Desenvolvimento Backend

Aula 14 - Documentando uma API

Material Didático do Instituto Metrópole Digital - IMD

Termo de uso

Apresentação

Nesta aula iremos abordar o assunto de documentação, afinal, é importante que programadores documentem seus códigos tanto para outros desenvolvedores, quanto para usuários da aplicação.

Objetivos

- Conhecer a importância da documentação
- Conhecer o padrão *Open API*
- Conhecer como documentar a API
- Conhecer como documentar rotas que precisam de autenticação

Importância da documentação e o que é Open API

Link do video da aula: https://youtu.be/1Xu04dwxV9w

Quando um programador desenvolve um código é importante lembrar que outros desenvolvedores e os usuários vão utilizá-lo. Por esse motivo, a documentação técnica é tão importante, pois ela auxiliará no entendimento do código e no funcionamento da aplicação.

Padrão *Open API*

Open API é uma iniciativa de diversas empresas em criar um padrão de documentação de API. Esse padrão facilita a comunicação de diversos serviços de empresas diferentes, facilitando o intercâmbio de informações.

Por ser *open source*, é possível visitar a especificação desse padrão no *GitHub* clicando aqui.

Por fim, para visualizar a API documentada, será utilizada a ferramenta *Swagger Petstore*.

Preparando o projeto

Link do video da aula: https://youtu.be/qgm7o2_Qj-Q

Antes de iniciar a documentação é preciso preparar o projeto. Então,

acompanhando a aula, faça as mudanças necessárias.

Instalando e configurando as bibliotecas

Para criar a documentação, será utilizado o *Swagger UI Express*. Para instalar, execute no terminal o comando abaixo:

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

Nessa documentação iremos usar o formato *yaml* e será necessário instalar a biblioteca *yamljs*, seguindo o comando abaixo:

```
npm install --save yamljs
```

Feitas as instalações, é preciso fazer o *setup* no arquivo *index.js*. Então, altere o arquivo e insira as linhas abaixo como indicado na aula. Serão feitas as importações das bibliotecas instaladas, o carregamento do arquivo e a configuração da rota.

```
const swaggerUi = require('swagger-ui-express');
const YAML = require('yamljs');
const swaggerDocument = YAML.load('./api.yaml');

app.use('/api-docs', swaggerUi.serve,
swaggerUi.setup(swaggerDocument));
```

Antes de criar o arquivo que será carregado, é interessante instalar uma extensão chamada "*OpenAPI (Swagger) Editor*" do *vscode* para facilitar a criação do arquivo. Logo após, com a ajuda da extensão, crie o arquivo '*api.yaml*' na raiz do projeto.

Agora, ao acessar a rota da documentação, já será possível visualizar a documentação da API.

Notas

Para haver o recarregamento automático do *nodemon* quando houver alterações, é preciso alterar o *script* de *start* adicionando os tipos de arquivos que devem ser observados. O *script* deve ficar como abaixo:

```
"start": "cross-env NODE_ENV=development nodemon -e yaml,js,json --
exec node src/index.js"
```

Documentando a API

Link do video da aula: https://youtu.be/mFmn8jB54dA

Dando continuidade à criação da documentação da *API*, vamos inicialmente verificar o retorno do *GET* de usuários. É possível ver que está sendo retornada a lista de todos os usuários cadastrados. No entanto, há informações que não devem ser retornadas, por serem informações confidenciais ou de tratamento interno.

Tratando os dados de retorno

Antes de retornar à lista de todos os usuários e todas as informações contidas em cada um, é interessante fazer um tratamento desses dados coletados e retornar somente as informações desejadas. Então, no arquivo de '*usuario.rota.js*' crie uma função para preparar os dados, como abaixo:

```
function prepararResultado(usuario){
  const result = Object.assign({}, usuario)
  if (result.createdAt) delete result.createdAt
  if (result.updatedAt) delete result.updatedAt
  if (result.senha) delete result.senha
  return result
}
```

Após criada a função, no mesmo arquivo, navegue até a rota mencionada acima e adicione a preparação dos dados antes de retorná-los. O código da rota deve ficar como abaixo:

```
router.get("/", async (req, res) => {
  const usuarios = await Usuario.findAll();
  const resultado = usuarios.map(user =>
prepararResultado(user.dataValues))
  res.json({ usuarios: resultado });
});
```

Ainda nas rotas de usuário, existe a rota que busca um usuário pelo *id* e essa rota retorna também todas as informações, sem tratamento de dados. Então, adicione a mesma preparação de dados e a rota deve ficar como abaixo:

```
router.get("/:id", async (req, res) => {
  const usuario = await Usuario.findByPk(req.params.id);
  if (usuario) {
```

```
    res.json({ usuario: prepararResultado(usuario.dataValues) });
  } else {
    res.status(400).json({ msg: "Usuário não encontrado!" });
  }
});
```

Agora, voltemos à documentação.

Iniciando a documentação

Para iniciar a documentação, acompanhe a aula e veja a organização do arquivo. Após isso, vamos trabalhar na parte inicial do arquivo, inserindo informações e alterando alguns dados que vieram por padrão. Ao final, seu arquivo deverá estar assim:

```
openapi: '3.0.2'
info:
  title: IMD Blog API!
  version: '1.0'
  description: API do blog do IMD
  contact:
    name: Gustavo Leitão
servers:
  - url: http://localhost:8080/api
    description: Servidor de teste
paths:
  /test:
   get:
      responses:
         '200' :
          description: OK
```

Feita essa parte inicial, passaremos a documentar os *paths* da *API*.

Documentando os paths da API (Parte 1)

Link do video da aula: https://youtu.be/ripznqoNkFQ

Como já falado anteriormente, o arquivo do tipo *yaml* é tabulado, então é importante ter cuidado em sua criação para não haver erros.

Documentando as primeiras rotas

Inicialmente, serão documentadas duas rotas: a que retorna a lista de usuários e a que busca um usuário pelo *id*. Então, acompanhe a aula para obter todos os detalhes e informações e ao final é desejável que seu arquivo de documentação esteja como se vê abaixo:

```
openapi: '3.0.2'
info:
  title: IMD Blog API!
  version: '1.0'
  description: API do blog do IMD
  contact:
    name: Gustavo Leitão
servers:
  - url: http://localhost:8080/api
    description: Servidor de teste
paths:
  /usuarios:
    get:
      operationId: getUsuarios
      summary: Obtém todos os usuários do sistema
      tags:
        - Usuários
      responses:
        '200':
          description: Lista de usuários
          content:
            appication/json:
              schema:
                type: object
                properties:
                  usuarios:
                    type: array
                    items:
                      $ref:  "#/components/schemas/Usuario"

  /usuarios/{id}:
    get:
      operationId: getById
      summary: Obtém um usuário a partir do id
      tags:
        - Usuários
      parameters:
```

```
        - in: path
          name: id
          schema:
            type: integer
          required: true
      responses:
        '200':
          description: Retorna um usuário
          content:
            appication/json:
              schema:
                type: object
                properties:
                  usuario:
                    type: object
                    $ref: "#/components/schemas/Usuario"

        '400':
          description: Bad Request
          content:
            appication/json:
              schema:
                $ref:  "#/components/schemas/Erro"

components:
  schemas:
    Usuario:
      type: object
      properties:
        id:
          type: integer
          description: Id do usuário
          example: 9
        email:
          type: string
          description: E-mail do usuário
          example: foo@bar.com.br
    Erro:
      type: object
      properties:
        msg:
          type: string
          description: Mensagem de erro
```

```
          example: Usuário não encontrado!
```

Documentando os paths da API (Parte 2)

Link do video da aula: https://youtu.be/HMIsyJAQmYQ

Já foi visto como documentar para parâmetros que estejam no *path*, mas agora será visto como documentar parâmetros que venham na url através da *query*.

Documentando a remoção de usuários

No nosso arquivo já foi documentado o *GET* de usuários, agora será visto como documentar o *DELETE*. Então, com o auxílio da extensão instalada no *vscode* e acompanhando a aula, implemente no arquivo '*api.yaml*' a documentação de *delete*.

Abaixo, é possível encontrar a parte implementada na aula de hoje:

```
delete:
      tags:
        - Usuários
      description: Remove um usuário de determinado ID
      operationId: deleteUserByID
      summary: Remove um usuário
      parameters:
        - in: query
          name: id
          schema:
            type: integer
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  msg:
                    type: string
                    example: "Usuário deletado com sucesso!"
        '400':
```

```
          description: 400 Bad Request
          content:
            application/json:
              schema:
                $ref:  "#/components/schemas/Erro"
```

Documentando a autenticação da API

Link do video da aula: https://youtu.be/wiQQBBWkpXc

É válido ressaltar que o nosso projeto possui uma forma de *login* e é importante documentar essa parte também. Além disso, outras rotas dependem da autenticação para funcionar. Então, devemos documentar o tipo de autenticação e como se faz o *login*.

Acompanhe a aula para compreender melhor as alterações que serão feitas.

Documentando o tipo de autenticação

Acessando a documentação do *Swagger*, podemos ver o tipo de autenticação que fizemos, que é *JWT*, usando o padrão *Bearer Authentication*.

Sabendo disso, é preciso seguir a documentação do *Swagger* para inserir as linhas no arquivo yaml. Abaixo é possível encontrar os comandos a serem inseridos:

```
# 1) Define the security scheme type (HTTP bearer)
components:
  securitySchemes:
    bearerAuth:            # arbitrary name for the security scheme
      type: http
      scheme: bearer
      bearerFormat: JWT    # optional, arbitrary value for
documentation purposes
# 2) Apply the security globally to all operations
security:
  - bearerAuth: []         # use the same name as above
```

Documentando como se faz um *login*

A parte de logar é em '*/usuario/login*', sendo um *path* no arquivo. Então, é preciso documentar.

Veja abaixo a documentação do *login*:

```
/usuarios/login:
    post:
      operationId: userLogin
      summary: Realiza autenticação na API
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
                  type: string
                  example: foo@bar.com
                senha:
                  type: string
                  example: algosecreto
      tags:
        - Autenticação
      responses:
        '200':
          description: ok
          content:
            application/json:
              schema:
                type: object
                properties:
                  accessToken:
                    type: string
                    example:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjUsImlzcyI6ImltZC1iYWN
rZW5kIiwiYXVkIjoiaW1kLWZyb250ZW5kIiwiZW1haWwiOiJuZXd0b25AZ21haWwuY29
tIiwia
        '403':
          description: Forbidden
          content:
            application/json:
              schema:
                $ref:  "#/components/schemas/Erro"
```

Documentando rotas fechadas

Link do video da aula: https://youtu.be/Fw1D4qkbpqI

Para finalizar a aula, veremos como documentar uma rota que precisa de autenticação para funcionar.

Documentando a inserção de um novo post

Para adicionar um novo *post* na *API*, serão necessárias informações do usuário que está fazendo aquela inserção. Então, para obter essas informações é preciso fazer a autenticação antes.

Para documentar esse *path*, acompanhe a aula e veja as alterações feitas; ao final, todo o seu arquivo de documentação deve ficar como se observa abaixo:

```
openapi: '3.0.2'
info:
  title: IMD Blog API!
  version: '1.0'
  description: API do blog do IMD
  contact:
    name: Gustavo Leitão
servers:
  - url: http://localhost:8080/api
    description: Servidor de teste
paths:
  /posts:
    post:
      description: Adiciona uma postagem no blog
      operationId: postPost
      summary: Adiciona um post
      tags:
        - Posts
      requestBody:
        content:
          application/json:
            schema:
              $ref: "#/components/schemas/Post"
      responses:
        '200':
          description: OK
        '401':
          description: Não autorizado
          content:
            application/json:
              schema:
```

```
                $ref:  "#/components/schemas/Erro"
        '403':
          description: Acesso negado
          content:
            application/json:
              schema:
                $ref:  "#/components/schemas/Erro"

  /usuarios/login:
    post:
      operationId: userLogin
      summary: Realiza autenticação na API
      requestBody:
        content:
          application/json:
            schema:
              type: object
              properties:
                email:
                  type: string
                  example: foo@bar.com
                senha:
                  type: string
                  example: algosecreto
      tags:
        - Autenticação
      responses:
        '200':
          description: ok
          content:
            application/json:
              schema:
                type: object
                properties:
                  accessToken:
                    type: string
                    example:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOjUsImlzcyI6ImltZC1iYWN
rZW5kIiwiYXVkIjoiaW1kLWZyb250ZW5kIiwiZW1haWwiOiJuZXd0b25AZ21haWwuY29
tIiwia
        '403':
          description: Forbidden
          content:
            application/json:
```

```
              schema:
                $ref:  "#/components/schemas/Erro"
  /usuarios:
    get:
      operationId: getUsuarios
      summary: Obtém todos os usuários do sistema
      tags:
        - Usuários
      responses:
        '200':
          description: Lista de usuários
          content:
            appication/json:
              schema:
                type: object
                properties:
                  usuarios:
                    type: array
                    items:
                      $ref:  "#/components/schemas/Usuario"

    delete:
      tags:
        - Usuários
      description: Remove um usuáriodeterminado ID
      operationId: deleteUserByID
      summary: Remove um usuário
      parameters:
        - in: query
          name: id
          schema:
            type: integer
      responses:
        '200':
          description: OK
          content:
            application/json:
              schema:
                type: object
                properties:
                  msg:
                    type: string
                    example: "Usuário deletado com sucesso!"
        '400':
```

```
          description: 400 Bad Request
          content:
            application/json:
              schema:
                $ref:  "#/components/schemas/Erro"

  /usuarios/{id}:
    get:
      operationId: getById
      summary: Obtém um usuário a partir do id
      tags:
        - Usuários
      parameters:
        - in: path
          name: id
          schema:
            type: integer
          required: true
      responses:
        '200':
          description: Retorna um usuário
          content:
            appication/json:
              schema:
                type: object
                properties:
                  usuario:
                    type: object
                    $ref: "#/components/schemas/Usuario"

        '400':
          description: Bad Request
          content:
            appication/json:
              schema:
                $ref:  "#/components/schemas/Erro"

components:
  securitySchemes:
    bearerAuth:            # arbitrary name for the security scheme
      type: http
      scheme: bearer
      bearerFormat: JWT    # optional, arbitrary value for
documentation purposes
```

```
  schemas:
    Usuario:
      type: object
      properties:
        id:
          type: integer
          description: Id do usuário
          example: 9
        email:
          type: string
          description: E-mail do usuário
          example: foo@bar.com.br
    Post:
      type: object
      properties:
        titulo:
          type: string
          example: Titulo do seu post
        texto:
          type: string
          example: Conteudo do seu post...
        userId:
          type: integer
          example: 9
    Erro:
      type: object
      properties:
        msg:
          type: string
          description: Mensagem de erro
          example: Usuário não encontrado!

security:
  - bearerAuth: []
```

Resumo

Nesta aula, abordamos a importância da documentação de uma *API* e como fazer uma boa documentação, utilizando ferramentas de fácil acesso. Ademais, foi visto como documentar *paths*, autenticações e rotas fechadas.