

## Conteúdo

### ***Módulo 1: Introdução ao Node.JS***

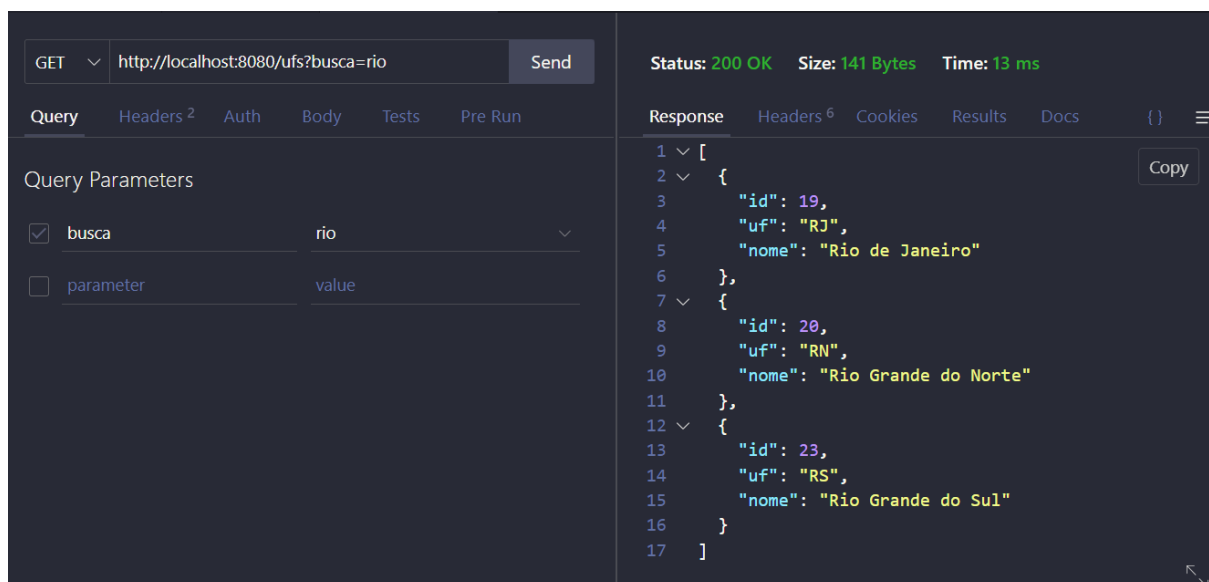
- APIs e coleção de dados
- Retornando a coleção
- Retornando um elemento único
- Tratamento de erros na API
- Aplicando buscas na API
- Conclusão
- Atividades Extras

Por fim vamos aplicar buscas na API

Nossa busca deve retornar todas as UFs que possuem o nome que colocarmos no seguinte parâmetro de pesquisa:

<http://localhost:8080/ufs?busca=rio>

A API deve retornar todas as UFs que possuem “rio” no nome.



The screenshot shows a web browser interface for testing an API. The URL bar displays `http://localhost:8080/ufs?busca=rio`. The 'Query Parameters' section shows a checked parameter `busca` with the value `rio`. The 'Response' section shows a successful status (200 OK) and a JSON array of three objects:

```
[{"id": 19, "uf": "RJ", "nome": "Rio de Janeiro"}, {"id": 20, "uf": "RN", "nome": "Rio Grande do Norte"}, {"id": 23, "uf": "RS", "nome": "Rio Grande do Sul"}]
```

Vamos implementar o código da busca em nosso **index.js**

Vamos atualizar o arquivo **index.js**, para que seja possível consumir a **API para buscar dados**.

Nesta versão, vamos implementar um método **filter** na primeira rota da **API**, através do endpoint: **http://localhost:8080/ufs?busca=rio**.

A **API** vai retornar para o cliente uma lista com o resultado da busca.

# UniSENAI

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

const app = express();

const buscarUfsPorNome = (nomeUf) => {
  return colecaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));
};
```

```
app.get('/ufs', (req, res) => {
  const nomeUf = req.query.busca;
  const resultado = nomeUf ? buscarUfsPorNome(nomeUf) : colecaoUf;
  if (resultado.length > 0) {
    res.json(resultado);
  } else {
    res.status(404).send({ "erro": "Nenhuma UF encontrada" });
  }
});
```

```
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');  
});
```

Há uma limitação no framework **Express**: não é possível criar um método `app.get` apenas para o consumo de buscas.

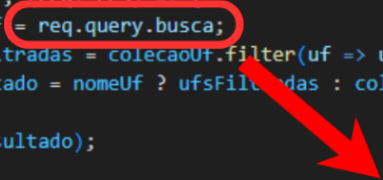
Por este motivo, aplicamos a **busca** no código da **primeira rota (/ufs)**:

```
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const ufsFiltradas = colecaoUf.filter(uf => uf.nome.includes(nomeUf));  
  const resultado = nomeUf ? ufsFiltradas : colecaoUf;  
  
  res.json(resultado);  
});
```

`req.query` armazena o valor do parâmetro enviado pelo cliente. Sintaxe:

`req.query.nome_do_parametro`

```
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const ufsFiltradas = colecaoUf.filter(uf => uf.nome.includes(nomeUf));  
  const resultado = nomeUf ? ufsFiltradas : colecaoUf;  
  
  res.json(resultado);  
});
```

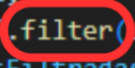
A red circle highlights the variable `req.query.busca` in the first line of the code. A red arrow points from this circle to the `filter` method in the second line, which also has a red circle around it. This illustrates how the value from the URL is passed to the filter function.

`localhost:8080/ufs?busca=Rio`

Neste exemplo, a constante `nomeUf` terá o valor 'Rio'

Em seguida, aplicamos o método `.filter` para criar uma nova lista de elementos filtrados (`ufsFiltradas`):

```
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const ufsFiltradas = colecaoUf.filter(uf => uf.nome.includes(nomeUf));  
  const resultado = nomeUf ? ufsFiltradas : colecaoUf;  
  
  res.json(resultado);  
});
```

A red circle highlights the `filter` method in the second line of the code. This highlights the specific method used to create the filtered list `ufsFiltradas`.

Usamos uma **condicional** para definir o retorno. Se a constante `nomeUf` tiver **algum valor**, o retorno será **`ufsFiltradas`**. Senão, o retorno será **`colecacaoUf`**.

```
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const ufsFiltradas = colecacaoUf.filter(uf => uf.nome.includes(nomeUf));  
  const resultado = nomeUf ? ufsFiltradas : colecacaoUf;  
  
  res.json(resultado);  
});
```

Note que `nomeUf` só terá valor caso o cliente esteja realizando uma busca

Vamos melhorar o código. O método **`.toLowerCase()`** foi usado para que a busca **não seja case sensitive**:

```
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const ufsFiltradas = colecacaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));  
  const resultado = nomeUf ? ufsFiltradas : colecacaoUf;  
  
  res.json(resultado);  
});
```

Vamos aplicar uma nova melhoria. Vamos deixar o código mais organizado, criando uma **função para buscar a UF na coleção**:

```
const buscarUfsPorNome = (nomeUf) => {  
  return colecacaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));  
};  
  
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const resultado = nomeUf ? buscarUfsPorNome(nomeUf) : colecacaoUf;
```

Note como o código ficou mais limpo e fácil de entender.

Por fim, vamos aplicar o **tratamento de erro** na rota:

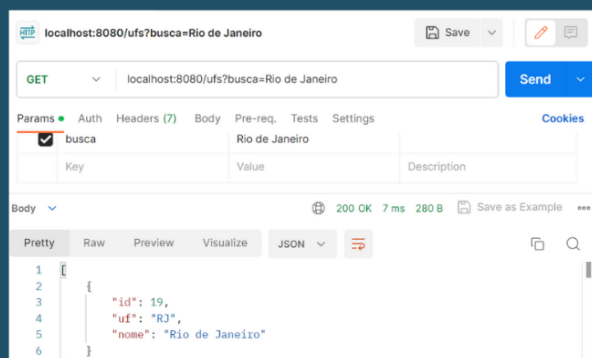
```
const buscarUfsPorNome = (nomeUf) => {  
  return colecaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));  
};  
  
app.get('/ufs', (req, res) => {  
  const nomeUf = req.query.busca;  
  const resultado = nomeUf ? buscarUfsPorNome(nomeUf) : colecaoUf;  
  
  if (resultado.length > 0) {  
    res.json(resultado);  
  } else {  
    res.status(404).send({ "erro": "Nenhuma UF encontrada" });  
  }  
});
```

A API vai retornar o **código 404** caso nenhuma UF seja encontrada na busca.

Testando a Busca:

Primeiro, vamos testar a busca com o **nome exato da UF**:

localhost:8080/ufs?busca=Rio de Janeiro





# Unisenai

Um detalhe sobre o uso de espaços na busca:

```
localhost:8080/ufs?busca=Rio de Janeiro
```

A forma **correta** para o cliente enviar os dados é:

```
localhost:8080/ufs?busca=Rio%20de%20Janeiro
```

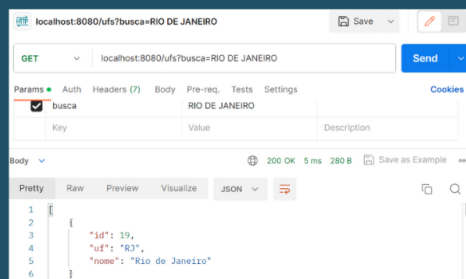
Ou seja, espaços devem ser substituídos por **%20**.  
Essa sintaxe é uma preocupação do **sistema Front-end**.

Prosseguindo, vamos testar com valores **case sensitive**.

Os consumos abaixo devem ter o **mesmo retorno**:

```
localhost:8080/ufs?busca=RIO DE JANEIRO
```

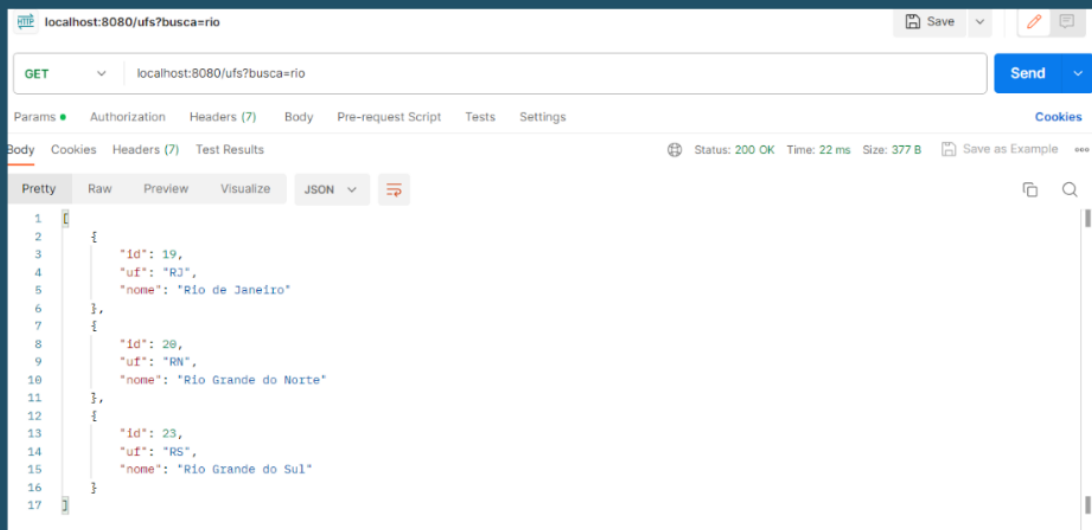
```
localhost:8080/ufs?busca=rio de janeiro
```





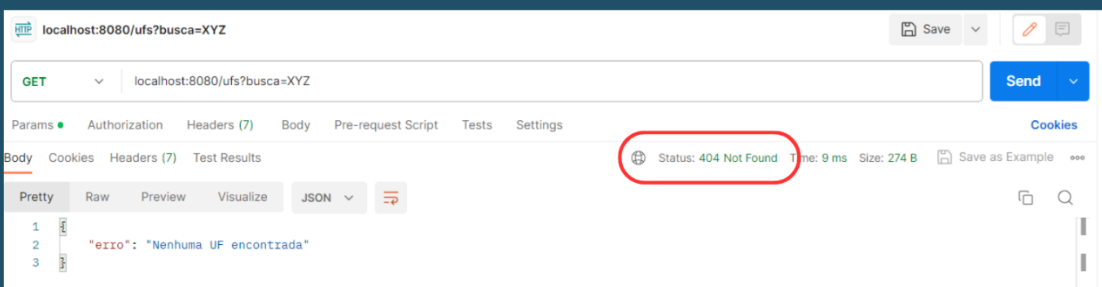
Busca a partir de um trecho:

`localhost:8080/ufs?busca=rio`



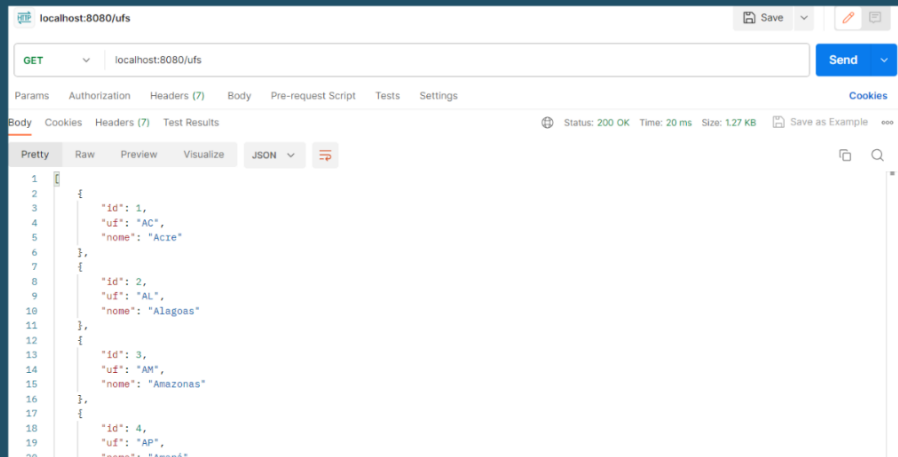
Vamos testar um caso de UF não encontrada:

`localhost:8080/ufs?busca=XYZ`



Por fim, testamos o consumo de **toda a coleção**:

localhost:8080/ufs



**Finalizamos nossa API mais completa do que nunca!**

## Atividades

**Realizar a criação da busca na nossa API de Games das Ots passadas e o tratamento de erros quando buscar por um id de um Game que não existe.**

**Enviar separada essa OT a API feita e a atividade dessa OT em pastas separadas para que o professor avalie.**