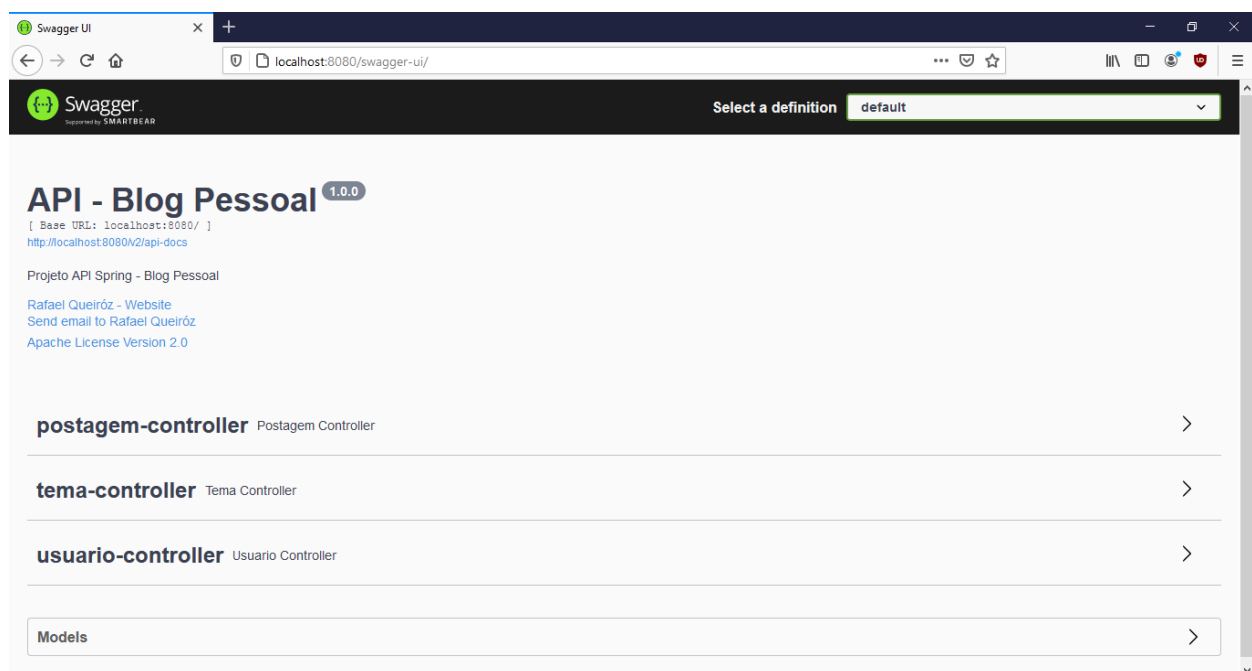


Documentação da API com o Swagger 3.0

O Swagger trata-se de uma aplicação open source, que auxilia as pessoas desenvolvedoras nos processos de definir, criar, documentar e consumir API's REST. O Swagger tem por objetivo padronizar este tipo de integração, descrevendo os recursos que uma API deve possuir, como endpoints, dados recebidos, dados retornados, códigos HTTP, 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. Para estas APIs, o Swagger facilita a modelagem, a documentação e a geração de código.



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

Passo 01- Dependências

Configurando o pom.xml

Insira a seguinte dependência no projeto:

```
<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-boot-starter</artifactId>
  <version>3.0.0</version>
</dependency>
```

 [Documentação: Dependência do Swagger](#)

 [Código fonte: pom.xml](#)

Passo 02 - SwaggerConfig

Crie uma nova package (pacote) no seu projeto chamada **configuration** , dentro dela crie uma classe chamada SwaggerConfig e configure segundo o modelo abaixo:

```
@Configuration
public class SwaggerConfig {
    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .select()
            .apis(RequestHandlerSelectors
                .basePackage("br.org.generation.blogpessoal.controller"))
            .paths(PathSelectors.any())
            .build()
            .apiInfo(metadata())
            .useDefaultResponseMessages(false)
            .globalResponses(HttpMethod.GET, responseMessage())
            .globalResponses(HttpMethod.POST, responseMessage())
            .globalResponses(HttpMethod.PUT, responseMessage())
            .globalResponses(HttpMethod.DELETE, responseMessage());
    }
}
```

```

public static ApiInfo metadata() {

    return new ApiInfoBuilder()
        .title("API - Blog Pessoal")
        .description("Projeto API Spring - Blog Pessoal")
        .version("1.0.0")
        .license("Apache License Version 2.0")
        .licenseUrl("https://github.com/rafaelq80")
        .contact(contact())
        .build();

}

private static Contact contact() {

    return new Contact("Rafael Queiróz",
        "https://github.com/rafaelq80",
        "rafaelproinfo@gmail.com");

}

private static List<Response> responseMessage() {

    return new ArrayList<Response>() {

        private static final long serialVersionUID = 1L;

        {

            add(new ResponseBuilder().code("200")
                .description("Sucesso!").build());
            add(new ResponseBuilder().code("201")
                .description("Criado!").build());
            add(new ResponseBuilder().code("400")
                .description("Erro na requisição!").build());
            add(new ResponseBuilder().code("401")
                .description("Não Autorizado!").build());
            add(new ResponseBuilder().code("403")
                .description("Proibido!").build());
            add(new ResponseBuilder().code("404")
                .description("Não Encontrado!").build());
            add(new ResponseBuilder().code("500")
                .description("Erro!").build());

        }

    };

}

}

```

1. Método public Docket api()

Define a package onde estão as classes do tipo `@RestController`, para que o Swagger mapeie todas as classes e seus respectivos endpoints para montar a documentação do projeto.

```
.apis(RequestHandlerSelectors.basePackage("br.org.generation.blogpessoal.controller"))
```

Seleciona todos os endpoints de todos os recursos.

```
.paths(PathSelectors.any())
```

Ignora as mensagens padrão de todas as Responses (Respostas das requisições).

```
.useDefaultResponseMessages(false)
```

Personaliza as mensagens de algumas Responses (Respostas da requisições), através do Método **responseMessage()**. Cada linha corresponde a um tipo de requisição.

```
.globalResponses(HttpMethod.GET, responseMessage())  
.globalResponses(HttpMethod.POST, responseMessage())  
.globalResponses(HttpMethod.PUT, responseMessage())  
.globalResponses(HttpMethod.DELETE, responseMessage())
```



[Documentação: RequestHandlerSelectors.basePackage](#)



[Documentação: .paths\(PathSelectors.any\(\)\)](#)



[Documentação: .useDefaultResponseMessages\(false\)](#)



[Documentação: .globalResponses\(\)](#)

2. Método public static ApiInfo metadata()

1) Define o título da sua aplicação que será exibida na documentação.

```
.title("Blog Pessoal")
```

2) Cria uma descrição para a sua aplicação.

```
.description("API do Projeto de blog pessoal")
```

3) Define a versão da sua aplicação.

```
.version("1.0.0")
```

4) Define o tipo de licença da sua aplicação.

```
.license("Apache License Version 2.0")
```

5) Informa o link de acesso da licença da sua aplicação (geralmente se aplica a licença no Github).

```
.licenseUrl(https://github.com/rafaelq80)
```

**Em nosso exemplo inserimos o endereço do Github somente para termos uma url válida.*

6) Define os dados para contato com o desenvolvedor inseridos no método `contact()`.

```
.contact(contact()).build()
```

 [Documentação: `ApiInfo\(\)`](#)

3. Método `private static Contact contact()`

Define os dados do Desenvolvedor (Nome, Website e o E-mail). Insira somente um e-mail e um endereço http.

```
return new Contact("Rafael Queiróz", "https://github.com/rafaelq80",  
"rafaelproinfo@gmail.com");
```

 [Documentação: `Contact\(\)`](#)

4. Método private static List responseMessage()

Define as mensagens personalizadas para os códigos de Resposta do protocolo http (http Response) para todos os verbos (GET, POST, PUT e DELETE). Cada linha é referente a um Status Code.

```
add(new ResponseBuilder().code("200").description("Sucesso!").build());
add(new ResponseBuilder().code("201").description("Criado!").build());
add(new ResponseBuilder().code("400").description("Erro na
requisição!").build());
add(new ResponseBuilder().code("401").description("Não Autorizado!").build());
add(new ResponseBuilder().code("403").description("Proibido!").build());
add(new ResponseBuilder().code("404").description("Não Encontrado!").build());
add(new ResponseBuilder().code("500").description("Erro!").build());
```



[Documentação: ResponseBuilder](#)

Salve todos arquivos e inicie a sua aplicação.



[Código fonte: SwaggerConfig.java](#)

Passo 03 - 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 **@ApiModelProperty**.

A anotação **@ApiModelProperty** 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:

```
@ApiModelProperty(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.

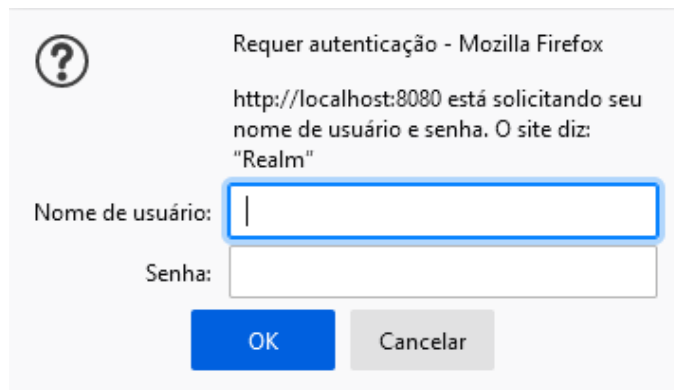


Passo 04 - Executando o Swagger

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

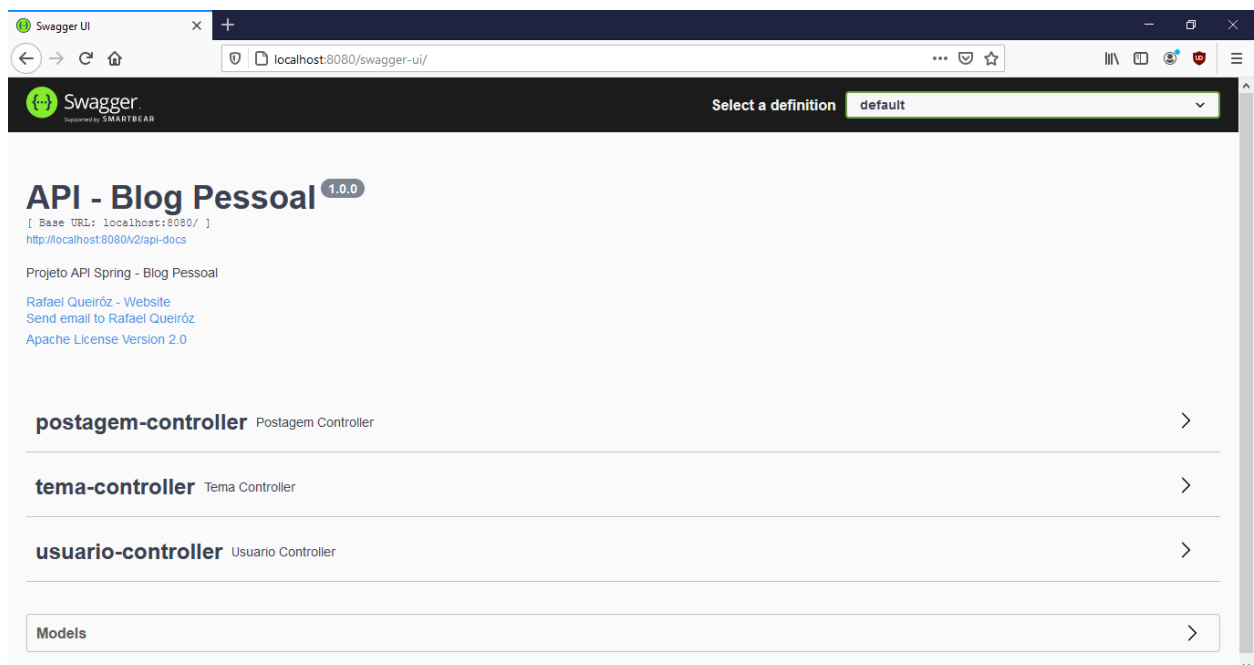
<http://localhost:8080/swagger-ui/>

2. Em aplicações com camadas de segurança, 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.



A screenshot of a Firefox authentication dialog box. The title bar says "Requer autenticação - Mozilla Firefox". The main text says "http://localhost:8080 está solicitando seu nome de usuário e senha. O site diz: 'Realm'". There are two input fields: "Nome de usuário:" and "Senha:". Below the fields are two buttons: "OK" (blue) and "Cancelar" (grey).

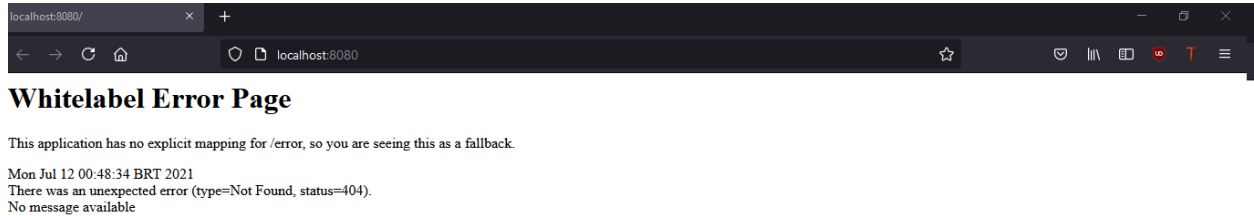
3. Pronto! A sua documentação no Swagger está funcionando.





Passo 05 - Definindo o Swagger como página principal da API

Vamos configurar o **Swagger** como página principal da nossa API, ou seja, ao digitarmos o endereço: <http://localhost:8080>, ao invés de abrir a página abaixo:



Abriremos a página do Swagger.

Na camada principal do nosso projeto Blog Pessoal (**br.org.generation.blogpessoal**) vamos abrir o arquivo **BlogpessoalApplication**

Vamos alterar o arquivo de:

```
package br.org.generation.blogpessoal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BlogpessoalApplication {

    public static void main(String[] args) {
        SpringApplication.run(BlogpessoalApplication.class, args);
    }

}
```

Para:

```
package br.org.generation.blogpessoal;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.servlet.ModelAndView;

@SpringBootApplication
@RestController
@RequestMapping("/")
public class BlogpessoalApplication {

    @GetMapping
    public ModelAndView swaggerUi() {
        return new ModelAndView("redirect:/swagger-ui/");
    }

    public static void main(String[] args) {
        SpringApplication.run(BlogpessoalApplication.class, args);
    }

}
```

As alterações acima transformam a classe principal da nossa API (**BlogpessoalApplication**) em uma classe do tipo **RestController**, que responderá à todas as requisições do tipo **GET** para o **endpoint "/"** (raiz do nosso projeto) e fará o redirecionamento para o Swagger, ou seja, o Swagger será a página home do nosso projeto.



[Código fonte: Projeto finalizado](#)

Passo 06 - Gerando o PDF da Documentação

1. No Swagger, clique no link: <http://localhost:8080/v2/api-docs> para visualizar a documentação no formato JSON.



API - Blog Pessoal

1.0.0

[Base URL: localhost:8080/]

<http://localhost:8080/v2/api-docs>

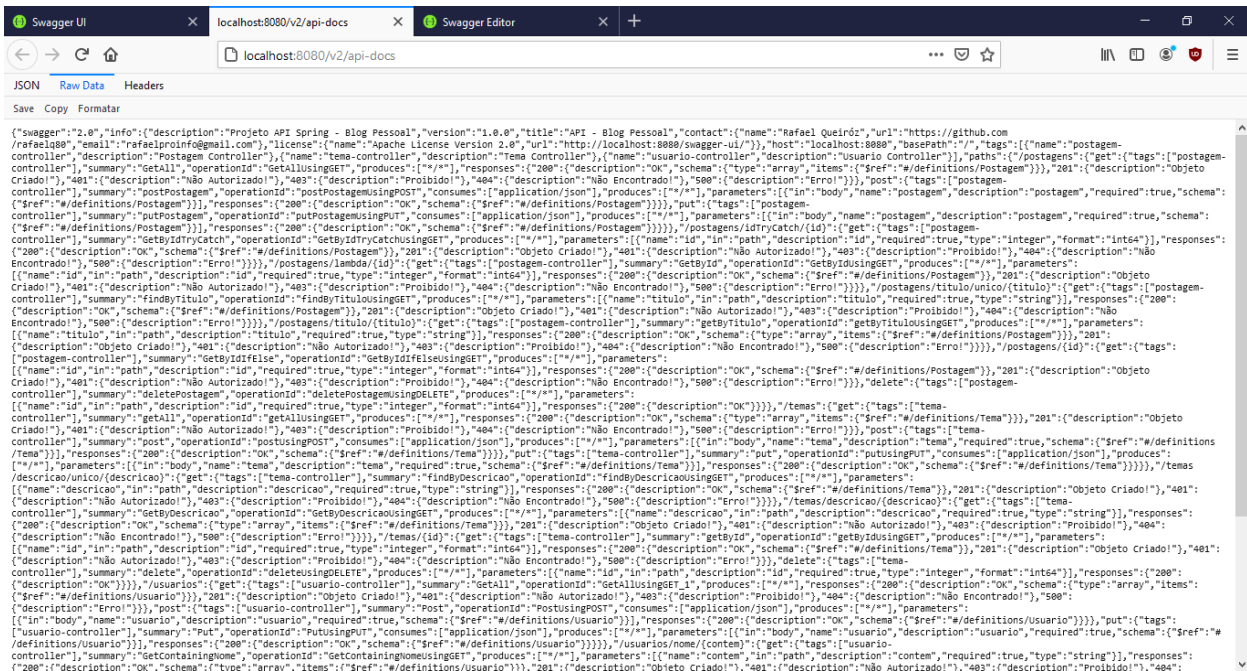
Projeto API Spring - Blog Pessoal

Rafael Queiróz - Website

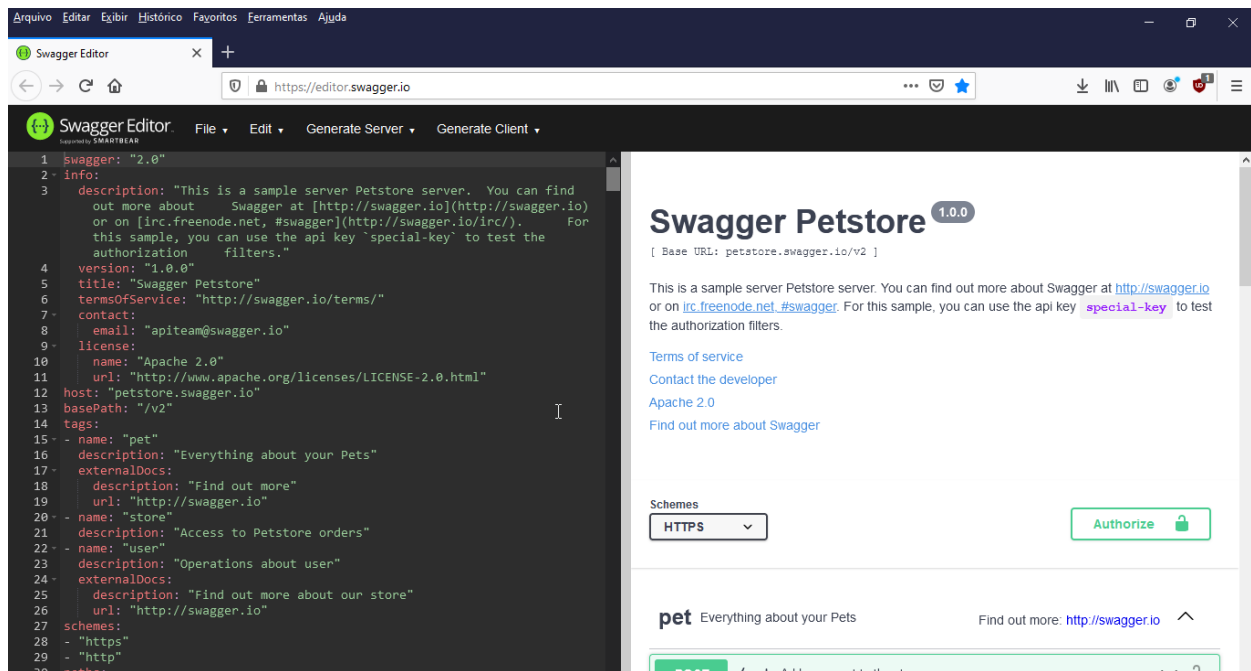
Send email to Rafael Queiróz

Apache License Version 2.0

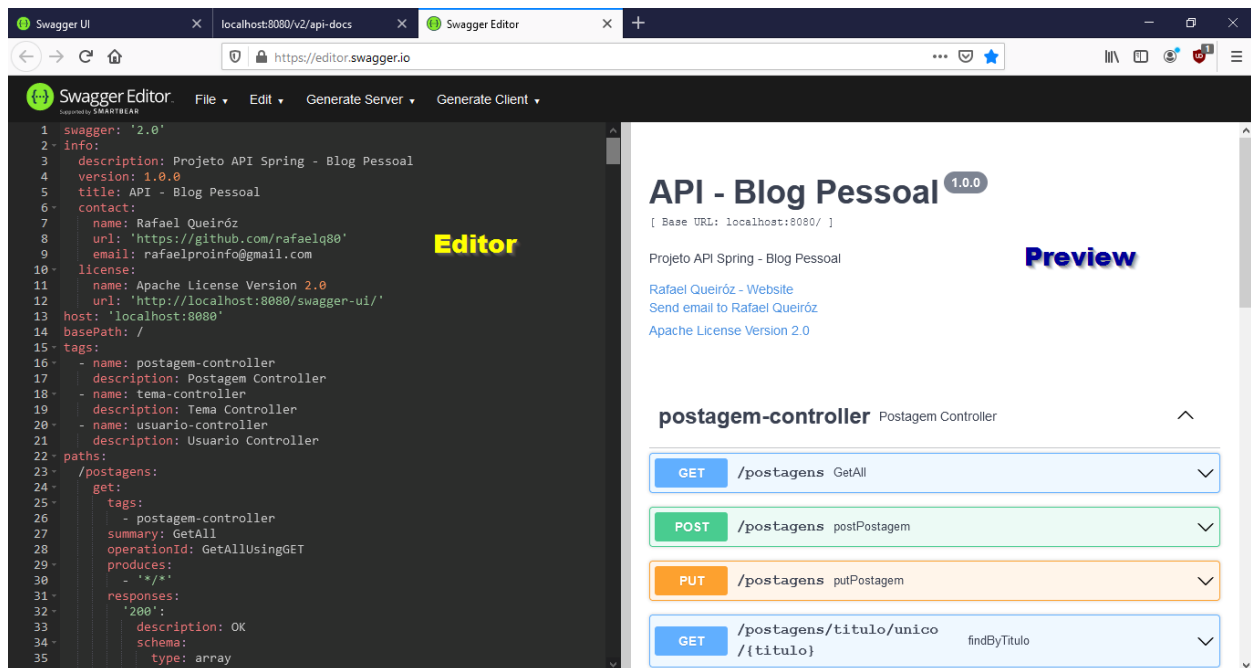
2. Visualize o código no formato Raw Data (No Chrome e no Edge é o formato padrão),
Selecione todo o código (Ctrl + A) e Copie (Ctrl + C).



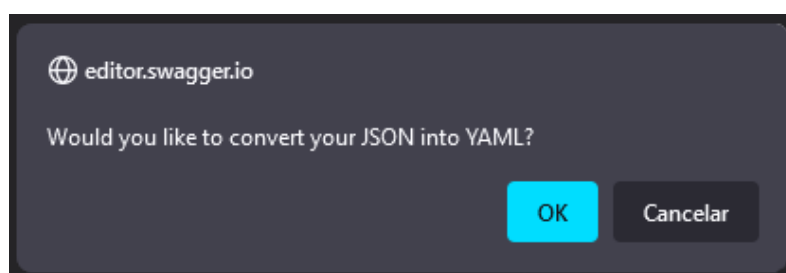
3. Acesse o site **Swagger Editor** (<https://editor.swagger.io/>).



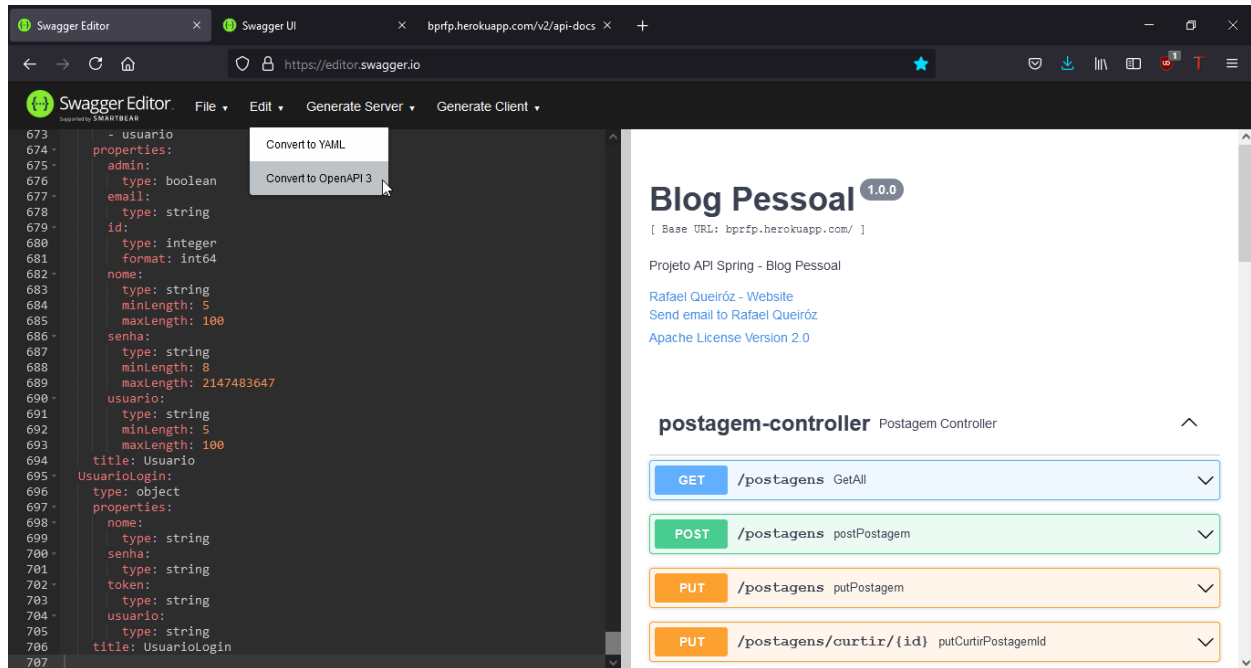
4. No **Swagger Editor**, apague o código exemplo e cole o código copiado da Documentação dentro do Editor (Ctrl + V).



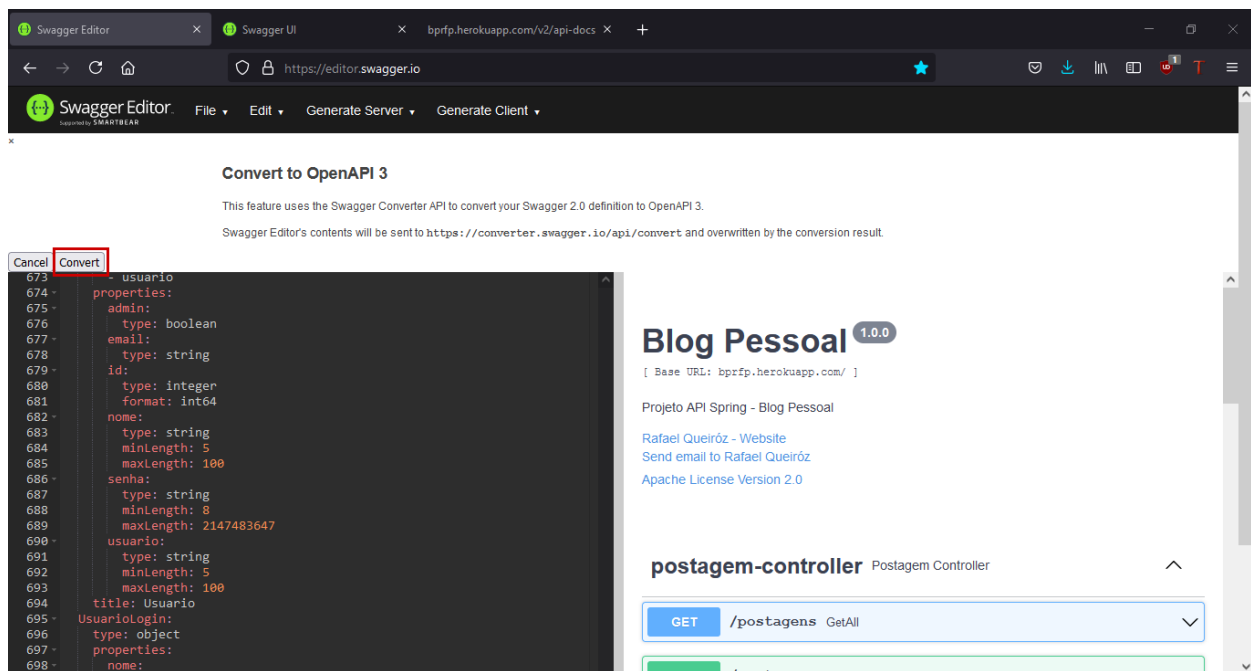
5. O Swagger Editor perguntará se você deseja converter o código JSON em YAML. Clique em OK para converter.



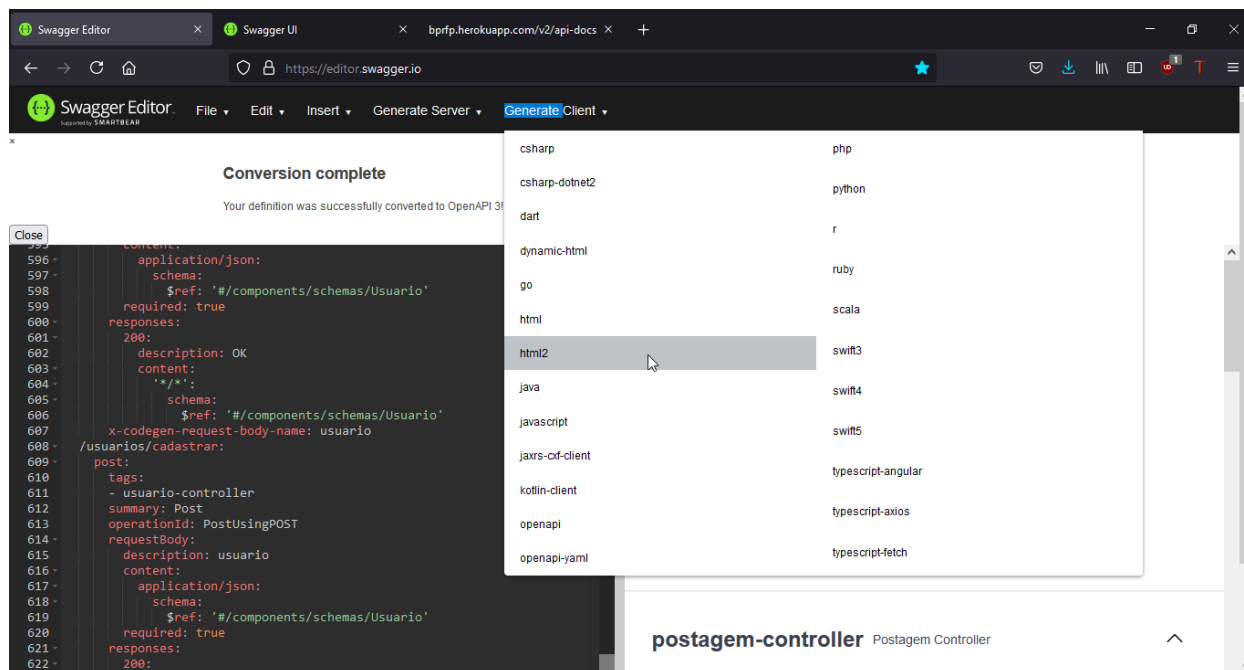
6. No Menu **Edit**, selecione a opção **Convert to OpenAPI3**



7. Em seguida, clique na guia **Convert**



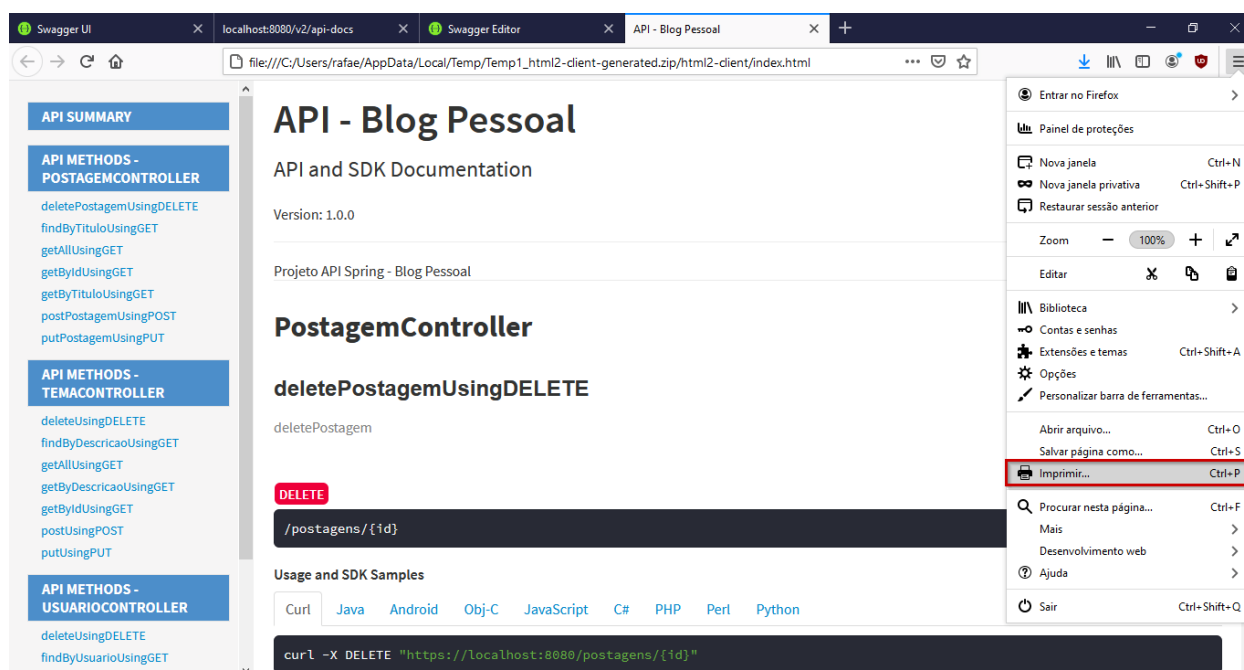
8) No menu **Generate Client**, selecione a opção **html2**.



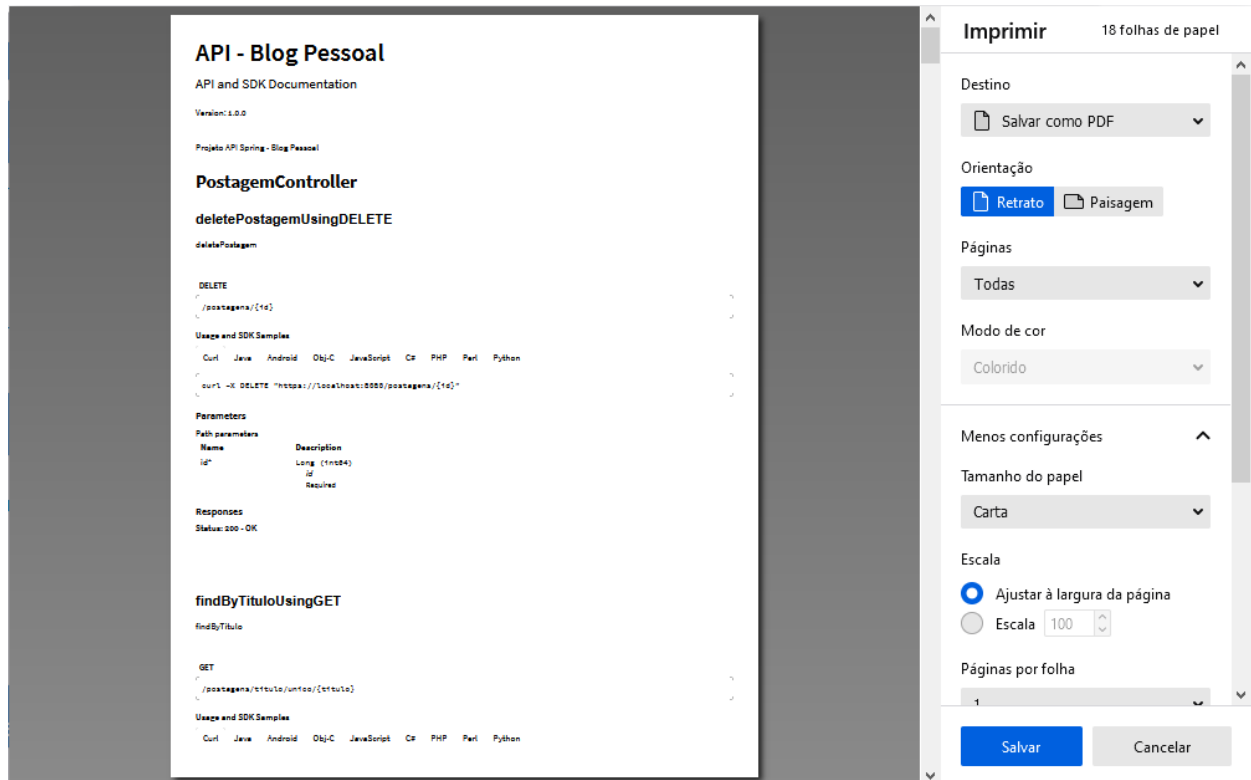
9. O **Swagger Editor** solicitará o download do arquivo **html2-client-generated.zip**. Faça o download do arquivo, descompacte no seu computador e abra o arquivo **index.html** no seu navegador.

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

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



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



Documentação em PDF gerada!

