Lista 5 de Exercício – API

Desenvolvimento Web – Lab. Profª Angelina V S Melaré

Parte 1:

Correção:

a) Pegue o projeto-consome-api da semana passada – Parte 2-2.
b) Execute o projeto.
c) No seu Browser veja na área do desenvolvedor (Inspecionar) a parte de Rede as requisições. Se for o caso atualize a página para ver. Veja que ficou em loop.

users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8
users/	200	fetch	App.js:8

d) Corrija o problema
e) A requisição tem que ser feita uma única vez, para isso coloque a chamada da função (que faz a requisição) dentro do useEffect()

```
React.useEffect(() => { funcaoAssync(); }, []);
```

Parte 2:

Projeto API

1) Esse projeto não será igual aos projetos anteriores, que usavam o modelo do React. Faremos a criação do zero. Para isso:
   a) Cria uma pasta para o projeto (md projeto-api)
   b) Entre nesta pasta (cd projeto-api)
   c) Dê o comando para criar o projeto e fazer as definições iniciais, digitando **npm init**

      Irão ser feitas algumas questões sobre o projeto. Pode dar "Enter" e aceitar tudo ou ainda se quiser poderia ter digitado **npm init -y** (que já ignora todas as perguntas)

d) Neste projeto serão usados o framework Express e o pacote Nodemon. Para isso instale os dois:
npm install express nodemon

e) Abra o arquivo do projeto "**package.json**" e veja todas as configurações iniciais que você fez estão lá e os pacotes instalados.

```
"author": "Angelina",
"license": "MIT",
"keywords": ["API","Express"],
"description": "Criação de API",
"dependencies": {
  "express": "^4.17.3",
  "nodemon": "^2.0.15"
}
```

f) No arquivo criado "**package.json**" crie o código que inicia seu projeto. Vá na parte de scripts e coloque o nodemon fazendo o controle e executando o arquivo principal. Veja que ele será nosso servidor e qualquer mudança no código ele fará refletir na execução.

```
"scripts": {
  "test": "echo \"Error: no test specified\" && exit 1",
  "start": "nodemon ./index.js localhost 3000"
},
```

g) Crie o arquivo "**index.js**"

h) Neste arquivo coloque os comandos para importar o express e criar a constante "app" sendo do tipo express. Veja que também será configurado o formato json como padrão de uso da API

```
const express = require('express')
const app= express()

app.use (express.urlencoded({extended:true}))
app.use (express.json())
```

Para configurar como a API agirá ao receber uma requisição "get" faça a sua configuração. No exemplo abaixo estamos definindo que quando for colocada a rota inicial do site (/) e numa requisição de consulta a resposta (res) será um json com a mensagem "Consulta realizada".

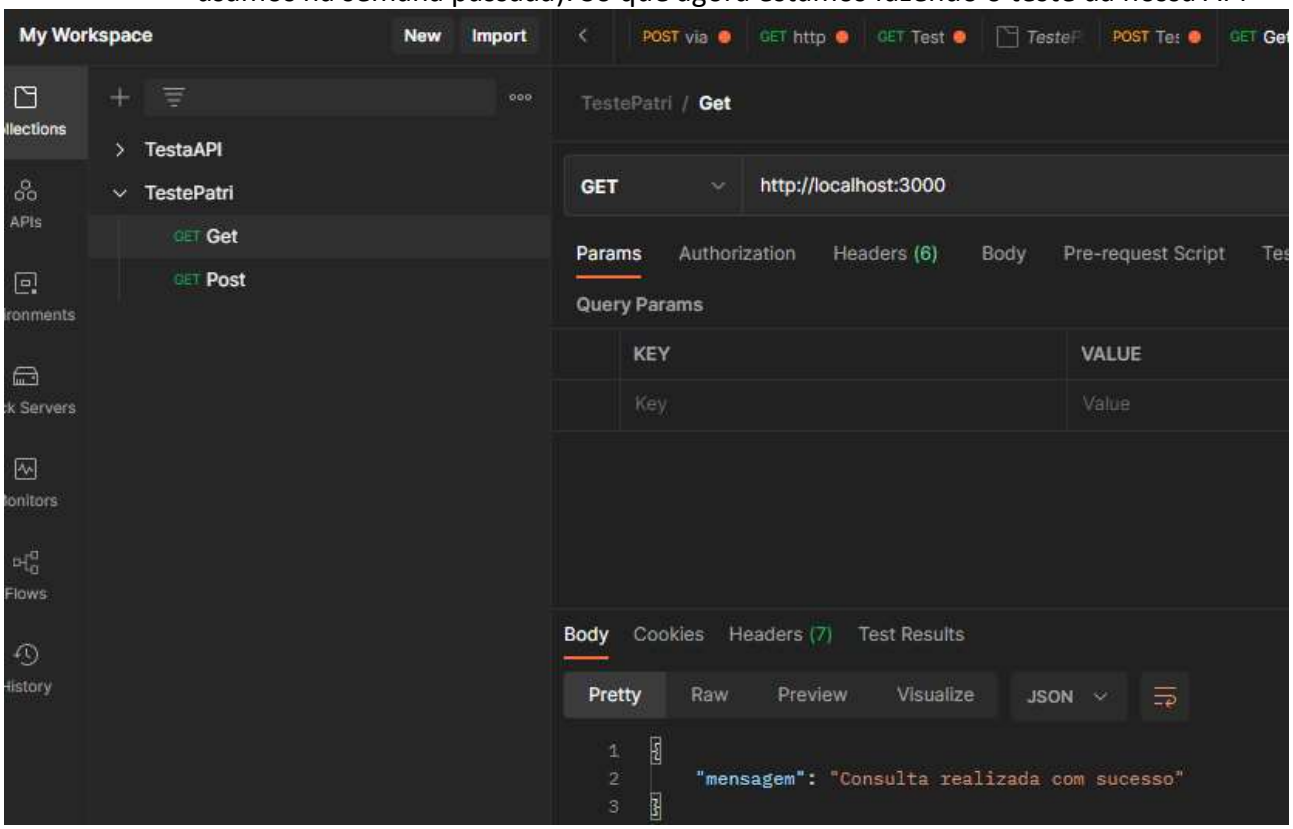
```
app.get ('/', (req,res) =>{
    res.json({'mensagem': 'Consulta realizada'})})
```

```
app.listen(3000)
```

Veja que estamos informando que a porta padrão para uso é a 3000.

i) Para ver como a sua API agirá numa requisição inicie seu projeto
NPM start e veja que abrirá o browser com a mensagem em formato json

j) Faça o teste no Postman. Crie umWorkspace de "TestaAPI", dentro dele dois "request" (aperte o botão da direita do mouse para ver a opção de Add)
Coloque o endereço da API ----- Veja que é igual a uma URL real (uma URL igual que usamos na semana passada). Só que agora estamos fazendo o teste da nossa API



Veja que na parte superior é a requisição e na parte de baixo aparece a resposta.

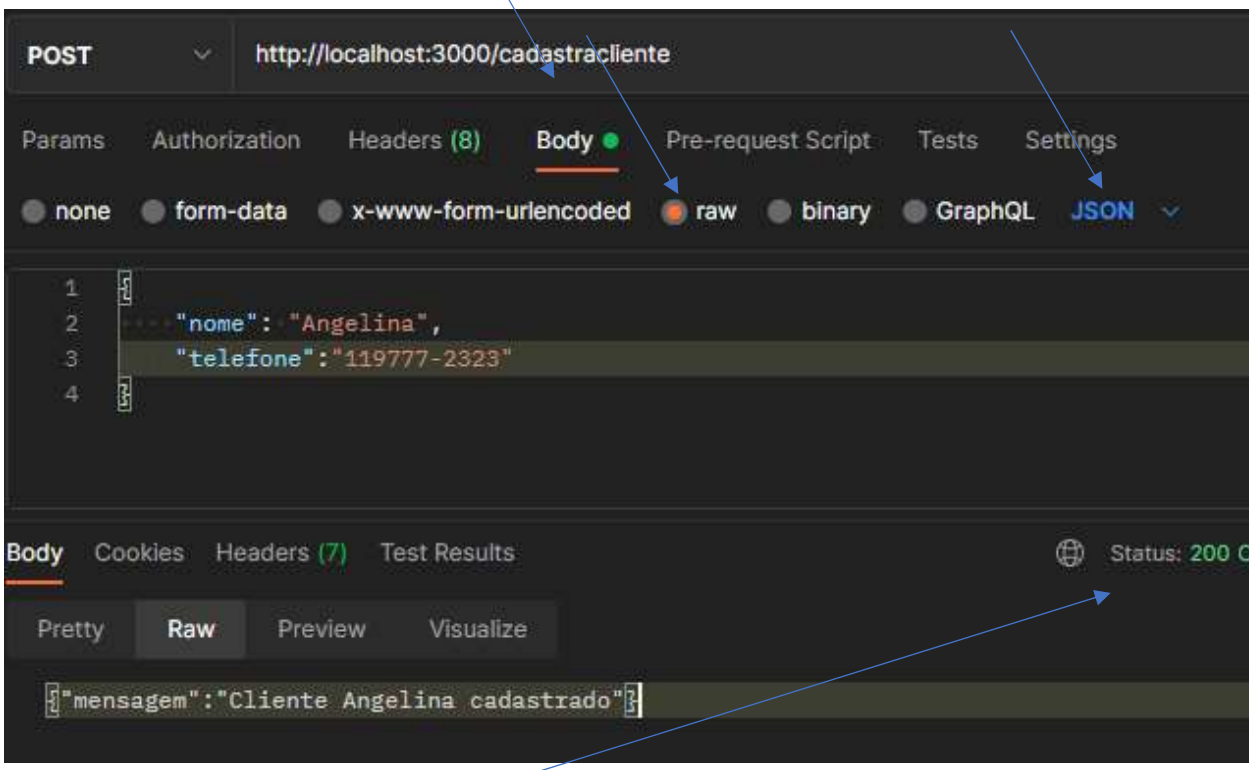
O mesmo processo foi feito para testar a API reqres (https://reqres.in/api/users?id=2) e a de correios (https://viacep.com.br/ws/SP/Sorocaba/Domingos/json/)

k) Agora acrescente o código de simulação de uma requisição inserindo dados, no caso usando o método "post"

```
app.post('/cadastracliente', (req,res) =>{
        const nome =req.body.nome
        const telefone = req.body.telefone
```

```
    console.log(`Cadastrado: ${nome} com o telefone ${telefone}`)
    res.json({'mensagem': `Cliente ${nome} cadastrado`})
})
```

l) Vá no Postman para testar
Para fazer o teste deverá ser mudado o método (verbo) de get para post
Como não tivemos o envio dos dados a serem inseridos (não temos o front-end enviando o json) faremos a inserção direto no "body". Veja que serão digitados os dados json a aba "Body"- escolhida a opção "raw" e o formato "Json"



Depois de inserir os dados aperte o "SEND" veja que a mensagem de retorno apareceu e o status foi o "200"

m) Veja dentro do VS Code que a resposta de retorno da requisição também apareceu com o nome da pessoa que você inseriu

```
Cadastrado: Klebi com o telefone 15-9388-8338
```

n) Agora faça algumas mudanças no código. Veja no site do "mozilla" tem todas as especificações de status de resposta
https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

Como o retorno correto da inserção é 201 e não 200 vamos mudar o código:

```
Mude o código de
   res.json({mensagem: `Cliente ${nome} cadastrado`})
 para
  res.status(201).json({mensagem: `Cliente ${nome} cadastrado`})
```

e ainda coloque :

```
if (!nome)
{
        res.status(422).json({mensagem: 'Campo nome é obrigatório'})
        return
}
```

Para que possa ser uma porta configurável faça as mudanças abaixo, criado uma constante:

```
const porta= 3000
app.listen(porta,()=> {console.log(`Rodando na porta ${porta}`)})
```

o) Desafio: crie o método “delete”
p) Para teste do “post” crie um array de objetos com dados de clientes no próprio código.
q) Capture a execução dos métodos no Postman e envio junto com o projeto.