

Linguagem de Programação Visual

RICARDO HENDGES



Documentação

A documentação permite que desenvolvedores tenham uma visão mais clara do seu produto e de como o seu código pode funcionar em conjunto com o software deles.



Por que e como documentar uma API?

A documentação de uma API é uma visualização mais técnica de conteúdo, contendo instruções sobre como usar e acessar efetivamente a solução.

Identificamos principalmente em uma documentação:

- Rotas (ex: /alunos)
- Parametros (ex: /alunos/:id)
- Retornos (ex: {id:1, nome: "ricardo"})

Dicas

1. Documente TUDO!
2. Disponibilize EXEMPLOS!
3. Vá direto ao ponto!
4. Cuidado com as excessões!
5. TESTE sua documentação!



SWAGGER

Swagger permite que você descreva a estrutura de APIs de uma forma facilitada. Swagger consegue ler a estrutura da sua API e gerar automaticamente uma documentação ou ler a documentação e gerar uma API.

Esta documentação descreve algumas informações:

- Quais são todas as operações que sua API suporta.
- Quais são os parâmetros de sua API e o que ela retorna.
- Se sua API precisa de alguma autorização.

Você pode escrever uma especificação Swagger para sua API manualmente, ou ter ela gerada automaticamente por anotações no seu código fonte.

Estrutura Básica

O Swagger pode ser escrito em JSON ou YAML

```
1. swagger: "2.0"
2. info:
3.   title: Sample API
4.   description: API description in Markdown.
5.   version: 1.0.0
6.
7. host: api.example.com
8. basePath: /v1
9. schemes:
10.  - https
11.
12. paths:
13.   /users:
14.     get:
15.       summary: Returns a list of users.
16.       description: Optional extended description in Markdown.
17.       produces:
18.         - application/json
19.       responses:
20.         200:
21.           description: OK
```

Entendendo a estrutura

Versão do Swagger - 2.0

Algumas informações sobre a identificação da API

```
"swagger": "2.0",  
"info": {  
  "version": "1.0.0",  
  "title": "Alunos Horus",  
  "description": "Documentação da API de alunos da Horus"  
},
```


Entendendo a estrutura

Dados sobre a URL a ser chamada pelo swagger.

```
"host": "localhost:3000",  
"basePath": "/v1",  
"schemes": [  
  "http"  
],
```

Dados acima indicam que o caminho base para cada chamada seria:
`http://localhost:3000/v1`

Entendendo a estrutura

Consumes e produces indicam que tipo de dado sera utilizado e gerado pela API

```
"consumes": [  
  "application/json"  
],  
"produces": [  
  "application/json"  
],
```

Entendendo a estrutura

A sessão Paths define um endpoint individual na sua API e qual o método HTTP suportado por este endpoint. Por exemplo GET /alunos

```
"paths": {  
  "/aluno": {  
    "get": {  
      "summary": "Obter dados de um Aluno da Horus!",  
      "description": "",  
      "responses": {  
        "201": {  
          "description": "Sucesso!"  
        }  
      }  
    }  
  }  
}
```

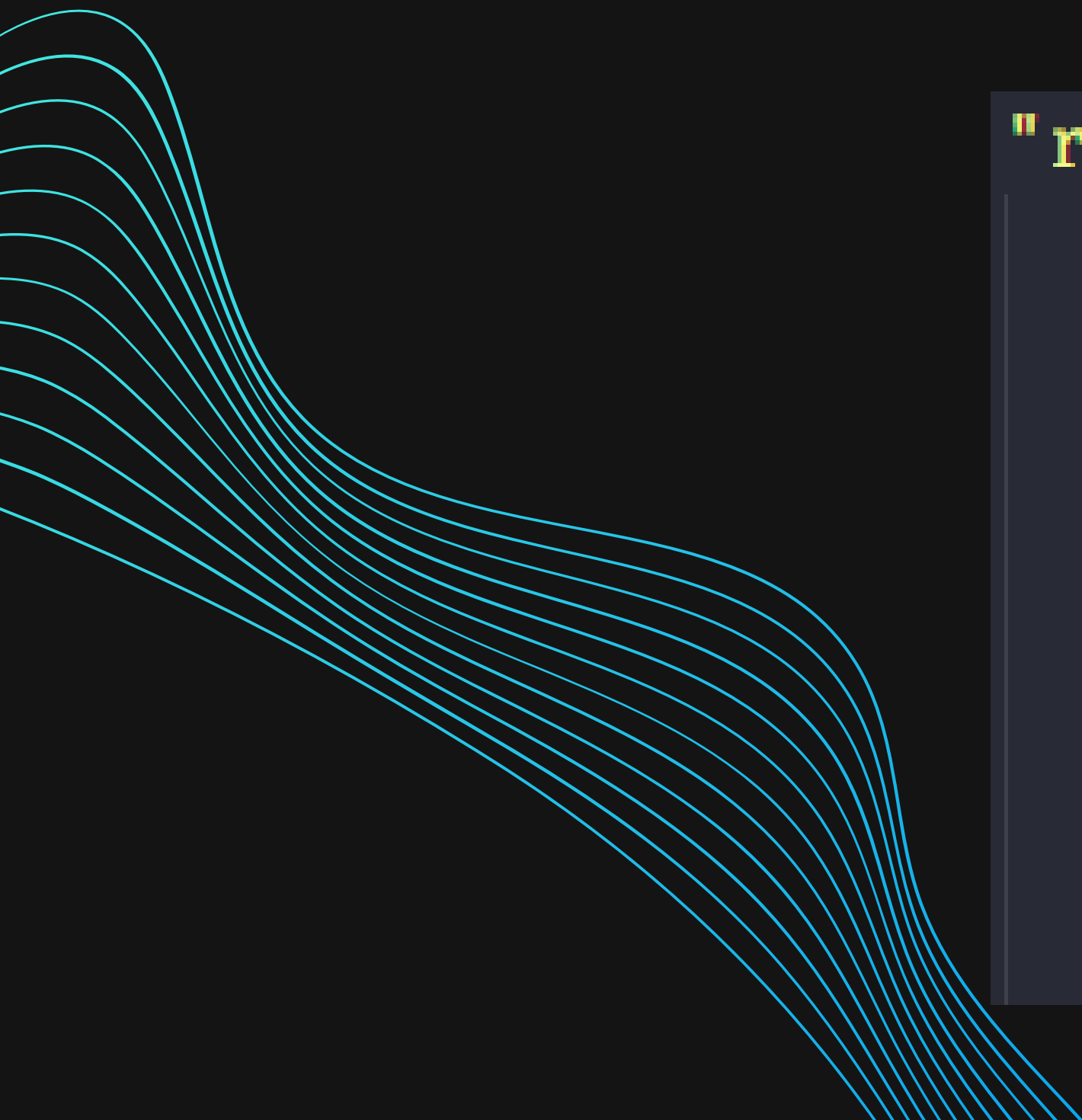
Entendendo a estrutura

As operações podem ter parâmetros que serão informados via URL Path (/aluno/{id}), Query String (/aluno?id=123), Headers (X-customHeader: Value) e Request Body. Você pode definir os tipos dos parâmetros, formato, obrigatoriedade e outros detalhes:

```
"/aluno/{id}": {  
  "delete": {  
    "summary": "Deleta um aluno",  
    "description": "",  
    "parameters": [  
      {  
        "name": "id",  
        "in": "path",  
        "required": true,  
        "type": "integer",  
        "description": "ID do aluno a ser removido.",  
        "value": 1
```

Entendendo a estrutura

Para cada operação você pode definir os status code possíveis, como 200 OK ou 404 Not Found, pode definir também o schema do Response body.



```
"responses": {  
  "204": {  
    "description": "Removido com Sucesso!"  
  },  
  "404": {  
    "description": "Aluno não encontrado!"  
  },  
  "500": {  
    "description": "Problema no servidor."  
  }  
}
```

`npm i swagger-ui-express`

`npm i swagger-autogen`

Criar pasta Views dentro de src.

Copie todo conteúdo do node-modules swagger-ui-dist para a pasta Views

Criar pasta Docs dentro de src.

Crie o arquivo swagger.js dentro da pasta src/services



Código swagger.js
dentro da pasta
src/services

```
const swaggerAutogen = require('swagger-autogen')('pt-BR');

const doc = {
  info: {
    version: "1.0.0",
    title: "API HORUS ALUNOS",
    description: "Documentação da API HORUS ALUNOS"
  },
  host: `localhost:3000`,
  basePath: "",
  schemes: ['http'],
  consumes: ['application/json'],
  produces: ['application/json'],
}

const outputFile = './src/docs/swagger.yaml';
const endpointsFiles = ['./src/routes/aluno.js'];

swaggerAutogen(outputFile, endpointsFiles, doc);
```


Buildando swagger.yaml

Adicione a linha do swaggerbuild dentro do package.json

```
"scripts": {  
  "swaggerbuild": "node ./src/services/swagger.js",  
  "dev": "nodemon ./src/index.js",  
  "prd": "node ./src/index.js"
```

execute: `npm run swaggerbuild`

index.js

Ajuste o arquivo para ter a chamada das rotas de documentação swagger.

```
const express = require('express')
const app = express()
app.use(express.json())
require('./services/swagger')

require('./routes')(app)
app.get('/', (req, res) => res.status(200).send('Hello World'))

app.use('/v1/docs', express.static('src/views'))
app.use('/docs/swagger.yaml', express.static('src/docs/swagger.yaml'))

app.listen(3000)
```

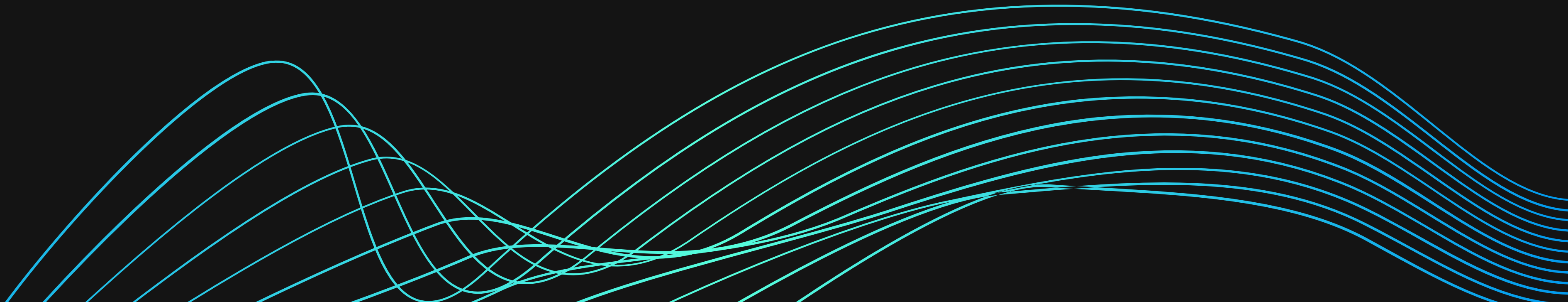
Swagger view

Ajuste o arquivo index.html dentro da pasta src/view na linha 42 conforme a imagem abaixo.

```
39 window.onload = function() {  
40     // Begin Swagger UI call region  
41     const ui = SwaggerUIBundle({  
42         url: "/docs/swagger.yaml",  
43         dom_id: '#swagger-ui',
```

Acesse

localhost:3000/v1/docs



Notations / Cabeçalho

`#swagger.tags` (ARRAY) para tirar a nossa rota do grupo 'default' na documentação e podermos agrupar escopos da API.

```
module.exports = (app) => {  
  app.get('/aluno', alunoController.getAluno  
  /**  
    #swagger.tags = ["aluno"]  
  */  
);
```

aluno

GET

/aluno

default

POST

/aluno

DELETE

/aluno/{id}

PUT

/aluno/{id}

PATCH

/aluno/{id}

Notations / Cabeçalho

#swagger.summary (STRING) para dar uma descrição breve da rota.

```
app.get('/aluno', alunoController.getAluno
/**
  #swagger.tags = ["aluno"]
  #swagger.summary = 'Consulta lista de alunos'
*/
```

aluno		
GET	/aluno	Consulta lista de alunos
default		
POST	/aluno	
DELETE	/aluno/{id}	
PUT	/aluno/{id}	
PATCH	/aluno/{id}	

Notations / Cabeçalho

#swagger.description
(STRING) para dar uma
descrição completa da rota.

aluno		
GET	/aluno	Consulta lista de alunos
Consulta lista de alunos, todos cadastrados		

```
app.get('/aluno', alunoController.getAluno
/**
  #swagger.tags = ["aluno"]
  #swagger.summary = 'Consulta lista de alunos'
  #swagger.description = 'Consulta lista de alunos, todos cadastrados'
*/
```

Notations / Parâmetros

#swagger.parameters (OBJECT) para indicar todos parâmetros que necessitamos.

```
#swagger.parameters['json'] = {  
  in: 'body',  
  description: 'Dados para inserir um novo aluno!',  
  type: 'json',  
  schema: {  
    id: 1,  
    nome: "Ricardo",  
    sobrenome: "Hendges",  
    periodo: 2,  
    observacao: "professor"  
  }  
}
```

exemplo com body

in: body, query, path

json
object
(body)

Dados para inserir um novo aluno!

Example Value | Model

```
{  
  "id": 1,  
  "nome": "Ricardo",  
  "sobrenome": "Hendges",  
  "periodo": 2,  
  "observacao": "professor"  
}
```


Notations / Parâmetros

#swagger.parameters (OBJECT) para indicar todos parâmetros que necessitamos.

```
app.put('/aluno/:id', alunoController.getAluno
/**
  #swagger.parameters['id'] = {
    description: 'Código do ALUNO!!!',
    in: 'path',
    name: 'id',
    type: 'integer',
    value: 2
  }
*/
```

exemplo com path

in: body, query, path

PUT /aluno/{id}	
Parameters	
Name	Description
id ★ required	Código do ALUNO!!!
integer (path)	<input type="text" value="2"/>

Notations / Parâmetros

#swagger.parameters (OBJECT) para indicar todos parâmetros que necessitamos.

```
app.get(['/aluno', alunoController.getAluno  
/**  
  #swagger.parameters['id'] = {  
    description: 'Código do aluno.',  
    in: 'query',  
    name: 'id',  
    type: 'integer',  
    value: 1  
  }  
}
```

exemplo com query

in: body, query, path

GET /aluno	
Parameters	
Name	Description
id	Código do aluno.
integer (query)	<input type="text" value="1"/>

Notations / Respostas

#swagger.responses (OBJECT) para descrever todos retornos conhecidos possíveis como sucessos, erros, validações, etc.

```
app.get('/aluno', alunoController.getAluno
/**
  #swagger.responses[200] = { description: 'Sucesso!'
    schema: {
      "total": 1,
      "alunos": [
        {
          "id": 1,
          "nome": "Ricardo",
          "sobrenome": "Hendges",
          "periodo": 5,
          "observacao": "Professor"
        }
      ]
    }
  }
```

200 - OK!

Responses

Code	Description
200	Sucesso!

Example Value | Model

```
{
  "total": 1,
  "alunos": [
    {
      "id": 1,
      "nome": "Ricardo",
      "sobrenome": "Hendges",
      "periodo": 5,
      "observacao": "Professor"
    }
  ]
}
```

Notations / Respostas

`#swagger.responses` (OBJECT) para descrever todos retornos conhecidos possíveis como sucessos, erros, validações, etc.

```
app.get('/aluno', alunoController.getAluno 400 - Bad Request!  
/**  
#swagger.responses[400] = { description: 'Bad request',  
  schema: {  
    mensagem: 'Campos faltando!',  
    detalhe: 'Os campos nome, periodo são obrigatórios.'  
  }  
}
```

400 Bad request

Example Value | Model

```
{  
  "mensagem": "Campos faltando!",  
  "detalhe": "Os campos nome, periodo são obrigatórios."  
}
```