Centro Universitário SENAC SP – Santo Amaro

21 de outubro de 2022

API REST

(ADO 03)

Autor: Matheus Ferreira Santos.

Professor: Carlos Verissimo.

Desenvolvimento

Primeira parte:

```
1   const express = require('express');
2   const server = express();
3   server.use(express.json());
```

```
73  server.listen(3000);
```

```
5   const usuarios = [];
6   const senhas = [];
```

(Linhas 1 a 3): Para dar início ao projeto passado (simulando um CRUD com vetores) eu defino algumas variáveis em que vou utilizar ao longo do projeto todo, a constante "express" que receberá o método "require('express')" como valor inicial padrão, defino também a constante "server" que receberá o método "express()" e também aviso para o programa, através do comando "server.use(express.json())", que estarei, em alguma parte do programa, enviando um JSON (onde basicamente servirá para fazer funcionar, posteriormente, as minhas Body Params Requests) por valor inicial padrão, os três serão fundamentais para o funcionamento a minha API.

Observações: Express é o que permitirá usar as rotas dentro da minha API, e nos nomes das constantes eu poderia utilizar qualquer nome, dês de que eu me lembre posteriormente de usar corretamente, por exemplo: se troco o nome da constante "server" por "ABC", também deveria trocar o "server.use" para "ABC.use" e assim por diante.

(Linha 73): Informa que a constante "server" irá ser escutada no browser/navegador na porta 3000, assim quando seu servidor for executado no console, deve se passar a URL, no browser/navegador, como "localhost:3000".

(Linhas 5 e 6): Logo após definir essas "configurações" padrões, começo de fato a me preocupar com a real funcionalidade da API, que seria: verificar se existem usuários previamente cadastrados em um "banco de dados", por meio de requisições, e informar algumas mensagens, para isso, eu crio 2 constantes, "usuários" e "senhas", ambas são auto explicativas, e funcionará para guardar, posteriormente, o nome dos usuários e as suas respectivas senhas, simulando assim um banco de dados.

Desenvolvimento

Segunda parte:

```
 8  server.get('/usuarios/logins', (requisicao, resposta) => {
 9      return resposta.json(usuarios);
10  });
11
12  server.get('/usuarios/senhas', (requisicao, resposta) => {
13      return resposta.json(senhas);
14  });
15
16  server.get('/usuarios/listar', (requisicao, resposta) => {
17      const index = requisicao.query.id;
18
19      return resposta.json({
20          Nome: usuarios[index - 1],
21          Senha: senhas[index - 1]
22      });
23  });
```

(Linhas 8 a 14): Ambas funções imprime todos os usuários, ou senhas, cadastradas até o momento (tudo irá depender de qual URL for informado, se for informado: "[...]/usuarios/logins" então mostrará todos os usuarios cadastrados, no entanto se for informado: "[...]/usuarios/senhas" então mostrará todas as senhas cadastradas).

Observação: Ambos, tanto usuários, quanto senhas, serão apresentados pela ordem de inserção, e não por ordenação que geralmente segue alguma regra ortográfica.

(Linhas 16 a 23): Essa função, por sua vez, espera receber um "id" por meio de uma Query Params Request na URL, que posteriormente será passado para uma constante "index", para buscar, através do id informado, quem é o usuário e qual é a sua senha, que já foram previamente cadastradas no "banco de dados".

Observação: É possível notar que utilizei uma subtração para informar o índice do vetor, pois, provavelmente um usuário comum nunca saberia que um vetor tem por sua primeira posição o índice 0, logo ele informaria o ID 1 para buscar o primeiro cadastro de usuário/senha, porém, sabemos que voltaria os dados do segundo usuário informado, por isso utilizei essa subtração, para prevenir que o usuário informe o ID de 1, pela Query Params Request, e me volte o usuário/senha que "possuem" o ID de número 2.

Desenvolvimento

Terceira parte:

```
25  server.get('/usuarios', (requisicao, resposta) => {
26      const { verificaUsuario } = requisicao.body;
27      const { verificaSenha } = requisicao.body;
28      let usuario = false;
29      let senha = false;
30
31      usuarios.forEach((texto) => {
32          if(texto == verificaUsuario){
33              usuario = true;
34          }
35      });
36
37      senhas.forEach((texto) => {
38          if(texto == verificaSenha){
39              senha = true;
40          }
41      });
42
43      if(senha == true && usuario == true){
44          return resposta.json('Usuario liberado');
45      } else if(senha == false && usuario == false){
46          return resposta.json('Não registrado');
47      } else if (usuario == false){
48          return resposta.json('Usuario incorreto');
49      } else {
50          return resposta.json('Senha incorreta');
51      }
52  });
53
```

(Linhas 25 a 52): Esse método é basicamente a alma da API, onde receberá 2 valores por meio de um Body Params Request, um valor com um nome de usuário e outro valor com uma senha, que serão atribuídos, respectivamente, a 2 constantes: "verificaUsuario" e "verificaSenha". É possível notar também que eu crio 2 variáveis de escopo local, "usuario" e "senha", que receberão 2 valores booleanos, atribuídos inicialmente em false/falso, fazendo alusão à: os usuários informados não existem.

(Linhas 31 a 35): Logo após essas definições de variáveis, eu solicito um ***ForEach*** a aplicação para verificar cada índice do vetor informado (vetor que contém todos os usuários), fazendo uma busca linear, para verificar se o nome de usuário, informado pela Body Params Request previamente alocada em "verificaUsuario", existe no vetor, caso ela exista passará para a constante "usuario" o valor true/verdadeiro, fazendo alusão, novamente, que o usuário realmente existe dentro do banco de dados.

Desenvolvimento

Continuação da Terceira parte:

(Linhas 37 a 41): Ele fará exatamente o mesmo que as linhas: 31 a 35, porém ao invés de verificar se realmente existe o usuário informado pela Body Params Request, verificará se realmente existe a senha informada pela Body Params Request, previamente armazenada na constante "verificaSenha", caso seja comprovado que realmente a senha seja compatível com a senha guardada no banco de dados, então atribuirá true/verdadeiro para a variável senha fazendo alusão que senha também existe no banco de dados.

Observação: *ForEach* é uma forma de percorrer um vetor em qualquer linguagem de programação, tanto Java, quanto Python, quanto JavaScript, utilizei esse tipo de busca pela facilidade que ele me concede.

(Linhas: 43 a 52): Fará alguns testes através das informações captadas para informar mensagens dinâmicas na tela, que são elas:

- "Usuário liberado", caso seja realmente comprovado que tanto o usuário quanto a senha informada existam no banco de dados;
- "Não registrado", caso seja realmente comprovado que tanto o usuário quanto a senha informada não existam no banco de dados;
- "Usuário incorreto", caso seja realmente comprovado que o Usuário da API informou a senha correta, mas o nome do usuário (registrado no banco de dados) incorreto;
- "Senha incorreta", caso seja realmente comprado que o Usuário da API informou o usuário correto, mas a digitação da senha (registrada no banco de dados) esteja incorreta.

Desenvolvimento

Quarta parte:

```
54  server.post('/usuarios', (requisicao, resposta) => {
55      let { novoUsuario } = requisicao.body;
56      let { novaSenha } = requisicao.body;
57
58      usuarios.push(novoUsuario);
59      senhas.push(novaSenha);
60
61      return resposta.json(usuarios);
62  });
```

(Linhas 54 a 62): Esse método também é uma das partes principais da API pois, através de um Body Params Request, solicita ao usuário um nome, para um novo usuário, e uma senha, para uma nova senha, que serão posteriormente atribuídos às variáveis locais "novoUsuario" e "novaSenha", onde:

- "novoUsuario" será adicionado ao vetor "usuarios";
- "novaSenha" será adicionada ao vetor "senhas".

Observação: Essa atribuição de ambas das variáveis aos respectivos vetores só é possível realizar através do método ".push(<nome da variável/nome qualquer>)".

É possível notar que ele retorna como resposta para o usuário da API somente o nome dos usuários que ele registrou no "banco de dados".

Desenvolvimento

Quinta parte:

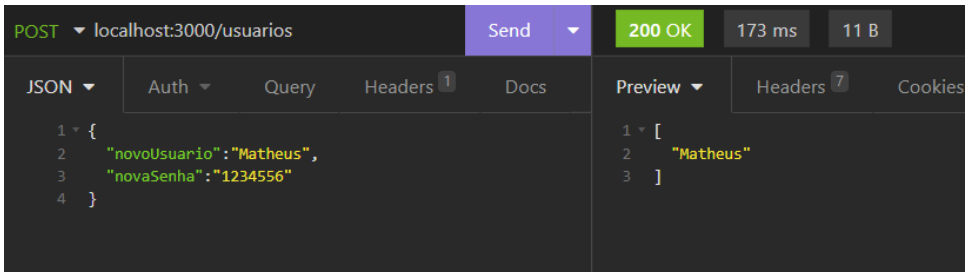
```
64  server.delete('/usuarios/:index', (requisicao, resposta) => {
65      const { index } = requisicao.params;
66
67      usuarios.splice((index - 1),1);
68      senhas.splice((index - 1),1);
69
70      return resposta.json('Usuário ' + index + ' deletado com sucesso');
71  });
```

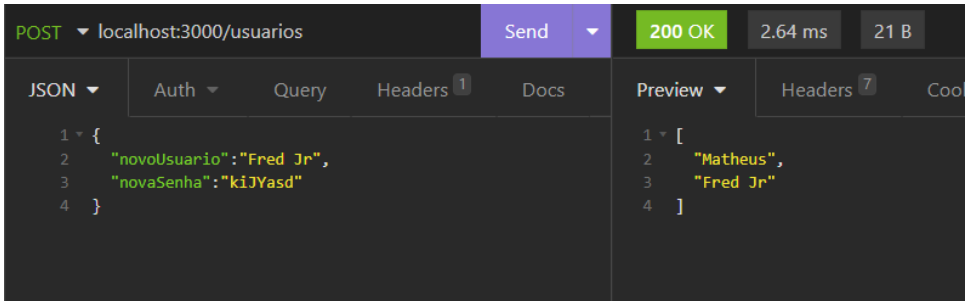
(Linhas 64 a 71): Através desse método é possível realizar a exclusão de um usuário informando o número do ID do usuário que quer excluir, pela URL, e por um Route Params Request pegará o ID informado e armazenará em uma constante Index, onde logo após isso irá ser passado como um dos parâmetros (e realizado uma conta matemática para evitar erros indesejados pelo usuário, da mesma forma que foi feita na segunda observação da parte 02 desse documento), dentro do método ***splice***, junto com a quantidade que você quer deletar a partir do índice do ID do usuário.

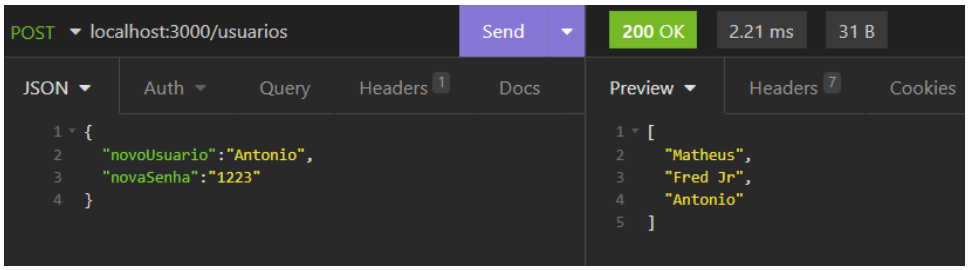
Observação: *Splice* é um método da linguagem JavaScript que serve para deletar algum índice de um vetor, onde deve ser passado como parâmetro, primeiramente, o índice do vetor que deseja excluir, e por segundo parâmetro, a quantidade que deseja excluir, a partir daquele determinado índice. Exemplo: "xuxu.splice(2, 1)" significa: "exclua para mim no vetor xuxu somente índice 2".

Imagens dos Testes

Testes com método Post:

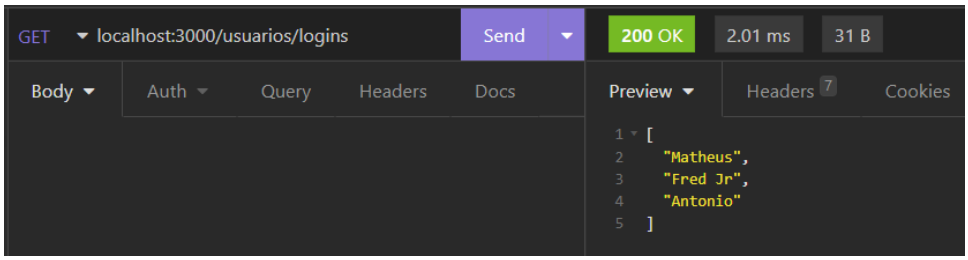


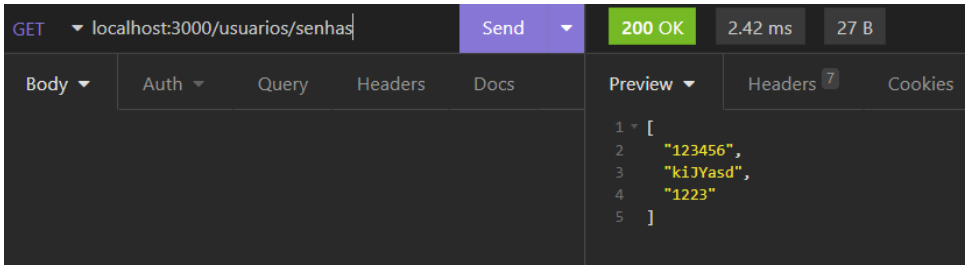


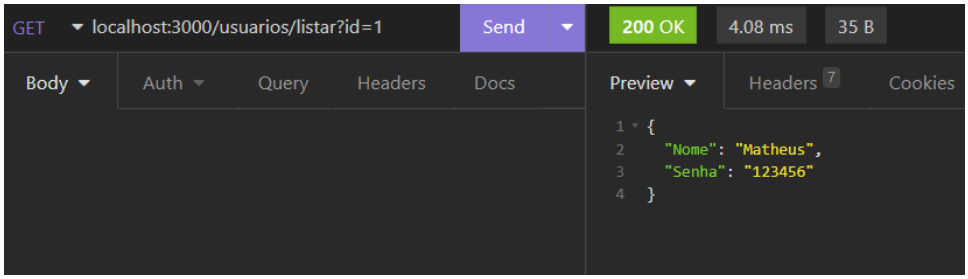


Imagens dos Testes

Testes com método Get (Um de cada método):

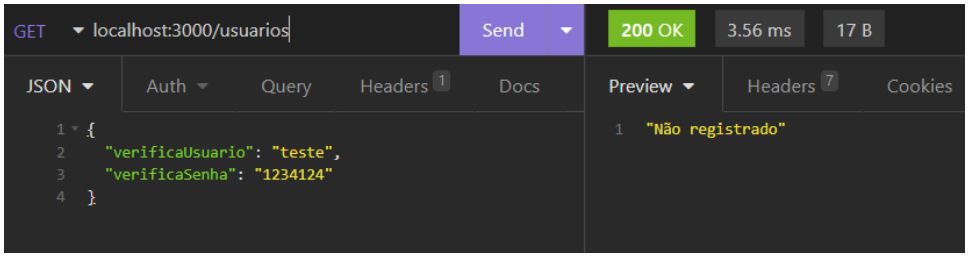


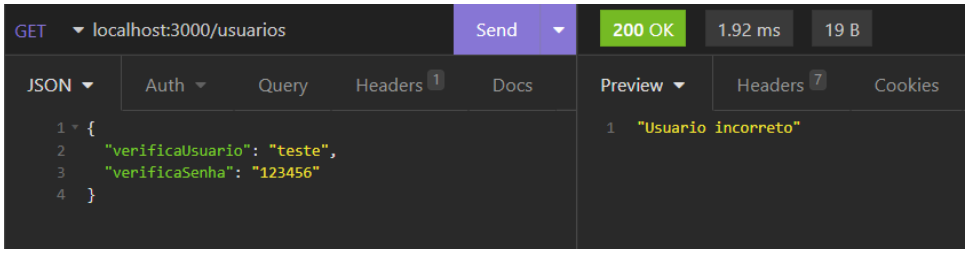


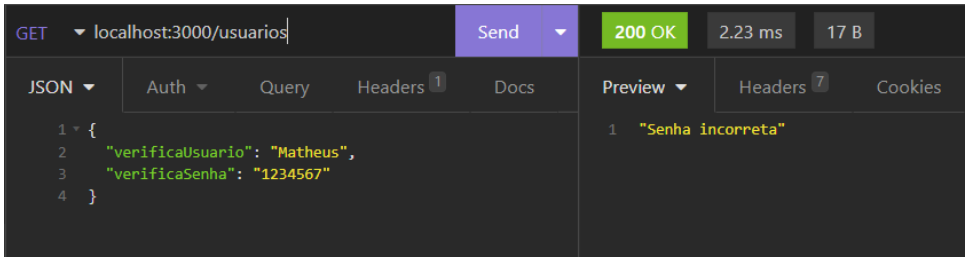


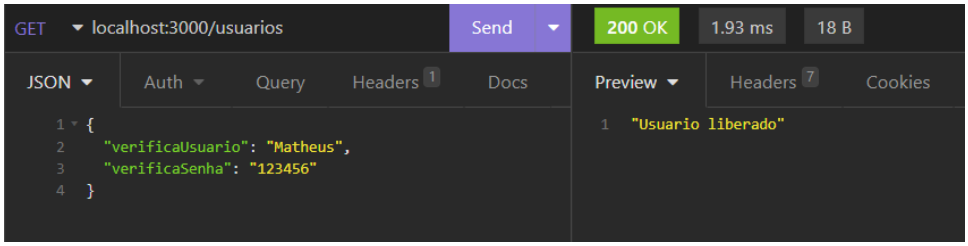
Imagens dos Testes

Testes com método Get (Verificação):



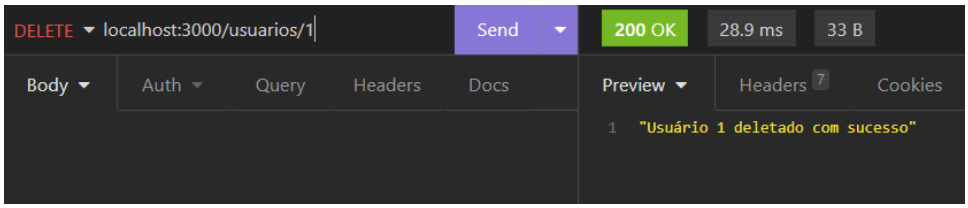




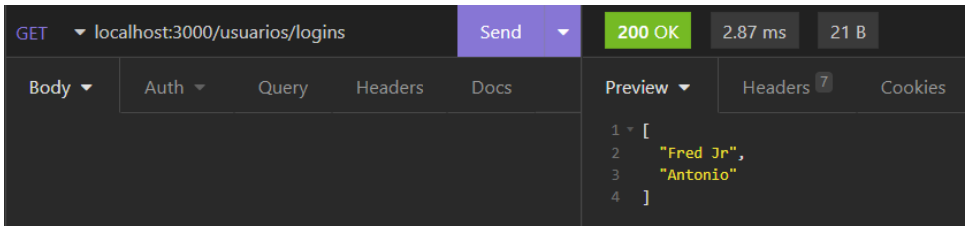


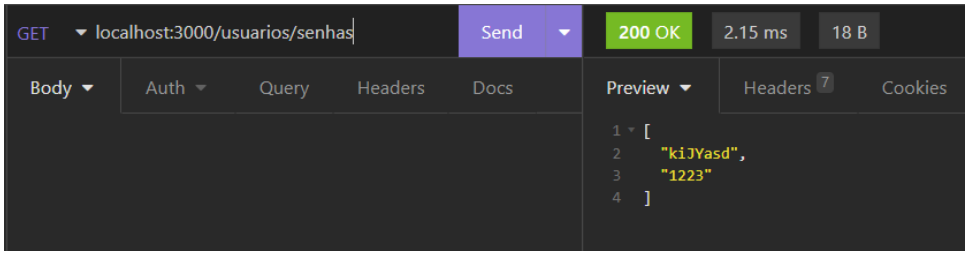
Imagens dos Testes

Testes com método Delete:



Verificando para ver se existe ainda no "banco de dados":





Conclusão

Essa é a API construída do 0 por mim, seguindo todos os padrões REST para desenvolvimento de APIs, para o exercício, em forma de ADO, passado em aula, consegui aprender bastante sobre desenvolvimento de APIs, mesmo sendo tudo uma simulação com 2 vetores e bastante lógica de programação, mas conseguiu me dar uma base muito boa sobre os tipos de requisição que existem muito obrigado por ler até aqui.

LinkedIn: https://www.linkedin.com/in/matheusfsantos9438/

É possível ver o código dessa API acessando o meu repositório do GitHub: Matheus-FSantos/SENAC-PWA107-1142496616-Matheus (github.com)

Após entrar, siga os passos: ADO03 (API REST) > v2 > index.js