Javascript Node

Criando Rotas com Express.js

Vamos aprender a configurar rotas com o Express.js, com exemplos utilizando as requisições GET, POST, PUT e DELETE.

Wesley Gado · há 2 anos 6 meses

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.

Caso esse assunto seja novidade para você, aconselho a leitura de dois artigos onde explicamos estes conceitos, são eles: O que HTTP? e REST não é simplesmente retornar JSON: indo além com APIs REST

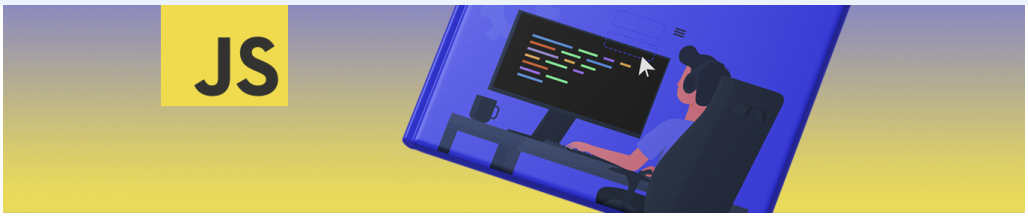
Servidor HTTP com Express

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.

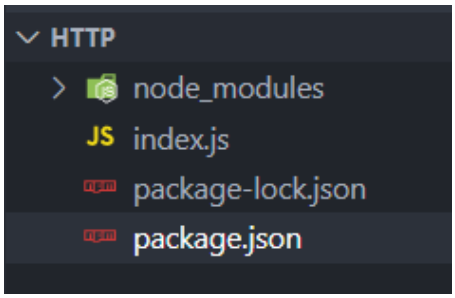
Esses passos já foram explicados em nosso artigo de como criar um servidor HTTP com Express.

Agora vamos entender o gerenciamento de rotas do Express.



Rota com Requisição GET

Vamos criar uma aplicação com a seguinte estrutura:



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

Copiar

```
import Express from 'Express';

const app = Express();

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

Agora precisamos criar a **rota** raiz, pra 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 /.**

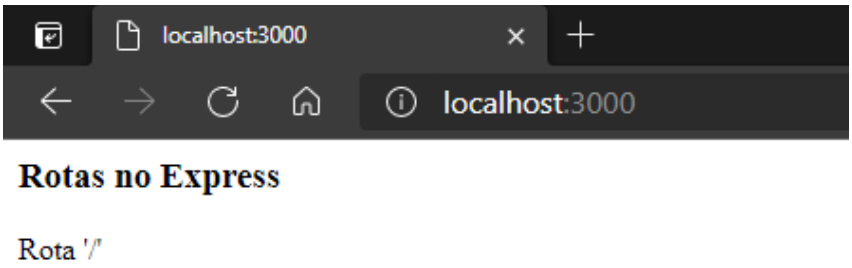
Por isso vamos começar pelo get. O Express permite o uso dos métodos (verbos) HTTP de maneira muito simples, neste caso vamos utilizar o método `app.get()`:

Copiar

```
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:



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**:

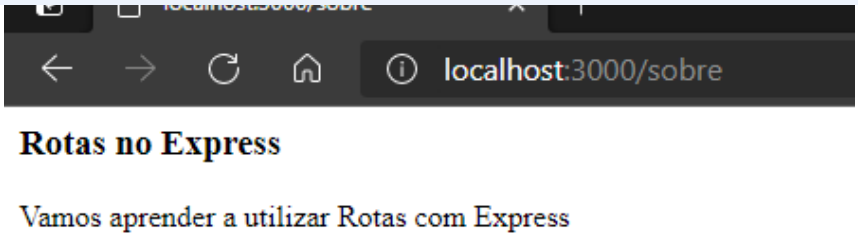
Copiar

```
import Express from 'Express';

const app = Express();

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")
);
```



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:

Copiar

```
import Express from 'Express';

const app = Express();

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

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

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

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:

Copiar

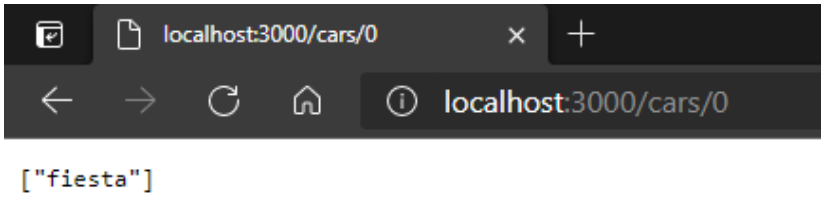
```
var carros = ['fiesta', 'saveiro'];
```

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:

Copiar

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

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



Que corresponde ao índice 0 de nosso array.

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

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

Copiar

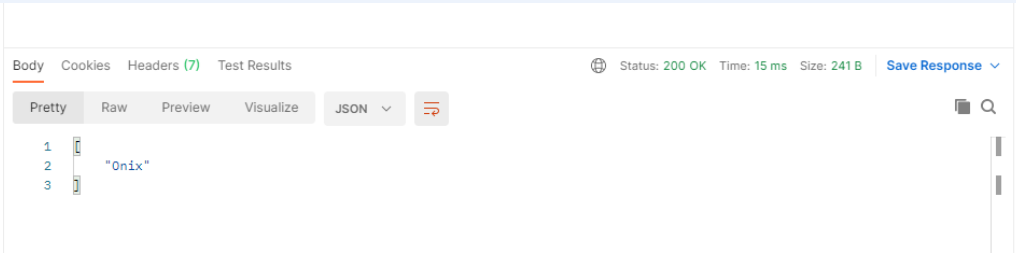
```
app.post('/cars/', (req, res) => {
    let name = req.body.name;
    carros[(carros.length)] = name;
    return res.json([carros[(carros.length - 1)]]);
});
```

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 urlenconded, e no caso o extended: true, para selecionar a biblioteca compatível com JSON. Para mais informações sobre a função `urlenconded();`, você pode estar acessando a documentação do Express.

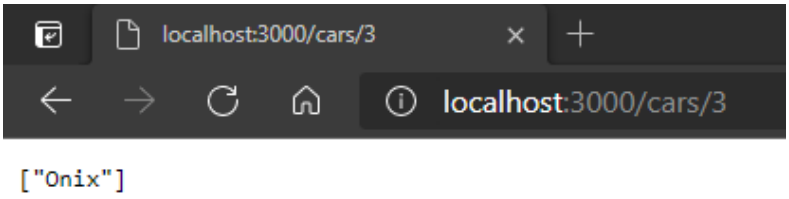
Utilizando POSTMAN

O **Postman** é uma ferramenta que podemos utilizar para testar requisições HTTP. Você pode acessar o site da ferramenta para saber mais sobre ela e efetuar o download (também existe a versão no navegador caso seja sua preferência).

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 Postman, vamos então efetivamente enviar a requisição:



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:



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

Copiar

```
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]);
```

```
    let id = req.params.id;
    return res.json([carros[id]])
});

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

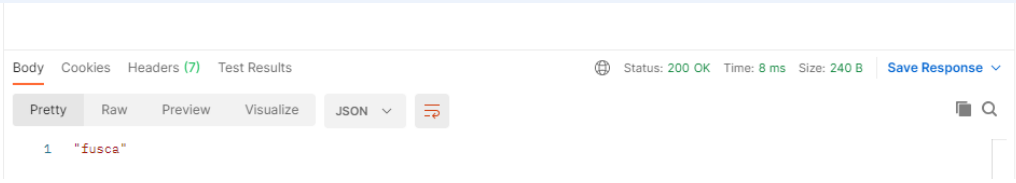
Rota com requisição PUT

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[]`:

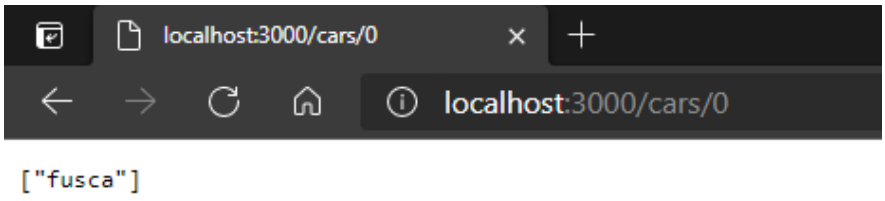
Copiar

```
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 Postman 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.



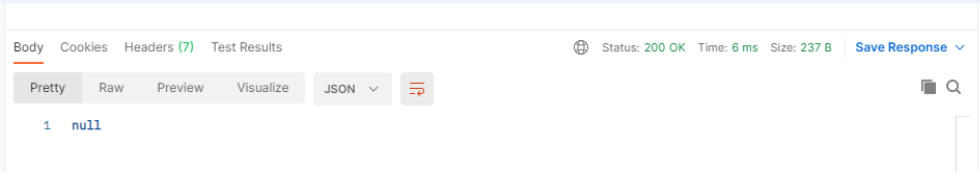
Rota com requisição DELETE

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:

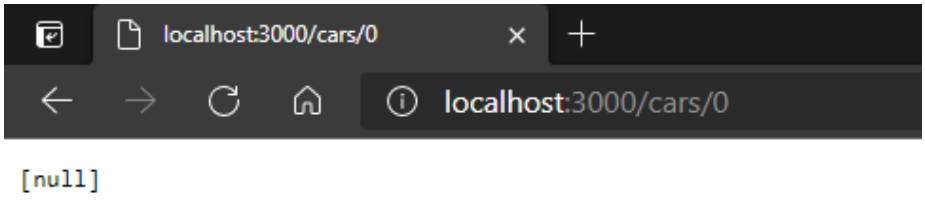
Copiar

```
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:



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.

Por fim, caso queira aprender mais sobre Express saiba que aqui na TreinaWeb temos o curso Express - Desenvolvendo aplicações web que possui **03h09 de vídeos** e um total de **18 exercícios**. Conheça também nossos outros cursos de Node.js.

- Conexão com banco de dados;
- Template Engine;
- Express Generator.

#JavaScript #Express.js #Node.js

Autor(a) do artigo

Wesley Gado

Formado em Análise e Desenvolvimento de Sistemas pelo Instituto Federal de São Paulo, atuou em projetos como desenvolvedor Front-End. Nas horas vagas grava Podcast e arrisca uns três acordes no violão.

Todos os artigos

Artigos relacionados

Ver todos →

aplicação com banco de dados MySQL usando Sequelize ORM com Node.js...

utilizado em conjunto com o Node.js. Ele possui características que facilit...

vamos fazer um compilado de tópicos úteis sobre o framework, desde cr...

Desenvolvimento Back-end

Como realizar upload de arquivos locais no NestJS

Neste artigo vamos aprender a utilizar o recurso de upload de arquivos locais com o NestJS utilizand...

Desenvolvimento Back-end

Criando o primeiro CRUD com NestJS

Neste artigo veremos como criar um CRUD com NestJS, o passo a passo utilizando ferramentas como o Ty...

Node

Trabalhando com Prisma ORM e NestJS

Neste artigo vamos aprender os primeiros passos para trabalhar com o Prisma ORM e o NestJS.

handlebars

Neste artigo vamos aprender a configurar o NestJS com o handlebars, uma template engine que permite...

transformer

Vamos abordar a utilização de duas biblitecas de serialização e validação fundamentais no NestJS: cl...

Nodemailer

Neste artigo vamos aprender como enviar emails utilizando o NestJS e o Nodemailer, um dos módulos ma...

Javascript

Express com Template Engine PUG

Neste artigo vamos estruturar um projeto utilizando Express com o Template Engine PUG e entender as...

Node

Utilizando template engine EJS com Node.js

Neste artigo vamos aprender a utilizar a Template Engine EJS com Node.js e também alguns recursos in...

Javascript

Conhecendo o MEAN Stack

Conheça neste artigo o que é o MEAN Stack.