

---

# Desenvolvimento WEB

-

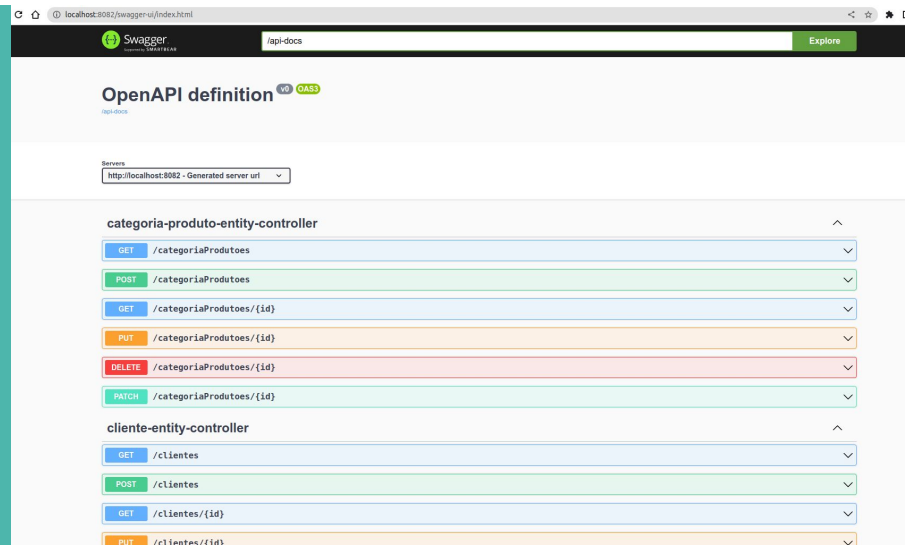
Full Stack Completo: Java + React

---

---

# Criando Documentação para a sua API

## com o Swagger



# Documentando a API com o Swagger



Documentar uma aplicação é um ponto essencial de qualquer projeto, muitas vezes negligenciado. Quando se trabalha em equipe, uma má documentação pode dificultar (e muito) o trabalho dos demais desenvolvedores.

# Documentando a API com o Swagger



- Primeiramente vamos entender do que se trata essa ferramenta que nos auxilia a criar documentação para os nossos serviços fornecidos por APIs.
- O **Swagger** é um projeto open source que contempla diversas ferramentas para documentação, consumo e visualização de serviços RESTful, entre essas ferramentas, temos o framework para documentação legível de APIs, esse framework implementa a especificação OpenAPI, que é uma linguagem para descrição de contratos de APIs REST.
- A OpenAPI define um formato JSON com campos padronizados, através de um JSON Schema para que você descreva recursos, modelo de dados, URIs, Content-Types, métodos HTTP aceitos e códigos de respostas.

# Documentando a API com o Swagger



- O framework do Swagger tem a implementação em diversas linguagens de programação e ao final fornece uma Interface de Usuário, o Swagger UI, para visualização interativa da documentação gerada automaticamente.
- A Swagger UI fornece uma página que lê um documento de especificação OpenAPI e gera um site de documentação interativo.

# Documentando a API com o Swagger



1) Acrescente no `pom.xml` as dependências do swagger abaixo:

...

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
  <version>2.2.0</version>
</dependency>
```

```
</dependencies>
```

...

# Documentando a API com o Swagger



2) Acrescente ao arquivo `application.properties` a linha **em vermelho** abaixo:

```
...  
  
# JPA  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=false  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLD  
ialect  
spring.jpa.properties.hibernate.temp.use_jdbc_metadata_defaults = false  
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQL9Dialect  
spring.jpa.generate-ddl=true  
  
springdoc.api-docs.path=/api-docs
```

# Documentando a API com o Swagger



3) Dentro da pasta "config", criar uma nova classe chamada "SwaggerConfig" e copie o código abaixo:

```
package br.com.ifpe.oxefood.config;

import org.springdoc.core.models.GroupedOpenApi;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.info.Info;

@Configuration
public class SwaggerConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .info(new Info()
                .title("OxeFood API")
                .version("1.0")
                .description("API do OxeFood")
                .contact(new Contact()
                    .name("Aluno IFPE")
                    .email("aluno@discente.ifpe.edu.br")));
    }

    @Bean
    public GroupedOpenApi customApi() {
        return GroupedOpenApi.builder()
            .group("api")
            .pathsToMatch("/api/**")
            .pathsToExclude("/error", "/actuator/**")
            .build();
    }
}
```



# Documentando a API com o Swagger



## 4) Adicione a documentação da API nos métodos do Controller:

```
...
import io.swagger.v3.oas.annotations.Operation;
...
@RestController
@RequestMapping("/api/cliente")
@CrossOrigin
public class ClienteController {

    @Autowired
    private ClienteService clienteService;

    @Operation(
        summary = "Serviço responsável por salvar um cliente no sistema.",
        description = "Exemplo de descrição de um endpoint responsável por inserir um cliente no sistema."
    )
    @PostMapping
    public ResponseEntity<Cliente> save(@RequestBody @Valid ClienteRequest request) {

        Cliente cliente = clienteService.save(request.build());
        return new ResponseEntity<Cliente>(cliente, HttpStatus.CREATED);
    }
    ...
}
```

# Documentando a API com o Swagger



5) Para acessar a página da documentação, acesse: `http://localhost:8080/swagger-ui/index.html`

A screenshot of the Swagger UI interface in a web browser. The browser's address bar shows the URL 'localhost:8082/swagger-ui/index.html'. The Swagger logo is in the top left, and a search bar with the text '/api-docs' and an 'Explore' button is in the top right. Below the header, it says 'OpenAPI definition' with a 'v0 OAS3' label. A 'Servers' section shows a dropdown menu with 'http://localhost:8082 - Generated server url'. The main content area lists two API controllers. The first, 'categoria-produto-entity-controller', has six endpoints: GET /categoriaProdutos, POST /categoriaProdutos, GET /categoriaProdutos/{id}, PUT /categoriaProdutos/{id}, DELETE /categoriaProdutos/{id}, and PATCH /categoriaProdutos/{id}. The second, 'cliente-entity-controller', has three endpoints: GET /clientes, POST /clientes, and PUT /clientes/{id}. Each endpoint is represented by a colored bar with the HTTP method in a small button on the left and a dropdown arrow on the right.

# Dúvidas



# Exercícios



- 1) Documente a API de **todas as rotas** implementadas nos Controllers até o momento.



The image features a white background with two large, solid green abstract shapes. One shape is a semi-circle on the left side, and the other is a more complex, organic shape on the right side. Centered between these shapes is the text "Obrigado !".

Obrigado !