WebTech network

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?

O 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 analizar 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.

Davi Cândido de Almeida

Estrutura Básica OpenAPI (YAML)

```yaml
openapi: 3.0.4
info:
  title: Sample API
  description: Optional multiline or single-line description in [CommonMark](http://commonmark.org/help/) or HTML.
  version: 0.1.9

servers:
  - url: http://api.example.com/v1
    description: Servidor principal (produção)
  - url: http://staging-api.example.com
    description: Servidor de homologação interno para testes

paths:
  /users:
    get:
      summary: Retorna uma lista de usuários.
      description: Descrição estendida (opcional).
      responses:
        "200":
          description: Um array JSON com nomes de usuários
          content:
            application/json:
              schema:
                type: array
                items:
                  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.

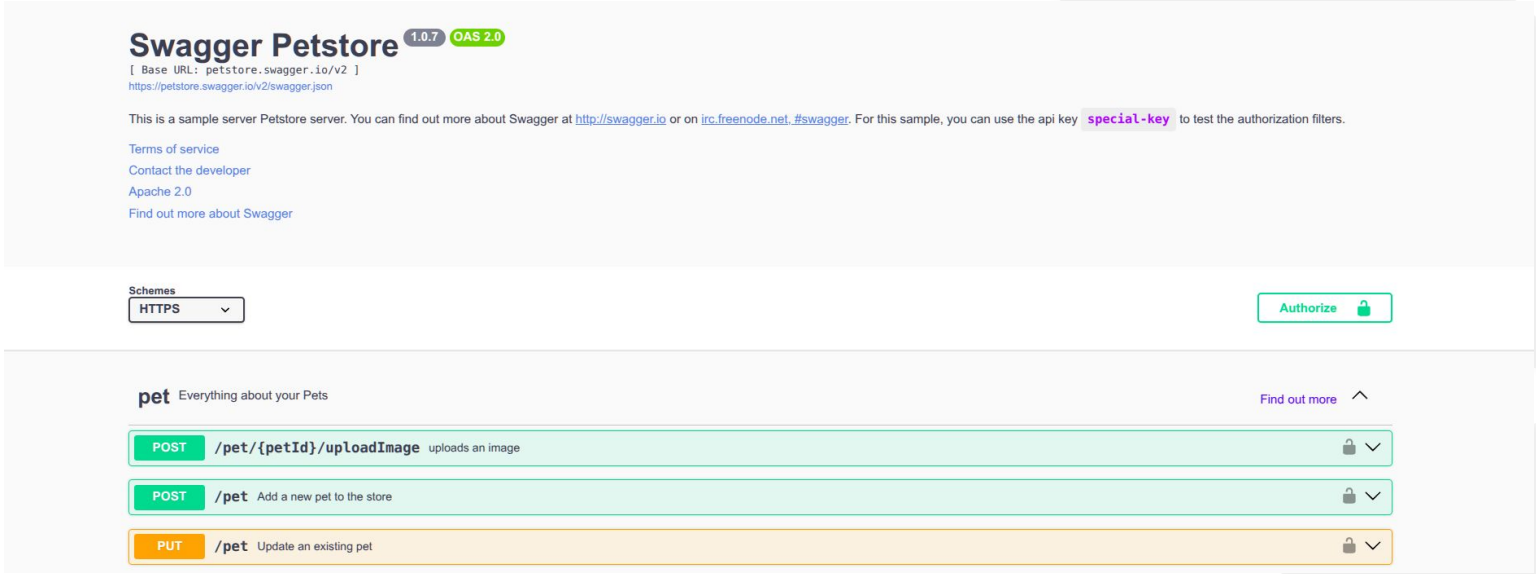
WebTech network

Pratica: Implementação Swagger

Documentando uma Api de crud em express com swagger-ui

O que queremos?

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



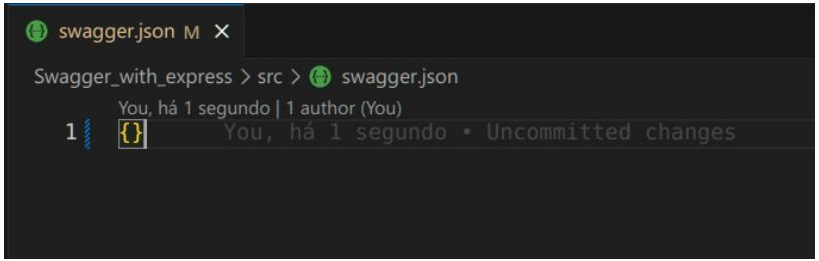
Você também pode acessar o JSON usado por essa documentação em: https://petstore.swagger.io/v2/swagger.json

Instalação de dependencias

1. Baixe a biblioteca Swagger UI Express:

```
npm install swagger-ui-express
```

2. Crie na raiz do projeto um arquivo chamado swagger.json ou swagger.yaml e coloque uma chave vazia em seu conteúdo:



Davi Cândido de Almeida

Importação e configurações

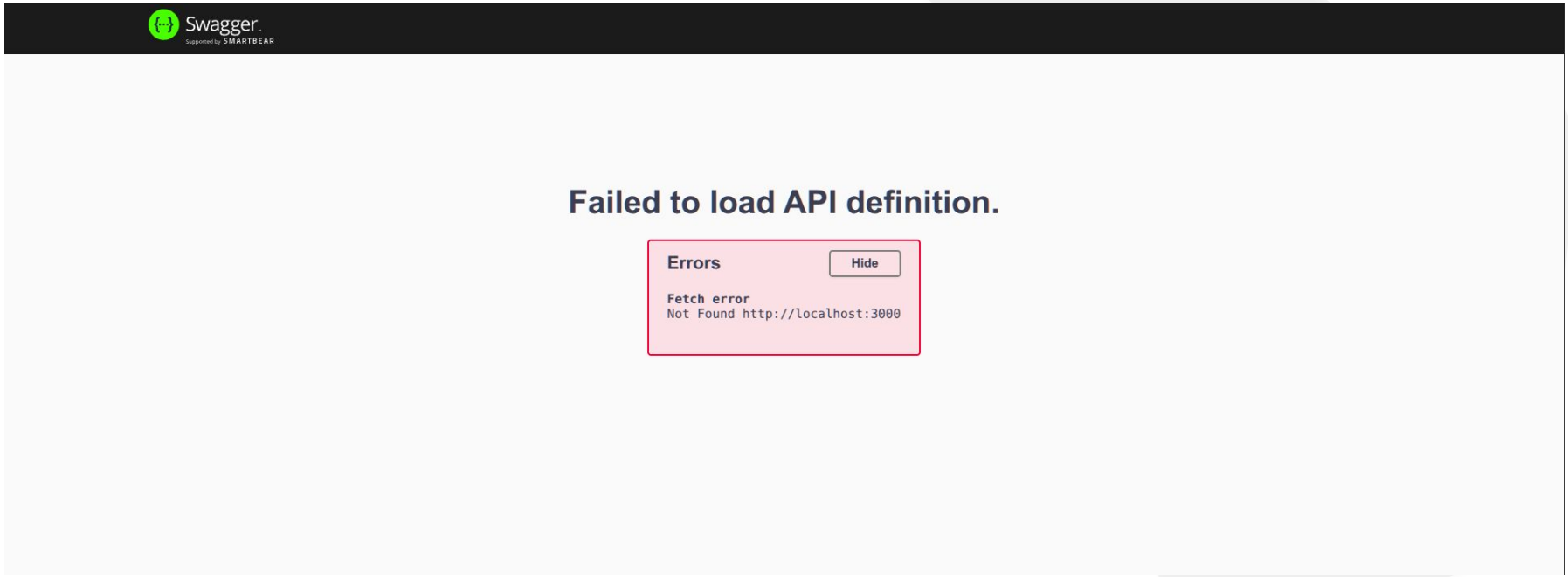
```
1  const express = require("express");
2  const app = express();
3
4  const swaggerUi = require("swagger-ui-express"); //commonjs
5  // ou
6  //import swaggerUi from "swagger-ui-express"; //ES6
```

Adicione o middleware do swagger

```
1  / Configurando o swagger
2  app.use("api-docs", swaggerUi.serve, swaggerUi.setup(require("./swagger.json")));
3
4  // Ou para melhor legibilidade:
5  // const swaggerDocs = require("./swagger.json");
6  // app.use("api-docs", swaggerUi.serve, swaggerUi.setup(swaggerDocs));
```

Importação e configurações

Visite http://localhost:3000/api-docs e vera algo proximo a isso:



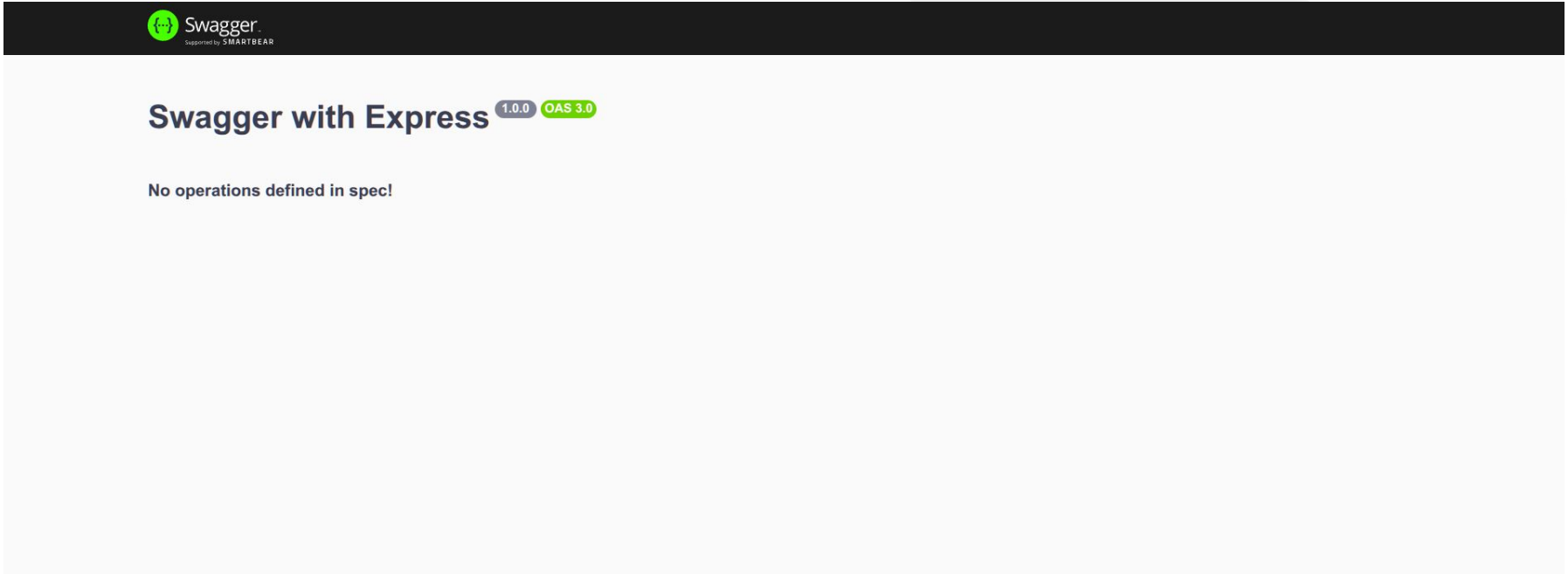
Estrutura básica do Swagger

Como não temos nenhuma definição até o momento a documentação ainda não estará acessível, adicione agora o cabeçalho de nossa documentação em swagger.json e acesse novamente http://localhost:3000/api-docs/:

```
{
  "openapi": "3.0.0",
  "info": {
    "title": "Swagger with Express",
    "description": "Essa api tem como objetivo demonstra o uso do swagger com express",
    "version": "1.0.0"
  }
}
```

Estrutura básica do Swagger

Vera algo proximo a isso:



Melhorando o cabeçalho

Agora vamos deixar essa cabeçalho um pouco mais completo, iremos adicionar 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 colocaremos um contato de referência

```
"openapi": "3.0.0",
   "info": {
       "title": "Swagger with Express",
       "description": "Essa api tem como objetivo demonstra o uso do swagger com express",
       "version": "1.0.0",
       "termsOfService": "http://localhost:3000/terms/",
       "contact": {
           "name": "Davi Cândido",
           "email": "davicandidopucminas@gmail.com"
       }
   },
```

Definindo servidores

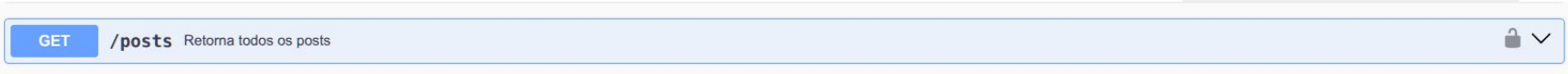
Em seguida vamos informar quais urls nossa Api será disponibilizada, separando entre ambiente de desenvolvimento e produção

```
"openapi": "3.0.0",
  "info": {
    ...
  },
  "servers": [{
      "url": "http://localhost:3000/v1",
      "description": "Ambiente de desenvolvimento"
   },
   {
      "url": "www.crudJourney.com/v2",
      "description": "Ambiente de produção"
   }
  ]
```

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:

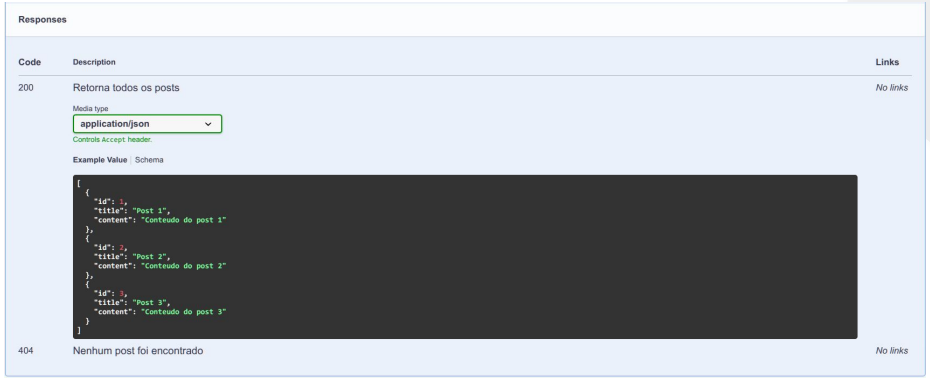
```
"openapi": "3.0.0",
    "info": {
      ...
    },
    "servers": ...,
    "paths": {
        "/posts":{
            "summary": "Retorna todos os posts",
            "description": "Essa rota será responsável por retorna todos os posts",
            "get": {
                "tags": ["Posts"],
                "responses": {
                    "200": {
                        "description": "Retorna todos os posts"
                    },
                    "404": {
                        "description": "Nenhum post foi encontrado"
                    }
                }
            }
        }
    }
```



Davi Cândido de Almeida

Mapeando rotas da API - Rota GET `/posts`

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



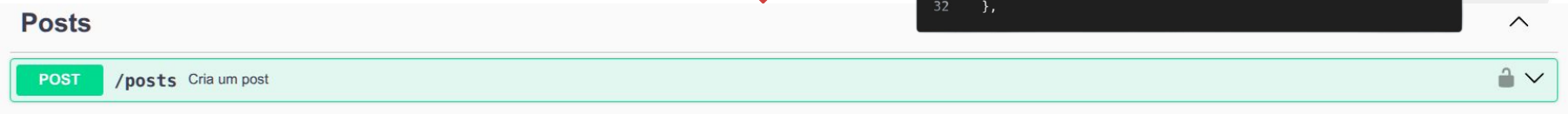

```
"paths": {
  "/posts":{
      "summary": "Retorna todos os posts",
      "description": "Essa rota será responsável por retorna todos os posts",
      "get": {
          "tags": ["Posts"],
          "responses": {
              "200": {
                  "description": "Retorna todos os posts",
                  "content": {
                    "application/json": {
                      "schema": {
                        "type": "array",
                        "example": [
                          {
                            "id": 1,
                            "title": "Post 1",
                            "content": "Conteudo do post 1"
                          },
                          {
                            "id": 2,
                            "title": "Post 2",
                            "content": "Conteudo do post 2"
                          },
                          {
                            "id": 3,
                            "title": "Post 3",
                            "content": "Conteudo do post 3"
                          }
                        ]
                      }
                    }
                  }
                }
              },
              "404": {
                  "description": "Nenhum post foi encontrado"
              }
          }
      }
  }
```


Davi Cândido de Almeida

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

Para o método post teremos 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":

```
"post": {
        "summary": "Cria um post",
        "description": "Essa rota cria um post",
        "tags": ["Posts"],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Post"
              },
              "examples": {
                "post": {
                  "value": {
                    "title": "Post 1",
                    "content": "Conteudo do post 1"
                  }
                }
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Post criado com sucesso"
          },
          "404": {
            "description": "Dados incorretos ou incompletos"
          }
        }
      }
    }
  },
```



Mapeando rotas da API - Referencia a componente de schema

```
"post": {
        "summary": "Cria um post",
        "description": "Essa rota cria um post",
        "tags": ["Posts"],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Post"
              },
              "examples": {
                "post": {
                  "value": {
                    "title": "Post 1",
                    "content": "Conteudo do post 1"
                  }
                }
              }
            }
          }
        },
        "responses": {
          "201": {
            "description": "Post criado com sucesso"
          },
          "404": {
            "description": "Dados incorretos ou incompletos"
          }
        }
      }
    }
  },
```

```
"components": {
    "schemas": {
      "Post": {
        "type": "object",
        "required": ["title", "content"],
        "properties": {
          "id": {
            "type": "number"
          },
          "title": {
            "type": "string"
          },
          "content": {
            "type": "string"
          }
        }
      }
    }
  }
```


Davi Cândido de Almeida

Mapeando rotas da API - Autenticação com JWT

Extra: Caso a api use autentificações 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

```
"components": {
    "schemas": {
      ...
    },
    "securitySchemes": {
      "bearerAuth": {
        "type": "http",
        "scheme": "bearer",
        "bearerFormat": "JWT"
      }
    }
  }
```

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

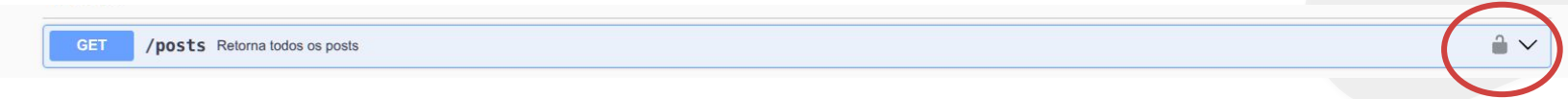
Davi Cândido de Almeida

Mapeando rotas da API - Proteção de rotas

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

```
"paths": {
    "/posts": {
      "get": {
        "summary": "Retorna todos os posts",
        "description": "Essa rota será responsável por retorna todos os posts",
        "tags": ["Posts"],
        "security": [{
          "bearerAuth": []
        }],
        ...
```

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 um rota autenticada:



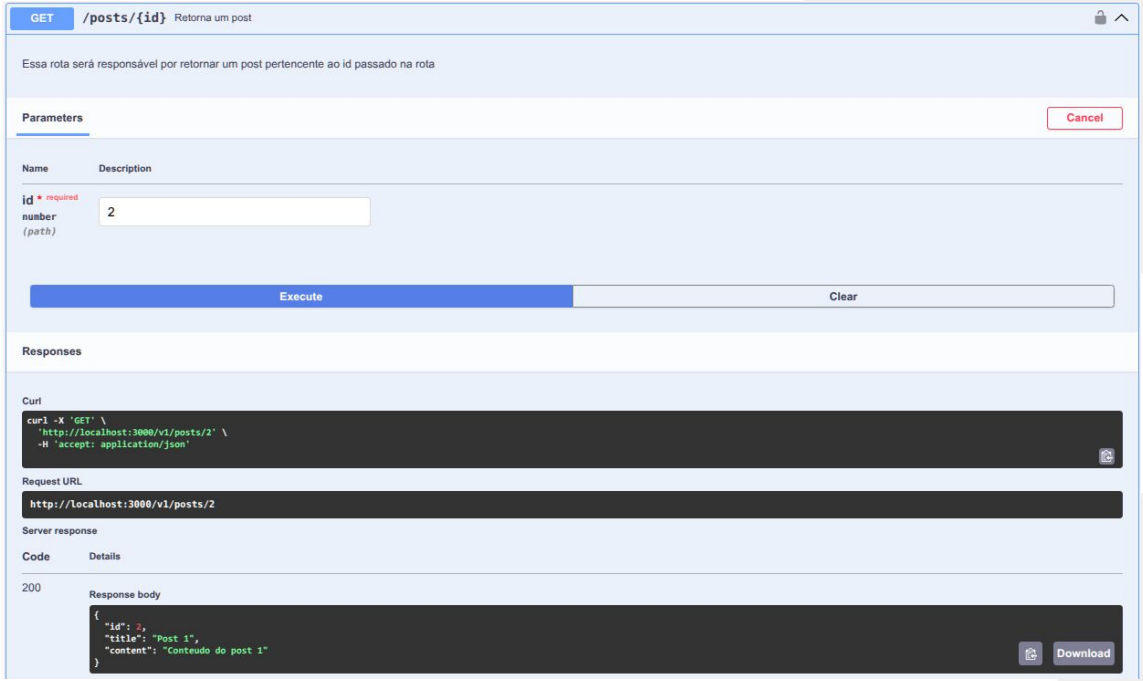
Parâmetros em rotas - GET `/posts/{id}`

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:

```
"paths": {
    "/posts": {
      "get": {
        ...
      },
      "post": {
        ...
      }
    },
    "/posts/{id}": {
      "get": {
        "summary": "Retorna um post",
        "description": "Essa rota será responsável por retornar um post pertencente ao id passado na rota",
        "tags": ["Posts"],
        "parameters": [
          {
            "name": "id",
            "in": "path", //ou query
            "required": true,
            "schema": {
              "type": "number"
            }
          }
        ],
        "security": [{
            "bearerAuth": []
        }],
        "responses": {
          "200": {
            "description": "Retorna o post pertencente ao {id}",
            "content": {
              "application/json": {
                "schema": {
                  "type": "object",
                  "example": {
                      "id": 1,
                      "title": "Post 1",
                      "content": "Conteudo do post 1"
                  }
                }
              }
            }
          },
          "404": {
            "description": "Nenhum post foi encontrado"
          }
        }
      }
    }
```

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:



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:

```
"/posts/{id}": {
      "get": {
        ...
      },
      "put": {
        "summary": "Atualiza completamente um post de acordo com o id passado na rota",
        "description": "Essa rota será responsável por atualizar completamente um post de acordo com o id passado na rota",
        "tags": ["Posts"],
        "parameters": [{
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "number"
            }
        }],
        "security": [{
            "bearerAuth": []
        }],
        "requestBody": {
          "content": {
            "application/json": {
              "schema": {
                "$ref": "#/components/schemas/Post"
              },
              "examples": {
                "post": {
                  "value": {
                    "title": "Post 1 atualizado",
                    "content": "Conteudo do post 1 atualizado"
                  }
                }
              }
            }
          }
        },
        "responses": {
          "200": {
            "description": "Retorna o post atualizado pertencente ao {id}",
            "content": {
              "application/json": {
                "schema": {
                  "type": "object",
                  "example": {
                    "message": "Post atualizado com sucesso",
                    "post": {
                      "id": 1,
                      "title": "Post 1 atualizado",
                      "content": "Conteudo do post 1 atualizado"
                    }
                  }
                }
              }
            }
          },
          "404": {
            "description": "Nenhum post foi encontrado"
          }
        }
      }
    }
```

Parâmetros em rotas

DELETE `/posts/{id}`

E para a rota de delete:

```
"/posts/{id}": {
      "get": {
        ...
      },
      "put": {
        ...
      },
      "delete": {
        "summary": "Deleta um post de acordo com o id passado na rota",
        "description": "Essa rota será responsável por deletar um post de acordo com o id passado na rota",
        "tags": ["Posts"],
        "parameters": [{
            "name": "id",
            "in": "path",
            "required": true,
            "schema": {
              "type": "number"
            }
        }],
        "security": [{
            "bearerAuth": []
        }],
        "responses": {
          "200": {
            "description": "Retorna se o post pertencente ao {id} foi deletado",
            "content": {
              "application/json": {
                "schema": {
                  "type": "object",
                  "example": {
                    "message": "Post deletado com sucesso"
                  }
                }
              }
            }
          },
          "404": {
            "description": "Nenhum post foi encontrado"
          }
        }
      }
    },
```


Davi Cândido de Almeida

Documentação JSON - Rota `/docs-swagger`

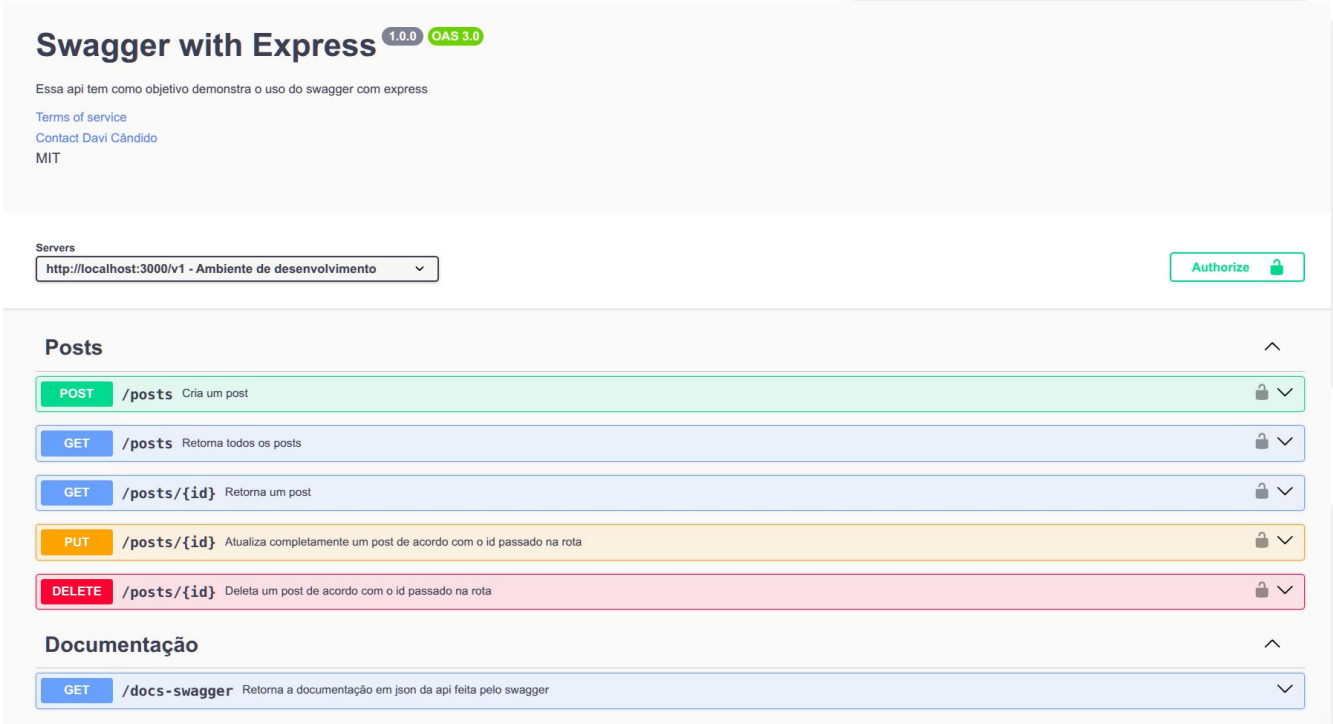
E para finalizar podemos documentar uma rota que será responsável por fornecer nossa documentação em json, fornecimento feito através do envio de um arquivo estático, neste caso nosso swagger.json:

```
1
2  // Rota para servir o arquivo swagger.json
3  app.get("/v1/docs-swagger", (req, res) => {
4    res.sendFile(path.join(__dirname, "swagger.json"));
5  });
6
```

```
1  "paths": {
2    "/posts": {
3      ...
4    },
5    "/posts/{id}": {
6      ...
7    },
8    "/docs-swagger": {
9      "get": {
10       "summary": "Retorna a documentação em json da api feita pelo swagger",
11       "description": "Essa rota retorna a documentação em json da api feita pelo swagger",
12       "tags": ["Documentação"],
13       "responses": {
14         "200": {
15           "description": "Retorna a documentação",
16           "content": {
17             "text/json": {}
18           }
19         }
20       }
21     }
22   }
23 },
```

Visualização final

Veja por fim como ficou nossa documentação de nossa api:



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!