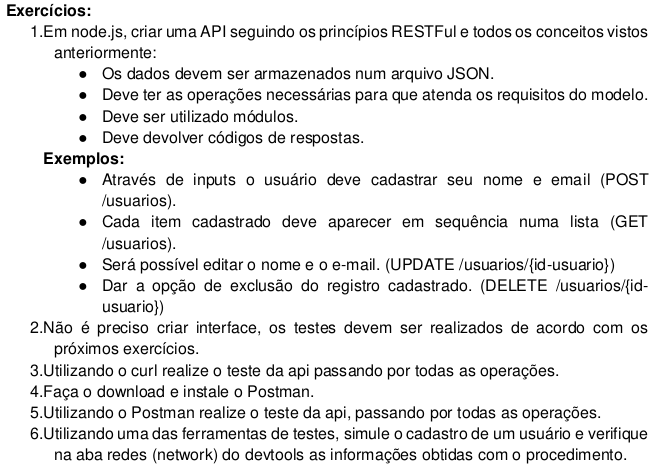
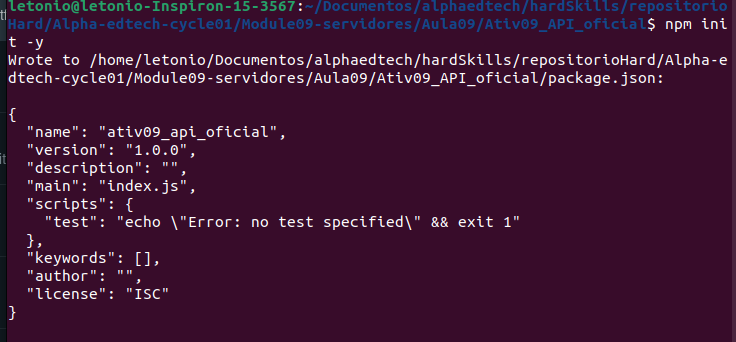
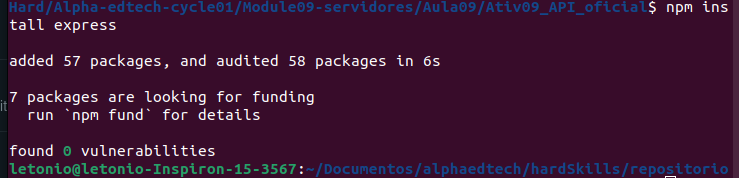
**Módulo 09 – Atividade 09**



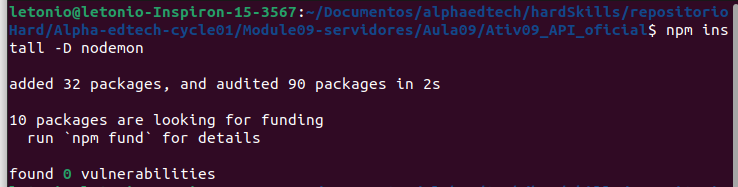
Q1) Comecei a api abrindo o terminal na raiz do projeto, e utilizando “npm init -y”.

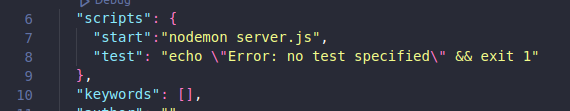
 O comando acima cria um package.json que conterá as informações do nosos projeto, além de scripts e dependências necessárias para o projeto funcionar. Há ainda algumas dependências que são instaladas como devDependencies, que são dependências usadas apenas durante o desenvolvimento do projeto, não são necessárias quando a aplicação já está em produção.

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

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.

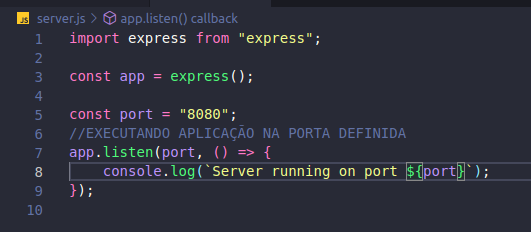
No arquivo “package.json”, se faz necessário adicionar “start”:”nodemon server.js” dentro da chave “scripts”, para que o processo fique automatizado:

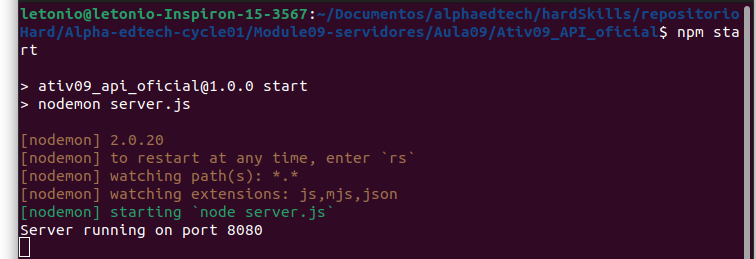
Basta ir no terminal e digitar npm start para inicializar o server, e sempre que fizermos alguma mudança, ele reiniciará automaticamente.

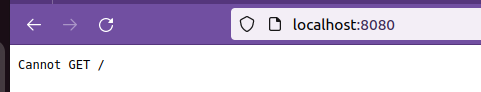
Precisamos criar o arquivo do nosso servidor, o server.js.

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”:

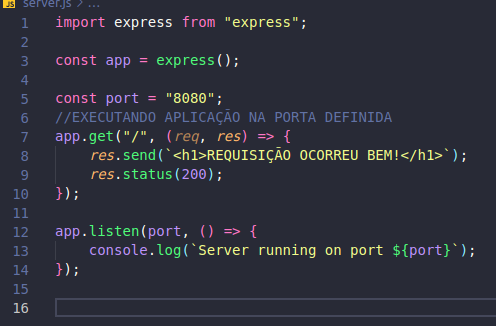
Vamos levantar o servidor, indo no terminal e usando “npm start”:

Pronto, nosso servidor está ativo, indo no navegador e acessando “[http://localhost:8080](http://localhost:8080/)”, temos o seguinte resultado:

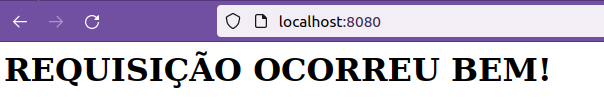


Estamos ouvindo na porta 8080, porém não tem nenhuma função que seja chamada quando o navegador faz uma requisição pelo método GET. Portanto, precisamos adicionar um “app.get()” e indicar o que vai acontecer quando alguém acessa nosso servidor e manda uma requisição pelo método GET.

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



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.

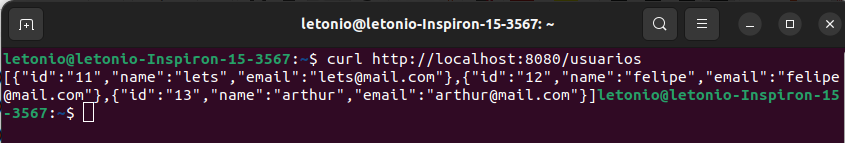
----------------------

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

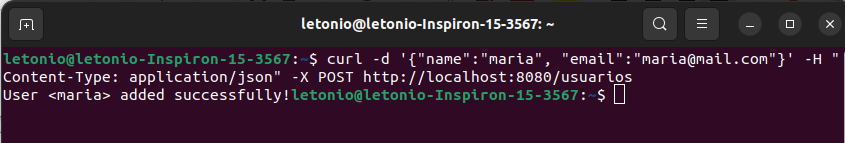


Nota-se que temos três usuários na resposta enviada pelo servidor.

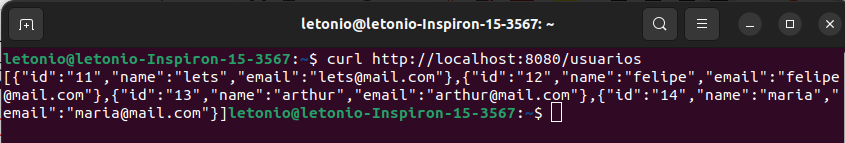
O próximo método testado será o método POST, escrevendo a seguinte linha de código:

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

O resultado foi o seguinte:

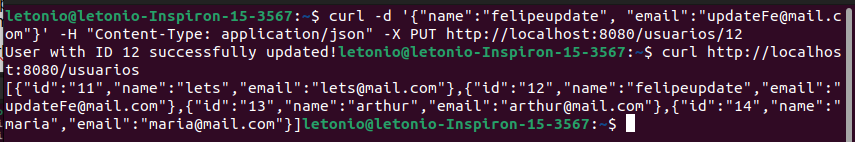
É possível confirmar que a mudança ocorreu de fato, através desse comando:

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

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

O comando empregado foi:

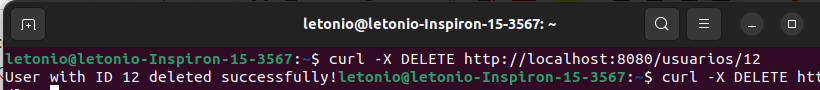
curl -d '{"name":"felipeupdate", "email":"updateFe@mail.com"}' -H "Content-Type: application/json" -X PUT http://localhost:8080/usuarios/12

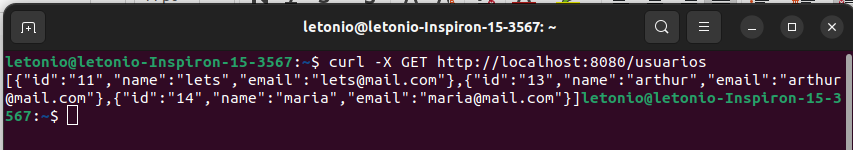
Aproveitei para verificar se a alteração havia ocorrido.

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.

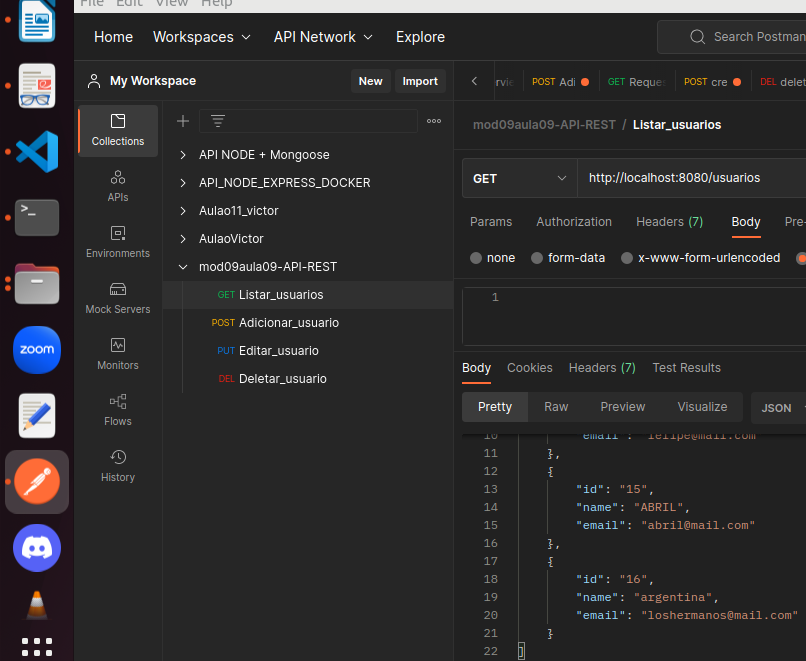
Optou-se por deletar o elemento que foi alterado anteriormente. Portanto, o comando executado foi esse:

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

Para confirmar que foi deletado, verifiquei com o GET

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

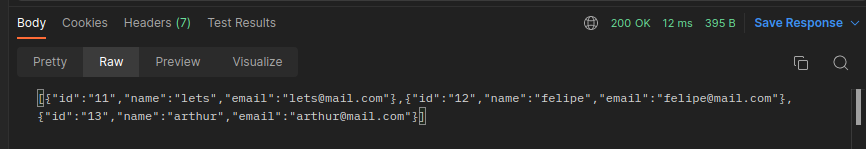
Q4) Instalei o postman:

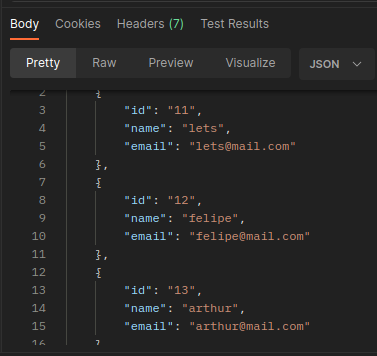


Q5) Primeiro teste é o método GET, lembrando que a URL é

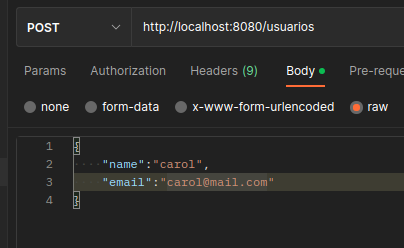
“<http://localhost:8080/usuarios>”

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

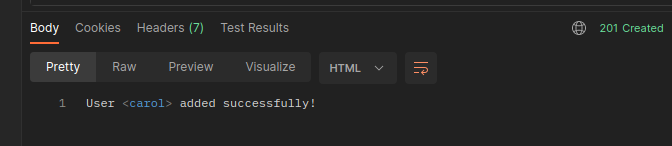
Nota-se que o status foi 200, ou seja, está tudo okay. A próxima imagem deixa num formato mais fácil de identificar os elementos:



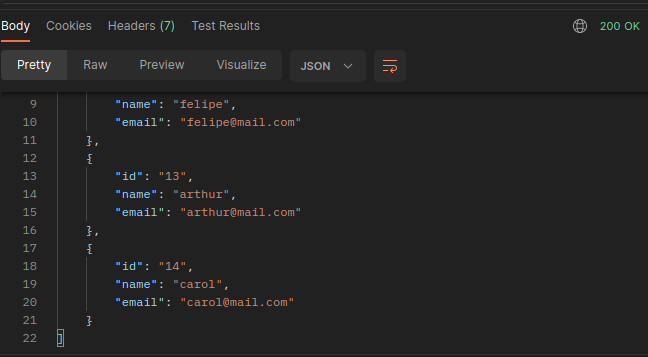
Testando o método POST, enviando no corpo da requisição o seguinte conteúdo:



Ao enviar o resultado é o seguinte:

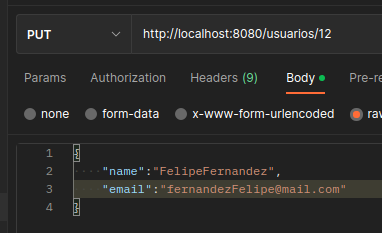
Nota-se o status 201, isso foi enviado pelo código, é interessante usar esse status porque indica que houve criação de recurso.

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

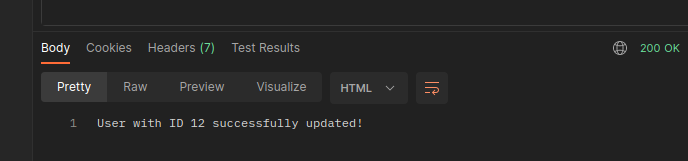
Nota-se que Carol foi adicionada ao final do arquivo.

O próximo método sera 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.

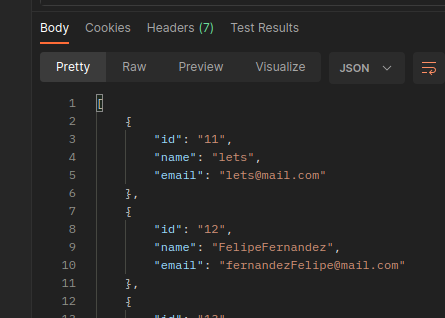
A URL será “<http://localhost:8080/usuarios/12>”, com o método PUT. A imagem a seguir mostra como ficou a requisição via POSTMAN:



A resposta obtida tem status 200:



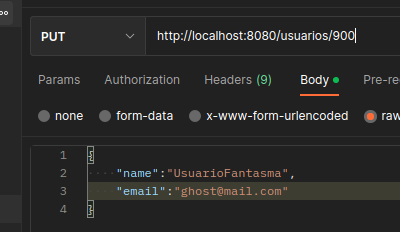
Para confirmar que a alteração aconteceu, usamos o método GET novamente.



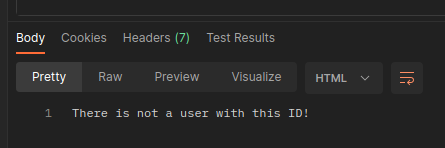
O elemento com ID 12 foi alterado corretamente.

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:

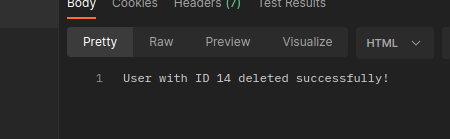


A resposta é uma mensagem de que não existe um usuário com essa ID.

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:

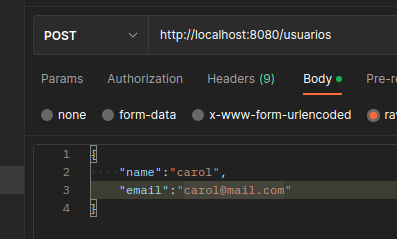
<http://localhost:8080/usuarios/14>:



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

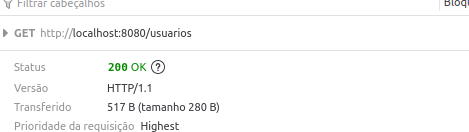


Q6) Vamos usar o método POST, através do Postman, para adicionar um usuário, enquanto observamos a aba de rede (network) na DEVTOOLS.



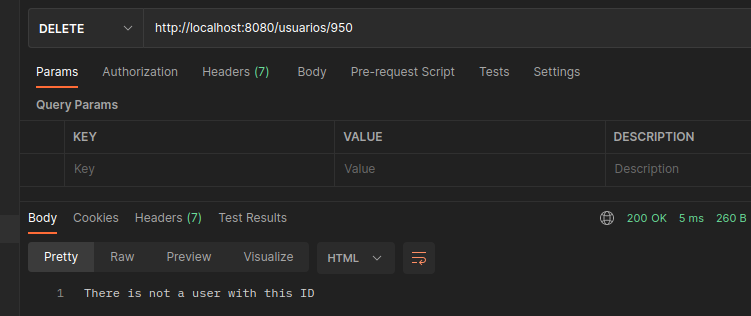
A página mostra o conteúdo que foi retornado na resposta.

Ao lado direito do devtools, nota-se:

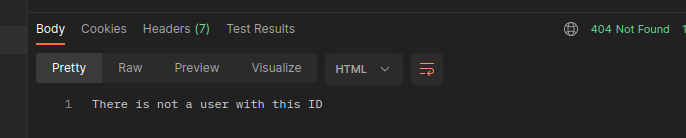


Q7) Ainda usando o POSTMAN, vamos tentar deletar um usuário com ID inexistente, por exemplo, ID=950. O caminho completo que deve ser adicionado no Postman será:

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

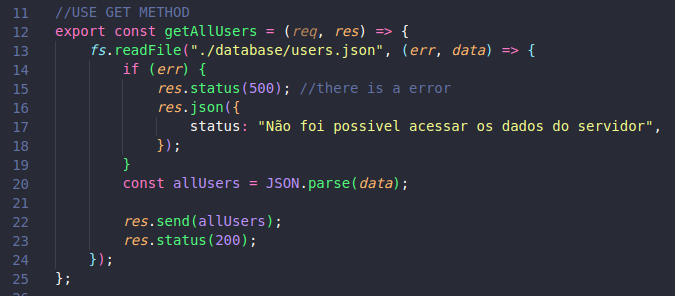
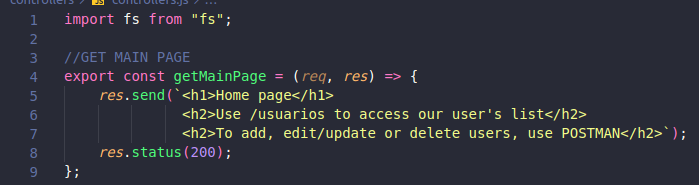


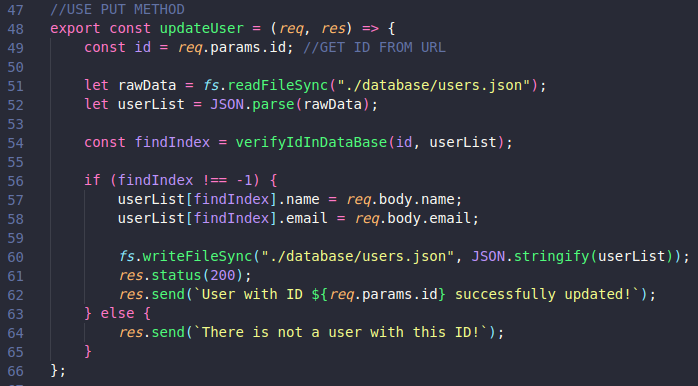
Coloquei para ele retornar status de 404 – Not found.

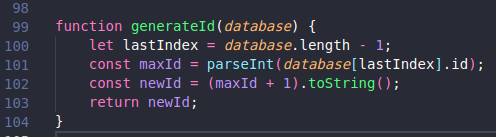
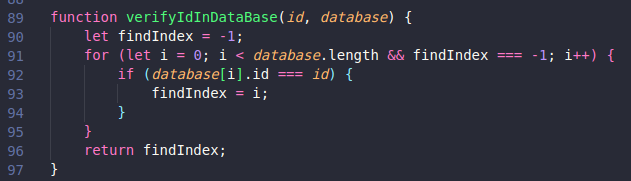
DEIXANDO REGISTRADO O MÓDULO DE MANIPULAÇÃO “FS”

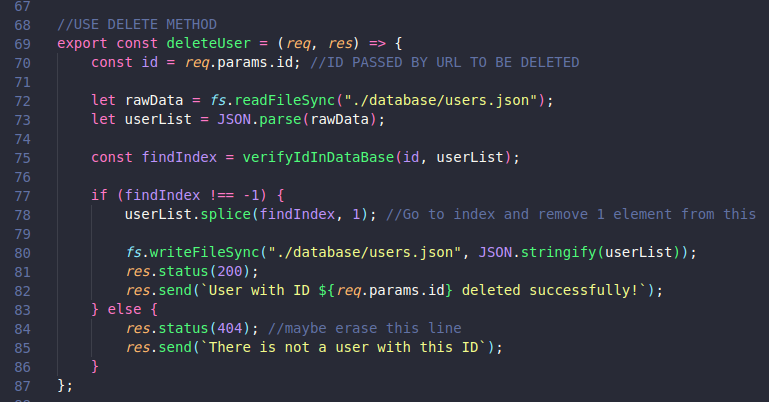
Manipulação de arquivos JSON como banco de dados. Para isso, utilizaremos o Módulo FS (file system module), que é um módulo build-in do Node, portanto, não é necessário fazer a instalação usando o npm.

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





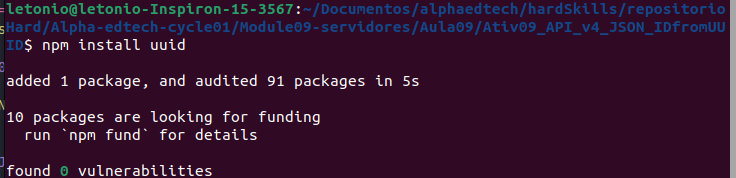




Por fim, ilustra-se o arquivo original json “users.js”, utilizado como banco de dados.



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”



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();