

Conteúdo

Módulo 1: Introdução ao Node.JS

- Rotas com Express
- CRUD com Express

A princípio, quando trabalharmos com servidores HTTP vamos ter que apontar para onde o navegador terá que ir com as URLs, desta forma precisamos trabalhar com as **rotas**.

Sempre que apontamos o navegador para alguma rota, ele faz uma requisição, se tratando do protocolo HTTP, nós temos os métodos ou verbos HTTP que definem o comportamento da requisição, os mais usados são get, post, put, delete.

Servidor HTTP com Express

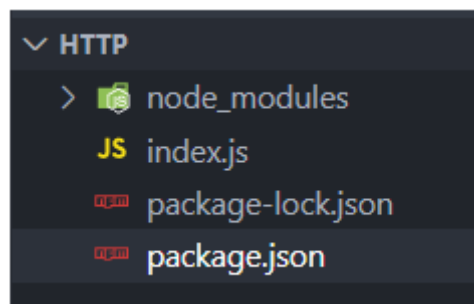
Primeiramente vamos precisar ter um servidor rodando em nosso ambiente, para isso vamos precisar:

- Configurar ambiente Node.js;
- Instalar o Express com o NPM;
- Importar o Express e iniciar o servidor na nossa aplicação.

Agora vamos entender o gerenciamento de rotas do Express.

Rota com Requisição GET

Vamos criar uma aplicação com a seguinte estrutura:



Para iniciarmos o package.json usamos o comando:

UniSENAI

```
npm init -y
```

```
npm install --save express
```

Nosso servidor está no arquivo index.js, da seguinte maneira:

```
import Express from 'Express';

const app = Express();

app.listen(3000, () =>
  console.log('Servidor iniciado na porta 3000')
);
```

Agora precisamos criar a **rota** raiz, para isso vamos utilizar o método (ou verbo) **get**, caso contrário ao entrarmos em localhost:3000 (endereço de nosso servidor local), vamos obter a seguinte mensagem no navegador: **Cannot GET /**.

Por isso vamos começar pelo `get`. O Express permite o uso dos métodos (verbos) HTTP de maneira muito simples, neste caso vamos utilizar o método

`app.get()`:

```
import Express from 'Express';

const app = Express();

app.get('/', (req, res) =>
  res.send("<h3>Rotas no Express</h3><p>Rota '/'")
);

app.listen(3000, () =>
  console.log('Servidor iniciado na porta 3000')
);
```

Como podemos ver, utilizamos o método **get** seguindo o primeiro parâmetro, onde apontamos a rota. Logo em seguida, passamos uma **arrow function**, que recebe uma **request** e um **response**, com o **response** podemos usar o método **send** e exibir o conteúdo que desejamos, neste caso, uma mensagem em formato **HTML**, porém no **response** você poderia passar um **JSON** por exemplo.

Ao executar o nosso servidor e entrar na rota principal, depois de configurado a rota, vamos obter o seguinte resultado:

teremos um erro se não corrigirmos no `package.json` e colocarmos o `type: module` :

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

agora rodando o programa :

node index.js teremos:

```
lenni@DESKTOP-VI20DSV MINGW64 ~/OneDrive/Área de Trabalho/Carlos/http
$ node index.js
Servidor iniciado na porta 3000
█
```

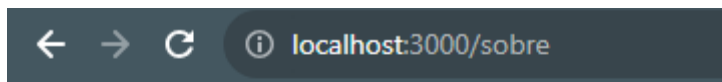
← → ↻ ⓘ localhost:3000

Rotas no Express

Rota '/'

Nós podemos, por exemplo, criar outras rotas de acordo com a necessidade da sua aplicação, como uma página sobre, desta forma, utilizando a rota **/sobre**:

Utilizando a mesmo conceito do método `app.get()`, crie a rota **/sobre**, com um conteúdo diferente da **rota** principal:



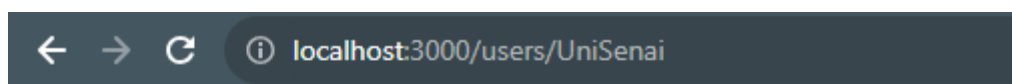
Rotas no Express

Vamos aprender a utilizar Rotas com Express

Nós podemos também receber parâmetro pela URL usando o método `get`, para isso precisamos, no momento de configurar a rota, utilizar `:parametro`, onde `:` é o que caracteriza a variável que vamos usar, conforme exemplo abaixo:

```
app.get('/users/:name', (req, res) => //recebe o parâmetro name
  res.send('Usuário:' + req.params.name) //exibe o parametro name
);
```

Dessa forma vamos receber um parâmetro e exibir na resposta da requisição:



Usuário:UniSenai

Rota com Requisição POST

Antes de tudo para exemplificar uma rota com express utilizando post vamos criar um **array**, vamos chamar esse array de carros e criar uma lista com o nome de carros aleatórios:

```
var carros = ['fiesta', 'saveiro'];
```

Em seguida vamos criar uma rota **GET** para consultar os dados deste array, utilizando o índice do vetor para acessar o valor de cada item do vetor, da seguinte forma:

```
app.get('/cars/:id', (req, res) => {  
  let id = req.params.id;  
  return res.json([carros[id]])  
});
```

```
JS index.js 1 X {} package.json
JS index.js > ...
13 app.get('/users/:name', (req, res) => //recebe o parâmetro name
14 |   res.send('Usuário:' + req.params.name) //exibe o parametro
15 | );
16
17 app.listen(3000, () =>
18 | console.log('Servidor iniciado na porta 3000')
19 | );
20
21 var carros = ['fiesta', 'saveiro'];
22
23 app.get('/cars/:id', (req, res) => {
24 |   let id = req.params.id;
25 |   return res.json([carros[id]])
26 | });
27
```

Se fizermos uma consulta passando o índice **0**, vamos obter o seguinte retorno:

```
← → ↻ ⓘ localhost:3000/cars/0
["fiesta"]
```

Que corresponde ao índice **0** de nosso **Array**.

Agora vamos cadastrar novos carros em nosso array utilizando o verbo **POST**, no Express podemos facilmente utilizar junto ao método `app.post()`. Então vamos criar uma rota e enviar o nome do novo carro pela requisição post da seguinte forma:

Vamos adicionar a seguinte linha após a declaração do array.

Unisenai

```
app.use(Express.urlencoded({ extended: true }));
```

Como podem ver abaixo :

```
var carros = ['fiesta', 'saveiro'];
app.use(Express.urlencoded({ extended: true }));

app.get('/cars/:id', (req, res) => {
  let id = req.params.id;
  return res.json([carros[id]])
});
```

E então, utilizar o método `app.post()`.

```
var carros = ['fiesta', 'saveiro'];
app.use(Express.urlencoded({ extended: true }));

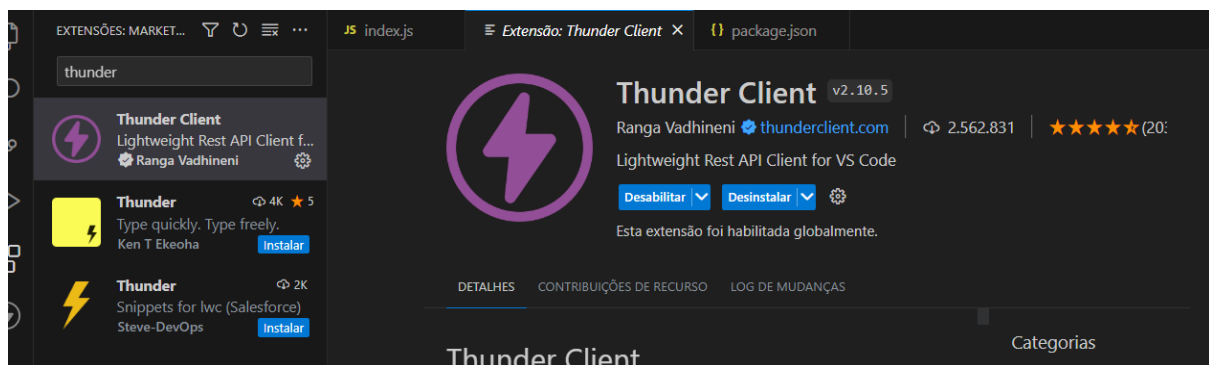
app.get('/cars/:id', (req, res) => {
  let id = req.params.id;
  return res.json([carros[id]])
});

app.post('/cars/', (req, res) => {
  let name = req.body.name;
  carros[carros.length] = name;
  return res.json([carros[carros.length - 1]]);
});
```

Unisenai

OBS: perceba que adicionamos a linha `app.use(Express.urlencoded({ extended: true }));`. Como estamos trabalhando com JSON, precisamos fazer o parsing do conteúdo que recebemos nas requisições. Para isso utilizamos o `urlencoded`, e no caso o `extended: true`, para selecionar a biblioteca compatível com JSON. Para mais informações sobre a função `urlencoded()`, você pode estar acessando a [documentação do Express](#).

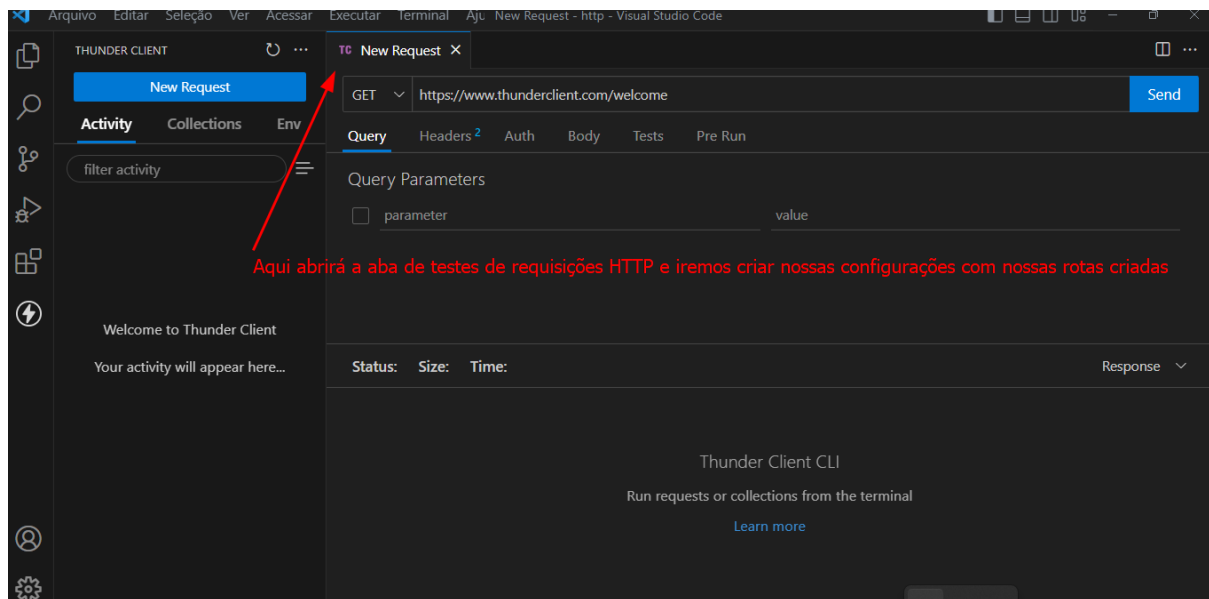
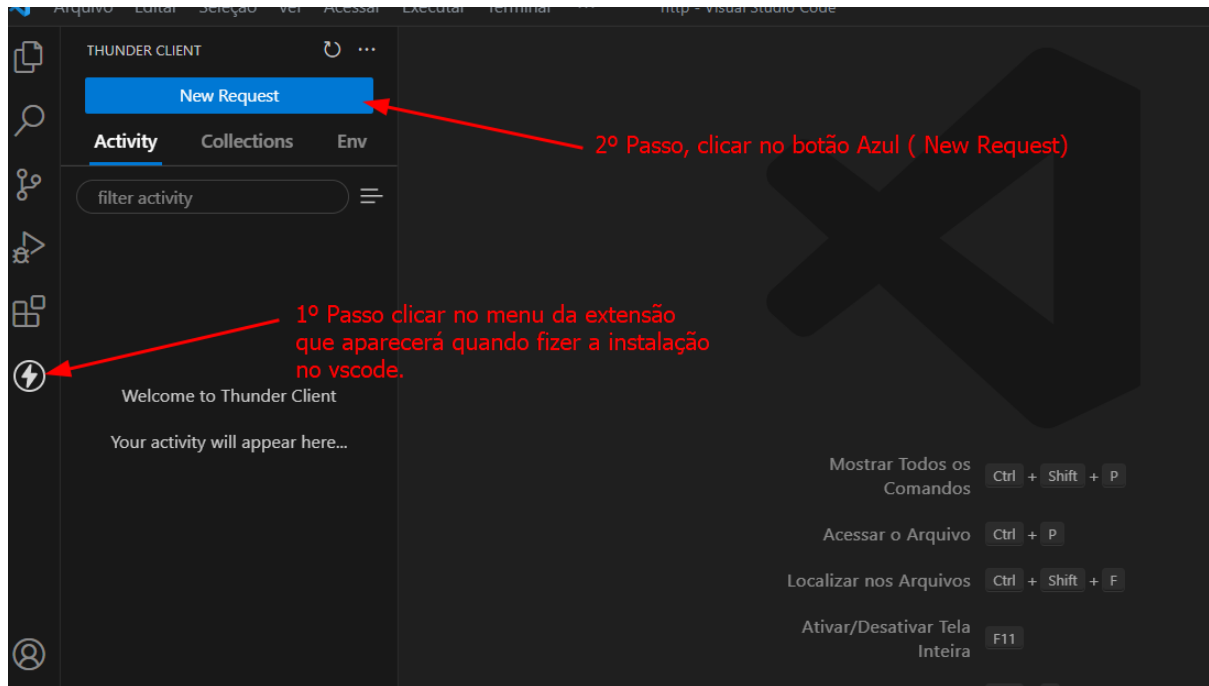
Aprendendo a utilizar a extensão do Vscode Thunder Client



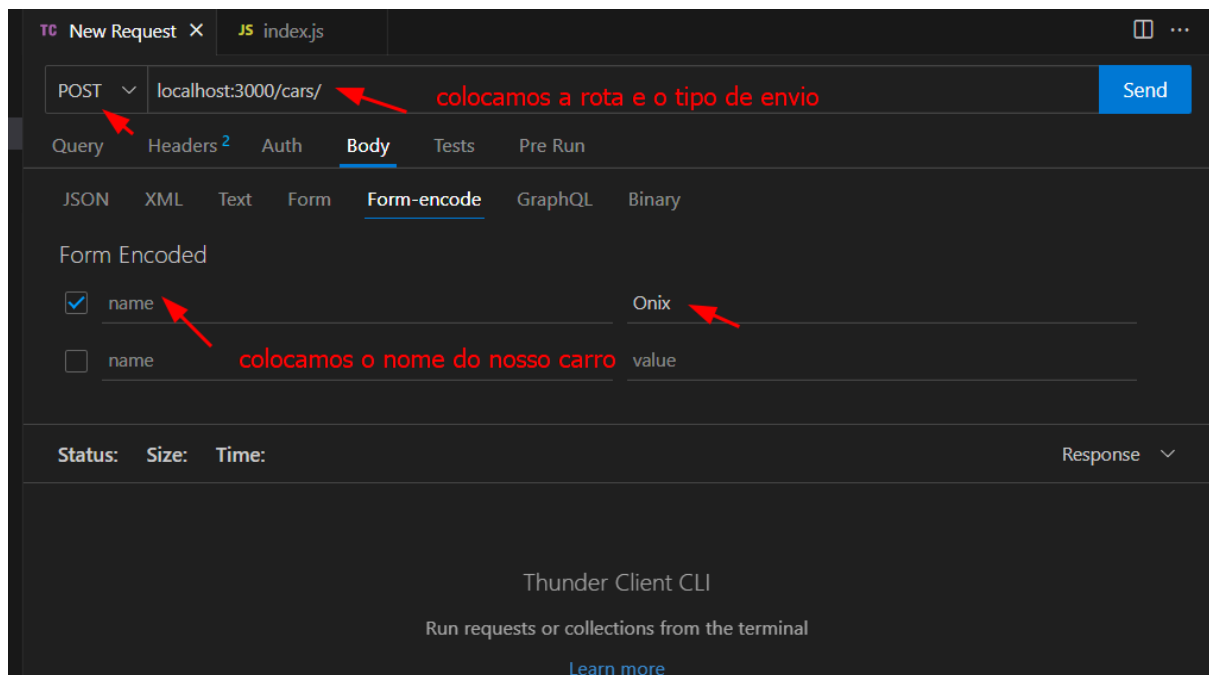
O **ThunderClient** é uma ferramenta que podemos utilizar para testar requisições HTTP. Você pode acessar o repositório no github para saber mais sobre ela e efetuar a instalação pela aba de extensões no vscode.

Logo, ao criar a **rota /cars**, vamos pegar a informação que mandaremos pela requisição (no caso o nome do carro) e atribuir este valor na **variável** `name`. Por último, vamos adicionar ao final do nosso array. Usando o **ThunderClient**, vamos então efetivamente enviar a requisição:

UnisenAI

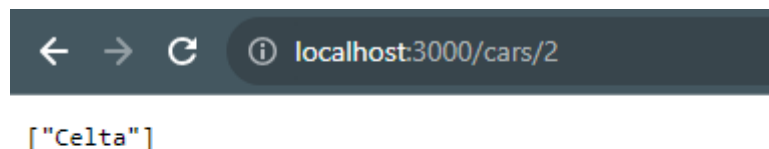
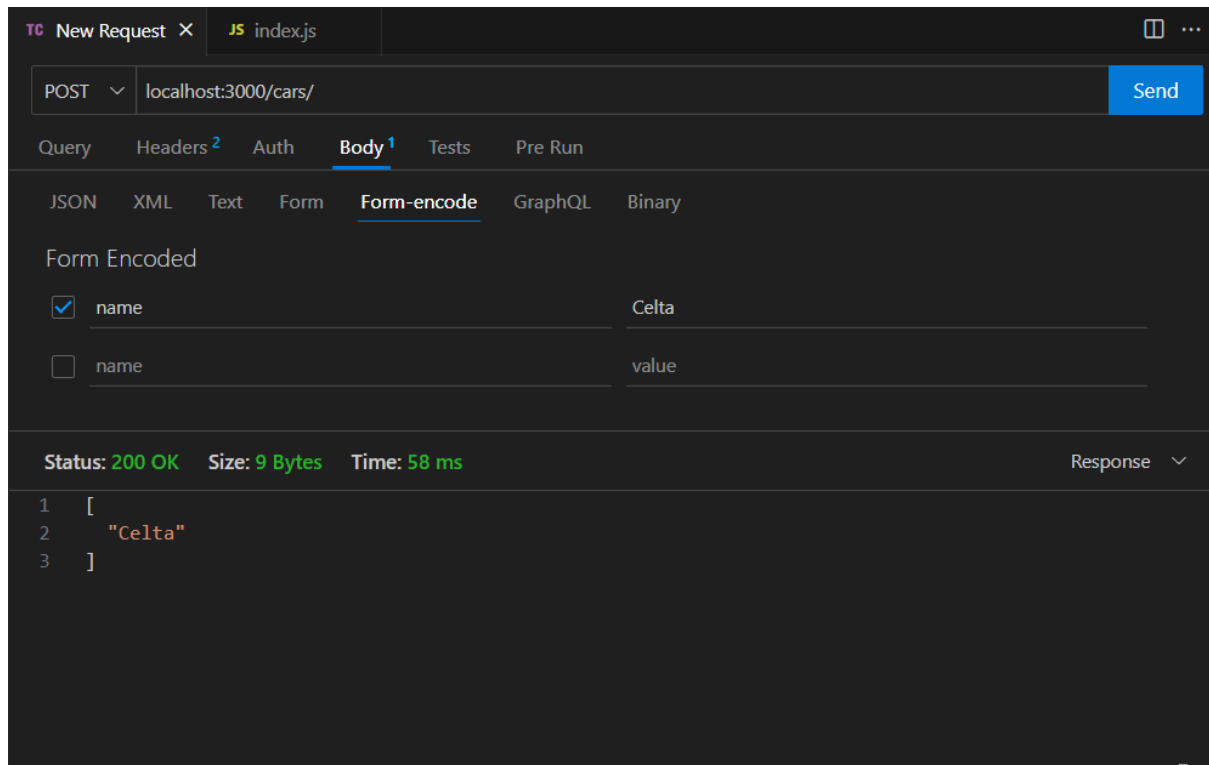


Unisenai



Para isto vamos usar a nossa **rota /cars**, selecionar como requisição post, e passar o valor “**onix**” conforme imagem acima. Observe que tivemos o retorno em **JSON**, para confirmar vamos fazer uma requisição **GET** passando o último índice do nosso array:

UniSENAI



Conforme retorno, nossa requisição post foi efetuada com sucesso. No final deste processo, nosso código estará conforme abaixo:

```
import Express from 'Express';

const app = Express();

var carros = ['fiesta', 'saveiro'];

app.use(Express.urlencoded({ extended: true }));

app.get('/', (req, res) =>
  res.send("<h3>Rotas no Express</h3><p>Rota '/'")
);

app.get('/sobre', (req, res) =>
  res.send("<h3>Rotas no Express</h3><p>Vamos aprender a utilizar Rotas com Express")
);

app.get('/users/:name', (req, res) =>{
  return res.json([name]);
});

app.post('/cars/', (req, res) => {
  let name = req.body.name;
  carros[(carros.length)] = name;
  return res.json([carros[(carros.length - 1)]]);
});

app.get('/cars/:id', (req, res) => {
  let id = req.params.id;
  return res.json([carros[id]])
});

app.listen(3000, () =>
  console.log('Servidor iniciado na porta 3000')
);
```

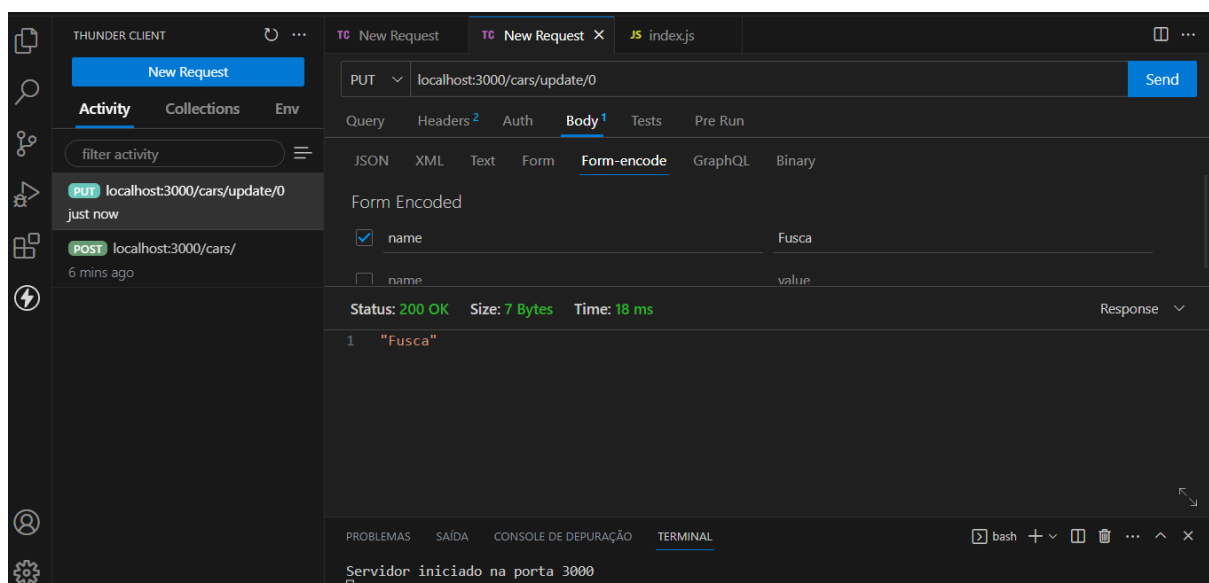
Rota com requisição PUT

Podemos criar também uma rota com express para atualizar os dados da nossa aplicação, para isso podemos utilizar a rota junto ao método `app.put()`. A requisição PUT segue o mesmo conceito da requisição POST, com a diferença que vamos atualizar uma informação. Neste exemplo vamos atualizar o nome do carro de índice 0 do nosso array já criado, `carros[]`:

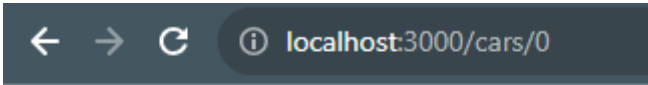
Unisenai

```
app.put('/cars/update/:id', (req, res) => {  
  let name = req.body.name;  
  carros[req.params.id] = name;  
  return res.json(carros[req.params.id]);  
});
```

Com a variável name pegamos o valor que vamos passar na requisição, localizar o índice do array com o parâmetro passado pela rota e, finalmente, atualizar os valores. Neste caso vamos atualizar o valor que está no índice 0 (fiesta), para “fusca”:



Usamos o Thunder Client para efetuar a requisição **PUT**, e logo em seguida fizemos uma requisição **GET** na rota /cars/:id para confirmar a atualização efetuada em nosso array.



["Fusca"]

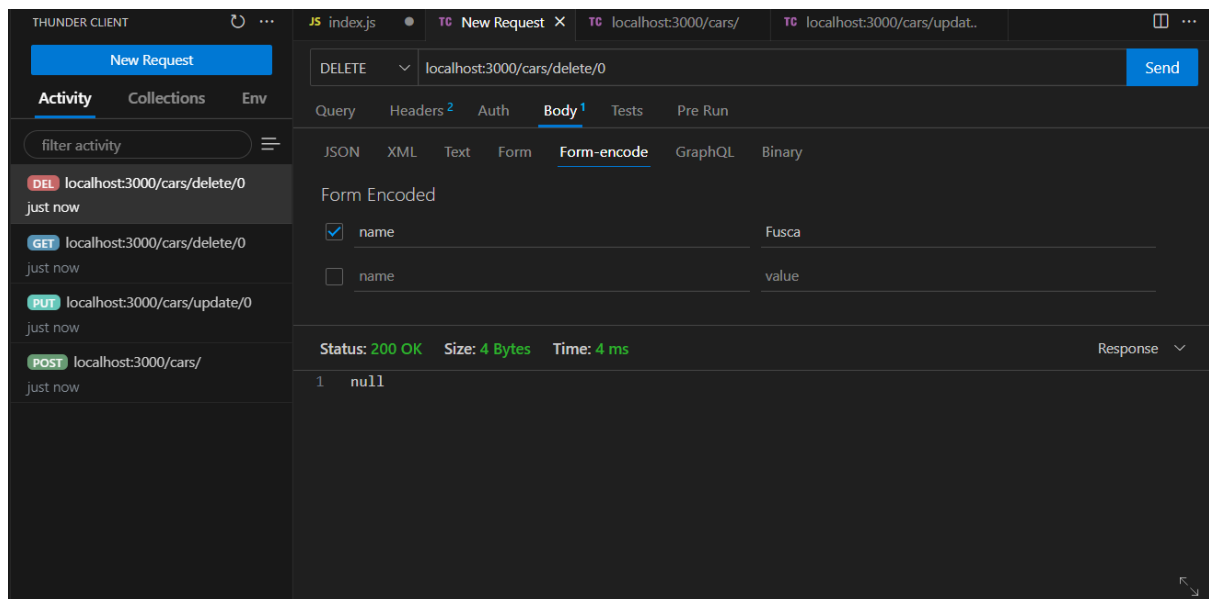
Rota com requisição DELETE

Agora, vamos criar uma rota para deletar algum dado da nossa aplicação, para isso vamos utilizar o método `app.delete()`. Vamos passar a rota `"/cars/delete/:id"`, onde o id será o índice do nosso array, desta forma:

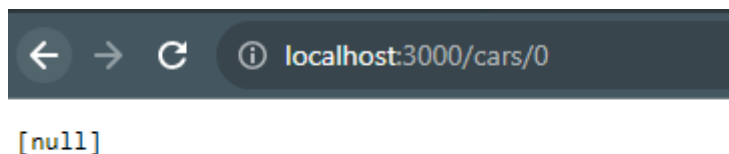
```
app.delete('/cars/delete/:id', (req, res) => {  
  let id = req.params.id;  
  carros[id] = null; //deletar item  
  return res.json(carros[id]);  
});
```

Assim, o conteúdo do nosso array de respectivo id, que foi passado por parâmetro, será null. Fizemos isso para simular a exclusão de um dado (estamos alterando para vazio), já que estamos utilizando somente um array, por exemplo, e não necessariamente acessando um banco de dados e excluindo algum item. Para testar, seguimos o mesmo exemplo da rota usando a requisição put:

UniSENAI



Por fim, acessamos a rota com a requisição get



Conclusão

Em suma, vale salientar que nós utilizamos um único arquivo com várias responsabilidades (como as rotas e o servidor), fizemos isto para fins **didáticos**, conforme a aplicação for crescendo o ideal é criarmos vários arquivos onde cada um tenha uma responsabilidade. Como resultado, aprendemos a utilizar as rotas com Express em diferentes cenários em relação às requisições **GET**, **POST**, **PUT** e **DELETE**, onde podemos utilizar para outros métodos/verbos **HTTP** possibilitando criar aplicações web e APIs de forma robusta e prática.

Atividade

Criar seu Array de lista de carros personalizados com no mínimo 6 modelos com nome, preço e marca e depois adicionar mais 3 modelos diferentes dos que já existem e realizar edição em 2 modelos e exclusão de pelo menos 2 modelos.