

Aula 2 - Configurando rotas

Docupedia Export

Author:Ferro Alisson (CtP/ETS)

Date:15-Aug-2023 16:46

Table of Contents

1 Verbos HTTP	3
1.1 GET	3
1.2 POST	3
1.3 DELETE	3
1.4 PUT	3
1.5 PATCH	3
2 Requisições POST	4
3 Status code	7
4 Requisições GET	8
4.1 Desafio: crie uma API para carros, crie os métodos get, post, get por ID, put e delete	8

1 Verbos HTTP

Os verbos HTTP fazem parte da nossa requisição, visto que um endpoint pode atribuir mais de um verbo, por exemplo, um *endpoint* pode atender requisições *get* e *post*.

1.1 GET

Pode passa ou não um ID, serve para buscar dados

1.2 POST

Normalmente usado sem passagem de parâmetro, passa informações via *body*

1.3 DELETE

Usado para remover um dado, passado id como parâmetro

1.4 PUT

Usado para atualizar um dado, atualiza totalmente os dados, ou seja, perde os dados que não forem passados, passando id como parâmetro e as informações via *body*

1.5 PATCH

Usado para editar um dado, mas só atualiza os dados que forem passados, não perdendo as outras informações, passando id como parâmetro e as informações via *body*.

Como iniciado na aula passada, vamos começar a criar rotas para nossa API.

2 Requisições *POST*

A primeira requisição que iremos fazer será com *POST* para começar a preencher nossos dados, para isso no arquivo *routes/person* e vamos criar as próximas rotas. Vamos primeiro ver o que recebemos na requisição.

```
const express = require('express');
const router = express.Router();

const people = [{}];

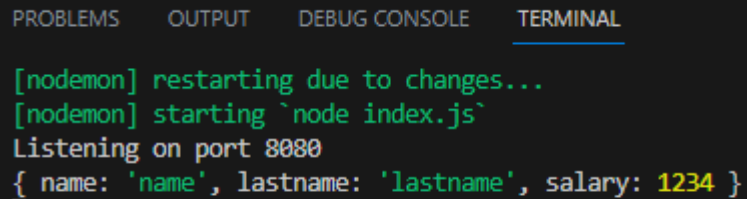
router
  .post('/api/person', (req, res) => {
    console.log(req.body);
    return;
  })

module.exports = router;
```

E no Postman vamos colocar nosso endpoint e o JSON para enviar as informações

```
{
  "name": "name",
  "lastname": "lastname",
  "salary": 1234
}
```

Como na API colocamos o `console.log`, então irá ser impresso uma mensagem semelhante à seguir



```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

[nodemon] restarting due to changes...
[nodemon] starting `node index.js`
Listening on port 8080
{ name: 'name', lastname: 'lastname', salary: 1234 }
```

Mas ainda não está salvando os dados, mesmo que sem persistir os dados, para isso vamos criar uma lista para armazenar as informações e adicionar com o método *push*.

```
const express = require('express');
const router = express.Router();

const people = [];

router
  .post('/api/person', (req, res) => {
    const { name, lastname, salary } = req.body;
    const person = {
      id: people.length,
      name: name,
      lastname: lastname,
      salary: salary
    }

    people.push(person);
    return res.status(201).send({ message: "Pessoa inserida com sucesso" });
  })

module.exports = router;
```

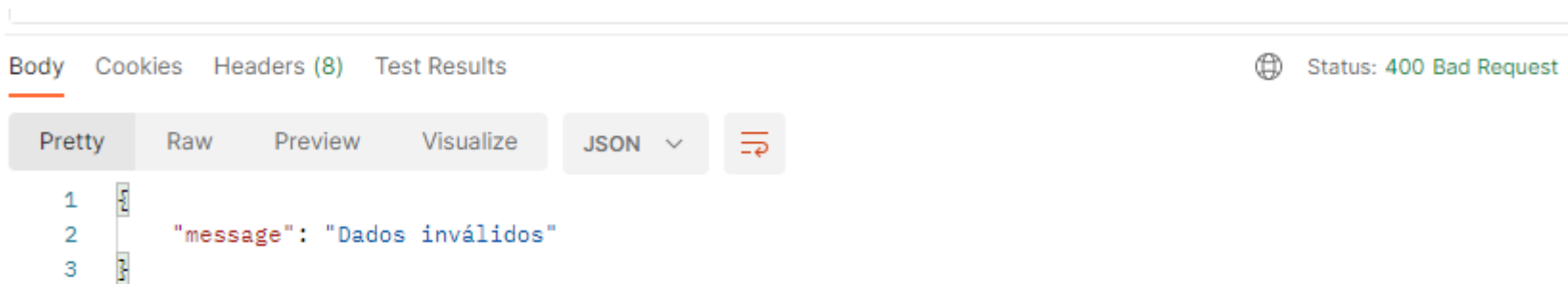
No Postman uma mensagem será exibida



Mas dessa forma ainda não está muito boa, visto que se enviarmos um nome nulo, a API esta aceitando e inserindo, para isso vamos adicionar uma verificação no nosso método.

```
if(!name || !lastname || !salary)
    return res.status(400).send({ message: "Dados inválidos" })
```

No Postman vamos excluir um ou mais dados para verificarmos se está atendendo



Observe que agora não foi inserido pois estavam faltando dados.

3 Status code

Vimos alguns status ao enviar as requisições, mas o que são esses status?

São códigos de resposta HTTP e indicam o status de uma requisição HTTP e são agrupados em

[Informational responses](#) (100 – 199)

[Successful responses](#) (200 – 299)

[Redirection messages](#) (300 – 399)

[Client error responses](#) (400 – 499)

[Server error responses](#) (500 – 599)

Podemos ver mais na [documentação](#).

4 Requisições GET

Para nossas requisições get, é mais simples basta enviar a lista com o *return*.

```
router
  .get('/api/person', (req, res) => {
    return res.status(200).send({ data: poeople });
  })
```

Assim, temos agora dois métodos, um *GET* e um *POST*.

4.1 Desafio: crie uma API para carros, crie os métodos get, post, get por ID, put e delete