

Documentação com o SpringDoc



Uma API bem construída deve ser simples de ser consumida, por isso é fundamental documentar a API da maneira correta. O objetivo da Documentação é apresentar uma espécie de “manual de instruções” que permita que qualquer pessoa (Desenvolvedora ou não), que ainda não tenha pleno conhecimento do domínio da sua aplicação, **consuma a API de maneira fácil, rápida, eficaz e autônoma**.

1. OpenApi

A **OpenApi**, trata-se de uma especificação Open Source, que auxilia as pessoas desenvolvedoras nos processos de **definir, criar, documentar e consumir** API's REST e RESTful. A OpenApi tem por objetivo padronizar este tipo de integração, descrevendo os recursos que uma API possui, os respectivos endpoints, os dados que serão solicitados nas Requisições, os dados que serão retornados nas Respostas, os Status HTTP, os modelos de dados, os métodos de autenticação, entre outros.

APIs REST são frequentemente usadas para a integração de aplicações, seja para consumir serviços de terceiros, seja para prover novos serviços. Para estas APIs, a especificação OpenApi facilita a modelagem, a documentação e a geração de código.



[Site Oficial: OpenApi](#)



[Documentação: Especificação OpenApi](#)

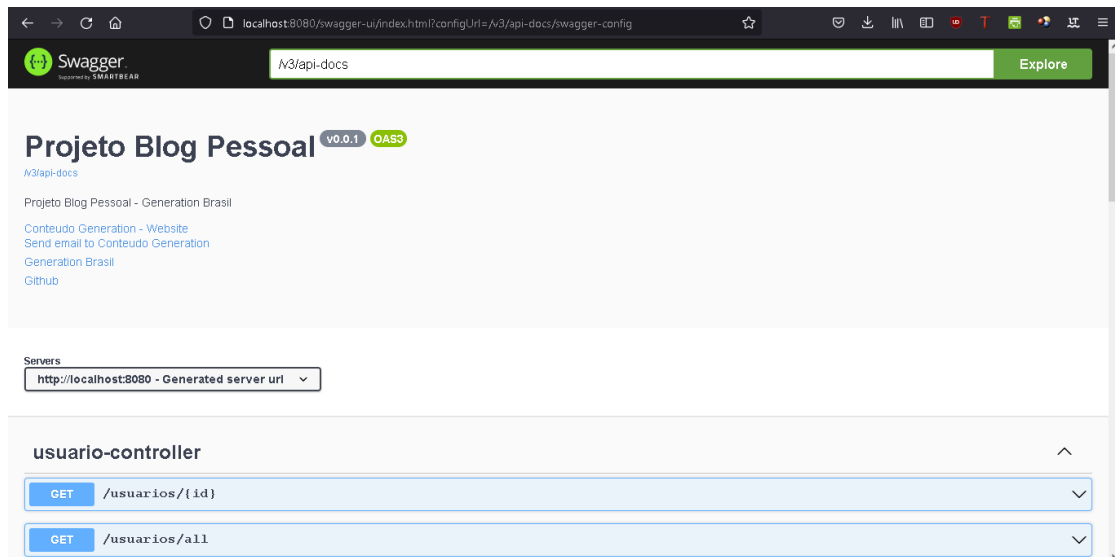
2. Swagger

O Swagger é uma poderosa ferramenta que ajuda pessoas Desenvolvedoras a **projetar, desenvolver, documentar e consumir serviços** web REST e RESTful, de forma interativa e dinâmica, aproveitando ao máximo todos os Recursos da especificação OpenApi. Apesar de ser conhecida principalmente por sua interface **Swagger UI**, o software inclui suporte para documentação automatizada da API, geração de código e testes.



[Site Oficial: Swagger](#)

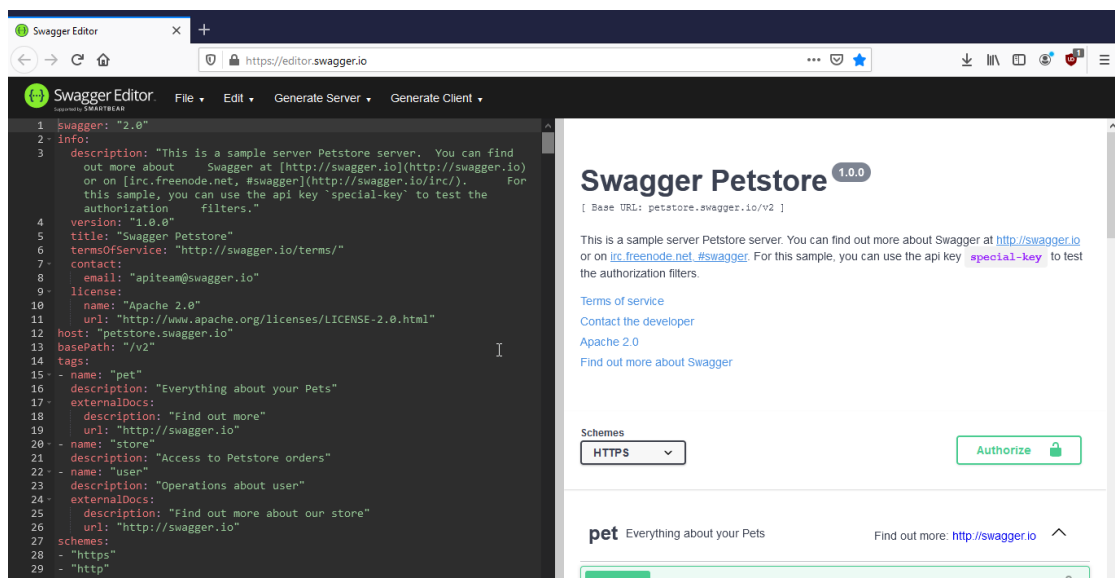
2.1. Swagger-UI



 [Site Oficial: Swagger-UI](https://swagger.io)

2.2. Swagger Editor

O Editor Swagger é um editor de código aberto para projetar, definir e documentar APIs RESTful na Especificação Swagger.



 [Site Oficial: Swagger-Editor](https://editor.swagger.io)

3. SpringDoc

A biblioteca Java **Springdoc-OpenApi** é uma implementação da OpenApi + Swagger, que ajuda a automatizar a geração de documentação API em projetos SpringBoot. A Springdoc-OpenApi examina um aplicativo em tempo de execução para Criar a Documentação da API, com base nas configurações do Spring, na estrutura das Classes e algumas Anotações. A documentação é gerada automaticamente no formato JSON / YAML e HTML. Esta documentação pode ser completada por comentários usando Anotações do Swagger.

YAML é um formato de serialização (codificação de dados) de dados legíveis por humanos inspirado no XML, sendo amplamente utilizada em arquivos de configuração, assim como o XML e o JSON. YAML é um acrônimo recursivo que significa “YAML Ain’t Markup Language” (em português, “YAML não é linguagem de marcação”). Como é usado frequentemente XML para serialização de dados e XML é uma autêntica linguagem de marcação de documentos, é razoável considerar o YAML como uma linguagem de marcação rápida.

 [Site Oficial: YAML](#)



[Site Oficial: SpringDoc](#)

Vamos criar a Documentação do nosso Projeto Blog Pessoal no formato Digital (Swagger-UI) através da Biblioteca **SpringDoc** e no formato Impresso (PDF), através do **Swagger Editor**.



Passo 01- Adicionar a Dependência do Spring Doc

Abra o arquivo pom.xml e insira a dependência do SpringDoc:

```
<dependency>
  <groupId>org.springdoc</groupId>
  <artifactId>springdoc-openapi-ui</artifactId>
  <version>1.5.13</version>
</dependency>
```



[Documentação: Dependência do SpringDoc](#)



ATENÇÃO: A Documentação sugere utilizar as versão 1.6.0 do SpringDoc, porém esta versão está apresentando alguns erros com as versões mais novas do SpringBoot. Utilize a versão 1.5.13, que está estabilizada no momento em que esta documentação foi escrita.



Passo 02- Configurar o arquivo application.properties

```
13
14 springdoc.api-docs.path=/v3/api-docs
15 springdoc.swagger-ui.path=/swagger-ui.html
16 springdoc.swagger-ui.operationsSorter=method
17 springdoc.swagger-ui.disable-swagger-default-url=true
18 springdoc.swagger-ui.use-root-path=true
19 springdoc.packagesToScan=br.org.generation.blogpessoal.controller
20
```

Linha 14: Define o Caminho para a Documentação do OpenAPI no formato JSON.

Linha 15: Define o Caminho para a Documentação do Swagger-ui no formato HTML.

Linha 16: Define a ordem em que os Recursos serão exibidos no Swagger-ui.

Linha 17: Desabilita a URL padrão do Swagger-ui (Exemplo no site do Swagger).

Linha 18: Define o Swagger-ui como a página inicial da aplicação.

Linha 19: Define o nome da Package da Camada Controladora (Controller)

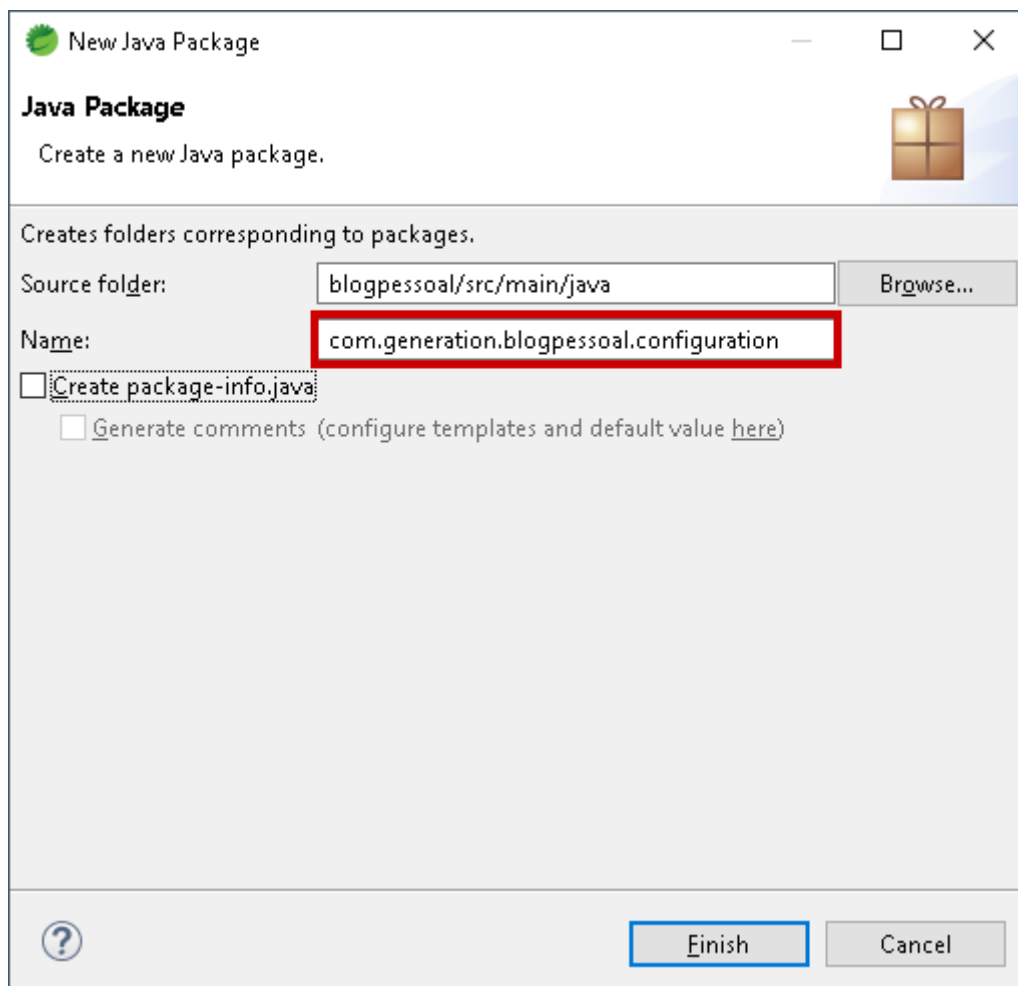


Documentação: [application.properties](#)

Passo 03 - Criar a Camada configuration e a Classe SwaggerConfig

A Camada **Configuration** é responsável por implementar configurações específicas da aplicação. Em nosso Projeto, vamos utilizar esta Camada para implementar a Documentação da API com o Swagger

1. No lado esquerdo superior, na Guia **Package explorer**, clique com o botão direito do mouse sobre a Package **com.generation.blogpessoal**, na Source Folder **src/main/java** e clique na opção **New → Package**.
2. Na janela **New Java Package**, no item **Name**, acrescente no final do nome da Package **.configuration**, como mostra a figura abaixo:



3. Clique no botão **Finish** para concluir.
4. Clique com o botão direito do mouse sobre o **Pacote Configuration** (**com.generation.blogpessoal.configuration**), na Source Folder Principal (**src/main/java**), e na sequência, clique na opção **New → Class**

5. Na janela **New Java Class**, no item **Name**, digite o nome da Classe (**SwaggerConfig**), como mostra a figura abaixo:

New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

6. Clique no botão **Finish** para concluir.

Agora vamos criar o código como mostra a figura abaixo:

```
14
15@Configuration
16public class SwaggerConfig {
17
18    @Bean
19    public OpenAPI springBlogPessoalOpenAPI() {
20        return new OpenAPI()
21            .info(new Info()
22                .title("Projeto Blog Pessoal")
23                .description("Projeto Blog Pessoal - Generation Brasil")
24                .version("v0.0.1")
25            ).license(new License()
26                .name("Generation Brasil")
27                .url("https://brazil.generation.org/"))
28            .contact(new Contact()
29                .name("Conteudo Generation")
30                .url("https://github.com/conteudoGeneration")
31                .email("conteudogeneration@gmail.com"))
32            .externalDocs(new ExternalDocumentation()
33                .description("Github")
34                .url("https://github.com/conteudoGeneration/"));
35    }
36}
```

Vamos analisar o código:

Linha 15: A Anotação (Annotation) **@Configuration** indica que a Classe é do tipo configuração, ou seja, define uma classe como fonte de definições de beans e é uma das anotações essenciais se você estiver usando a configuração baseada em Java.

Linha 18: A Anotação **@Bean** utilizada em métodos de uma classe, geralmente marcada com **@Configuration**, indica ao Spring que ele deve invocar aquele método e gerenciar o objeto retornado por ele, ou seja, agora este objeto pode ser injetado em qualquer ponto da sua aplicação.

Bean: No Spring, os objetos que formam a espinha dorsal da sua aplicação e que sejam gerenciados pelo Spring são chamados de beans. Um bean é um objeto que é instanciado, montado e gerenciado pelo Spring IoC (Inversão de Controle) container. Existem diversas forma de se criar beans no Spring, você pode criar classes anotadas com **@Configuration** ou **@Service** para serem gerenciadas pelo Spring. Assim como pode usar o **@Bean** em um método, e tornar a instância retornada pelo método como um objeto gerenciado pelo Spring (seja de uma classe própria ou de terceiros).

Estas classes, que na visão do Spring são os beans, para você nada mais são do que classes você irá escrever as regras de funcionamento da sua aplicação.

Linha 20: Cria um Objeto da Classe OpenAPI, que gera a documentação no Swagger utilizando a especificação OpenAPI.

Linhas 21 a 24: Insere as informações sobre a API (Nome do projeto (Title), Descrição e Versão)

Linhas 25 a 27: Insere as informações referentes a licença da API (Nome e Link)

Linhas 28 a 31: Insere as informações de contato da pessoa Desenvolvedora (Nome, Site e e E-mail)

Linhas 32 a 24: Insere as informações referentes a Documentações Externas (Github, Gitpage e etc), onde são informados o Nome e o Link.



[Artigo: O Lado Legal do Open Source](#)



[Documentação: @Configuration](#)



[Artigo: Spring Bean](#)



[Documentação: Java Beans](#)



[Documentação: OpenApi\(\)](#)

```
36
37 @Bean
38 public OpenApiCustomiser customerGlobalHeaderOpenApiCustomiser() {
39
40     return openApi -> {
41         openApi.getPaths().values().forEach(pathItem -> pathItem.readOperations().forEach(operation -> {
42
43             ApiResponseResponses apiResponses = operation.getResponses();
44
45             apiResponses.addApiResponse("200", createApiResponse("Sucesso!"));
46             apiResponses.addApiResponse("201", createApiResponse("Objeto Persistido!"));
47             apiResponses.addApiResponse("204", createApiResponse("Objeto Excluido!"));
48             apiResponses.addApiResponse("400", createApiResponse("Erro na Requisição!"));
49             apiResponses.addApiResponse("401", createApiResponse("Acesso Não Autorizado!"));
50             apiResponses.addApiResponse("404", createApiResponse("Objeto Não Encontrado!"));
51             apiResponses.addApiResponse("500", createApiResponse("Erro na Aplicação!"));
52
53         }));
54     };
55 }
56
57 private ApiResponse createApiResponse(String message) {
58
59     return new ApiResponse().description(message);
60 }
61
62
63 }
64
```

A Classe **OpenApiCustomiser** permite personalizar o Swagger, baseado na Especificação OpenAPI. O Método acima, personaliza todas as mensagens HTTP Responses (Respostas das requisições) do Swagger.

Linha 40: Cria um Objeto da Classe OpenAPI, que gera a documentação no Swagger utilizando a especificação OpenAPI.

Linha 41: Cria um primeiro looping que fará a leitura de todos os recursos (Paths) através do Método `getPaths()`, que retorna o caminho de cada endpoint. Na sequência, cria um segundo looping que Identificará qual Método HTTP (Operations), está sendo executado em cada endpoint através do Método `readOperations()`. Para cada Método, todas as mensagens serão lidas e substituídas pelas novas mensagens.

Linha 43: Cria um Objeto da **Classe `ApiResponse`**s, que receberá as Respostas HTTP de cada endpoint (Paths) através do método **`getResponses()`**.

Linhas 44 a 51: Adiciona as novas Respostas no endpoint, substituindo as atuais e acrescentando as demais, através do Método **`addApiResponse()`**, identificadas pelo HTTP Status Code (200, 201 e etc).

Linhas 57 a 61: O Método **`createApiResponse()`** adiciona uma descrição (Mensagem), em cada Resposta HTTP.



[Documentação: Paths and Operations](#)



[Documentação: ApiResponse](#)



[Documentação: Operation](#)



[Documentação: PathItem](#)

Passo 04 - Alteração na Classe Usuario

Vamos configurar o atributo **usuario**, da **Classe Usuario**, para emitir um lembrete no Swagger de que deve ser digitado um e-mail no valor do atributo. Para isso, utilizaremos a anotação **@Schema**.

A anotação **@Schema** nos permite controlar as definições específicas do Swagger, como descrição (valor), nome, tipo de dados, valores de exemplo e valores permitidos para as propriedades do modelo. No atributo **usuario**, vamos utilizar a propriedade **example**.

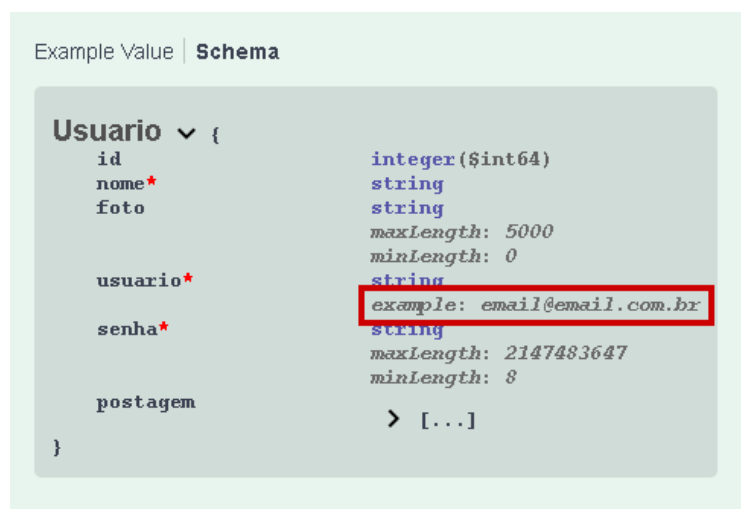
Abra o arquivo **Usuario**, da **Camada Model**, localize o atributo **usuario** e altere de:

```
@NotNull(message = "O atributo Usuário é Obrigatório!")
>Email(message = "O atributo Usuário deve ser um email válido!")
private String usuario;
```

Para:

```
@Schema(example = "email@email.com.br")
@NotNull(message = "O atributo Usuário é Obrigatório!")
>Email(message = "O atributo Usuário deve ser um email válido!")
private String usuario;
```

Observe na figura abaixo que o Swagger indicará que o atributo **usuario** deve ser um e-mail.

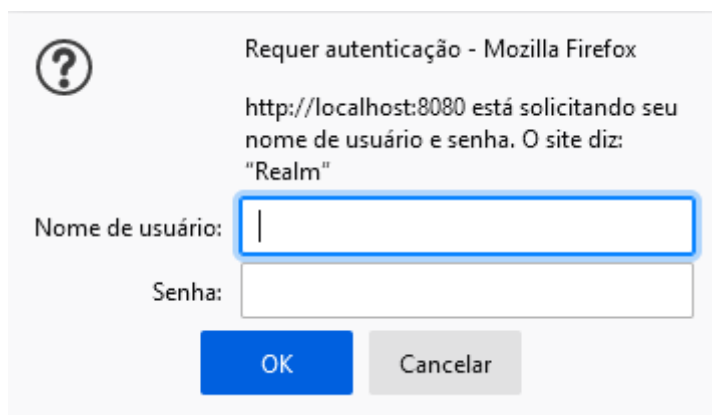


Passo 05 - Executar o Swagger

1. Abra o seu navegador na Internet e digite o seguinte endereço abaixo para abrir a sua documentação.

<http://localhost:8080/swagger-ui.html>

2. Em aplicações com a segurança habilitada, você precisará efetuar o login com uma conta de usuário antes de exibir a sua documentação no Swagger. Utilize o usuário em memória (root) ou um usuário cadastrado no Banco de Dados local via Postman para fazer o login.



Requer autenticação - Mozilla Firefox

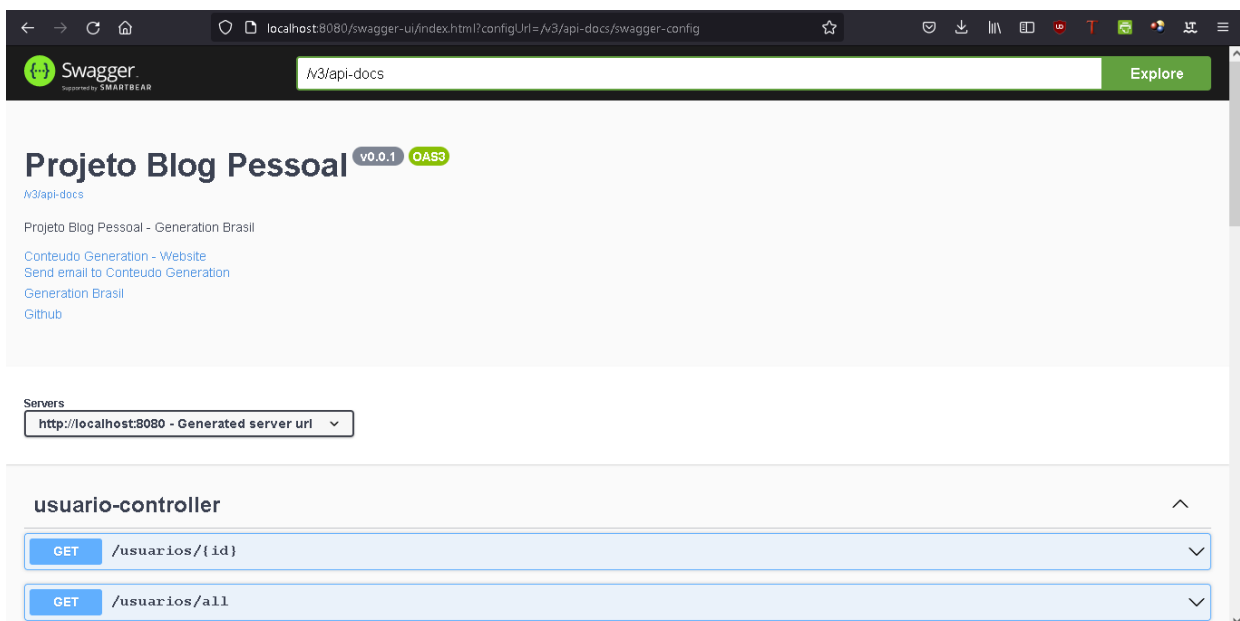
http://localhost:8080 está solicitando seu nome de usuário e senha. O site diz: "Realm"

Nome de usuário:

Senha:

OK Cancelar

3. Pronto! A sua documentação no Swagger será exibida.

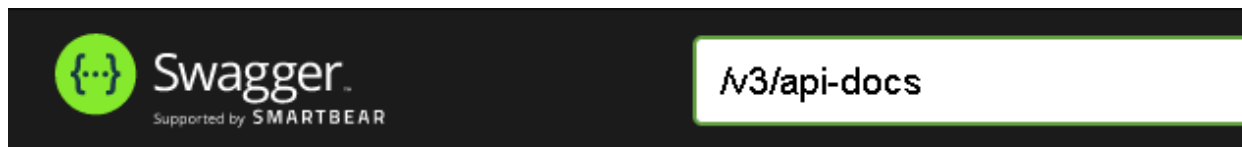


The screenshot shows the Swagger UI interface in a web browser. The address bar displays `localhost:8080/swagger-ui/index.html?configUrl=/v3/api-docs/swagger-config`. The Swagger logo is in the top left, and a search bar with `/v3/api-docs` and an `Explore` button is in the top right. The main heading is **Projeto Blog Pessoal** with version tags `v0.0.1` and `OAS3`. Below this, there are links for `/v3/api-docs`, the project name, and a list of links: `Conteúdo Generation - Website`, `Send email to Conteúdo Generation`, `Generation Brasil`, and `Github`. A `Servers` dropdown menu shows `http://localhost:8080 - Generated server url`. The `usuario-controller` section is expanded, showing two endpoints: `GET /usuarios/{id}` and `GET /usuarios/all`, each with a dropdown arrow.



Passo 06 - Gerar o PDF da Documentação

1. No Swagger, clique no link: <http://localhost:8080/v3/api-docs> para visualizar a documentação no formato JSON.



Projeto Blog Pessoal

v0.0.1

OAS3

[v3/api-docs](#)

Projeto Blog Pessoal - Generation Brasil

[Conteudo Generation - Website](#)

[Send email to Conteudo Generation](#)

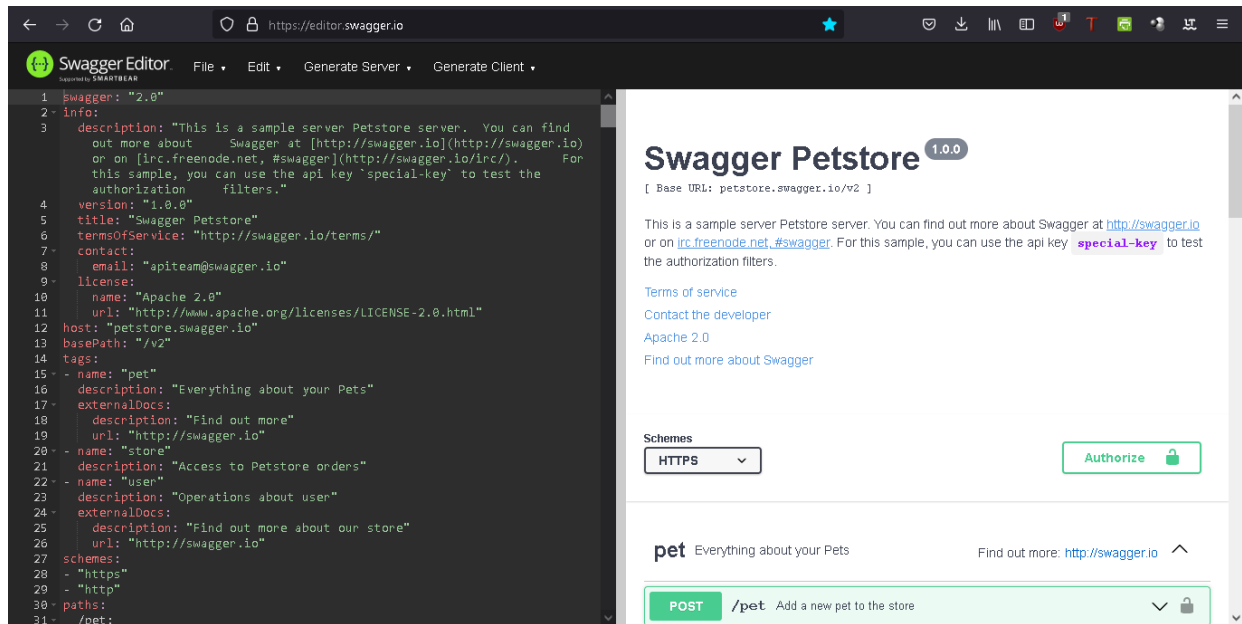
[Generation Brasil](#)

[Github](#)

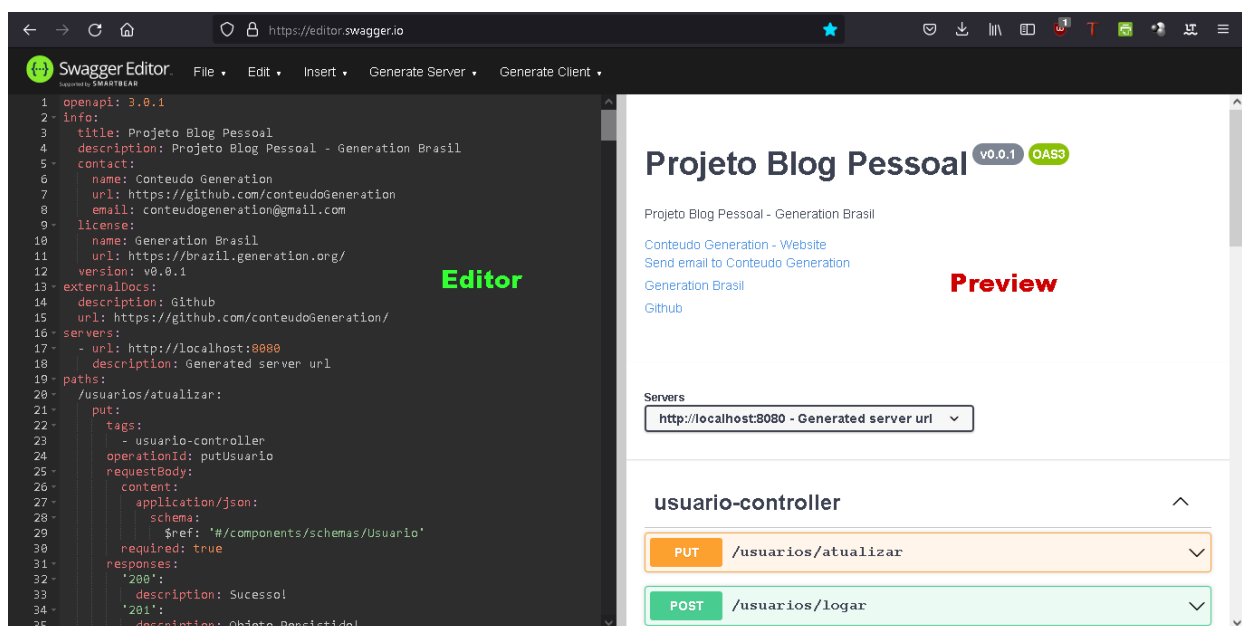
2. Visualize o código no formato Raw Data ou Dados brutos (No Chrome e no Edge é o formato padrão), Selecione todo o código (**Ctrl + A**) e Copie (**Ctrl + C**).



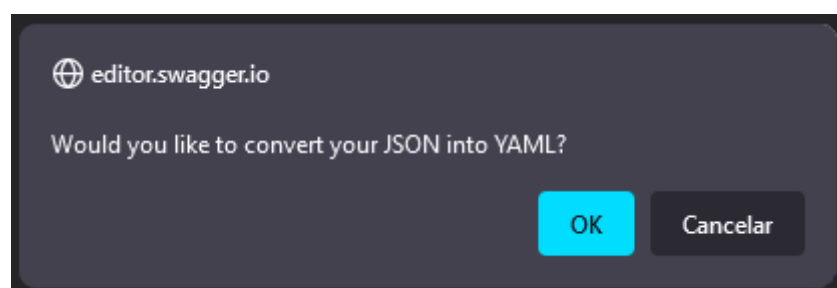
3. Acesse o site **Swagger Editor** (<https://editor.swagger.io/>).



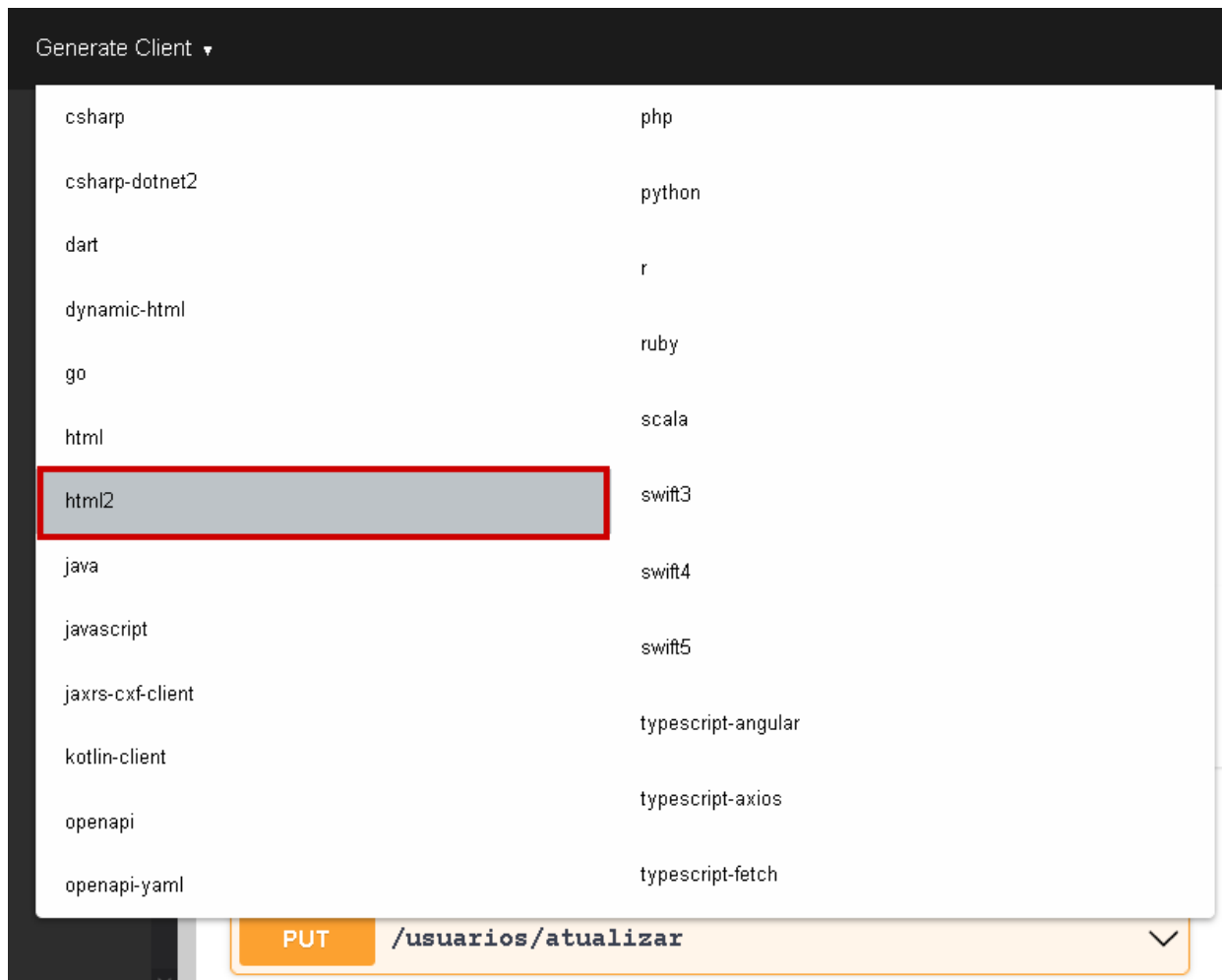
4. No **Swagger Editor**, apague o código exemplo e cole o código copiado da Documentação dentro do Editor (**Ctrl + V**).



5. O Swagger Editor perguntará se você deseja converter o código JSON em YAML. Clique em OK para converter.



6. No menu **Generate Client**, selecione a opção **html2**.

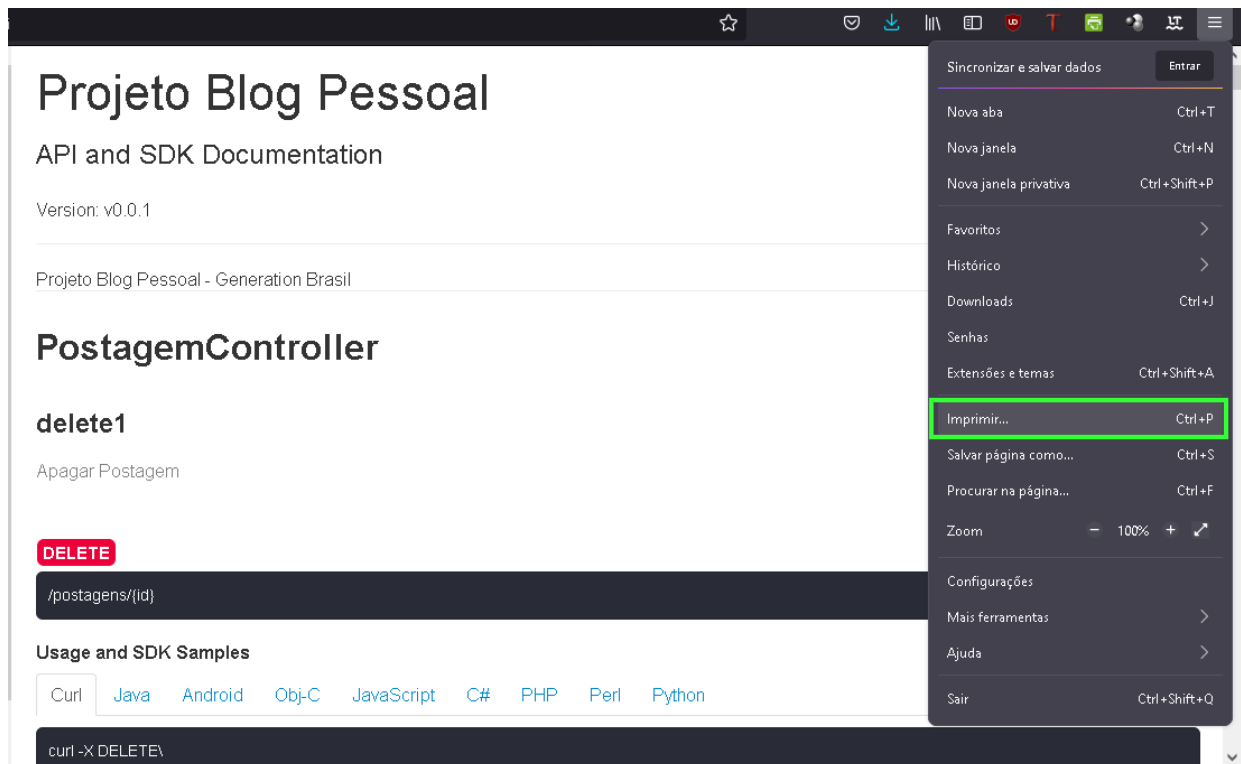


7. O **Swagger Editor** solicitará o download do arquivo **html2-client-generated.zip**.

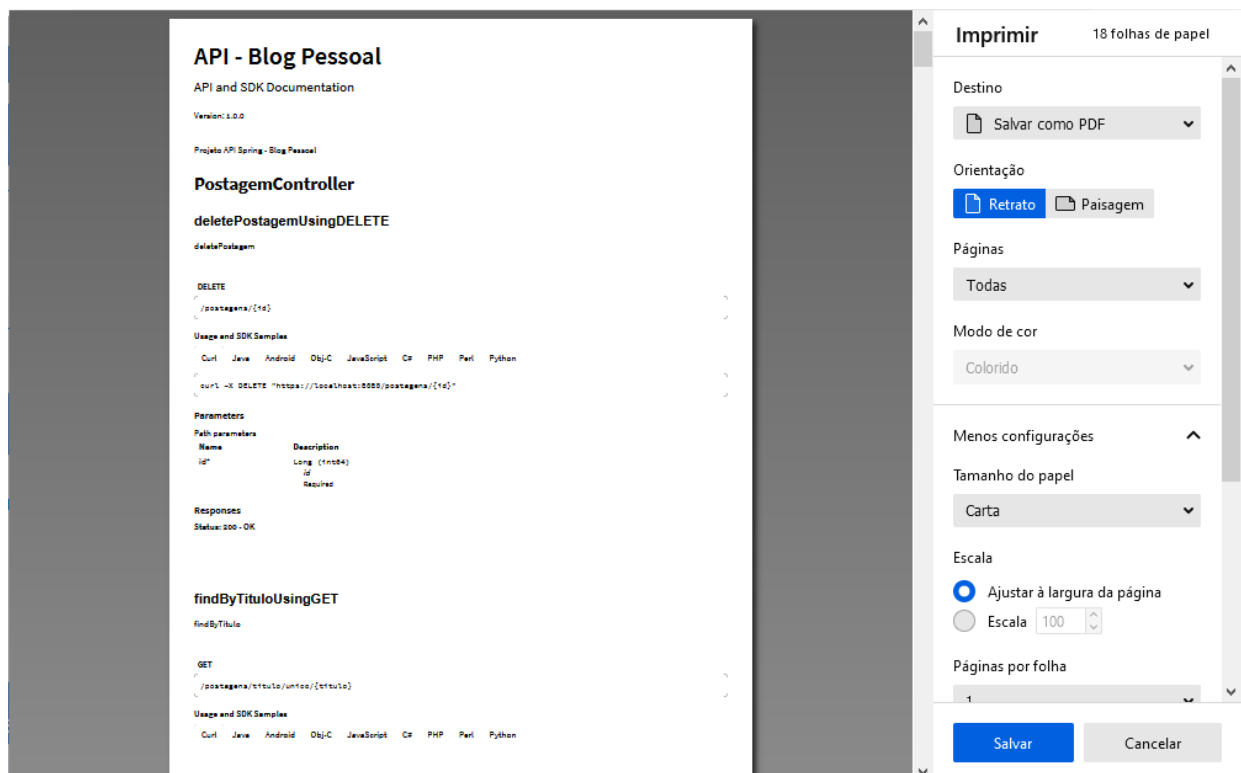
Faça o download do arquivo, descompacte no seu computador e abra o arquivo **index.html** no seu navegador.

Nome	Tipo	Tamanho Compact...
.swagger-codegen	Pasta de arquivos	
.swagger-codegen-ignore	Arquivo SWAGGER-CODE...	1 KB
index.html	Firefox Document	162 KB

8. No seu navegador, no menu principal, clique em **Imprimir**.



9. Na janela de impressão, no item **Destino**, selecione a opção **Salvar em PDF** e clique no Botão **Salvar**.



10. Documentação em PDF gerada!

