

## Módulo 09 – Atividade 09

### Exercícios:

1. Em node.js, criar uma API seguindo os princípios RESTful e todos os conceitos vistos anteriormente:

- Os dados devem ser armazenados num arquivo JSON.
- Deve ter as operações necessárias para que atenda os requisitos do modelo.
- Deve ser utilizado módulos.
- Deve devolver códigos de respostas.

### Exemplos:

- Através de inputs o usuário deve cadastrar seu nome e email (POST /usuarios).
  - Cada item cadastrado deve aparecer em sequência numa lista (GET /usuarios).
  - Será possível editar o nome e o e-mail. (UPDATE /usuarios/{id-usuario})
  - Dar a opção de exclusão do registro cadastrado. (DELETE /usuarios/{id-usuario})
2. Não é preciso criar interface, os testes devem ser realizados de acordo com os próximos exercícios.
3. Utilizando o curl realize o teste da api passando por todas as operações.
4. Faça o download e instale o Postman.
5. Utilizando o Postman realize o teste da api, passando por todas as operações.
6. Utilizando uma das ferramentas de testes, simule o cadastro de um usuário e verifique na aba redes (network) do devtools as informações obtidas com o procedimento.
7. Idem ao item 6 (seis), tente excluir um usuário de id inexistente e verifique o que acontece pelo Devtools.

Q1) Comecei a api abrindo o terminal na raiz do projeto, e utilizando “npm init -y”.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_oficial$ npm init -y
Wrote to /home/letonio/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_oficial/package.json:

{
  "name": "ativ09_api_oficial",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

O comando acima cria um package.json que conterá as informações do nosso projeto, além de scripts e dependências necessárias para o projeto

funcionar. Há ainda algumas dependências que são instaladas como devDependencies, que são dependências usadas apenas durante o desenvolvimento do projeto, não são necessárias quando a aplicação já está em produção.

Utilizaremos o Express, portanto, faz-se necessária a sua instalação, através desse comando: “npm install express”

```
Hard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_oficial$ npm install express
added 57 packages, and audited 58 packages in 6s
7 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
```

Nota-se a presença de um diretório chamado node\_modules, que conterá módulos necessários para o projeto.

Outro módulo interessante é o nodemon, que permitirá automatizar o processo de derrubar e levantar o servidor sempre que fizermos alguma alteração: “npm install -D nodemon”. A flag “-D” implica numa dependência de desenvolvimento.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_oficial$ npm install -D nodemon
added 32 packages, and audited 90 packages in 2s
10 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities
```

No arquivo “package.json”, se faz necessário adicionar “start”: “nodemon server.js” dentro da chave “scripts”, para que o processo fique automatizado:

```
6   "scripts": {
7     "start": "nodemon server.js",
8     "test": "echo \"Error: no test specified\" && exit 1"
9   },
10  "keywords": [],
```

Basta ir no terminal e digitar npm start para inicializar o server, e sempre que fizermos alguma mudança, ele reiniciará automaticamente.

Precisamos criar o arquivo do nosso servidor, o server.js.

Além disso, utilizaremos a forma de importação do ESM, portanto, precisamos adicionar no “package.json” uma chave:valor com “type”: “module”. Caso não fizéssemos isso, precisaríamos fazer importação, utilizando o “require” (padrão common JS).

A imagem abaixo mostra como ficou o conteúdo inicial do servidor “server.js”:

```
server.js > app.listen() callback
1  import express from "express";
2
3  const app = express();
4
5  const port = "8080";
6  //EXECUTANDO APLICAÇÃO NA PORTA DEFINIDA
7  app.listen(port, () => {
8    console.log(`Server running on port ${port}`);
9  });
10
```

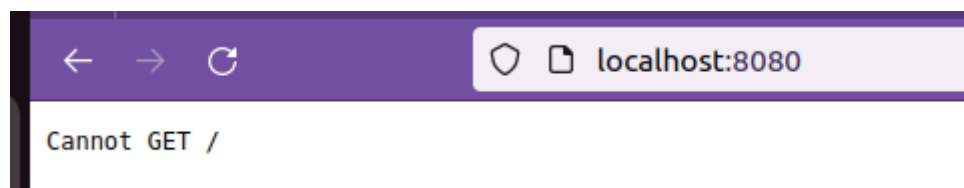
Vamos levantar o servidor, indo no terminal e usando “npm start”:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_oficial$ npm sta
rt

> ativ09_api_oficial@1.0.0 start
> nodemon server.js

[nodemon] 2.0.20
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node server.js`
Server running on port 8080
█
```

Pronto, nosso servidor está ativo, indo no navegador e acessando “<http://localhost:8080>”, temos o seguinte resultado:



Estamos ouvindo na porta 8080, porém não tem nenhuma função que seja chamada quando o navegador faz uma requisição pelo método GET. Portanto, precisamos adicionar um “app.get()” e indicar o que vai acontecer quando alguém acessa nosso servidor e manda uma requisição pelo método GET.

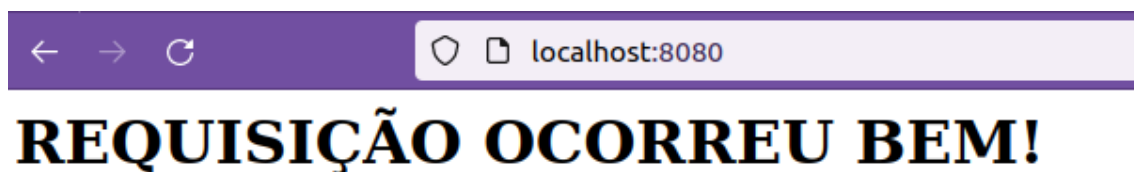
Para um teste inicial, o arquivo do servidor ficou assim:

```

1  import express from "express";
2
3  const app = express();
4
5  const port = "8080";
6  //EXECUTANDO APLICAÇÃO NA PORTA DEFINIDA
7  app.get("/", (req, res) => {
8      res.send(`<h1>REQUISIÇÃO OCORREU BEM!</h1>`);
9      res.status(200);
10 });
11
12 app.listen(port, () => {
13     console.log(`Server running on port ${port}`);
14 });
15
16

```

Recarregando a página no navegador, o resultado é o seguinte:



Está funcionando.

Agora, vamos procurar adicionar `app.post`, `app.put`, `app.delete`. Além disso, vamos modificar a rota para já ir se adaptando aos requisitos dessa atividade, pois pede-se que acessemos `"/usuarios"`.

Consegui adicionar cada método! Ficou simples e, por enquanto, não está modularizado.

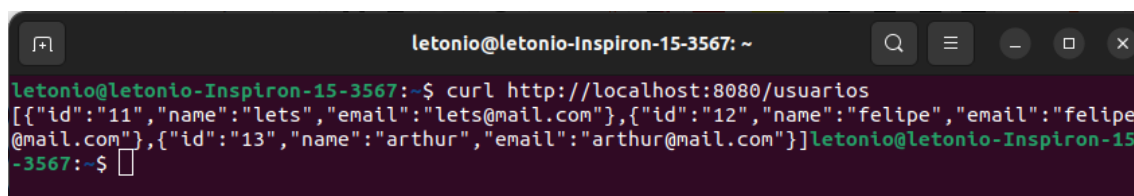
-----

Q1) Fiz a API REQUERIDA.

Q2) Essa questão não pede nada.

Q3) Usando o CURL para fazer o teste de cada método, temos que, para GET, basta fazer:

curl <http://localhost:8080/usuarios>

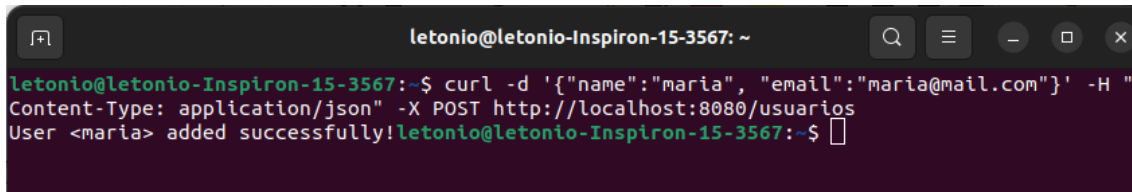


Nota-se que temos três usuários na resposta enviada pelo servidor.

O próximo método testado será o método POST, escrevendo a seguinte linha de código:

```
curl -d '{"name":"maria", "email":"maria@mail.com"}' -H "Content-Type: application/json" -X POST http://localhost:8080/usuarios
```

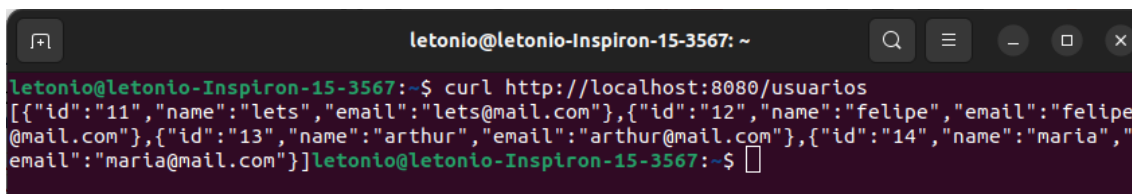
O resultado foi o seguinte:



```
letonio@letonio-Inspiron-15-3567: ~  
letonio@letonio-Inspiron-15-3567:~$ curl -d '{"name":"maria", "email":"maria@mail.com"}' -H "Content-Type: application/json" -X POST http://localhost:8080/usuarios  
User <maria> added successfully!letonio@letonio-Inspiron-15-3567:~$
```

É possível confirmar que a mudança ocorreu de fato, através desse comando:

curl <http://localhost:8080/usuarios>

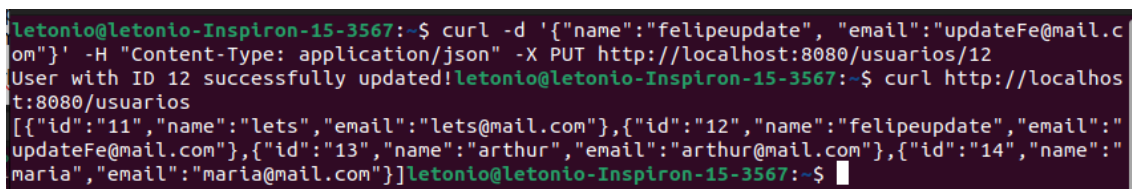


```
letonio@letonio-Inspiron-15-3567:~$ curl http://localhost:8080/usuarios  
[{"id":"11","name":"lets","email":"lets@mail.com"}, {"id":"12","name":"felipe","email":"felipe@mail.com"}, {"id":"13","name":"arthur","email":"arthur@mail.com"}, {"id":"14","name":"maria","email":"maria@mail.com"}]letonio@letonio-Inspiron-15-3567:~$
```

O próximo é o método PUT, para fazer a atualização de um elemento, para isso precisamos passar no body as novas informações e na URL informar o ID do elemento requerido. Faremos mudanças no usuário com índice "12".

O comando empregado foi:

```
curl -d '{"name":"felipeupdate", "email":"updateFe@mail.com"}' -H "Content-Type: application/json" -X PUT http://localhost:8080/usuarios/12
```



```
letonio@letonio-Inspiron-15-3567:~$ curl -d '{"name":"felipeupdate", "email":"updateFe@mail.com"}' -H "Content-Type: application/json" -X PUT http://localhost:8080/usuarios/12  
User with ID 12 successfully updated!letonio@letonio-Inspiron-15-3567:~$ curl http://localhost:8080/usuarios  
[{"id":"11","name":"lets","email":"lets@mail.com"}, {"id":"12","name":"felipeupdate","email":"updateFe@mail.com"}, {"id":"13","name":"arthur","email":"arthur@mail.com"}, {"id":"14","name":"maria","email":"maria@mail.com"}]letonio@letonio-Inspiron-15-3567:~$
```

Aproveitei para verificar se a alteração havia ocorrido.

Para finalizar, utilizarei o método DELETE. Nesse método, não é necessário enviar conteúdo pelo body da requisição, basta informar o ID do elemento que deverá ser excluído.

Optou-se por deletar o elemento que foi alterado anteriormente. Portanto, o comando executado foi esse:

```
curl -X DELETE http://localhost:8080/usuarios/12
```

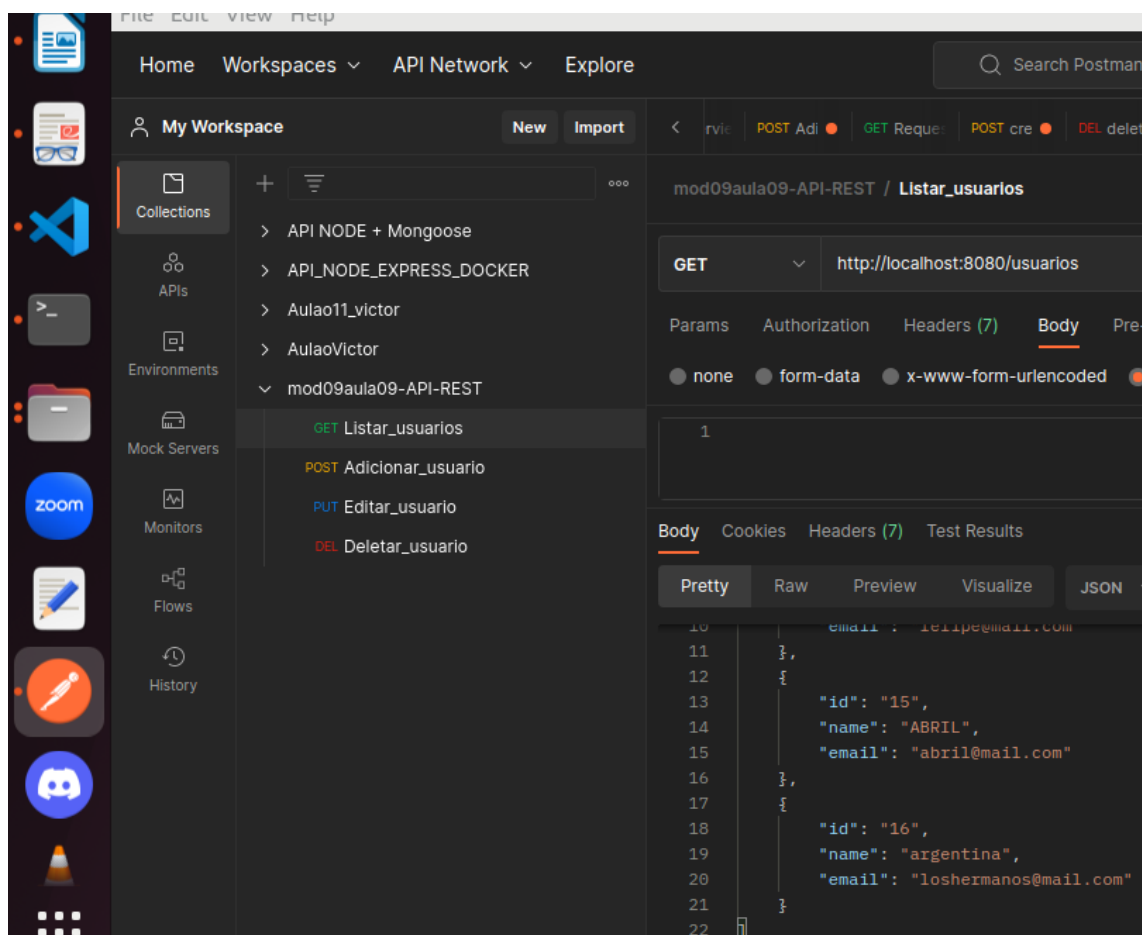
```
letonio@letonio-Inspiron-15-3567: ~  
letonio@letonio-Inspiron-15-3567:~$ curl -X DELETE http://localhost:8080/usuarios/12  
User with ID 12 deleted successfully!letonio@letonio-Inspiron-15-3567:~$ curl -X DELETE ht
```

Para confirmar que foi deletado, verifiquei com o GET

```
letonio@letonio-Inspiron-15-3567:~$ curl -X GET http://localhost:8080/usuarios  
[{"id":"11","name":"lets","email":"lets@mail.com"}, {"id":"13","name":"arthur","email":"arthur@mail.com"}, {"id":"14","name":"maria","email":"maria@mail.com"}]letonio@letonio-Inspiron-15-3567:~$
```

Nota-se que o usuário com ID=12 não está mais presente na lista.

Q4) Instalei o postman:



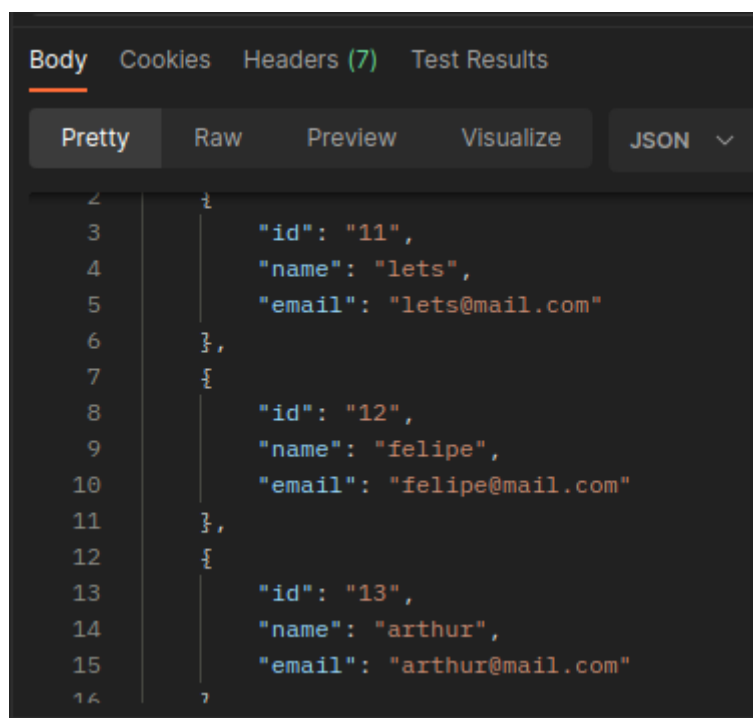
Q5) Primeiro teste é o método GET, lembrando que a URL é

"http://localhost:8080/usuarios"

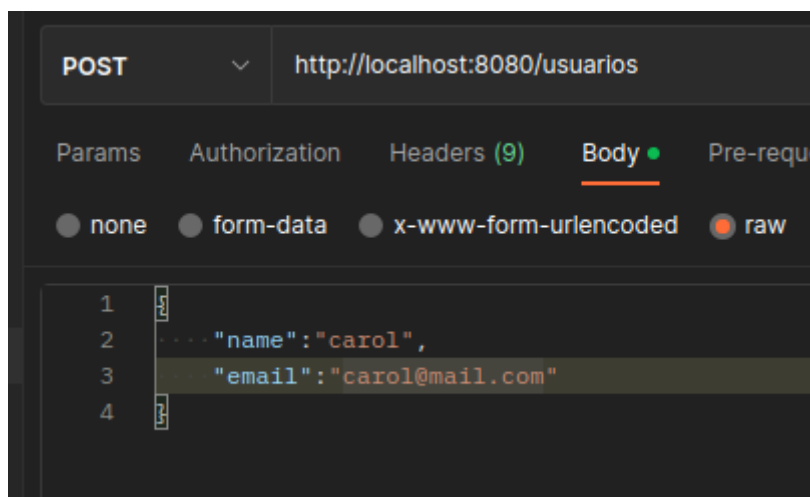
O arquivo inicialmente tem 3 elementos dentro. A imagem a seguir ilustra o resultado:



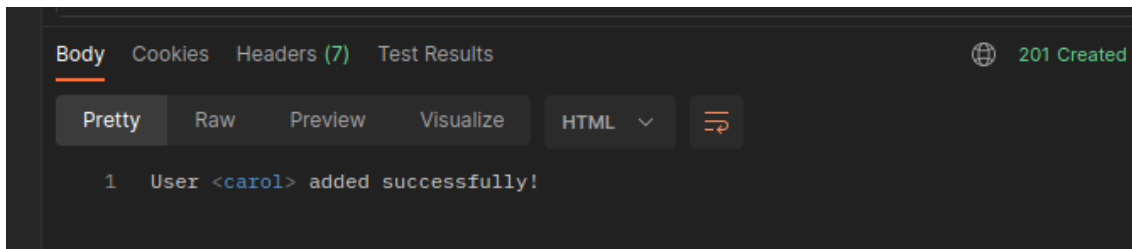
Nota-se que o status foi 200, ou seja, está tudo okay. A próxima imagem deixa num formato mais fácil de identificar os elementos:



Testando o método POST, enviando no corpo da requisição o seguinte conteúdo:

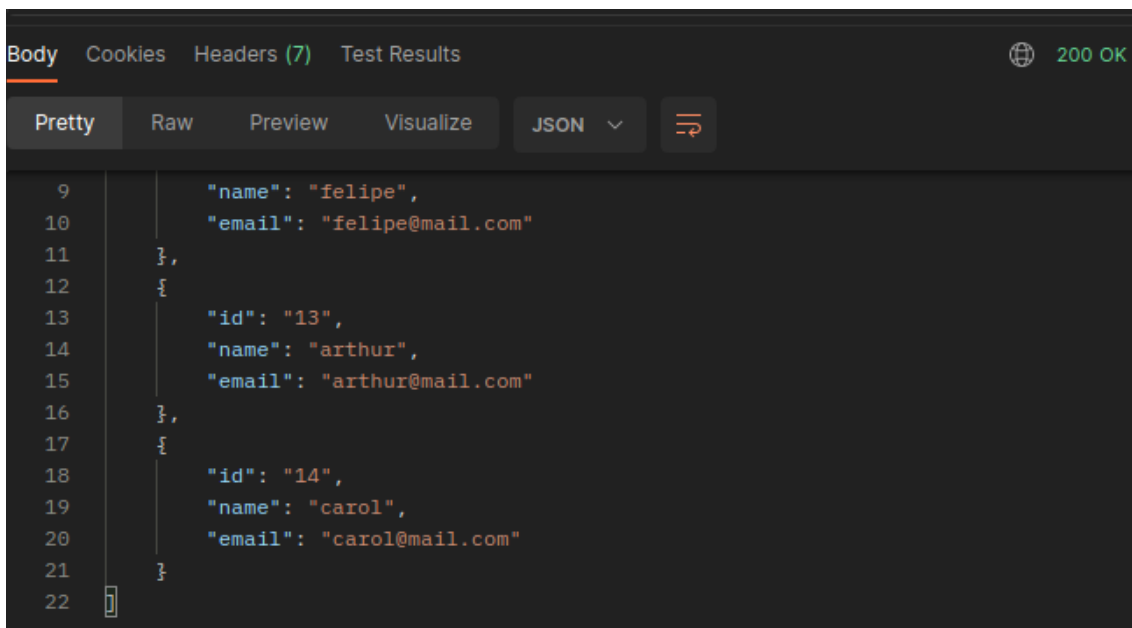


Ao enviar o resultado é o seguinte:



Nota-se o status 201, isso foi enviado pelo código, é interessante usar esse status porque indica que houve criação de recurso.

Para confirmar que houve a adição de um usuário, usaremos o método GET mais uma vez:

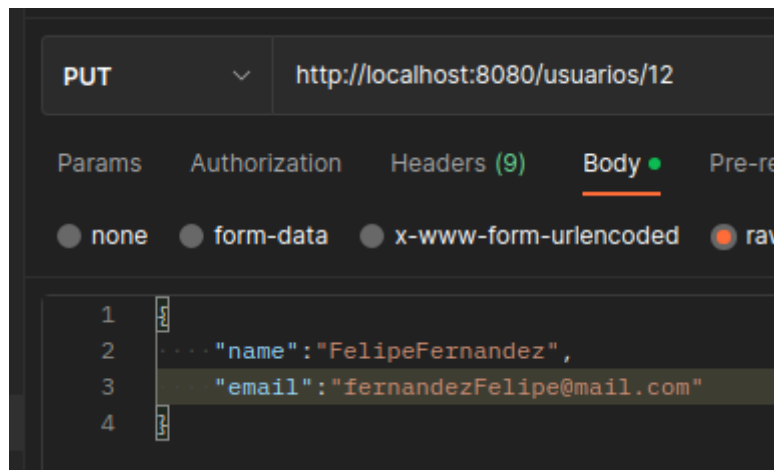


Nota-se que Carol foi adicionada ao final do arquivo.

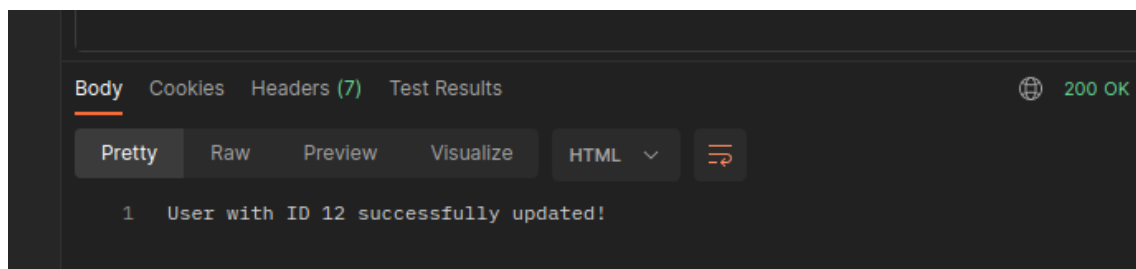
O próximo método será o PUT, onde faremos a alteração do nome e e-mail do usuário felipe, cuja id é "12". Obtei por utilizar string porque a ideia é posteriormente, utilizar um módulo que gere ID's únicos pseudo-aleatórios.

A URL será "<http://localhost:8080/usuarios/12>", com o método PUT. A imagem a seguir mostra como ficou a requisição via POSTMAN:

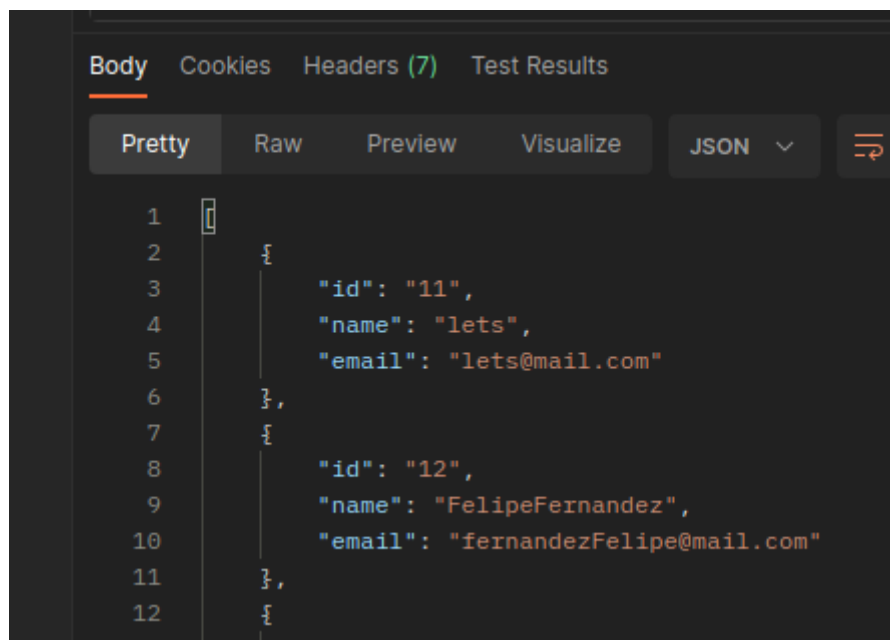




A resposta obtida tem status 200:



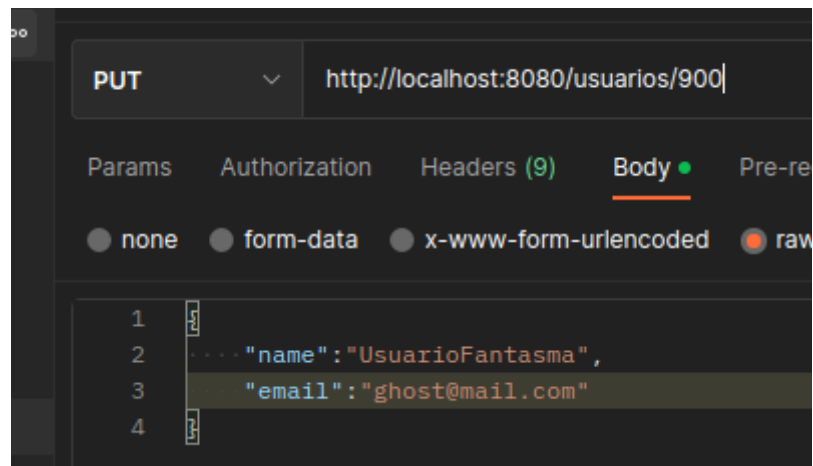
Para confirmar que a alteração aconteceu, usamos o método GET novamente.



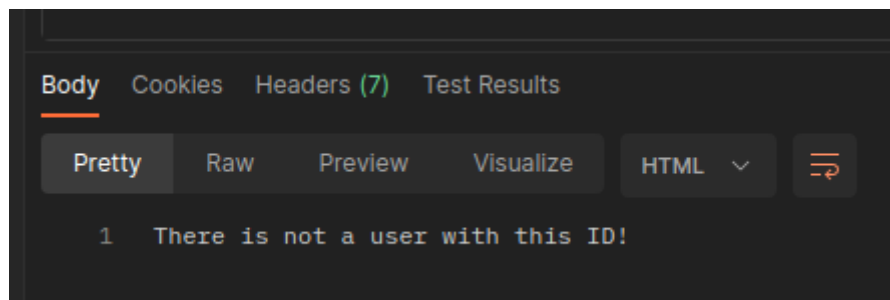
O elemento com ID 12 foi alterado corretamente.

Vamos tentar o método PUT passando um identificador que não existe,

“<http://localhost:8080/usuarios/900>”, conforme ilustrado a seguir:



Ao fazer a requisição, o resultado é o seguinte:

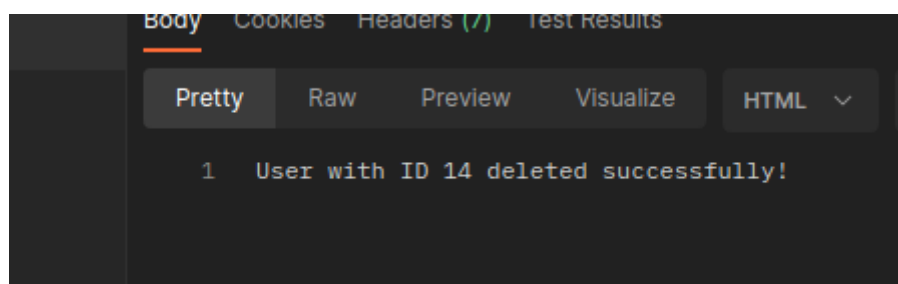


A resposta é uma mensagem de que não existe um usuário com essa ID.

Por fim, o método DELETE, que não precisa de um body, basta o ID do elemento que desejamos deletar.

Apagarei o elemento com id 14, portanto, o caminho completo precisa ser:

<http://localhost:8080/usuarios/14>:



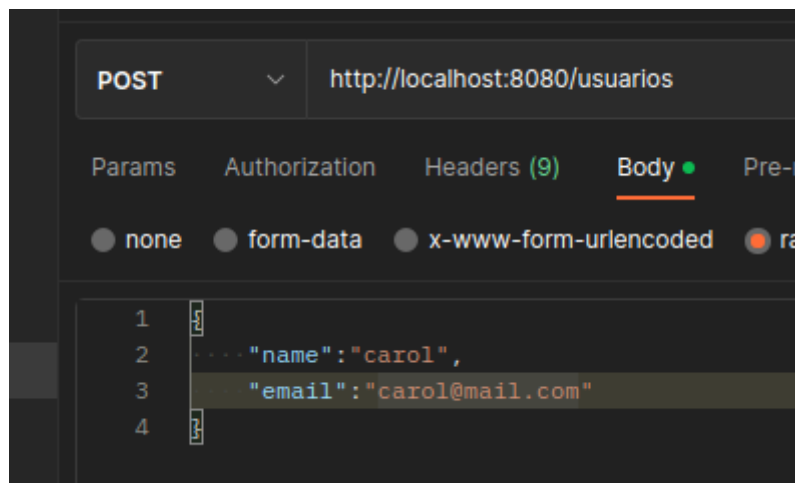
Elemento apagado com sucesso, apenas para confirmar usamos o GET mais uma vez. Nota-se que o último usuário tem ID=13.

```

6      },
7      {
8          "id": "12",
9          "name": "FelipeFernandez",
10         "email": "ferandezFelipe@mail.com"
11     },
12     {
13         "id": "13",
14         "name": "arthur",
15         "email": "arthur@mail.com"
16     }
17 ]

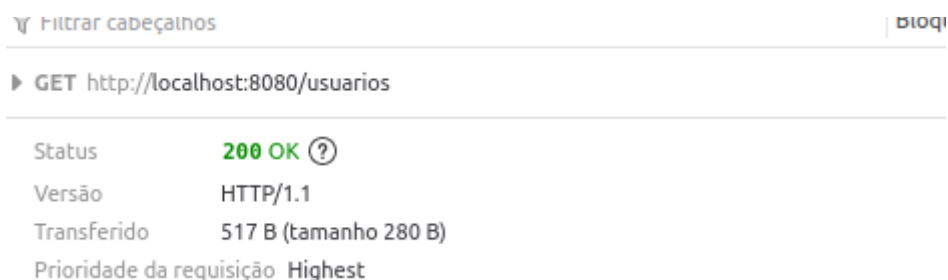
```

Q6) Vamos usar o método POST, através do Postman, para adicionar um usuário, enquanto observamos a aba de rede (network) na DEVTOOLS.



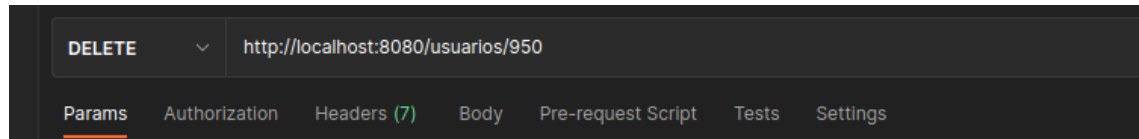
A página mostra o conteúdo que foi retornado na resposta.

Ao lado direito do devtools, nota-se:

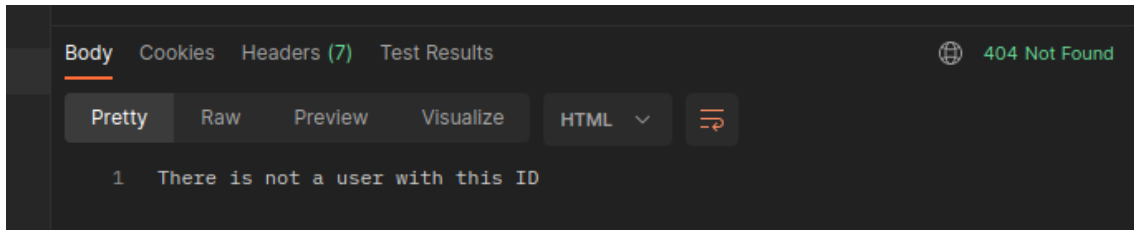


Q7) Ainda usando o POSTMAN, vamos tentar deletar um usuário com ID inexistente, por exemplo, ID=950. O caminho completo que deve ser adicionado no Postman será:

DELETE <http://localhost:8080/usuarios/950>



Coloquei para ele retornar status de 404 – Not found.



## DEIXANDO REGISTRADO O MÓDULO DE MANIPULAÇÃO “FS”

Manipulação de arquivos JSON como banco de dados. Para isso, utilizaremos o Módulo FS (file system module), que é um módulo build-in do Node, portanto, não é necessário fazer a instalação usando o npm.

A seguir colocarei alguns prints do código, mostrando como ficou o código fazendo manipulação de arquivo no “controllers.js”:

```
1 import fs from "fs";
2
3 //GET MAIN PAGE
4 export const getMainPage = (req, res) => {
5   res.send(`<h1>Home page</h1>
6   <h2>Use /usuarios to access our user's list</h2>
7   <h2>To add, edit/update or delete users, use POSTMAN</h2>`);
8   res.status(200);
9 };
```

```
11 //USE GET METHOD
12 export const getAllUsers = (req, res) => {
13   fs.readFile("./database/users.json", (err, data) => {
14     if (err) {
15       res.status(500); //there is a error
16       res.json({
17         status: "Não foi possível acessar os dados do servidor",
18       });
19     }
20     const allUsers = JSON.parse(data);
21
22     res.send(allUsers);
23     res.status(200);
24   });
25 };
```

```

47 //USE PUT METHOD
48 export const updateUser = (req, res) => {
49     const id = req.params.id; //GET ID FROM URL
50
51     let rawData = fs.readFileSync("./database/users.json");
52     let userList = JSON.parse(rawData);
53
54     const findIndex = verifyIdInDataBase(id, userList);
55
56     if (findIndex !== -1) {
57         userList[findIndex].name = req.body.name;
58         userList[findIndex].email = req.body.email;
59
60         fs.writeFileSync("./database/users.json", JSON.stringify(userList));
61         res.status(200);
62         res.send(`User with ID ${req.params.id} successfully updated!`);
63     } else {
64         res.send(`There is not a user with this ID!`);
65     }
66 };

```

```

88
89 function verifyIdInDataBase(id, database) {
90     let findIndex = -1;
91     for (let i = 0; i < database.length && findIndex === -1; i++) {
92         if (database[i].id === id) {
93             findIndex = i;
94         }
95     }
96     return findIndex;
97 }

```

```

98
99 function generateId(database) {
100     let lastIndex = database.length - 1;
101     const maxId = parseInt(database[lastIndex].id);
102     const newId = (maxId + 1).toString();
103     return newId;
104 }

```

```

27 //USE POST METHOD
28 export const createUser = (req, res) => {
29   let rawData = fs.readFileSync("./database/users.json");
30   let userList = JSON.parse(rawData);
31
32   const newId = generateId(userList);
33   const newUser = {
34     id: newId,
35     ...req.body, //create a copy of body content request
36   };
37   //counter++; //increase counter that is used to create unique ID's
38
39   userList.push(newUser);
40
41   fs.writeFileSync("./database/users.json", JSON.stringify(userList));
42
43   res.json({ status: "User added successfully" });
44   res.status(201); //created
45 };

```

```

67
68 //USE DELETE METHOD
69 export const deleteUser = (req, res) => {
70   const id = req.params.id; //ID PASSED BY URL TO BE DELETED
71
72   let rawData = fs.readFileSync("./database/users.json");
73   let userList = JSON.parse(rawData);
74
75   const findIndex = verifyIdInDataBase(id, userList);
76
77   if (findIndex !== -1) {
78     userList.splice(findIndex, 1); //Go to index and remove 1 element from this
79
80     fs.writeFileSync("./database/users.json", JSON.stringify(userList));
81     res.status(200);
82     res.send(`User with ID ${req.params.id} deleted successfully!`);
83   } else {
84     res.status(404); //maybe erase this line
85     res.send(`There is not a user with this ID`);
86   }
87 };

```

Por fim, ilustra-se o arquivo original json “users.js”, utilizado como banco de dados.

```
1  [
2    {
3      "id": "10",
4      "name": "lets",
5      "email": "lets@mail.com"
6    },
7    {
8      "id": "11",
9      "name": "felipe",
10     "email": "felipe@mail.com"
11   },
12   {
13     "id": "12",
14     "name": "arthur",
15     "email": "arthur@mail.com"
16   },
17   {
18     "id": "13",
19     "name": "joao",
20     "email": "joao@mail.com"
21   }
22 ]
```

Para a aula 10, usei essa versão de gerador de ID, porém, fiz outra versão, que utiliza o módulo UUID (Universal Unique Identifier). Basta instalar o módulo, utilizando o comando “npm install uuid”

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorio
Hard/Alpha-edtech-cycle01/Module09-servidores/Aula09/Ativ09_API_v4_JSON_IDfromUU
ID$ npm install uuid

added 1 package, and audited 91 packages in 5s

10 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

Ao importar, caso estejamos usando commonJS, adiciona-se essa linha no código:

```
const {v4 : uuidv4} = require("uuid");
```

Caso estejamos usando ESM(EcmaScript Module), então escreve-se:

```
import { v4 as uuidv4 } from "uuid";
```

Por exemplo, para obter um ID único, usa-se

```
const newId = uuidv4();
```