



Desenvolver aplicações Backend para WEB

Prof. Renan Ponick



Lema

"Nenhum homem é livre se não puder se controlar"

Pitágoras

Agenda

17/08/2023 - Postman + Post

18/08/2023 - Exercícios

24/08/2023 - Teste Unitário

25/08/2023 - Exercícios + Testes

31/08/2023 - Review

01/09/2023 - Exercícios + Testes

07/09/2023 - FERIADO

08/09/2023 - FERIADO

14/09/2023 - Boas Práticas + Arq. REST e RESTFull

15/09/2023 - Exercícios + Testes

21/09/2023 - MVC?

22/09/2023 - CRUD

29/08/2023 - MySQL - Async - Await

29/09/2023 - Exercícios + Testes

Nodemon

Nodemon

Instale o nodemon apenas para desenvolvimento

```
PROBLEMS  GL  OUTPUT  DEBUG CONSOLE  GL  TERMINAL  
○ renan@ng0502:~/Documents/senac/JP/javascript$ npm install --save-dev nodemon
```

Crie um script no package.json para iniciar a API

```
7  "scripts": {  
8    "start": "nodemon index.js",  
9    "test": "echo \"Error: no test specified\" && exit 1"  
10 },  
11 "repository": {
```

Nodemon

Inicialize a API agora somente com **npm start**

```
○ renan@ng0502:~/Documents/senac/JP/javascript$ npm start  
  
> javascript@1.0.0 start  
> nodemon index.js  
  
[nodemon] 3.0.1  
[nodemon] to restart at any time, enter `rs`  
[nodemon] watching path(s): *.*  
[nodemon] watching extensions: js,mjs,cjs,json  
[nodemon] starting `node index.js`  
Servidor rodando na porta 3000  
█
```



Antes de Avançar

Resumindo

A comunicação entre APIs ocorre por meio de solicitações e respostas entre sistemas computacionais.

Solicitação (**Request**): Um aplicativo cliente envia uma solicitação para a API. Essa solicitação inclui informações como o tipo de operação que o cliente deseja realizar, os parâmetros necessários e quaisquer dados adicionais relevantes. As solicitações podem ser feitas usando métodos HTTP, como **GET, POST, PUT ou DELETE**.

Processamento na API: A API recebe a solicitação e a processa de acordo com as regras e lógica definidas.

Resposta (**Response**): Após processar a solicitação, a API gera uma resposta. Essa resposta contém informações relevantes para o cliente, como dados solicitados ou informações sobre o **status** da operação. A resposta é geralmente estruturada em um formato como **JSON** ou **XML**, que é facilmente compreendido tanto pelo cliente quanto pela API.

JSON

JavaScript Object Notation é um formato de intercâmbio de dados leve e amplamente utilizado. Ele é uma maneira de representar informações estruturadas usando uma sintaxe de texto simples, tornando-o fácil de ler e escrever, além de ser facilmente interpretado.

A estrutura básica consiste em pares de chave-valor, onde as chaves são strings (identificadores) e os valores são qualquer coisa (quase qualquer coisa). Na aula anterior vimos ele sendo utilizado no package.json, porém aqui está outro exemplo:

```
1  {  
2    "nome": "João",  
3    "idade": 30,  
4    "casado": false,  
5    "interesses": ["programação", "música", "esportes"]  
6  }
```

REQUEST



lado do cliente

Método: GET, POST, PUT, DELETE;

URI: Caminho e params;

Headers: Authorization, Content-Type, User-Agent;

Body (só POST e PUT): Contém os dados a serem enviados (JSON);

Parâmetros de Consulta: query: { num1: '2', num2: '3' },

Para mais informações console o req

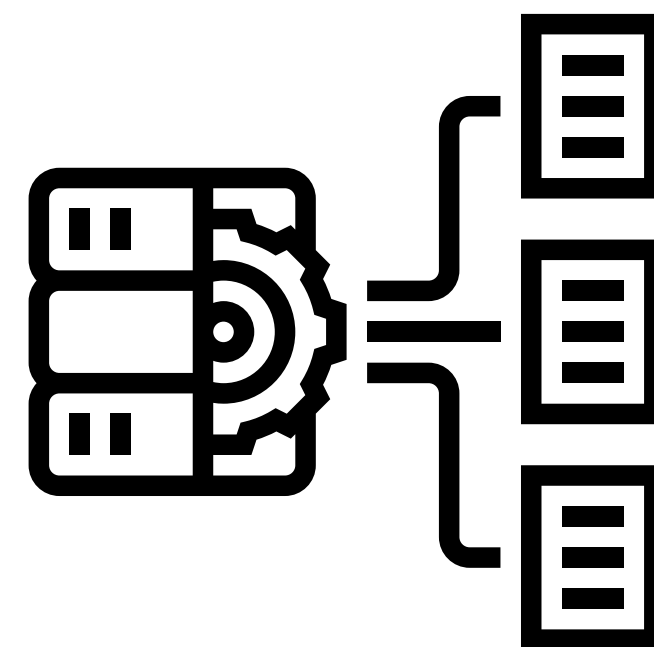
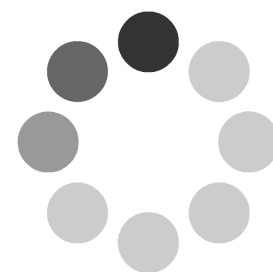
```
21 app.get("/api/exercicioum", (req, res) => {  
22 |   console.log(req)
```

REQUEST

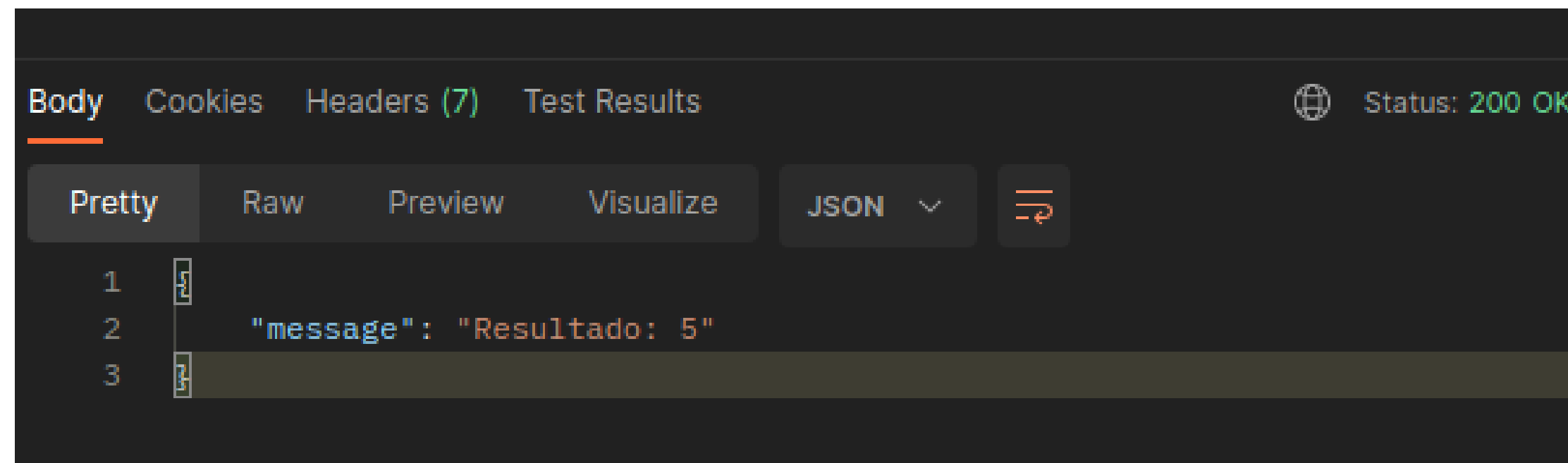


lado do cliente

Loading

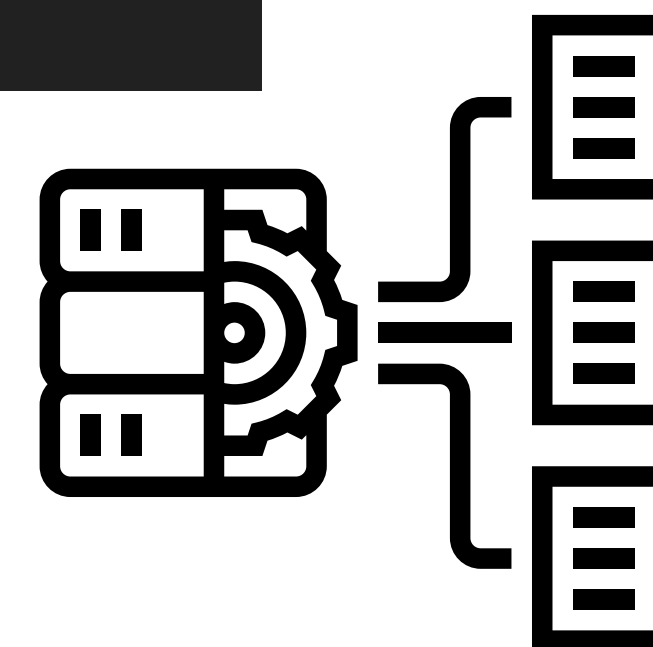


lado do servidor



lado do cliente

JSON || XML
Status



lado do servidor



RESPONSE

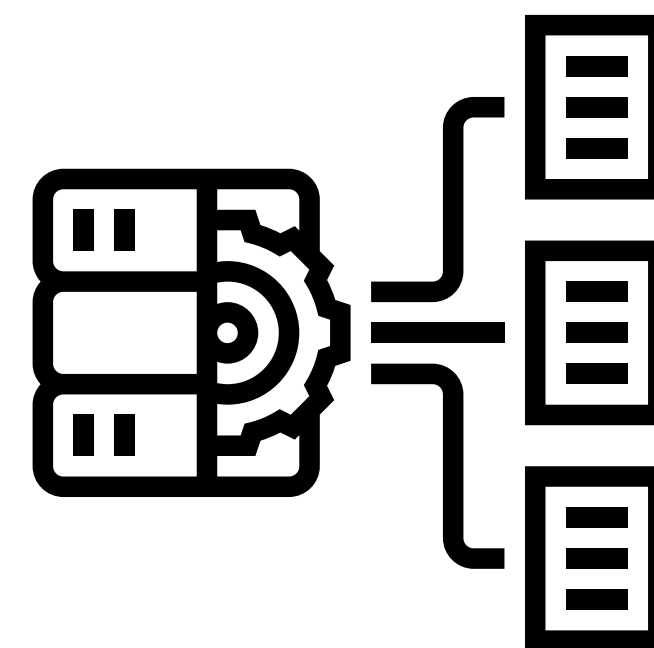
Para mais informações console o res

```
21 app.get("/api/exercicioum", (req, res) => {  
22   console.log(res)
```



lado do cliente

JSON || XML
Status



lado do servidor



RESPONSE

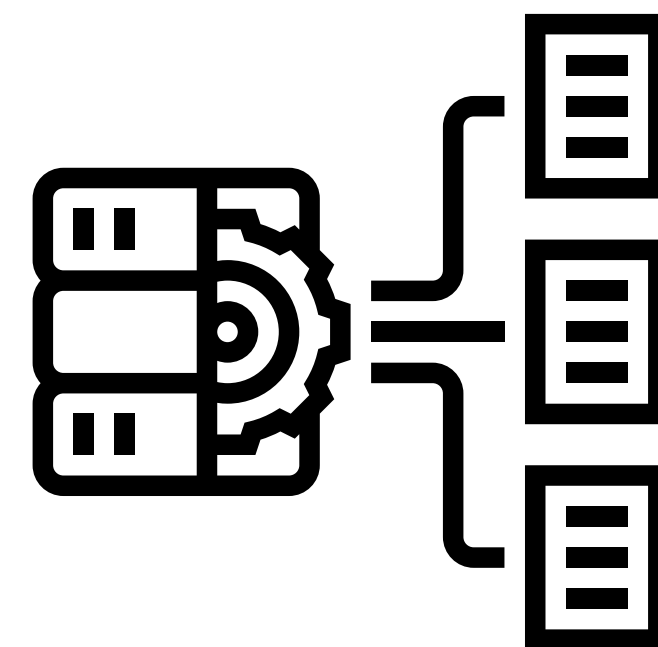
REQUEST



JSON || **XML**



lado do cliente



lado do servidor



RESPONSE



Até aqui tranquilo?

Métodos POST e PUT

POST e PUT

POST é usado para enviar dados para o servidor para que ele possa ser processado e armazenado.

PUT é usado para atualizar um recurso existente no servidor ou criar um recurso se ele ainda não existir.

Esses valores são passados pelo **body**, Contém os dados a serem enviados (JSON);

POST

Ajustando o código para pegar os valores esperados no **body** da **req**uiisição

```
31 app.post("/api/exercicioum", (req, res) => {  
32   const num1 = req.body.num1;  
33   const num2 = req.body.num2;  
34   const result = num1 + num2;  
35  
36   res.json({ message: `Resultado: ${result}` });  
37 });
```

Para executar um metodo POST e passar os valores no body, recomendo utilizar o postman.

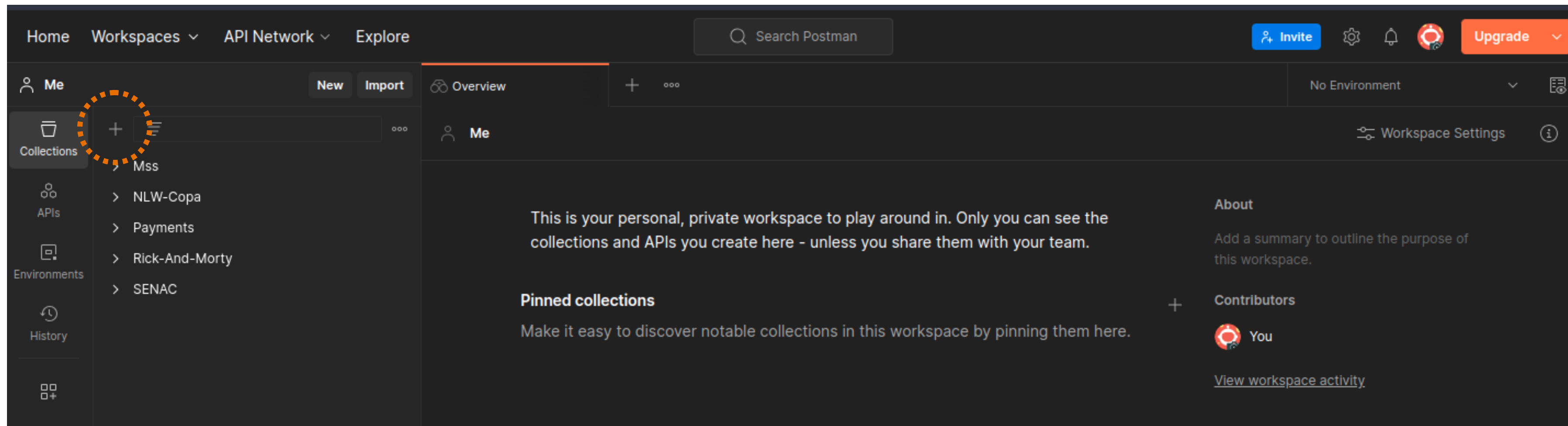


POSTMAN

Antes de mais nada, baixe o postman.
Com estaremos trabalhando com localhost, não será utilizado
o postman WEB para realizar as requisições.

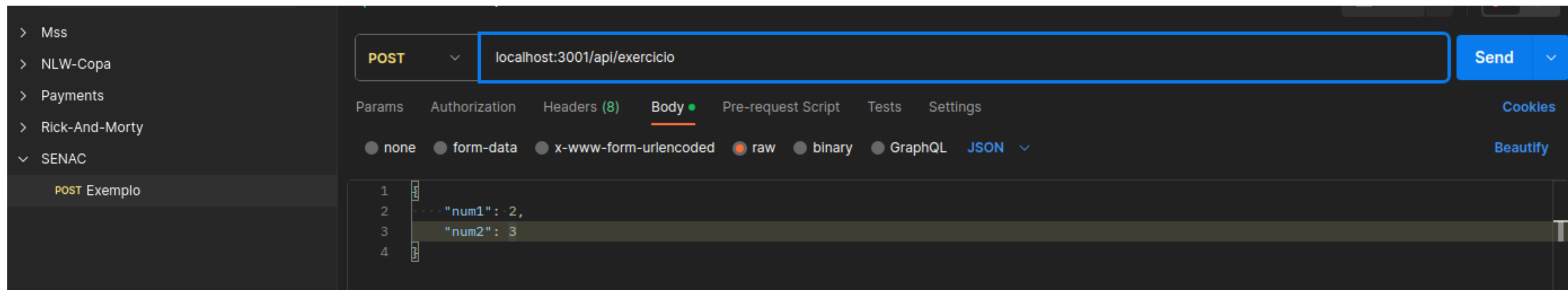
POSTMAN

Crie uma nova collection para salvar suas requisições



POSTMAN

Dentro da pasta crie uma nova requisição e no body da requisição defina como raw - Json e passe os valores abaixo:



Agora clique em send.

Erro...

```
Body Cookies Headers (8) Test Results
Status: 500 Internal Server Error Time: 11 ms Size: 1.69 KB Save as Example

Pretty Raw Preview Visualize HTML
1 <!DOCTYPE html>
2 <html lang="en">
3
4 <head>
5   <meta charset="utf-8">
6   <title>Error</title>
7 </head>
8
9 <body>
10   <pre>TypeError: Cannot read properties of undefined (reading '&#39;num1&#39;)<br> &nbsp; &nbsp;at file:///home/renan/Documents/senac/JP/
    javascript/index.js:35:27<br> &nbsp; &nbsp;at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/
    node_modules/express/lib/router/layer.js:95:5)<br> &nbsp; &nbsp;at next (/home/renan/Documents/senac/JP/javascript/node_modules/
    express/lib/router/route.js:144:13)<br> &nbsp; &nbsp;at Route.dispatch (/home/renan/Documents/senac/JP/javascript/node_modules/
    express/lib/router/route.js:114:3)<br> &nbsp; &nbsp;at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/
    node_modules/express/lib/router/layer.js:95:5)<br> &nbsp; &nbsp;at /home/renan/Documents/senac/JP/javascript/node_modules/express/
    lib/router/index.js:284:15<br> &nbsp; &nbsp;at Function.process_params (/home/renan/Documents/senac/JP/javascript/node_modules/
    express/lib/router/index.js:346:12)<br> &nbsp; &nbsp;at next (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/
    router/index.js:280:10)<br> &nbsp; &nbsp;at expressInit (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/
    middleware/init.js:40:5)<br> &nbsp; &nbsp;at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/
    node_modules/express/lib/router/layer.js:95:5)</pre>
11 </body>
```

```
[nodeemon] Starting node index.js
Servidor rodando na porta 3000
undefined
TypeError: Cannot read properties of undefined (reading 'num1')
    at file:///home/renan/Documents/senac/JP/javascript/index.js:35:27
    at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/layer.js:95:5)
    at next (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/route.js:144:13)
    at Route.dispatch (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/route.js:114:3)
    at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/layer.js:95:5)
    at /home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/index.js:284:15
    at Function.process_params (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/index.js:346:12)
    at next (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/index.js:280:10)
    at expressInit (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/middleware/init.js:40:5)
    at Layer.handle [as handle_request] (/home/renan/Documents/senac/JP/javascript/node_modules/express/lib/router/layer.js:95:5)
```

Middleware

Precisamos definir para o express que o corpo da requisição é um JSON.

```
5 // Cria uma instância do aplicativo Express.  
6 const app = express();  
7  
8 // Middleware para permitir que o Express interprete o corpo da requisição como JSON  
9 app.use(express.json());  
10
```

Agora clique em send no postman novamente.

Para mais informações console o req

```
21 app.get("/api/exercicioum", (req, res) => {  
22 |   console.log(req)
```


Agora sim

The screenshot displays a REST client interface with a sidebar on the left containing a list of collections: MISS, NLW-Copa, Payments, Rick-And-Morty, and SENAC. The 'SENAC' collection is expanded, showing a 'POST Exemplo' item. The main panel is configured for a POST request to 'localhost:3001/api/exercicio'. The 'Body' tab is active, showing a JSON payload:

```
{  "num1": 2,  "num2": 3}
```

. The 'raw' radio button is selected for the body type, and the 'JSON' format is chosen. The bottom panel shows the response in the 'Body' tab, which is a JSON object:

```
{  "message": "Resultado: 5"}
```

. The status bar at the bottom right indicates a successful response with 'Status: 200 OK', a response time of '14 ms', and a size of '261 B'. Other tabs in the bottom panel include 'Cookies', 'Headers (7)', and 'Test Results'. A 'Send' button is located in the top right corner of the request configuration area.

POST localhost:3001/api/exercicio

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings

none form-data x-www-form-urlencoded **raw** binary GraphQL JSON

```
1 {
2   "num1": 2,
3   "num2": 3
4 }
```

Body Cookies Headers (7) Test Results

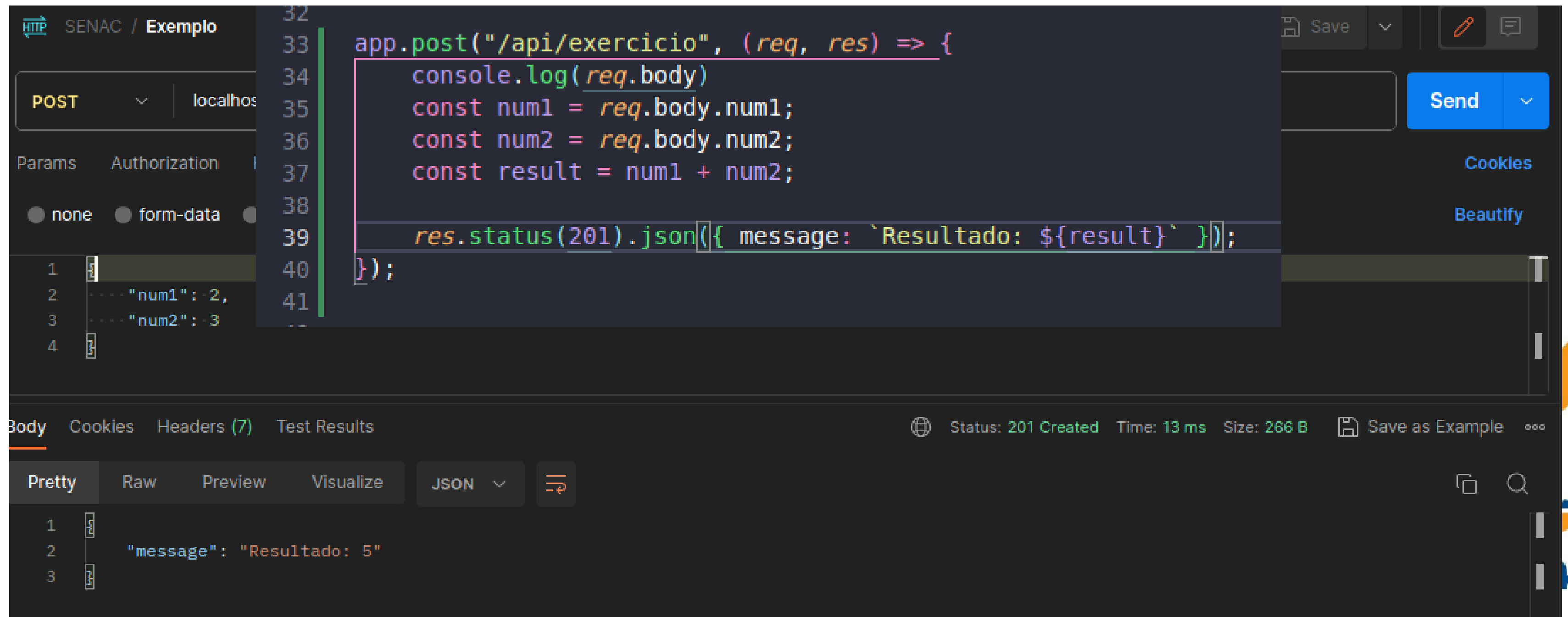
Pretty Raw Preview Visualize JSON

```
1 {
2   "message": "Resultado: 5"
3 }
```

Status: 200 OK Time: 14 ms Size: 261 B Save as Example

Status

Cada resposta possui também um status diferente, cada qual com seu significa, para informá-lo basta:



The screenshot shows a REST client interface in VS Code. On the left, a request is configured with the method **POST** and the URL `localhost:3000/api/exercicio`. The request body is a JSON object: `{ "num1": 2, "num2": 3 }`. The main editor displays the following JavaScript code for the request handler:

```
32
33 app.post("/api/exercicio", (req, res) => {
34   console.log(req.body)
35   const num1 = req.body.num1;
36   const num2 = req.body.num2;
37   const result = num1 + num2;
38
39   res.status(201).json({ message: `Resultado: ${result}` });
40 }
41
```

On the right, there are buttons for **Save**, **Send**, **Cookies**, and **Beautify**. Below the code editor, the response details are shown: **Status: 201 Created**, **Time: 13 ms**, and **Size: 266 B**. The response body is displayed in the **Body** tab, showing the JSON: `{ "message": "Resultado: 5" }`.



Até aqui tranquilo?

Pesquisa

Pesquise sobre todos os possíveis status de uma requisição HTTP e seus respectivos significados

Exercícios

Agora sim, pode fazer a lista toda incluindo a lista extra...





Desafio

Só recomendo pesquisar por funções e tentar estruturar seus arquivos com elas...