

Swagger e OpenAPI

Guia de Documentação de APIs REST com Swagger e OpenAPI

O que é OpenAPI?

A Especificação OpenAPI (anteriormente conhecida como Especificação Swagger) é um formato de descrição de API para APIs REST. Um arquivo OpenAPI permite descrever toda a sua API, incluindo:

- Endpoints disponíveis (**/users**) e operações em cada endpoint (**GET /users, POST /users**)
- Parâmetros de entrada e saída para cada operação
- Métodos de autenticação
- Informações de contato, licença, termos de uso e outras

As especificações podem ser escritas em **YAML** ou **JSON**, e são legíveis por humanos e máquinas.

O que é o Swagger?

Swagger é um conjunto de **ferramentas de código** aberto desenvolvido com base na especificação OpenAPI, que ajuda você a **projetar, construir, documentar e consumir APIs REST**. As principais ferramentas incluem:

- Swagger Editor – **Editor online** baseado em navegador para escrever definições OpenAPI
- Swagger UI – **Gera uma documentação interativa** para APIs
- Swagger Codegen – Gera **stubs de servidor e bibliotecas de cliente** a partir de definições
- Swagger Core – Bibliotecas em Java para trabalhar com OpenAPI
- Swagger Parser – Biblioteca para analisar arquivos OpenAPI
- Swagger APIDom – Estrutura unificada para **descrever APIs em diferentes formatos**

Veja mais em: <https://swagger.io/docs/>

Por que usar OpenAPI?

A descrição da estrutura da API é a base da proposta do OpenAPI. Depois de definida, a especificação pode ser usada para:

- Gerar stubs de servidor com o Swagger Codegen
- Criar documentação interativa com o Swagger UI
- Integrar a especificação com ferramentas como o SoapUI para testes automatizados

Obs: Ao criarmos um stub de um servidor criamos automaticamente um **esqueleto (estrutura base)** de uma aplicação backend a partir da documentação OpenAPI. Ou seja, o Swagger Codegen gera o código inicial do servidor (rotas, endpoints, controladores etc.) com base no que está definido no seu arquivo .yaml ou .json.

Estrutura Básica OpenAPI (YAML)

```
1 openapi: 3.0.4
2 info:
3   title: Sample API
4   description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
5   version: 0.1.9
6
7 servers:
8   - url: http://api.example.com/v1
9     description: Servidor principal (produção)
10  - url: http://staging-api.example.com
11    description: Servidor de homologação interno para testes
12
13 paths:
14   /users:
15     get:
16       summary: Retorna uma lista de usuários.
17       description: Descrição estendida (opcional).
18       responses:
19         "200":
20           description: Um array JSON com nomes de usuários
21           content:
22             application/json:
23               schema:
24                 type: array
25                 items:
26                   type: string
```

Cabeçalho com metadados essenciais

Definição dos servidores da nossa aplicação

Rotas, com suas descrições, schemas e status code de requisição e resposta

Links Uteis:

- [Editor Online do Swagger](#)
- [Documentação Oficial OpenAPI](#)
- [Swagger UI no GitHub](#)
- [swagger-jsdoc \(npm\)](#)
- [swagger-ui-express \(npm\)](#)

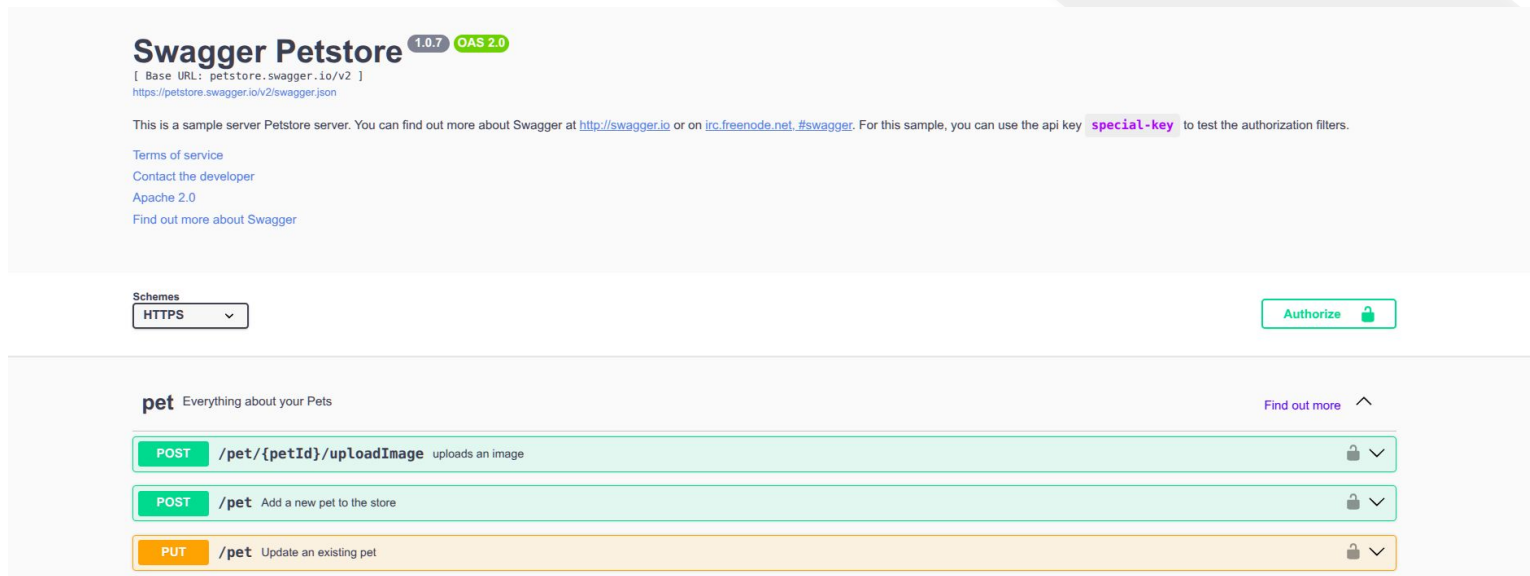
Com essas ferramentas, você poderá **criar APIs bem documentadas**, interativas e profissionais, **facilitando o desenvolvimento** e a integração com outros serviços.

Prática: Implementação Swagger

Documentando uma Api de crud em
express com swagger-ui e jsDoc

O que queremos?

O Swagger possui uma documentação de teste em: <https://petstore.swagger.io/> — é assim que queremos que nossa documentação final fique:



Você também pode acessar o JSON usado por essa documentação em: <https://petstore.swagger.io/v2/swagger.json>

Instalação de dependências

1. Baixe a biblioteca Swagger UI Express:



```
1 npm install swagger-ui-express  
2
```

2. E posteriormente a dependencia do Swagger com jsdoc:



```
1 npm install swagger-jsdoc
```

Importação e configurações

Adicione o middleware do swagger-ui

```
1 const express = require("express");
2 const app = express();
3
4 // Biblioteca para documentação
5 const swaggerUi = require("swagger-ui-express");
6 const swaggerJSDoc = require("swagger-jsdoc"); // commonjs
7 // Ou para
8 // const swaggerJSDoc from "swagger-jsdoc" // ES6
```

Adicione o middleware do swaggerJSDocs

```
1 const option = {
2   definition: {
3     openapi: "3.0.0",
4     info: {
5       title: "Swagger with Express",
6       description: "Essa api tem como objetivo demonstra o uso do swagger com express",
7       version: "1.0.0",
8       license: {
9         name: "MIT"
10      },
11       termsOfService: "http://localhost:3000/terms/",
12       contact: {
13         name: "Davi Candido",
14         email: "davicandidopucminas@gmail.com"
15      },
16     },
17     servers: [
18       {
19         url: "http://localhost:3000/v1",
20         description: "Ambiente de desenvolvimento"
21       },
22       {
23       },
24     ],
25   },
26   apis: ["./routes/*.js"]
27 }
28 const specs = swaggerJSDoc(option);
29
30 app.use("/api-docs", swaggerUi.serve, swaggerUi.setup(specs));
```

Importação e configurações

Visite <http://localhost:3000/api-docs> e veja algo próximo a isso:



Swagger with Express 1.0.0 OAS 3.0

Essa api tem como objetivo demonstra o uso do swagger com express

[Terms of service](#)

[Contact Davi Cândido](#)

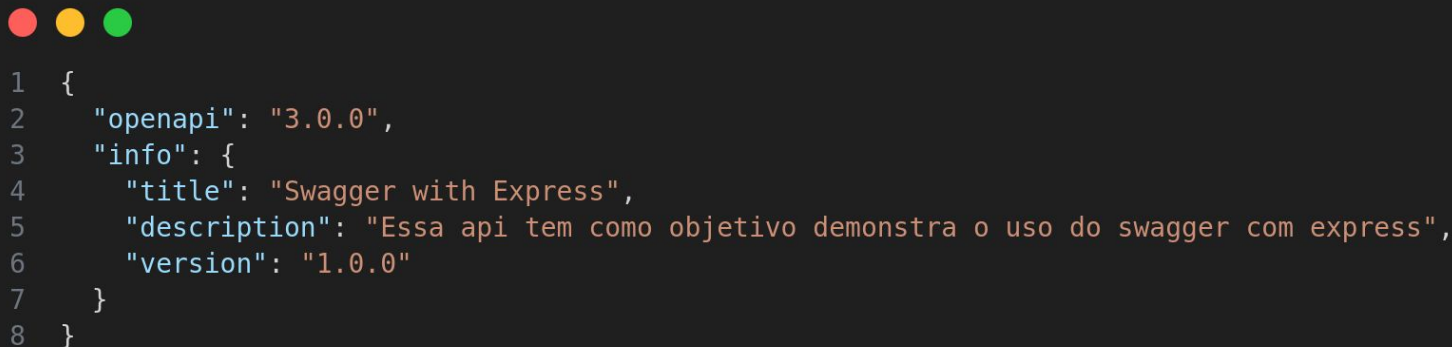
MIT

Servers

[http://localhost:3000/v1 - Ambiente de desenvolvimento](#)

Estrutura básica do Swagger

Voltando ao que foi feito em option fizemos a definição do cabeçalho de nossa documentação passando a versão da especificação da openapi utilizada e um série de informações (metadados), como título, descrição e a versão do documento



```
1  {
2    "openapi": "3.0.0",
3    "info": {
4      "title": "Swagger with Express",
5      "description": "Essa api tem como objetivo demonstra o uso do swagger com express",
6      "version": "1.0.0"
7    }
8  }
```

Melhorando o cabeçalho

Deixamos esse cabeçalho um pouco mais completo, adicionando uma rota de acesso aos termos de uso de nossa api que poderá ser disponibilizada através de uma página estática fornecida pelo servidor ou uma rota a parte, também colocamos um contato de referência

```
1  "openapi": "3.0.0",
2    "info": {
3      "title": "Swagger with Express",
4      "description": "Essa api tem como objetivo demonstra o uso do swagger com express",
5      "version": "1.0.0",
6      "termsOfService": "http://localhost:3000/terms/",
7      "contact": {
8        "name": "Davi Cândido",
9        "email": "davicandidopucminas@gmail.com"
10     }
11  },
```

Definindo servidores

Em seguida informamos em quais urls nossa Api foi disponibilizada, separando entre ambiente de desenvolvimento e produção

```
1  "openapi": "3.0.0",
2  "info": {
3    ...
4  },
5  "servers": [{
6    "url": "http://localhost:3000/v1",
7    "description": "Ambiente de desenvolvimento"
8  },
9  {
10   "url": "www.crudJourney.com/v2",
11   "description": "Ambiente de produção"
12 }
13 ]
```

Swagger with Express 1.0.0 OAS 3.0

Essa api tem como objetivo demonstra o uso do swagger com express

[Terms of service](#)

[Contact Davi Cândido](#)

MIT

Servers

http://localhost:3000/v1 - Ambiente de desenvolvimento



Mapeando rotas da API - Rota GET `/posts`

Agora iremos fazer um mapeamento de todas as rotas presente ou que futuramente estarão presentes em nossa api, vamos iniciar mapeando a nossa primeira rota de get:

```
1 /**
2  * @openapi
3  * /posts:
4  *   get:
5  *     summary: Retorna todos os posts
6  *     description: Essa rota será responsável por retornar todos os posts
7  *     tags: [Posts]
8  *     security:
9  *       - bearerAuth: []
10 *     responses:
11 *       200:
12 *         description: Retorna todos os posts
13 *       404:
14 *         description: Nenhum post foi encontrado
15 */
16 router.get("/", crudController.index);
```



GET /posts Retorna todos os posts



Mapeando rotas da API - Rota GET `/posts`

Podemos também colocar um conteúdo de exemplo que será retornado ao obtermos o status de resposta 200:



Responses		
Code	Description	Links
200	Retorna todos os posts	No links
<div>Media type</div> <div>application/json</div> <div>Controls Accept header.</div> <div>Example Value Schema</div> <pre>{ "id": 1, "title": "Post 1", "content": "Conteúdo do post 1" }, { "id": 2, "title": "Post 2", "content": "Conteúdo do post 2" }, { "id": 3, "title": "Post 3", "content": "Conteúdo do post 3" } </pre>		
404	Nenhum post foi encontrado	No links

```
1 /**
2  * @openapi
3  * /posts:
4  *   get:
5  *     summary: Retorna todos os posts
6  *     description: Essa rota retorna todos os posts disponíveis
7  *     tags: [Posts]
8  *     security:
9  *       - bearerAuth: []
10 *     responses:
11 *       200:
12 *         description: Lista de posts
13 *         content:
14 *           application/json:
15 *             schema:
16 *               type: array
17 *               items:
18 *                 $ref: '#/components/schemas/Post'
19 *             example:
20 *               - id: 1
21 *                 title: Post 1
22 *                 content: Conteúdo do post 1
23 *               - id: 2
24 *                 title: Post 2
25 *                 content: Conteúdo do post 2
26 *       404:
27 *         description: Nenhum post foi encontrado
28 *         content:
29 *           application/json:
30 *             schema:
31 *               type: object
32 *             example:
33 *               message: Nenhum post foi encontrado
34 */
35 router.get("/", crudController.index);
```


Mapeando rotas da API - `POST`/posts` com`requestBody``

Para o método post termos um tratamento um pouco diferente, teremos uma propriedade chamada `requestBody` que será responsável por dizer o tipo de dado requerido, que no nosso caso é o `application/json`, que por sua vez conterá o schema, ou seja o esquema referente ao formato como esses dados devem ser enviados, com seus atributos e tipos, veja que o esquema é feito através de um componente que é referenciado através do `$ref`, o esquema em si foi definido em "componentes":

```
1 /**
2  * @openapi
3  * /posts:
4  *   post:
5  *     summary: Cria um post
6  *     description: Essa rota cria um post
7  *     tags: [Posts]
8  *     security:
9  *       - bearerAuth: []
10 *     requestBody:
11 *       required: true
12 *       content:
13 *         application/json:
14 *           schema:
15 *             $ref: '#/components/schemas/Post'
16 *           examples:
17 *             post:
18 *               value:
19 *                 title: Post 1
20 *                 content: Conteúdo do post 1
21 *       responses:
22 *         201:
23 *           description: Post criado com sucesso
24 *           content:
25 *             application/json:
26 *               schema:
27 *                 type: object
28 *                 example:
29 *                   message: Post created successfully
30 *                   post:
31 *                     id: 1
32 *                     title: Post 1
33 *                     content: Conteúdo do post 1
34 *         400:
35 *           description: Dados incorretos ou incompletos
36 *           content:
37 *             application/json:
38 *               schema:
39 *                 type: object
40 *                 example:
41 *                   message: Title and content are required
42 */
43 router.post("/", crudController.save);
```

Posts

POST `/posts` Cria um post



Mapeando rotas da API - Referencia a componente de schema

```
1 /**
2  * @openapi
3  * /posts:
4  *   post:
5  *     summary: Cria um post
6  *     description: Essa rota cria um post
7  *     tags: [Posts]
8  *     security:
9  *       - bearerAuth: []
10 *     requestBody:
11 *       required: true
12 *       content:
13 *         application/json:
14 *           schema:
15 *             $ref: '#/components/schemas/Post'
16 *           examples:
17 *             post:
18 *               value:
19 *                 title: Post 1
20 *                 content: Conteúdo do post 1
21 *       responses:
22 *         201:
23 *           description: Post criado com sucesso
24 *           content:
25 *             application/json:
26 *               schema:
27 *                 type: object
28 *               example:
29 *                 message: Post created successfully
30 *                 post:
31 *                   id: 1
32 *                   title: Post 1
33 *                   content: Conteúdo do post 1
34 *             400:
35 *               description: Dados incorretos ou incompletos
36 *               content:
37 *                 application/json:
38 *                   schema:
39 *                     type: object
40 *                   example:
41 *                     message: Title and content are required
42 */
43 router.post("/", crudController.save);
```

```
1 * components:
2 *   schemas:
3 *     Post:
4 *       type: object
5 *       required:
6 *         - title
7 *         - content
8 *       properties:
9 *         id:
10 *           type: integer
11 *         title:
12 *           type: string
13 *         content:
14 *           type: string
15 */
```

Mapeando rotas da API - Autenticação com JWT

Extra: Caso a api use autenticações de segurança como JWT, também se é preciso informar em nossa documentação, para isso criamos um campo de securitySchemes no mesmo nível do schema em componentes



```
1
2 /**
3  * @openapi
4  * components:
5  *   schemas:
6  *     Post:
7  *       [...]
8  *     securitySchemes:
9  *       bearerAuth:
10 *         type: http
11 *         scheme: bearer
12 *         bearerFormat: JWT
13 */
```

Entenda melhor a estrutura do schemas de autenticação:

Campo	Significado
"type": "http"	Diz que o tipo de segurança é baseado em HTTP.
"scheme": "bearer"	Indica que a autenticação é via o esquema Bearer Token. Ex: Authorization: Bearer
"bearerFormat": "JWT"	Apenas uma dica para ferramentas como Swagger UI saberem que o token é um JWT. Não afeta a lógica da API

Mapeando rotas da API - Proteção de rotas

Agora nas rotas protegidas adicione uma tag de security, no mesmo nível do summary, description e tags, dessa forma:

```
1  /**
2   * @openapi
3   * /posts:
4   *   get:
5   *     summary: Retorna todos os posts
6   *     description: Essa rota será responsável por retornar todos os posts
7   *     tags: [Posts]
8   *     security:
9   *       - bearerAuth: []
10  *     responses:
11  *       [...]
12  */
13  router.get("/", crudController.index);
```

Acesse novamente nossa documentação em <http://localhost:3000/api-docs/> e veja que temos agora um cadeado no canto direito de nossa rota get, informando que esta rota é uma rota autenticada:

GET /posts Retorna todos os posts



Parâmetros em rotas - GET `/posts/{id}`

Para rotas que exigem a passagem de parâmetros (params ou query) criamos uma rota no mesmo nível da anterior rota defina como /posts, no entanto agora definimos em parameters o tipo de parâmetro exigido, informando o nome do parâmetro, se sera através de query, ou params (path), se seu envio é obrigatório (required) e seu esquema de tipo, veja o exemplo:

```
1 /**
2  * @openapi
3  * /posts/{id}:
4  *   get:
5  *     summary: Retorna um post
6  *     description: Essa rota será responsável por retornar um post pelo id
7  *     tags: [Posts]
8  *     parameters:
9  *       - in: path
10 *         name: id
11 *         required: true
12 *         schema:
13 *           type: number
14 *     security:
15 *       - bearerAuth: []
16 *     responses:
17 *       200:
18 *         description: Post encontrado
19 *       404:
20 *         description: Nenhum post foi encontrado
21 */
22 router.get("/:id", crudController.show);
```

Parâmetros em rotas - GET `/posts/{id}`

Veja que agora na rota apresentada em nossa documentação será criado um campo de teste onde poderemos adicionar um id de busca, tornando a busca especifica pelo {id} passado:

GET /posts/{id} Retorna um post

Essa rota será responsável por retornar um post pertencente ao id passado na rota

Parameters Cancel

Name	Description
id * required number (path)	<input type="text" value="2"/>

Execute Clear

Responses

Curl

```
curl -X 'GET' \
'http://localhost:3000/v1/posts/2' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:3000/v1/posts/2
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 2, "title": "Post 1", "content": "Conteúdo do post 1" }</pre> Download

Parâmetros em rotas

PUT `/posts/{id}` – Atualização completa

De forma semelhante agora podemos criar a documentação de nossas rotas de PUT (atualização total) e DELETE, para a rota de PUT segue o exemplo abaixo, veremos que nada mais do que se trata de uma adição de um requestBody para informar os dados que serão enviados para atualização do post pertencente ao id informado em parameters, veja:

```
1  /**
2   * @openapi
3   * /posts/{id}:
4   *   put:
5   *     summary: Atualiza um post
6   *     description: Atualiza completamente um post pelo id
7   *     tags: [Posts]
8   *     parameters:
9   *       - in: path
10      *         name: id
11      *         required: true
12      *         schema:
13      *           type: number
14      *     security:
15      *       - bearerAuth: []
16      *     requestBody:
17      *       required: true
18      *       content:
19      *         application/json:
20      *           schema:
21      *             $ref: '#/components/schemas/Post'
22      *           examples:
23      *             post:
24      *               value:
25      *                 title: Post Atualizado
26      *                 content: Conteudo atualizado
27      *     responses:
28      *       200:
29      *         description: Post atualizado com sucesso
30      *       404:
31      *         description: Post não encontrado
32      */
33  router.put("/:id", crudController.update);
```

Parâmetros em rotas

DELETE `/posts/{id}`

E para a rota de delete:

```
1 /**
2  * @openapi
3  * /posts/{id}:
4  *   delete:
5  *     summary: Deleta um post
6  *     description: Deleta um post pelo id
7  *     tags: [Posts]
8  *     parameters:
9  *       - in: path
10 *         name: id
11 *         required: true
12 *         schema:
13 *           type: number
14 *     security:
15 *       - bearerAuth: []
16 *     responses:
17 *       200:
18 *         description: Post deletado com sucesso
19 *       404:
20 *         description: Post não encontrado
21 */
22 router.delete("/:id", crudController.destroy);
```


Documentação JSON - Rota `/docs-swagger``

E para finalizar podemos documentar uma rota que será responsável por fornecer nossa documentação em json. O fornecimento será feito através do envio do próprio specs anteriormente definido:

```
1  /**
2  * @openapi
3  * /docs-swagger:
4  *   get:
5  *     summary: Retorna a documentação em JSON da API
6  *     description: Essa rota retorna a especificação Swagger gerada para a API
7  *     tags: [Documentação]
8  *     responses:
9  *       200:
10 *         description: Documentação da API
11 *         content:
12 *           application/json: {}
13 *       404:
14 *         description: Documentação não encontrada
15 *         content:
16 *           application/json:
17 *             schema:
18 *               type: object
19 *               example:
20 *                 message: Documentação não encontrada
21 */
22 router.get("/docs-swagger", (req, res) => {
23   res.json(specs);
24 });
```

Visualização final

Veja por fim como ficou nossa documentação de nossa api:

Swagger with Express 1.0.0 OAS 3.0

Essa api tem como objetivo demonstra o uso do swagger com express

[Terms of service](#)
[Contact Davi Cândido](#)
MIT

Servers

http://localhost:3000/v1 - Ambiente de desenvolvimento

Authorize

Posts

POST

/posts

Cria um post

GET

/posts

Retorna todos os posts

GET

/posts/{id}

Retorna um post

PUT

/posts/{id}

Atualiza completamente um post de acordo com o id passado na rota

DELETE

/posts/{id}

Deleta um post de acordo com o id passado na rota

Documentação

GET

/docs-swagger

Retorna a documentação em json da api feita pelo swagger

Considerações Finais

Lembre-se que isso não é tudo. Muito mais pode ser explorado.

Agora sua documentação Swagger está completa, com suporte para autenticação, rotas REST, schemas e testes interativos.

Use <https://editor.swagger.io> para validar seu swagger.json!

Acesse os código no GitHub:

<https://github.com/DaviKandido/Documentacao-Swagger-Express.git>

Esse tutorial foi escrito por Davi Cândido – PUC Minas. Compartilhe com colegas desenvolvedores!