

Conteúdo

Módulo 1: Introdução ao Node.JS

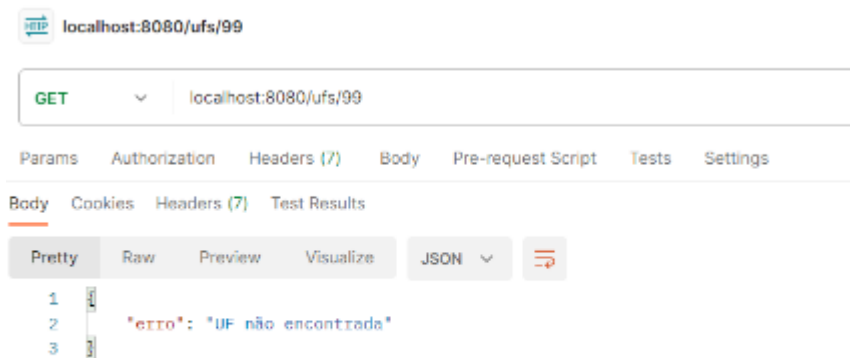
- APIs e coleção de dados
- Retornando a coleção
- Retornando um elemento único
- Tratamento de erros na API
- Aplicando buscas na API
- Conclusão
- Atividades Extras

Tratamento de erros na API

Agora iremos aplicar o tratamento de erros na nossa API de UFs. Por exemplo, se o cliente consumir:

<http://localhost:8080/ufs/99>

A API deveria retornar o código de status 404 (elemento não encontrado) - visto que não existe uma UF com ID = 99. Veja o print:



Codando o tratamento de erro

```
JS index.js > ...
1  import express from 'express';
2  import colecaoUf from '../dados/dados.js';
3
4  const app = express();
5
6  app.get('/ufs', (req, res) => {
7    res.json(colecaoUf);
8  });
9
10 app.get('/ufs/:iduf', (req, res) => {
11   const idUF = parseInt(req.params.iduf);
12   let uf = colecaoUf.find(u => u.id === idUF);
13
14   if (uf) {
15     res.json(uf);
16   } else {
17     res.status(404).send();
18   }
19 });
20
21 app.listen(8080, () => {
22   console.log('Servidor iniciado na porta 8080');
23 });
```

A sintaxe para retorno de um **código de status** é:

```
res.status(código de retorno).send();
```

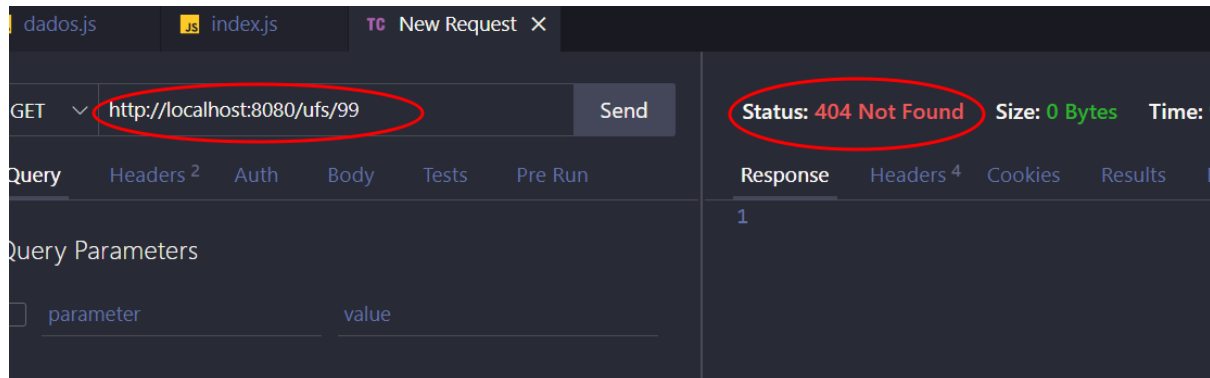
```
10 app.get('/ufs/:iduf', (req, res) => {
11   const idUF = parseInt(req.params.iduf);
12   let uf = colecaoUf.find(u => u.id === idUF);
13
14   if (uf) {
15     res.json(uf);
16   } else {
17     res.status(404).send();
18   }
19 });
```

- 1 Com `res.status` conseguimos retornar um código de erro. Note que **404** foi passado como **parâmetro**.
- 2 O código **404** será enviado se o método `find` não encontrar nenhuma ID.

```
10 app.get('/ufs/:iduf', (req, res) => {
11   const idUF = parseInt(req.params.iduf);
12   let uf = colecaoUf.find(u => u.id === idUF);
13
14   if (uf) {
15     res.json(uf);
16   } else {
17     res.status(404).send();
18   }
19 });
```

Testando agora 👍 com um valor que não existe e um valor inválido exemplos a seguir:

Testando com o valor ID=99 e como não temos precisa dar a msg "404 Not Found".



E agora vamos testar com letras passando como parâmetro:

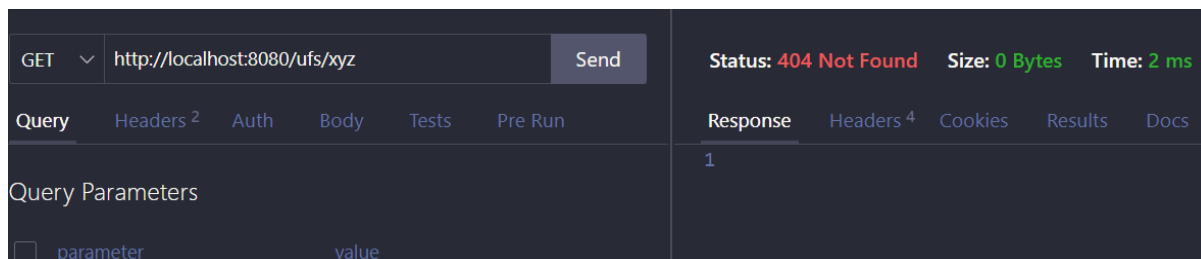
Mt bom para certificarmos que tudo corra bem, vamos fazer um if para tratar isso também.

```
app.get('/ufs/:iduf', (req, res) => {
  const idUF = parseInt(req.params.iduf);
  let uf;

  if (!(isNaN(idUF))) {
    uf = colecaoUf.find(u => u.id === idUF);
  }

  if (uf) {
    res.json(uf);
  } else {
    res.status(404).send();
  }
});
```

Unisenai



Por exemplo, se o cliente **consumir**:

`https://localhost:8080/ufs/xyz`

Código	Valor
<pre>const idUF = parseInt(req.params.iduf);</pre>	idUF terá o valor undefined;
<pre>if (!isNaN(idUF)) { uf = colecaoUf.find(u => u.id === idUF); }</pre>	<pre>isNaN(undefined) = true !(isNaN(undefined)) = false</pre> O método .find não será executado;
<pre>else { res.status(404).send(''); }</pre>	<pre>if (undefined) = false</pre> O método res.status(404) será executado.

Finalizamos essa parte inserindo mensagens para retornar quando tiver um erro, abaixo segue o código para inserimos:

Unisenai

```
import express from 'express';
import colecaoUf from './dados/dados.js';

const app = express();

app.get('/ufs', (req, res) => {
  res.json(colecaoUf);
});

app.get('/ufs/:iduf', (req, res) => {
  const idUF = parseInt(req.params.iduf);
  let mensagemErro = '';
  let uf;

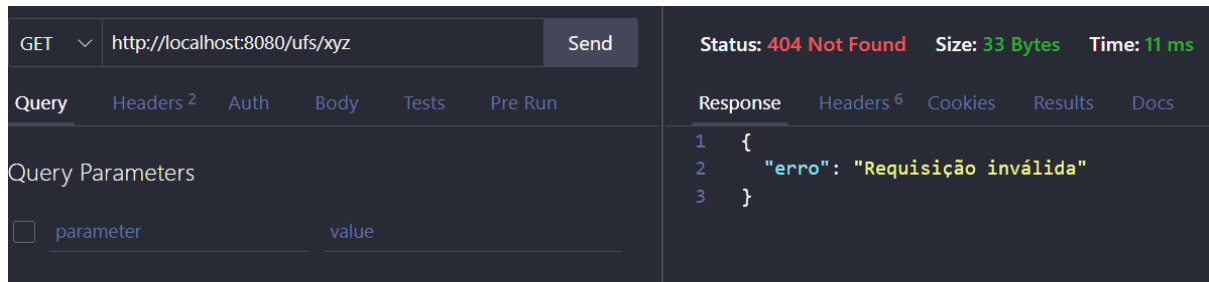
  if (!(isNaN(idUF))) {
    uf = colecaoUf.find(u => u.id === idUF);
    if (!uf) {
      mensagemErro = 'UF não encontrada';
    }
  } else {
    mensagemErro = 'Requisição inválida';
  }

  if (uf) {
    res.json(uf);
  } else {
    res.status(404).send({ "erro": mensagemErro });
  }
});

app.listen(8080, () => {
  console.log('Servidor iniciado na porta 8080');
});
```

Unisenai

Ao testarmos agora:



GET ☐ http://localhost:8080/ufs/xyz Send

Status: 404 Not Found Size: 33 Bytes Time: 11 ms

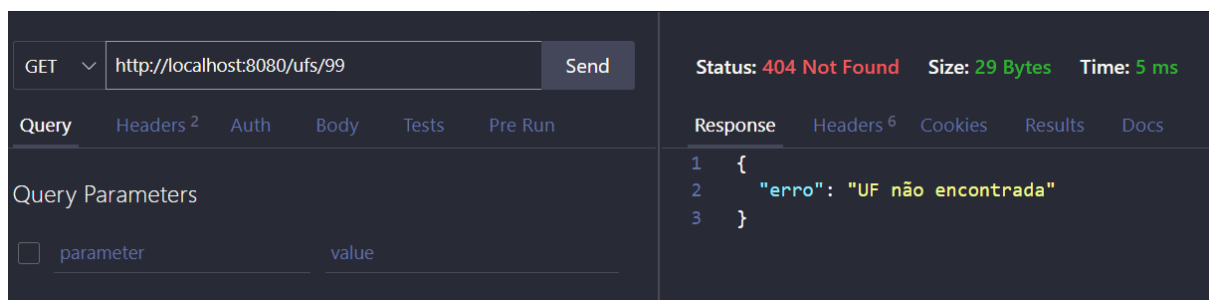
Query Headers 2 Auth Body Tests Pre Run

Response Headers 6 Cookies Results Docs

Query Parameters

☐ parameter value

```
1 {
2   "erro": "Requisição inválida"
3 }
```



GET ☐ http://localhost:8080/ufs/99 Send

Status: 404 Not Found Size: 29 Bytes Time: 5 ms

Query Headers 2 Auth Body Tests Pre Run

Response Headers 6 Cookies Results Docs

Query Parameters

☐ parameter value

```
1 {
2   "erro": "UF não encontrada"
3 }
```

Veja o código atualizado. Basicamente, setamos um **valor personalizado** à variável `mensagemErro`, dependendo do **tipo de erro**.

```
app.get('/ufs/:iduf', (req, res) => {
  const idUF = parseInt(req.params.iduf);
  let mensagemErro = '';
  let uf;

  if (!isNaN(idUF)) {
    uf = colecaoUf.find(u => u.id === idUF);
    if (!uf) {
      mensagemErro = 'UF não encontrada';
    } else {
      mensagemErro = 'Requisição inválida';
    }
  }

  if (uf) {
    res.json(uf);
  } else {
    res.status(404).send({ "erro": mensagemErro });
  }
});
```

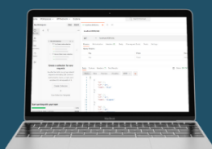


```
if (!isNaN(idUF)) {  
  uf = colecaoUF.find(u => u.id === idUF);  
  if (!uf) {  
    mensagemErro = 'UF não encontrada';  
  }  
} else {  
  mensagemErro = 'Requisição inválida';  
}
```

- 1 Se o `.find` não encontrar ocorrência, a variável será “UF não encontrada”
- 2 Se o valor de `(!(isNaN(idUF)))` for `false`, significa que o parâmetro enviado não é um número. Nesse caso, a variável recebe “Requisição inválida”.

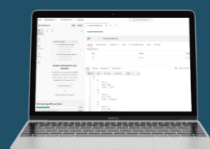
Vamos aplicar uma última evolução na nossa API. Agora, vamos retornar uma mensagem de erro diferente para cada cenário de erro:

Cenário 1: ID inexistente
localhost:8080/ufs/99



Resposta:
"UF não encontrada"

Cenário 2: Requisição inválida
localhost:8080/ufs/xyz



Resposta:
"Requisição inválida"

Finalizamos esta OT, mas iremos continuar o conteúdo na próxima.