

## Conteúdo

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

- Refatorando o código da aplicação anterior
- Criando camadas de serviços na aplicação

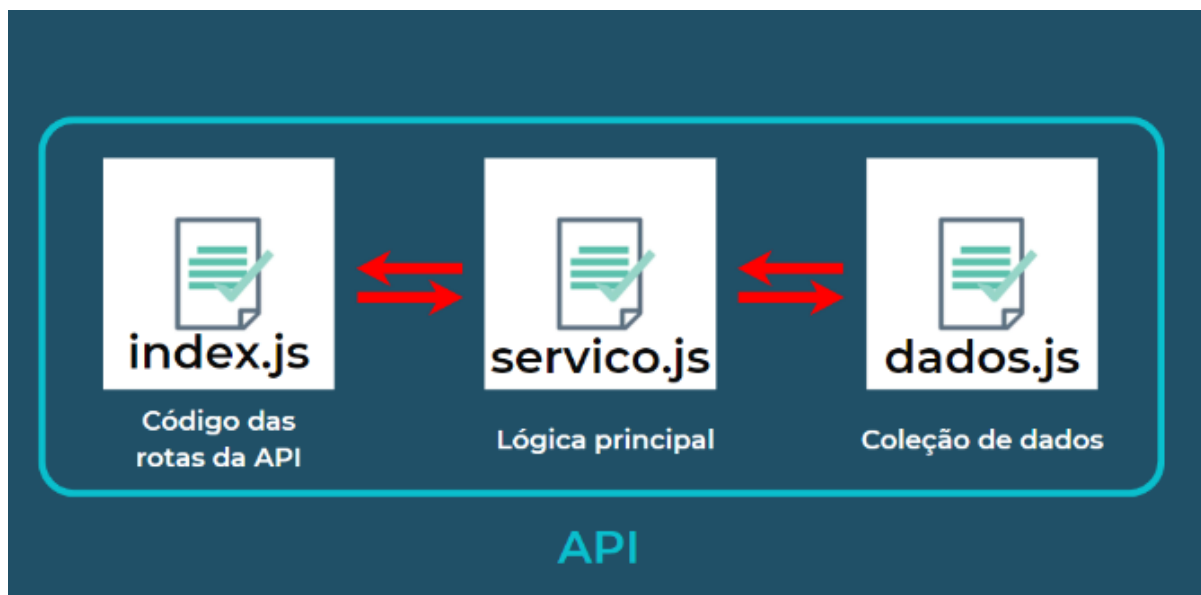
## Refatorando o código

Nossa API já está completamente funcional, com diferentes rotas de consumo. Antes de finalizar a missão, falta apenas organizar o código.

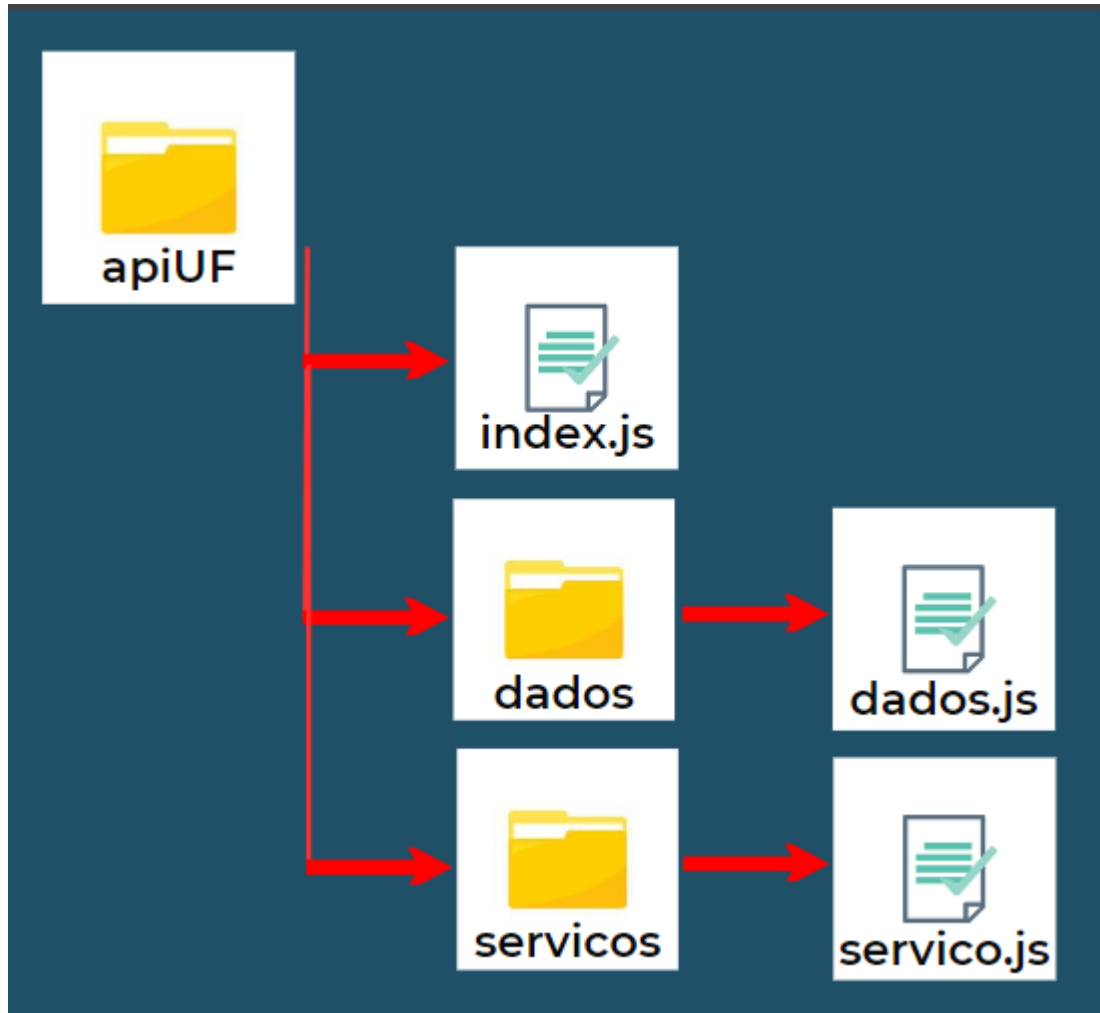
É importante separar o código em camadas, a fim de manter o código mais limpo e organizado.

Veremos nessa aula como aplicar este conceito na nossa API de UFs, através da criação da camada de serviço, a qual chamaremos de “**servico.js**”, que terá a lógica principal do sistema.

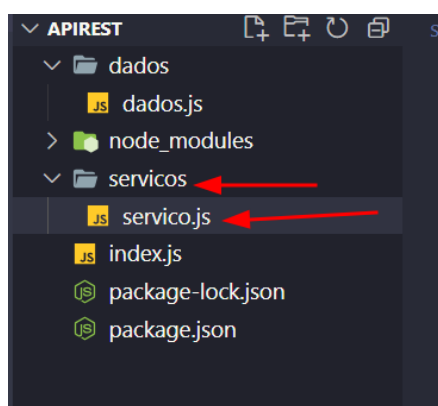
Veja como deve funcionar o consumo interno da API abaixo:



A estrutura de arquivos será utilizada conforme a imagem abaixo:



Antes de começar vamos iniciar mudando a nossa estrutura, acrescentando a pasta `servicos` e criando o arquivo `servicos.js` conforme imagem acima.



O objetivo desta aula é criar a **camada de serviço**, responsável pela **lógica principal** da API.



Na camada de serviço ficará toda nossa lógica da aplicação conforme imagem abaixo:

A imagem mostra uma captura de tela de um editor de código com o arquivo **servico.js** aberto. O explorador de arquivos à esquerda mostra a estrutura do projeto: **APIREST** > **dados** > **dados.js**, **node\_modules**, **servicos** > **servico.js**, **index.js**, **package-lock.json** e **package.json**. O código no editor é o seguinte:

```
1 import colecaoUf from '../dados/dados.js';
2
3 export const buscarUfs = () => {
4   return colecaoUf;
5 }
6
7 export const buscarUfsPorNome = (nomeUf) => {
8   return colecaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));
9 };
10
11 export const buscarUfsPorId = (id) => {
12   const idUF = parseInt(id);
13   return colecaoUf.find(uf => uf.id === idUF);
14 }
```

As funções representam **todas as possibilidades** de consumo da API:

```
1 export const buscarUfs = () => {  
  return colecaoUf;  
}  
  
2 export const buscarUfsPorNome = (nomeUf) => {  
  return colecaoUf.filter(uf => uf.nome.toLowerCase().includes(nomeUf.toLowerCase()));  
};  
  
3 export const buscarUfPorId = (id) => {  
  const idUF = parseInt(id);  
  return colecaoUf.find(uf => uf.id === idUF);  
};
```

- 1 Retorno de **toda a coleção**;
- 2 Busca de UF **através do nome**;
- 3 Retorno da UF de **ID específico**.

Olhe a mudança no **index.js**.

Observe o **novo** arquivo **index.js**.

Vamos entender o que mudou.

```
JS index.js > ...  
1 import express from 'express';  
2 import { buscarUfs, buscarUfPorId, buscarUfsPorNome } from './servicos/servico.js';  
3  
4  
5 const app = express();  
6  
7  
8 app.get('/ufs', (req, res) => {  
9   const nomeUf = req.query.busca;  
10  const resultado = nomeUf ? buscarUfsPorNome(nomeUf) : buscarUfs();  
11  if (resultado.length > 0) {  
12    res.json(resultado);  
13  } else {  
14    res.status(404).send({ "erro": "Nenhuma UF encontrada" });  
15  }  
16 });  
17  
18 app.get('/ufs/:iduf', (req, res) => {  
19  const uf = buscarUfPorId(req.params.iduf);  
20  if (uf) {  
21    res.json(uf);  
22  } else if (isNaN(parseInt(req.params.iduf))) {  
23    res.status(400).send({ "erro": "Requisição inválida" });  
24  } else {  
25    res.status(404).send({ "erro": "UF não encontrada" });  
26  }  
27 });  
28  
29 app.listen(8080, () => {  
30   console.log('Servidor iniciado na porta 8080');  
31 });
```

index.js

O primeiro ponto é que **index.js** não lida mais diretamente com a **coleção de dados** :

```
JS index.js > ...
1  import express from 'express';
2  import colecaoUf from './dados/dados.js';
3
4  const app = express();
5
```

Antigo

```
JS index.js > ...
1  import express from 'express';
2  import { buscarUfs, buscarUfPorId, buscarUfsPorNome } from './servicos/servico.js';
3
4
5  const app = express();
6
```

Novo

Outro ponto é sobre a **lógica da API**, que agora se encontra na **camada de serviço**:

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

app.get('/ufs', (req, res) => {
  const nomeUf = req.query.buscar;
  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 });
  }
});
```

Antigo

```
import express from 'express';
import { buscarUfs, buscarUfPorId, buscarUfsPorNome } from './servicos/servico.js';

const app = express();

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

app.get('/ufs/:iduf', (req, res) => {
  const uf = buscarUfPorId(req.params.iduf);
  if (uf) {
    res.json(uf);
  } else if (!isNaN(parseInt(req.params.iduf))) {
    res.status(400).send({ "erro": "Requisição inválida" });
  } else {
    res.status(404).send({ "erro": "UF não encontrada" });
  }
});
```

Novo

```
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" });  
  }  
});
```

Antigo

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

Novo

- 1 O index.js não contém mais o código da função **buscarUfsPorNome**;
- 2 A rota '/ufs' também **não acessa** mais a **coleção** diretamente. Agora ela consome a função **buscarUfs** do **servico.js**;

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

Antigo

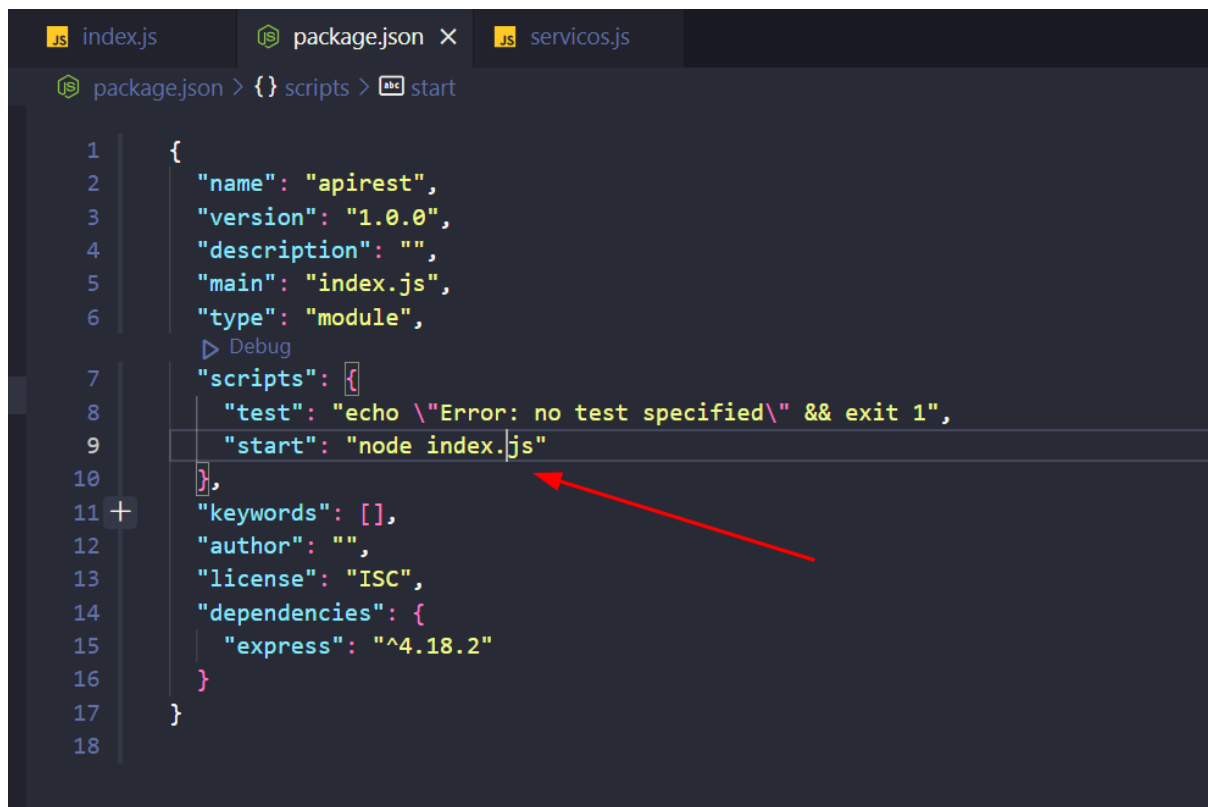
```
app.get('/ufs/:iduf', (req, res) => {  
  const uf = buscarUfPorId(req.params.iduf);  
  if (uf) {  
    res.json(uf);  
  } else if (isNaN(parseInt(req.params.iduf))) {  
    res.status(400).send({ "erro": "Requisição inválida" });  
  } else {  
    res.status(404).send({ "erro": "UF não encontrada" });  
  }  
});
```

Novo

- 3 A rota **/ufs:id** também mudou. Agora ela consome a função **buscarUFporID**.

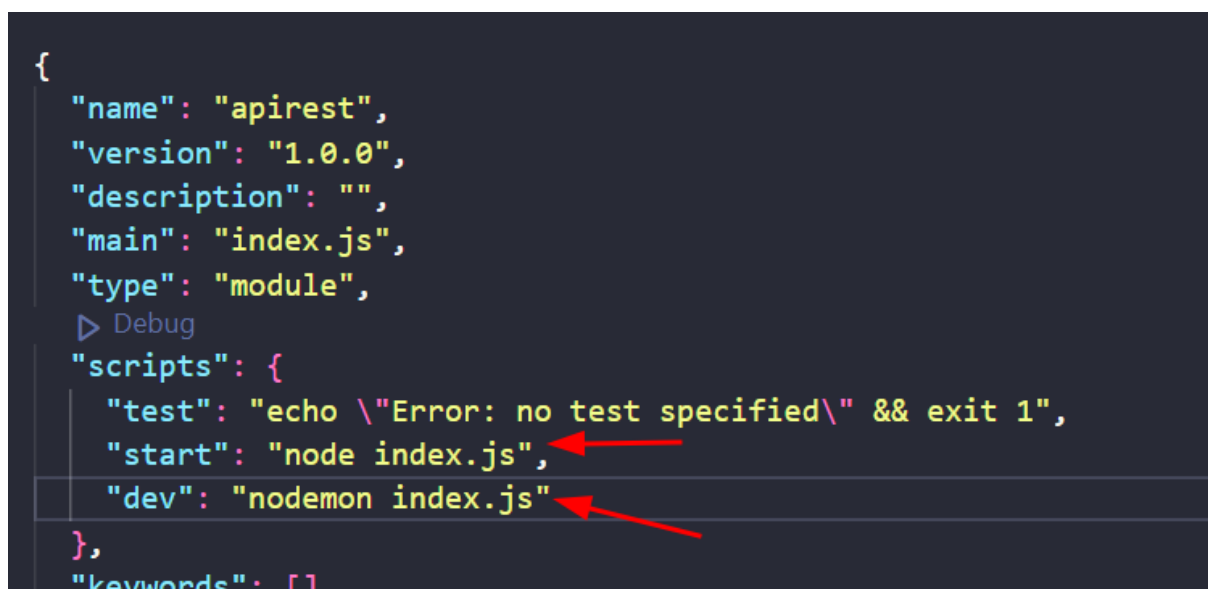
Por fim vamos criar um script para testarmos nossa API, no arquivo **package.json**, em **scripts** criem o script **"start"** igual na imagem abaixo:

# UnisenAI



```
index.js package.json X servicios.js
package.json > {} scripts > start

1  {
2    "name": "apiREST",
3    "version": "1.0.0",
4    "description": "",
5    "main": "index.js",
6    "type": "module",
7    "scripts": {
8      "test": "echo \"Error: no test specified\" && exit 1",
9      "start": "node index.js"
10   },
11   "keywords": [],
12   "author": "",
13   "license": "ISC",
14   "dependencies": {
15     "express": "^4.18.2"
16   }
17 }
18
```



```
{
  "name": "apiREST",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node index.js",
    "dev": "nodemon index.js"
  },
  "keywords": [],
```

Ao término basta acessarmos o terminal e rodarmos o comando “npm run start” ou “npm run dev” com o nodemon, para isso precisamos instalar ele também como já aprendemos via NPM ok?



# Unisenai

94%

feito isso basta acessarmos a nossa API e testarmos e veremos como ficou bem melhor dividido por camadas e sem deixar toda a responsabilidade no arquivo index.js.

```
localhost:8080/ufs
[{"id":1,"uf":"AC","nome":"Acre"}, {"id":2,"uf":"AL","nome":"Alagoas"}, {"id":3,"uf":"AM","nome":"Amazonas"}, {"id":4,"uf":"AP","nome":"Amapá"}, {"id":5,"uf":"BA","nome":"Bahia"}, {"id":6,"uf":"CE","nome":"Ceará"}, {"id":7,"uf":"DF","nome":"Distrito Federal"}, {"id":8,"uf":"ES","nome":"Espírito Santo"}, {"id":9,"uf":"GO","nome":"Goiás"}, {"id":10,"uf":"MA","nome":"Maranhão"}, {"id":11,"uf":"MG","nome":"Minas Gerais"}, {"id":12,"uf":"MS","nome":"Mato Grosso do Sul"}, {"id":13,"uf":"MT","nome":"Mato Grosso"}, {"id":14,"uf":"PA","nome":"Paraná"}, {"id":15,"uf":"PB","nome":"Paraíba"}, {"id":16,"uf":"PE","nome":"Pernambuco"}, {"id":17,"uf":"PI","nome":"Piauí"}, {"id":18,"uf":"PR","nome":"Paraná"}, {"id":19,"uf":"RJ","nome":"Rio de Janeiro"}, {"id":20,"uf":"RN","nome":"Rio Grande do Norte"}, {"id":21,"uf":"RO","nome":"Rondônia"}, {"id":22,"uf":"RR","nome":"Roraima"}, {"id":23,"uf":"RS","nome":"Rio Grande do Sul"}, {"id":24,"uf":"SC","nome":"Santa Catarina"}, {"id":25,"uf":"SE","nome":"Sergipe"}, {"id":26,"uf":"SP","nome":"São Paulo"}, {"id":27,"uf":"TO","nome":"Tocantins"}]
```

The screenshot shows a VS Code editor with a REST client request in `index.js` and a terminal window at the bottom.

**REST Client Request:**

- Method: GET
- URL: localhost:8080/ufs
- Status: 200 OK
- Size: 1.04 KB
- Time: 5 ms

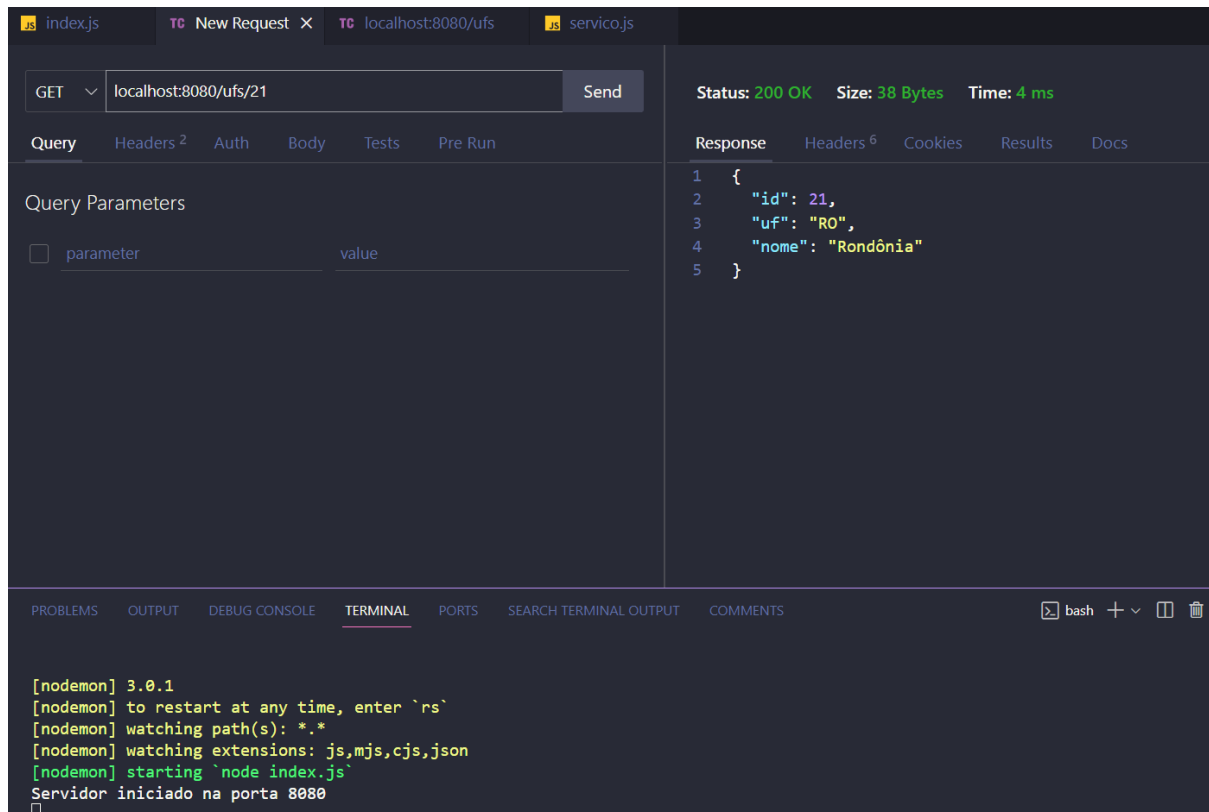
**Response:**

```
[{"id": 1, "uf": "AC", "nome": "Acre"}, {"id": 2, "uf": "AL", "nome": "Alagoas"}, {"id": 3, "uf": "AM", "nome": "Amazonas"}, {"id": 4, "uf": "AP", "nome": "Amapá"}, {"id": 5, "uf": "BA", "nome": "Bahia"}, {"id": 6, "uf": "CE", "nome": "Ceará"}, {"id": 7, "uf": "DF", "nome": "Distrito Federal"}, {"id": 8, "uf": "ES", "nome": "Espírito Santo"}, {"id": 9, "uf": "GO", "nome": "Goiás"}, {"id": 10, "uf": "MA", "nome": "Maranhão"}, {"id": 11, "uf": "MG", "nome": "Minas Gerais"}, {"id": 12, "uf": "MS", "nome": "Mato Grosso do Sul"}, {"id": 13, "uf": "MT", "nome": "Mato Grosso"}, {"id": 14, "uf": "PA", "nome": "Paraná"}, {"id": 15, "uf": "PB", "nome": "Paraíba"}, {"id": 16, "uf": "PE", "nome": "Pernambuco"}, {"id": 17, "uf": "PI", "nome": "Piauí"}, {"id": 18, "uf": "PR", "nome": "Paraná"}, {"id": 19, "uf": "RJ", "nome": "Rio de Janeiro"}, {"id": 20, "uf": "RN", "nome": "Rio Grande do Norte"}, {"id": 21, "uf": "RO", "nome": "Rondônia"}, {"id": 22, "uf": "RR", "nome": "Roraima"}, {"id": 23, "uf": "RS", "nome": "Rio Grande do Sul"}, {"id": 24, "uf": "SC", "nome": "Santa Catarina"}, {"id": 25, "uf": "SE", "nome": "Sergipe"}, {"id": 26, "uf": "SP", "nome": "São Paulo"}, {"id": 27, "uf": "TO", "nome": "Tocantins"}]
```

**Terminal Output:**

```
[nodeemon] 3.0.1
[nodeemon] to restart at any time, enter `rs`
[nodeemon] watching path(s): *.*
[nodeemon] watching extensions: js,mjs,cjs,json
[nodeemon] starting `node index.js`
Servidor iniciado na porta 8080
```

# Unisenai



GET localhost:8080/ufs/21

Status: 200 OK Size: 38 Bytes Time: 4 ms

Response

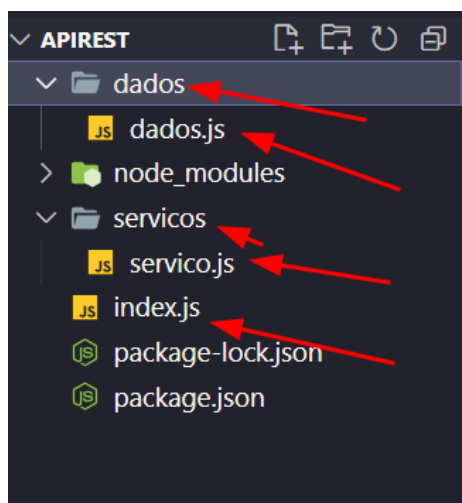
```
1 {
2   "id": 21,
3   "uf": "RO",
4   "nome": "Rondônia"
5 }
```

Query Parameters

parameter	value
-----------	-------

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS SEARCH TERMINAL OUTPUT COMMENTS

```
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Servidor iniciado na porta 8080
```



Chegou até aqui , meus parabéns, a nossa API de UFs está 100% funcional e dividida em camadas.