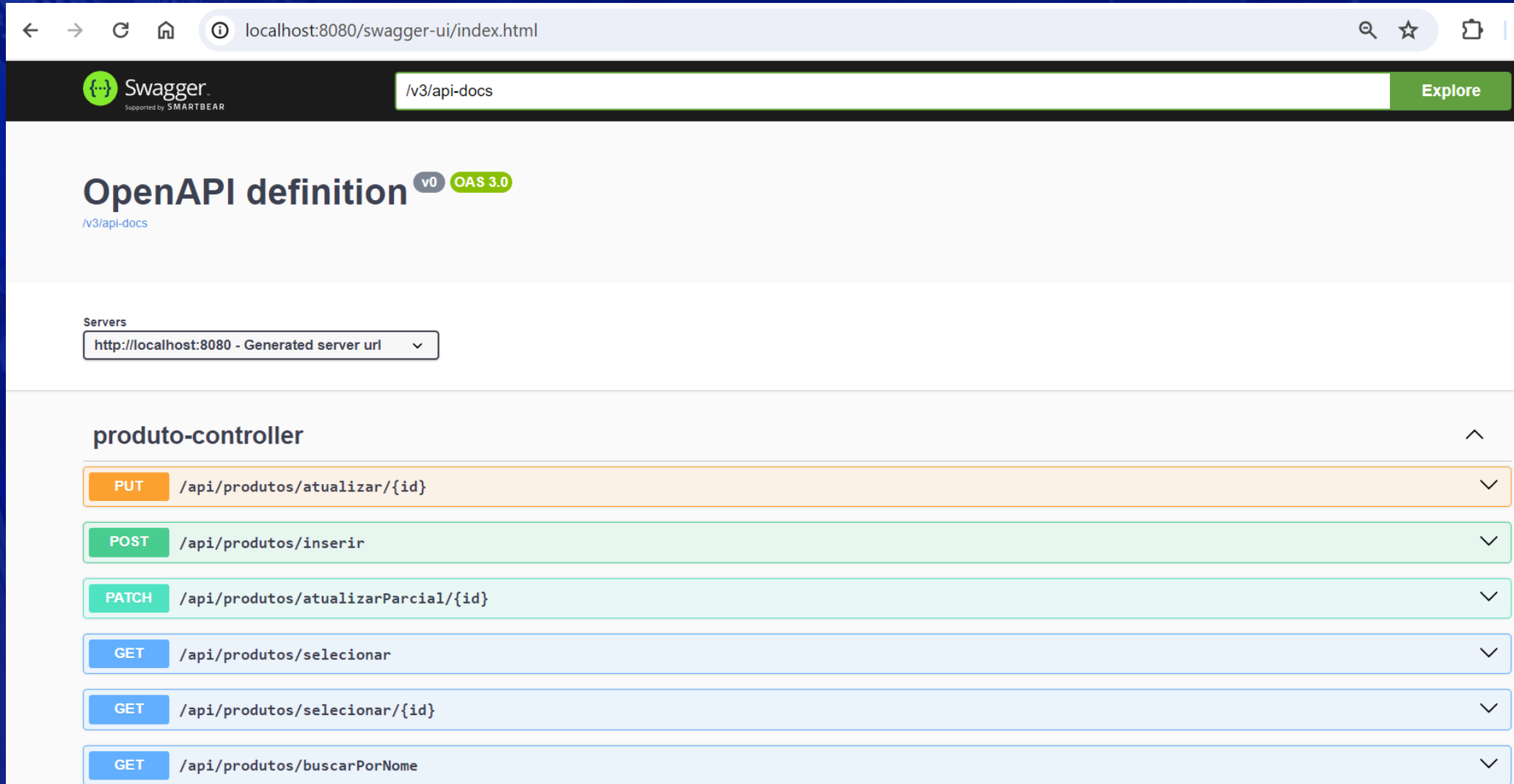
- A documentação de APIs é uma parte crucial do desenvolvimento de software
- Especialmente em aplicações web onde serviços precisam se comunicar de forma eficiente e precisa
- Swagger, junto com a especificação OpenAPI, é uma das ferramentas mais populares e poderosas para criar, gerenciar e consumir a documentação de APIs



OpenAPI

- Anteriormente conhecida como Swagger Specification, é uma especificação padrão para a criação de APIs RESTful
- Define uma forma consistente de descrever a estrutura de uma API RESTful, incluindo detalhes sobre seus endpoints, métodos HTTP suportados, parâmetros, e formatos de resposta

Exemplo de Documentação Swagger



The screenshot displays the Swagger UI interface in a web browser. The address bar shows the URL `localhost:8080/swagger-ui/index.html`. The Swagger logo is visible in the top left, and the text `/v3/api-docs` is in the top right. A green **Explore** button is located on the right side of the top bar.

The main content area features the title **OpenAPI definition** with a `v0` version tag and an **OAS 3.0** specification tag. Below the title, the text `/v3/api-docs` is displayed.

A **Servers** section contains a dropdown menu with the selected value `http://localhost:8080 - Generated server url`.

The **produto-controller** section lists six API endpoints, each with a method, path, and a collapse icon:

- PUT** `/api/produtos/atualizar/{id}`
- POST** `/api/produtos/inserir`
- PATCH** `/api/produtos/atualizarParcial/{id}`
- GET** `/api/produtos/selecionar`
- GET** `/api/produtos/selecionar/{id}`
- GET** `/api/produtos/buscarPorNome`





OpenAPI e Swagger

- OpenAPI: especificação para descrever APIs RESTful
- Swagger: conjunto de ferramentas para gerar, visualizar e interagir com APIs que seguem a especificação OpenAPI
- Benefícios: Facilita a documentação, teste e desenvolvimento de APIs

Benefícios da Documentação OpenAPI



- **Padronização:** fornece um formato padronizado para descrever APIs de forma consistente e precisa
- **Autodocumentação:** facilita a vida dos desenvolvedores, que podem entender rapidamente como interagir com a API sem precisar de documentação externa adicional
- **Teste e Depuração:** permite testar e depurar APIs diretamente a partir da documentação com ferramentas como Swagger UI e Swagger Inspector
- **Gerenciamento de Ciclo de Vida:** facilita a manutenção e evolução das APIs, garantindo que as mudanças sejam refletidas imediatamente na documentação

Ferramentas do Ecossistema Swagger



- **Swagger UI:** gera uma documentação interativa e navegável a partir de uma especificação OpenAPI, permitindo que desenvolvedores testem endpoints diretamente no navegador
- **Swagger Editor:** interface web que permite criar e editar especificações OpenAPI de forma interativa, oferecendo validação em tempo real e visualização da documentação
- **Swagger Codegen:** gera código cliente e servidor em diversas linguagens a partir de uma especificação OpenAPI, acelerando o desenvolvimento e garantindo a consistência entre a API e suas implementações.



Requisitos para o Projeto

- IntelliJ IDEA: IDE para desenvolvimento em Java.
- Spring Boot: Framework para criar aplicações Java.
- Maven:
 - Plugin: springdoc-openapi-maven-plugin
 - Dependência: springdoc-openapi-starter-webmvc-ui

Adicionar Dependência ao Projeto



```
<dependency>  
  <groupId>org.springdoc</groupId>  
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>  
  <version>2.5.0</version>  
</dependency>
```

Acessar Documentação do Swagger

- Acessar a documentação gerada automaticamente navegando para <http://localhost:8080/swagger-ui/index.html>
- Ajuste o URL conforme a porta que seu servidor estiver usando



Adicionar Plugin ao Projeto

(caso não funcione só com a dependência)

```
<plugin>  
  <groupId>org.springdoc</groupId>  
  <artifactId>springdoc-openapi-maven-plugin</artifactId>  
</plugin>
```

Configurações Adicionais (Opcional)

- Pode-se personalizar ainda mais o Swagger com algumas opções
- Proporciona uma visão mais clara e detalhada da API, incluindo exemplos personalizados dos parâmetros e saídas dos métodos
- Ajuda os desenvolvedores a entenderem rapidamente a funcionalidade da API e como usá-la
- Personalização pode ser feita por anotações nos controladores



Anotar Controladores

- `@Operation`: Define uma operação de API

Exemplo: `@Operation(summary = "Lista todos os produtos", description = "Retorna uma lista de todos os produtos disponíveis")`

- `@Parameter`: Define um parâmetro de entrada para uma operação de API

Exemplo: `@Parameter(description = "Produto a ser inserido")`

- `@ApiResponse`: Define uma resposta da API

Exemplo: `@ApiResponse(responseCode = "200", description = "Lista de produtos retornada com sucesso")`

- `@Schema`: Define um esquema (modelo) para a API

Exemplo: `content = @Content(mediaType = "application/json", schema = @Schema(implementation = Produto.class))`

@Schema



- Usada para definir e descrever propriedades do modelo de dados
- Pode ser aplicada a classes, métodos ou parâmetros para especificar o esquema de um objeto
- Descreve um modelo especificando a estrutura e os detalhes de uma classe que será usada como modelo de dados em suas APIs
- Detalha campos fornecendo descrições detalhadas para cada campo dentro de um modelo, incluindo tipos de dados, valores padrão, exemplos, etc.

@Schema - Exemplos

```
@Schema(description = "Representa um produto no sistema")
```

```
public class Produto {
```

```
    @Schema(description = "ID único do produto", example = "1")
```

```
    private Long id;
```

```
    @Schema(description = "Nome do produto", example = "Notebook")
```

```
    private String nome;
```



@Content

- Usada para descrever o conteúdo da resposta ou da solicitação, especialmente em termos de mídia e esquemas
- Frequentemente utilizada em conjunto com `@ApiResponse` para especificar o tipo de conteúdo que a API retorna ou espera
- Especifica o tipo de mídia do conteúdo (por exemplo, `application/json`, `application/xml`)
- Define o esquema do conteúdo, que geralmente é um modelo de dados descrito com a anotação `@Schema`.

@Content - Exemplo

```
@ApiResponse(responseCode = "200", description = "Produto encontrado",  
    content = @Content(mediaType = "application/json", schema =  
        @Schema(implementation = Produto.class)))
```

Anotar Controladores – Exemplo 1



```
@Operation(summary = "Lista todos os produtos",  
    description = "Retorna uma lista de todos os produtos disponíveis")  
@ApiResponses(value = {  
    @ApiResponse(responseCode = "200", description = "Lista de produtos retornada com sucesso",  
        content = @Content(mediaType = "application/json",  
            schema = @Schema(implementation = Produto.class))),  
    @ApiResponse(responseCode = "500", description = "Erro interno do servidor", content = @Content)  
})  
public ResponseEntity<List<Produto>> listarTodosProdutos() {
```

Anotar Controladores – Exemplo 2



```
@GetMapping("/selecionar/{id}")
@Operation(summary = "Busca produto por ID", description = "Retorna um produto pelo seu ID")
@ApiResponses(value = {
    @ApiResponse(responseCode = "200", description = "Produto encontrado",
        content = @Content(mediaType = "application/json",
            schema = @Schema(implementation = Produto.class))),
    @ApiResponse(responseCode = "404",
        description = "Produto não encontrado", content = @Content)
})
public ResponseEntity<?> buscarProdutoPorId(@Parameter(description = "ID do produto a ser buscado")
    @PathVariable Long id) {
```

Perguntas???





Exercício

Modificar o projeto Spring MVC da API Produto para gerar a documentação Swagger:

- Inserir o plugin e a dependência no Maven
- Testar o link para acessar a documentação
- Inserir as anotações no controlador para melhorar a documentação