Unifacisa

Swagger em Spring Boot

Documentação Automatizada de APIs

Unifacisa

Sumário

Unifacisa

Introdução

O que é documentação de API

- Manual de Instruções de como integrar e usar a API para os desenvolvedores.

Por que documentar APIs

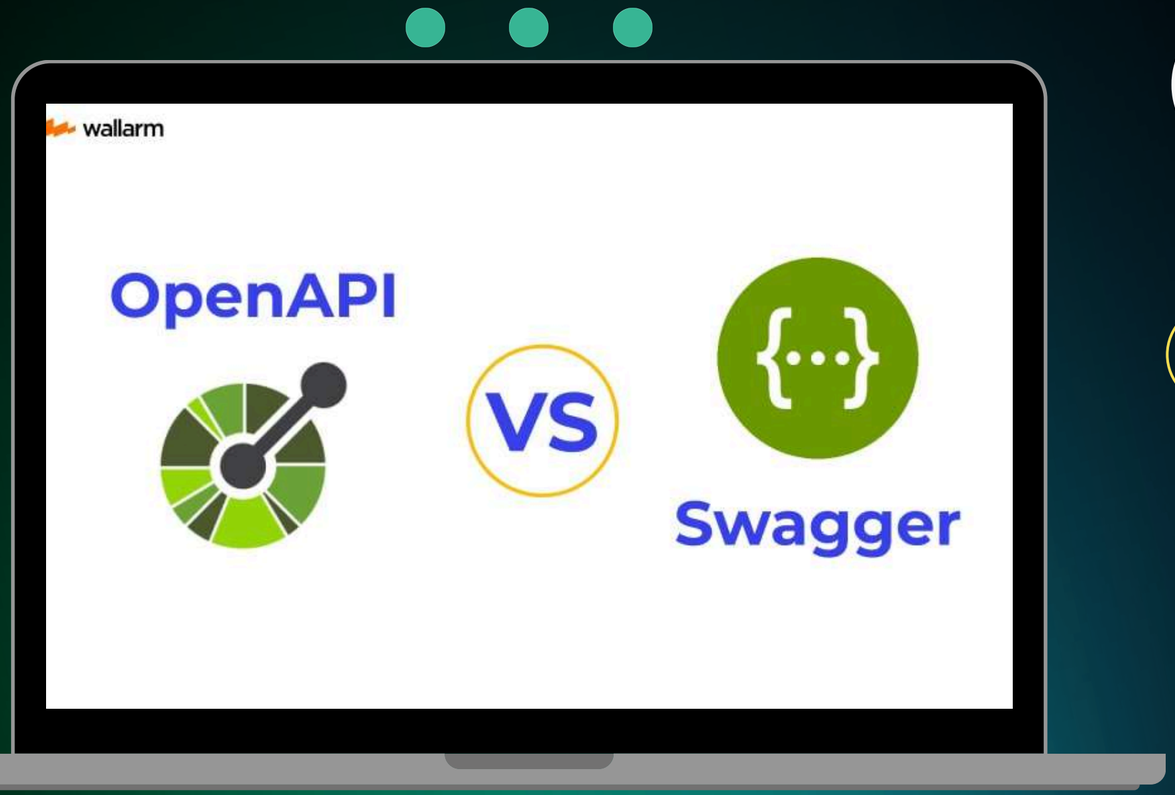
- Facilita o consumo por outros desenvolvedores.
- Reduz erros na integração.
- Melhora a manutenibilidade.

Desafios da Documentação Manual

- Trabalhosa e demorada.
- Desatualização frequente.
- Falta de padronização.

Unifacisa

Swagger VS OpenAPI



OpenAPI

- Especificação (padrão de descrição de APIs em YAML/JSON)
- Padronização (como documentar APIs)
- A linguagem padrão para descrever APIs

Swagger

- Conjunto de ferramentas (UI, Editor, Codegen)
- Implementação (ferramentas para visualizar e testar)
- Ferramentas que usam a OpenAPI

Unifacisa

Anotações Swagger

@Schema

- *Descreve modelos*
- *Gera exemplos pré-preenchidos*
- *Mostra descrições e regras de validação*

@Operation

- *Documenta um endpoint*
- *Exibe um resumo e descrição do endpoint*
- *Agrupa endpoints por tags*

@ApiResponse

- *Define possíveis respostas HTTP*
- *Lista códigos de status e mensagens de erro*
- *Pode incluir exemplos de respostas*

@Parameter

- *Documenta parâmetros de endpoints (path, query, header)*
- *Mostra descrições e exemplos ao lado dos parâmetros*

@Tag

- *Agrupa endpoints relacionados na UI*
- *Cria seções organizadas*

@ExampleObject

- *usada para definir exemplos concretos de requests/responses*
- *Para documentar casos específicos*

Unifacisa

Exemplo Prático

Sistema de agendamento de consultas médicas usando Spring Boot e Swagger

Unifacisa

Dependências Utilizadas

Spring Web

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Lombok

```
<dependency>
<groupId>org.projectlombok</groupId<artifactId>lombok</artifactId><optional>true</optional>
</dependency>
```

SpringDoc OpenAPI

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

CÓDIGOS DO POM.XML ✓

Serve para gerar automaticamente a documentação da sua API REST em formato OpenAPI/Swagger.

Unifacisa

Classe de Consulta

```java
package com.example.model;

import io.swagger.v3.oas.annotations.media.Schema;

@Data
@Schema(description = "Modelo de uma consulta médica agendada")
public class Consulta {
    @Schema(description = "ID da consulta", example = "1")
    private Long id;

    @Schema(description = "Nome do paciente", example = "João Silva", required = true)
    private String paciente;

    @Schema(description = "Especialidade médica", example = "Cardiologia")
    private String especialidade;

    @Schema(description = "Data e hora da consulta", example = "2025-12-15T14:30:00")
    private LocalDateTime dataHora;

    @Schema(description = "Status da consulta", example = "AGENDADA")
    private String status = "AGENDADA";
}
```

Unifacisa

Classe de Controller

```java
package com.example.controller;

import java.util.ArrayList;

@RestController
@RequestMapping("/consultas")
@Tag(name = "Consultas Médicas", description = "Agendamento e gerenciamento de consultas")
public class ConsultaController {

    private final List<Consulta> consultas = new ArrayList<>();
    private Long nextId = 11;

    @Operation(
        summary = "Listar todas as consultas",
        description = "Retorna todas as consultas agendadas, filtradas por status se fornecido"
    )
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "Lista de consultas retornada com sucesso"),
        @ApiResponse(responseCode = "204", description = "Nenhuma consulta encontrada")
    })
    @GetMapping
    public ResponseEntity<List<Consulta>> listar(
        @Parameter(description = "Filtrar por status (AGENDADA, CANCELADA, REALIZADA)", example = "AGENDADA")
        @RequestParam(required = false) String status) {

        List<Consulta> resultado = status != null ?
            consultas.stream().filter(c -> c.getStatus().equals(status)).toList() :
            consultas;

        return resultado.isEmpty() ?
            ResponseEntity.noContent().build() :
            ResponseEntity.ok(resultado);
    }

    @Operation(
        summary = "Agendar nova consulta",
        description = "Cria um novo agendamento de consulta médica"
    )
    @ApiResponses({
        @ApiResponse(
            responseCode = "201",
            description = "Consulta agendada com sucesso",
            content = @Content(schema = @Schema(implementation = Consulta.class))
        ),
        @ApiResponse(responseCode = "400", description = "Dados inválidos fornecidos")
    })
    @PostMapping
    public ResponseEntity<Consulta> agendar(
        @io.swagger.v3.oas.annotations.parameters.RequestBody(
            description = "Dados da consulta a ser agendada",
            required = true,
            content = @Content(
                examples = @ExampleObject(
                    value = "{\"paciente\": \"Maria Oliveira\", \"especialidade\": \"Dermatologia\", \"dataHora\": \"2025-11-20T10:00:00\"}"
                )
            )
        )
        @RequestBody Consulta consulta) {

        // Validação simples
        if (consulta.getPaciente() == null || consulta.getDataHora() == null) {
            return ResponseEntity.badRequest().build();
        }

        consulta.setId(nextId++);
        consulta.setStatus("AGENDADA");
        consultas.add(consulta);
        return ResponseEntity.status(201).body(consulta);
    }

    @Operation(
        summary = "Cancelar consulta",
        description = "Atualiza o status de uma consulta para CANCELADA"
    )
    @ApiResponses({
        @ApiResponse(responseCode = "200", description = "Consulta cancelada com sucesso"),
        @ApiResponse(responseCode = "404", description = "Consulta não encontrada")
    })
    @PatchMapping("/{id}/cancelar")
    public ResponseEntity<Void> cancelar(
        @Parameter(description = "ID da consulta a ser cancelada", example = "1")
        @PathVariable Long id) {

        Optional<Consulta> consulta = consultas.stream()
            .filter(c -> c.getId().equals(id))
            .findFirst();

        if (consulta.isPresent()) {
            consulta.get().setStatus("CANCELADA");
            return ResponseEntity.ok().build();
        }
        return ResponseEntity.notFound().build();
    }
}
```

Unifacisa

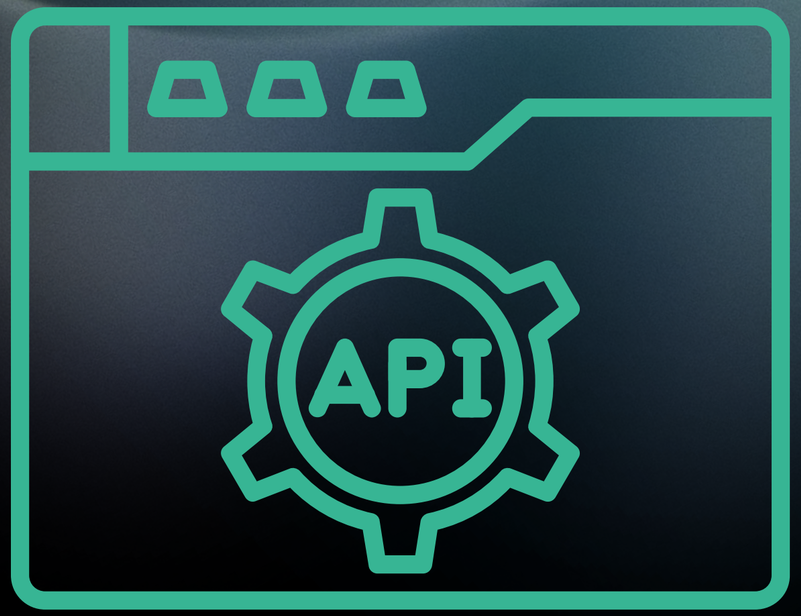
Classe de Configuração

```java
package com.example.config;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("Sistema de Agendamento Médico")
                .version("1.0")
                .description("API para demonstração de documentação com Swagger"));
    }
}
```

Unifacisa

Testando o Projeto

VIA NAVEGADOR



Tendo feito todos os passos anteriores, já se pode testar a aplicação.

PRIMEIRO PASSO

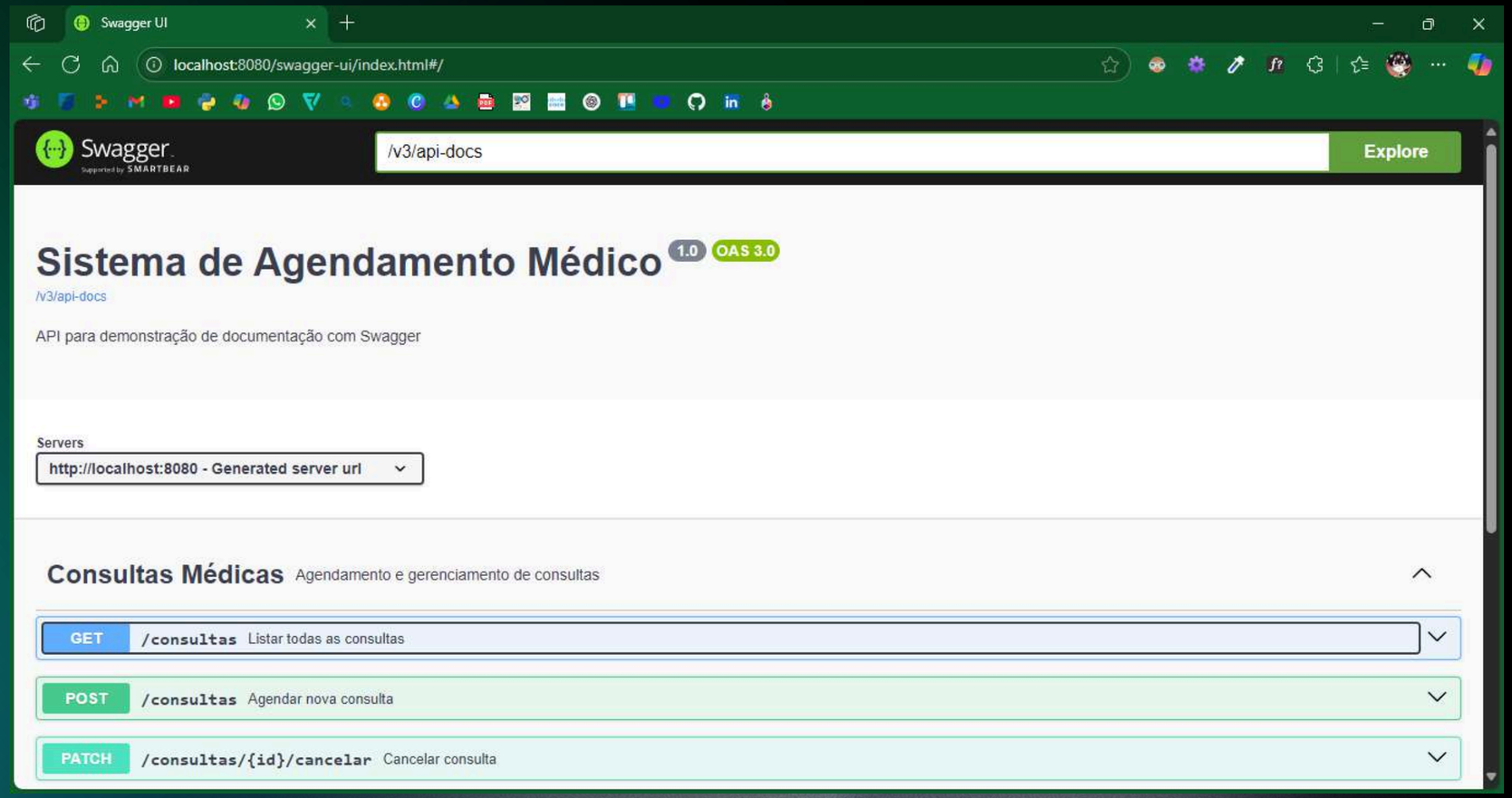
Rodar o projeto spring

SEGUNDO PASSO

Acessar a interface Swagger na url padrão
http://localhost:8080/swagger-ui.html

Unifacisa

Interface Swagger



Unifacisa

Thank You

Documentação viva, desenvolvimento ágil!

</> Tópicos Avançados de Programação - 2025