

DESIGN DE INTERFACE PARA WEB

AULA 12 – INTRODUÇÃO AO NODEJS



WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@UNIPE.EDU.BR

Mapa de Estudos

AULA 01
Arquitetura de Sistemas Web e sua evolução em camadas

AULA 03
Formulários HTML

AULA 02
Introdução ao HTML, principais TAGS

AULA 04
Introdução ao JavaScript.
Funções declaradas e eventos HTML

AULA 05
Tipos de dados JavaScript.
Principais operadores e introdução ao DOM com getElementById

AULA 06
Estruturas de decisão, repetição e arranjos em JavaScript.

Avaliação A1

JS

Mapa de Estudos

AULA 07

Funções:
addEventListener,
createElement(),
appendChild() e
removeChild().
Hoisting e expressões
de função

AULA 08

Orientação a Objetos de
forma prototípica (antes do
ECMA Script 6)

Criação de classes (depois do
ECMA Script 6)

AULA 11

JQuery

AULA 12, 13, 14...

NodeJS

Avaliação A2

JS

Aula de Hoje

- Introdução à plataforma Node.js.
- Criação de Servidor HTTP básico usando o módulo http do Node.js.
- Introdução à definição de rotas para manipulação de diferentes URLs em um servidor Node.js.
- Utilizar o framework Express.js para criação de aplicações.



O que é o NodeJS?

“NodeJS é JavaScript!”

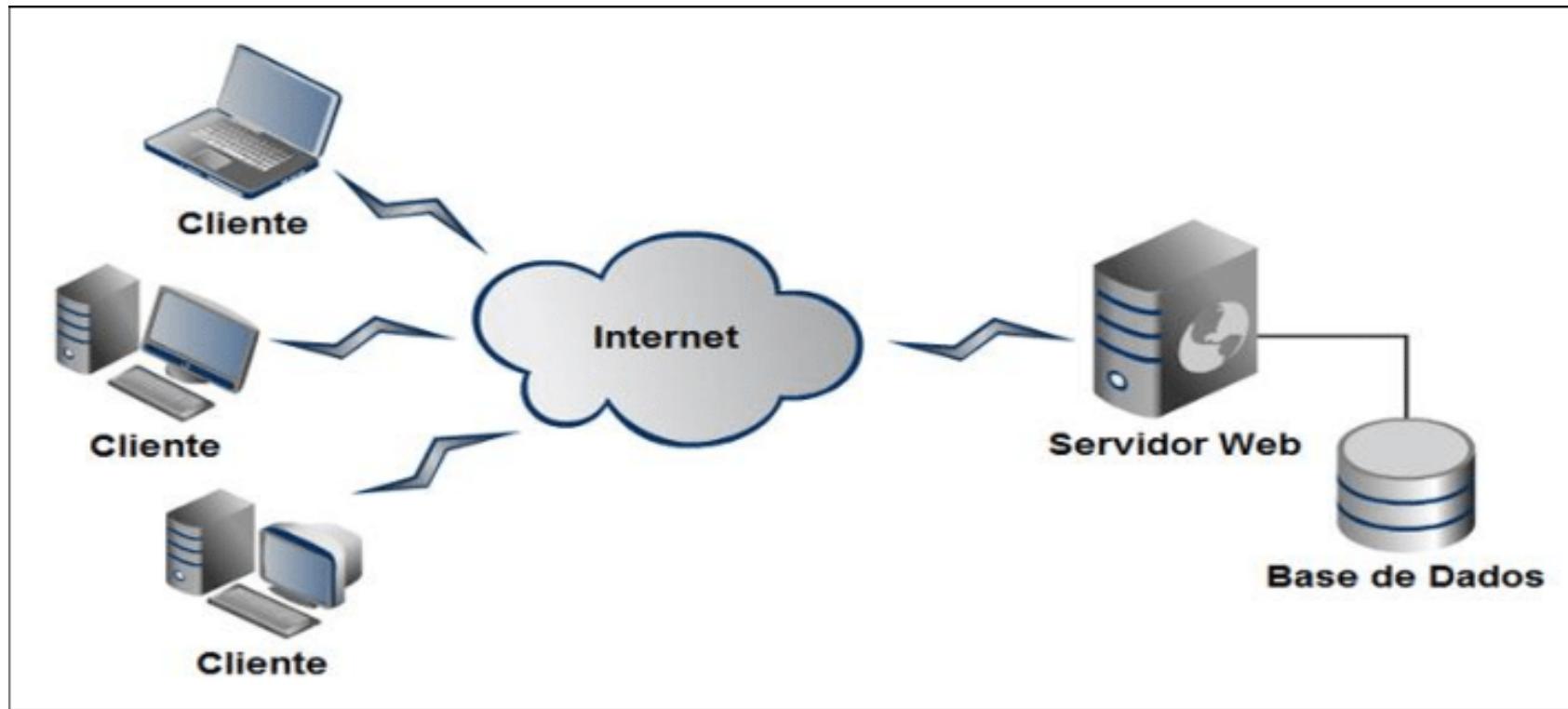
- Isso está incorreto.
- NodeJS é na verdade um **sistema** que você instala em uma determinada máquina para que ela consiga entender o JavaScript.
- O JavaScript por padrão só é entendido pelos navegadores.
- Quem possibilita essa interpretação é a **Engine** do navegador.

Engine

- A "engine" de um navegador é o componente responsável por processar e exibir conteúdo da web. É o **núcleo do navegador** que interpreta e executa o código HTML, CSS e JavaScript de uma página da web para que os usuários possam visualizá-la de maneira adequada.
- As principais partes de uma engine são: **Engine de layout (ou rendering)**, **Engine de JavaScript** e **Engine de Networking**.
- Existem diferentes tipos de engines de navegadores :O Google Chrome utiliza o Blink, que é baseado no WebKit (usado pelo Safari), enquanto o Firefox usa o Gecko e o Microsoft Edge utiliza o Blink (anteriormente usava o EdgeHTML).

Engine

O navegador (cliente) tem a Engine que consegue interpretar o JavaScript. O servidor, em geral, não possui o navegador instalado, por este motivo, não interpreta o JavaScript.



**E se quiséssemos
usar o JavaScript
para outras coisas?**

Aí que entra o NodeJS

NodeJS

- Exemplos de coisas que podemos fazer com o Node: **construir um backend ou API com endpoints, realizar comunicação com banco de dados, autenticação, etc.**
- Ele vai ampliar as possibilidades de uso do JavaScript para além do cliente e do front-end.
- **NodeJS = V8 (engine do Chrome) + libuv + conjunto de módulos**
- Ele é single thread e non blocking I/O, isso significa que todo o código JavaScript é executado em um único thread, permitindo uma execução simples e sem bloqueios na aplicação.

Questão 1

Resposta: D

Node.js é uma Linguagem baseada no motor de JavaScript V8 do Chrome. Quanto a sua orientação e arquitetura, o Node.js é uma linguagem que é orientada a:

- a) objeto e usa um esquema multi-threading, bloqueante.
- b) objeto e possui um modelo de E/S não bloqueante.
- c) eventos e possui uma arquitetura multi-threading e bloqueante.
- d) eventos e possui um modelo de E/S não bloqueante.

Questão 1 - Resposta

- A **libuv** é uma biblioteca multiplataforma focada em I/O (Input/Output) assíncrono. Ela fornece funcionalidades para manipulação de eventos, sistema de arquivos, rede e comunicação com o sistema operacional.
- Essa capacidade de I/O **não bloqueante** é fundamental para a eficiência do Node.js. Isso significa que, ao realizar operações de I/O, **ele não aguarda o término de uma operação para iniciar outra**. Em vez disso, o Node.js utiliza chamadas de retorno (callbacks), promessas ou, mais recentemente, `async/await` para lidar com operações assíncronas, permitindo que o código continue executando outras tarefas enquanto aguarda a conclusão da operação de I/O. Isso **melhora a escalabilidade e a eficiência em aplicações que dependem fortemente de I/O**.

**Vamos construir uma API Rest
com NodeJS e Express**

O que vamos precisar?

Configurando o Ambiente Node e Express no VS Code

1. Crie uma pasta usando o powershell chamada projectnode.
Entre nela e digite **code .** para abrir o VS Code.
2. Abram o terminal (**control + j**) e
3. **npm init - y** – iniciar novo projeto node. Cria o package.json para gerenciar as dependências. O –y é para responder sim a tudo.
4. **npm i express** – Instala a dependência **express**. Serve para gerenciar as rotas e criar o servidor. Vai criar a pasta **node-modules**.
5. Crie uma pasta **src** e um arquivo **servidor.js** dentro desta pasta. É nesse arquivo que iremos criar as rotas para nossa aplicação.

Express

<https://expressjs.com/pt-br/>

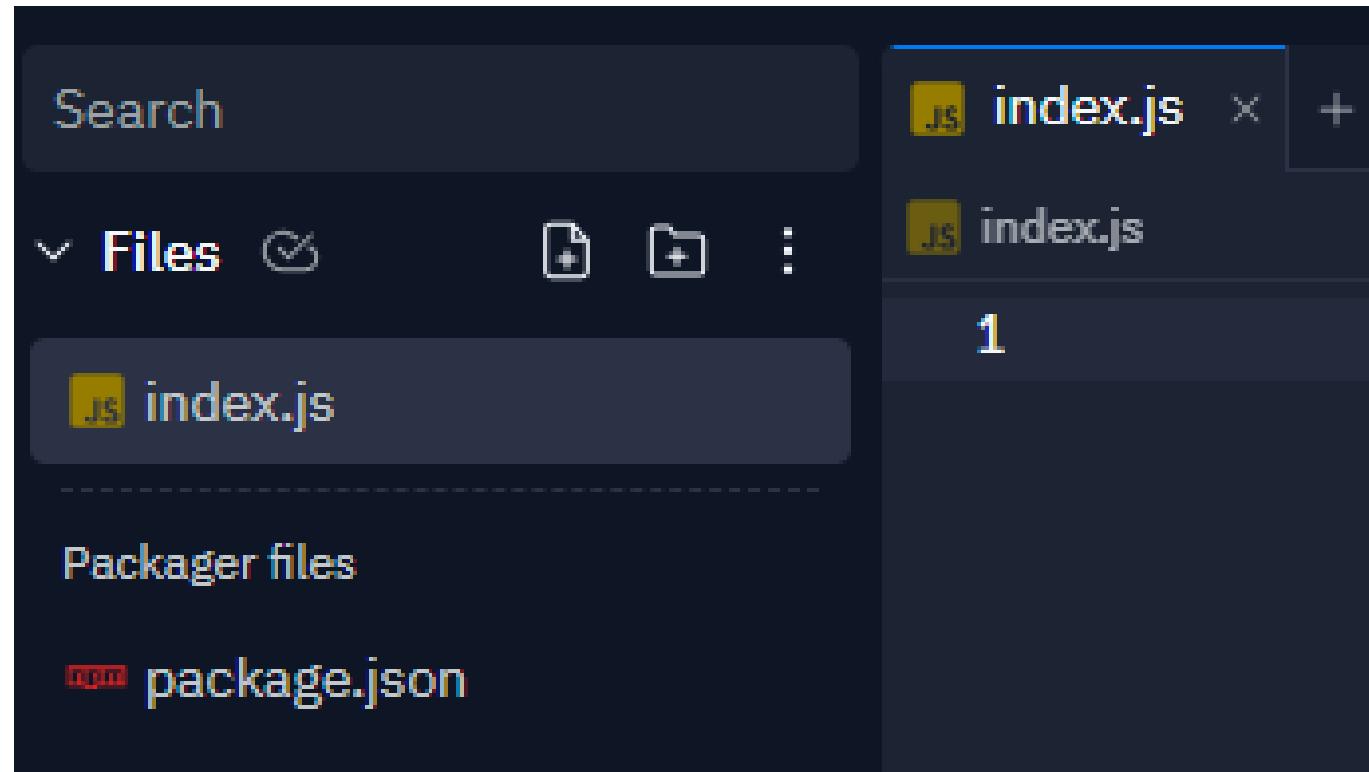
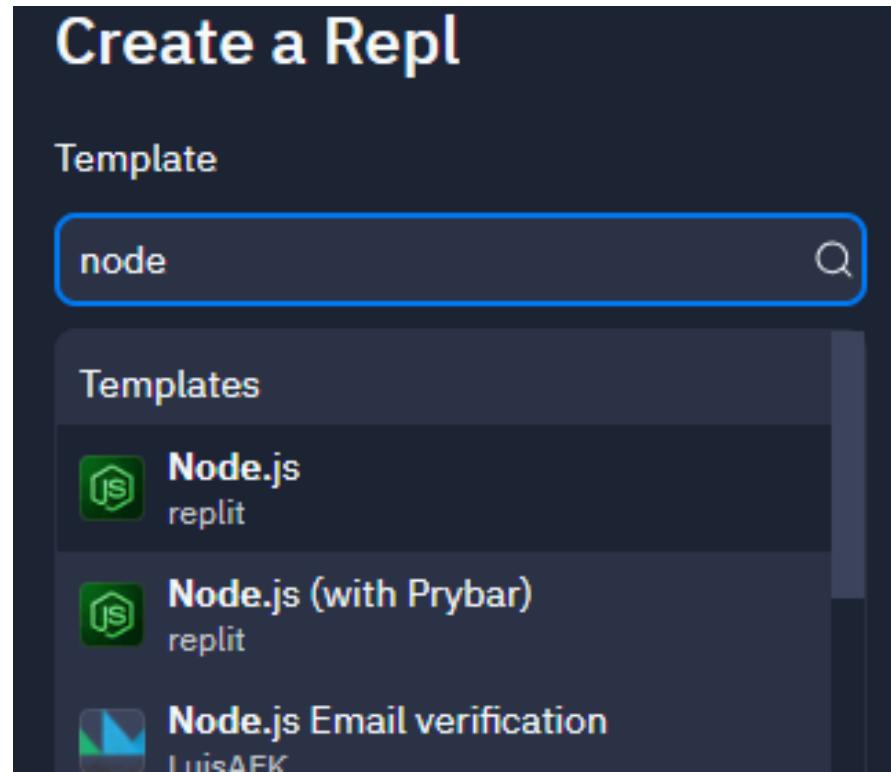
- O Express é um framework Node.js que simplifica o processo de criação de aplicativos web e APIs.
- Ele fornece um conjunto robusto de recursos para lidar com solicitações HTTP, roteamento, manipulação de middleware e renderização de views.

Aplicativos da Web

O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel.

Configurando o Ambiente Node e Express no Replit

Apesar de ter uma opção para o Express, ao selecionarmos a opção Node.js será feita a instalação no Repl do Express assim que começarmos a codificar no arquivo `index.js`.



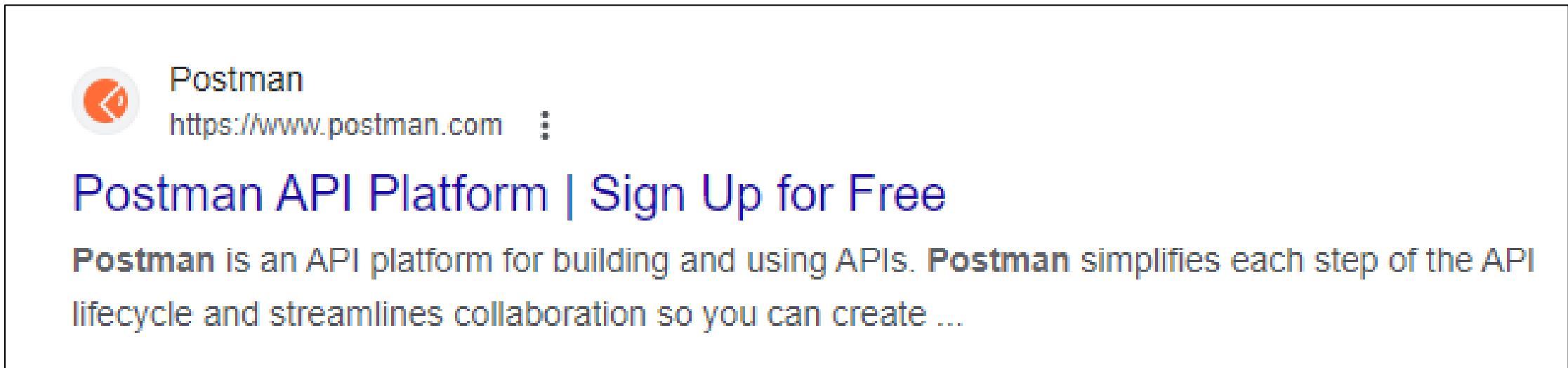
Também vamos precisar do Postman

- Apesar do navegador fazer todas as requisição além da GET, podemos utilizar o Postman pra criar as requisições e simular uma aplicação cliente.
- O Postman é uma plataforma de colaboração e um conjunto de ferramentas para testar, documentar e depurar APIs. Inicialmente, era uma extensão do Google Chrome, mas evoluiu para um aplicativo de desktop independente que oferece uma série de recursos úteis para desenvolvedores, testadores e equipes que trabalham com APIs.

Postman - download

<https://www.postman.com/downloads/>

Será necessário criar uma conta (gratuita).



The screenshot shows a web browser window with the Postman logo and URL in the address bar. The main content area displays the Postman API Platform landing page, featuring a large blue "Sign Up for Free" button. Below the button, a descriptive text explains that Postman is an API platform for building and using APIs, emphasizing its simplification of the API lifecycle and collaboration.

Postman

<https://www.postman.com> :

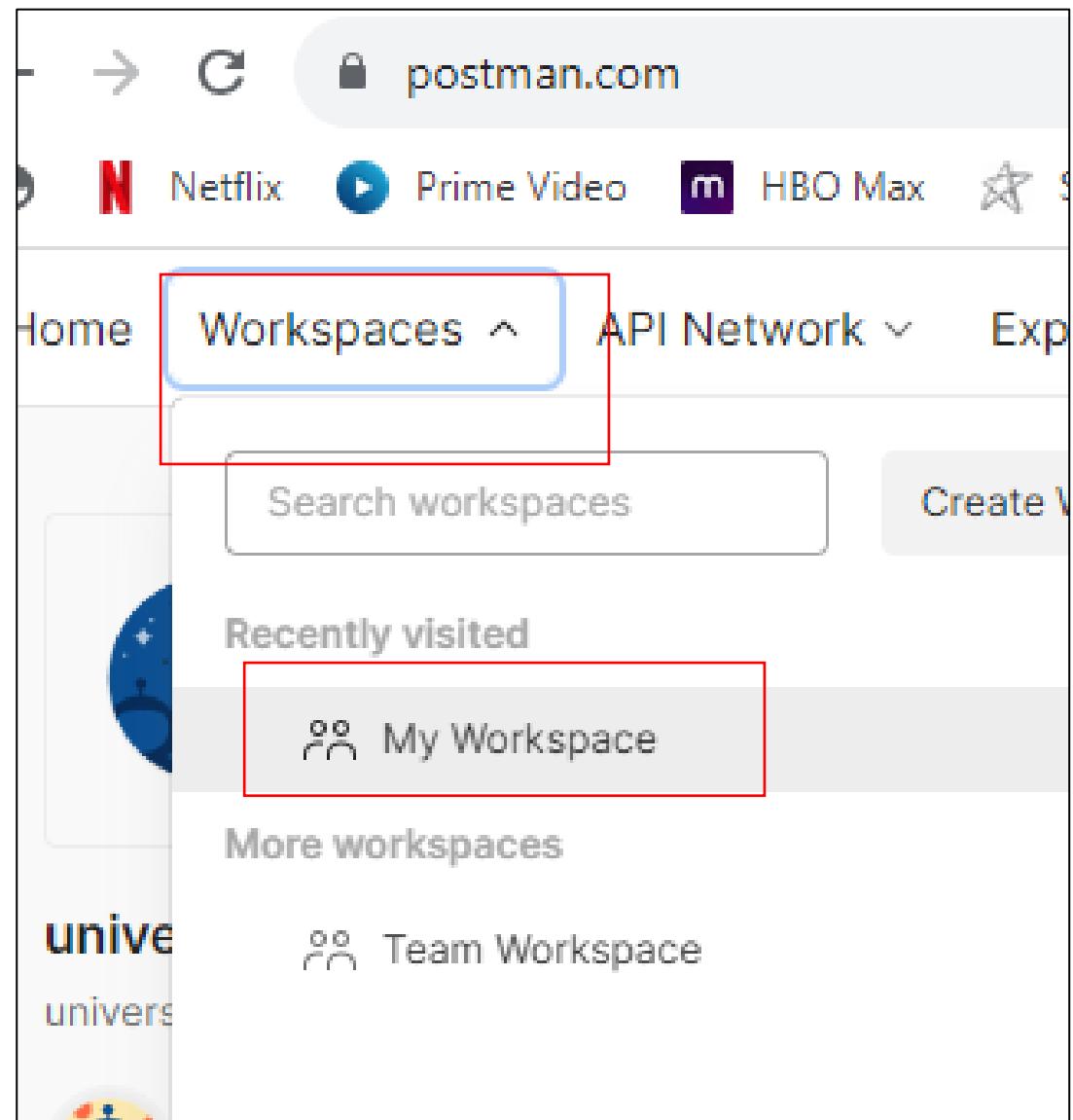
[Postman API Platform | Sign Up for Free](#)

Postman is an API platform for building and using APIs. Postman simplifies each step of the API lifecycle and streamlines collaboration so you can create ...

Postman - workspace

<https://www.postman.com/>

Clica no **Workspace** e em
seguida em **My Workspace**.



Um pouco de teoria antes de codificar

Requisição HTTP

Verbos HTTP

HTTP codes

Parâmetros da Requisição HTTP

E por fim... API REST

Requisição HTTP

- A comunicação entre o front-end (cliente) e o back-end (servidor) vai acontecer principalmente por meio de **requisições http**.
- Uma requisição HTTP (Hypertext Transfer Protocol) é **um pedido que um cliente** (geralmente um navegador da web ou uma aplicação) **envia a um servidor para solicitar recursos ou ações específicas**.
- O protocolo HTTP é usado para definir como essas requisições e respostas devem ser estruturadas, permitindo a comunicação entre o cliente e o servidor na internet.

Verbos HTTP

Os verbos HTTP, também chamados de métodos HTTP, são parte integrante do protocolo HTTP e definem a ação (a finalidade) a ser realizada em um recurso identificado por uma URL (Uniform Resource Locator). Eles especificam a operação a ser executada no recurso quando uma requisição é feita a um servidor web.

- GET – ler ou extrai um valor.
- POST – enviar um valor.
- PUT – alterar ou atualizar um valor.
- PATCH – alterar um valor específico.
- DELETE – deletar um valor.

HTTP Codes

Informam o status de nossas requisições http.

- 1xx – status informativos
- 2xx – status de confirmação ou sucesso da requisição..
- 3xx – status de redirecionamento.
- 4xx – status de erros no cliente (cliente tá fazendo algo errado).
- 5xx – status de erro no servidor (a API apresenta algum tipo de erro).

Parâmetros de uma Requisição HTTP

Basicamente é a maneira como vamos enviar as informações em nossa requisição http. Cada um desempenhando um papel distinto no processo de comunicação entre um cliente e um servidor.
Podem ser de 4 tipos:

- Route params
- Query params
- Body params
- Header params

Parâmetros de Rota - Route Params

- Eles permitem que determinados valores sejam passados dentro da própria URL. O valor é enviado para o servidor dentro da URL do endpoint.
- Supondo que temos o endpoint `https://urlteste.com.br/`
- E que queremos passar um id de valor **123**, passamos esse valor (após a barra) com `https://urlteste.com.br/123`
- No lado do servidor, o valor "123" seria extraído do parâmetro de rota para ser usado no processamento da requisição.
- Lembrando que, o backend tem que estar configurado pra receber esse valor.

Parâmetros de Consulta - Query Params

- Os parâmetros de consulta são usados para enviar dados como parte da URL de uma requisição GET. Eles são visíveis na própria URL após o ponto de interrogação ? e geralmente possuem o formato chave=valor, podendo incluir vários pares de chave e valor separados por &.
- Supondo que temos o endpoint `https://urlteste.com.br/`
- E que queremos passar um **id** de valor **123** e o valor **Walter** para a propriedade **nome** passamos esse valor (após a interrogação) com `https://urlteste.com.br?id=123&nome=walter`
- Lembrando que, o backend tem que estar configurado pra receber esse valor.

Body Params

- O corpo da requisição é onde os dados são enviados para o servidor em uma requisição HTTP, comumente utilizado em requisições do tipo POST, PUT, DELETE, entre outras.
- Dados no corpo da requisição podem ser em formatos como JSON, XML, formulários etc.
- Esse componente da requisição é invisível na URL e é útil para enviar grandes volumes de informações ou dados sensíveis que não devem ser visíveis na URL.
- Por exemplo, ao enviar um formulário, os dados do formulário (como nome, email, senha etc.) são frequentemente enviados no corpo da requisição.

Header Params

- Os cabeçalhos são **partes adicionais** de informações que são enviadas na requisição HTTP, contendo detalhes importantes sobre a requisição ou a resposta.
- Eles consistem em **pares chave-valor** e podem conter informações como o tipo de conteúdo aceito, autenticação, cache, tipo de codificação, idioma, entre outros detalhes.
- Alguns cabeçalhos comuns são **Content-Type**, **Authorization**, **Accept**, **User-Agent**, entre outros.

API x API REST

- Uma API (Application Programming Interface) é um **conjunto de definições, protocolos e ferramentas que permitem a comunicação e interação entre diferentes softwares**. Ela define como diferentes componentes de software devem interagir e se comunicar uns com os outros.
- Por outro lado, uma API REST (Representational State Transfer) é um **estilo arquitetural específico para o desenvolvimento de APIs na web**. Ela é baseada nos princípios do protocolo HTTP e define um **conjunto de restrições e diretrizes para criar serviços web**.

API REST – Segundo o chatGPT

Uma API RESTful segue os princípios do REST, incluindo:

- 1. Recursos identificados:** Cada recurso (como um objeto, dados ou serviço) é identificado por um URI (Uniform Resource Identifier).
- 2. Operações sobre recursos:** As operações (como GET, POST, PUT, DELETE) são usadas para manipular os recursos.
- 3. Representações de recursos:** As informações sobre um recurso podem ser representadas em vários formatos, como JSON, XML, HTML, entre outros, geralmente especificados através do cabeçalho "Accept" na requisição.
- 4. Sem estado (stateless):** Cada requisição do cliente para o servidor deve conter todas as informações necessárias para entender e processar a requisição. O servidor não mantém nenhum estado da sessão do cliente entre requisições.
- 5. Interface uniforme:** A API segue um conjunto de princípios de design para promover a simplicidade e a padronização.

API REST – Segundo Profissão Programador

- 1. Client / Server** – Cada um tem sua responsabilidade. Cada um tem seu papel e não deve ter preocupação com o papel do outro.
- 2. Stateless** – servidor não armazena o estado das requisições.
- 3. Cache** – a nossa aplicação deve ter condições de uma implementação de cache (armazenamento de informações temporárias).
- 4. Interface uniforme:** Identificação de recursos, Representação de recursos, Mensagens descritivas, hateoas.
- 5. Aplicação dividida em camadas.**
- 6. Entregar código sob demanda** – conforme a aplicação necessita.

Vamos programar

index.js (Replit) ou no servidor.js (VS Code)

```
js index.js

1 // Importa o express, puxa a dependência.
2 const express = require('express');
3
4 // É o express sendo executado.
5 // A partir da variável app conseguimos criar as primeiras rotas.
6 const app = express();
7
8 // O formato das rotas que iremos criar será assim...
9 app.get();
10 app.post();
11 app.patch();
```

Mas o que são essas rotas?

- Rotas em uma API REST representam os caminhos ou URLs específicos que o servidor utiliza para responder a requisições feitas pelos clientes.
- Cada rota está associada a um recurso ou a uma operação específica que a API oferece.
- Elas definem como a API responde a diferentes requisições HTTP em um servidor.

index.js (Replit) ou no servidor.js (VS Code)

Note que colocar apenas **app.get()** não é suficiente para criar a rota. Precisamos de mais parâmetros. O 1º argumento é o caminho, quando tem apenas / indica que é o mesmo endereço da aplicação.

```
app.get('/', (request, response) => {  
    response.send('Você acessou o servidor');  
})
```

Na seta =>
temos uma
arrow function

O 2º argumento são o **request** e **response**. Toda requisição tem uma **request** e uma **response**. Request é o que o cliente está requisitando e o **response** é a resposta que o servidor irá mostrar.

Arrow Functions

- As arrow functions (funções de seta) em JavaScript são uma forma concisa de escrever funções.
- Ela geralmente consiste apenas nos parâmetros e na expressão que deve ser retornada.
- Aqui está um exemplo simples de uma arrow function que adiciona dois números:

```
const soma = (a, b) => a + b;
```

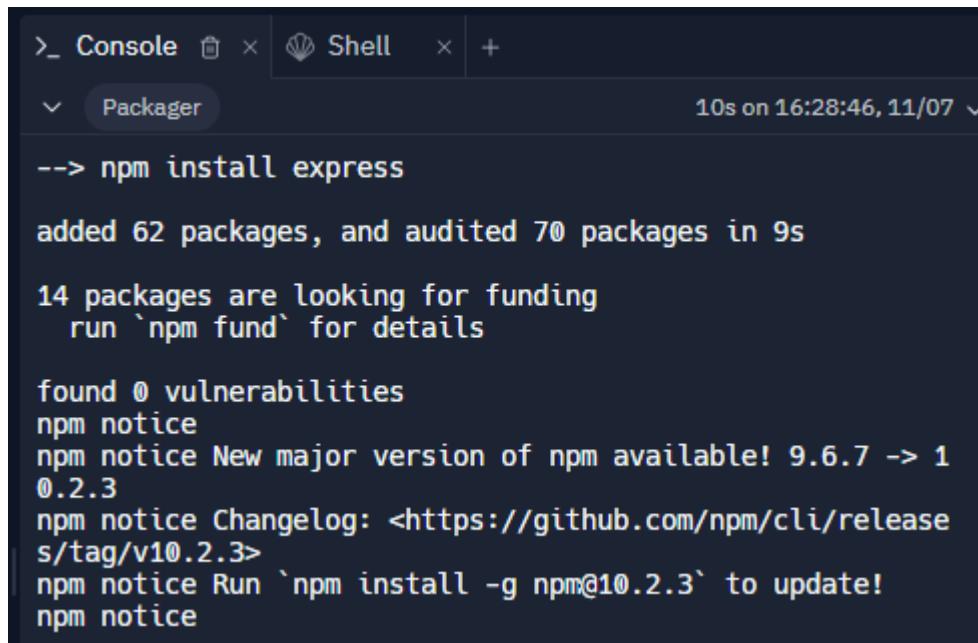
index.js (Replit) ou no servidor.js (VS Code)

```
.js index.js

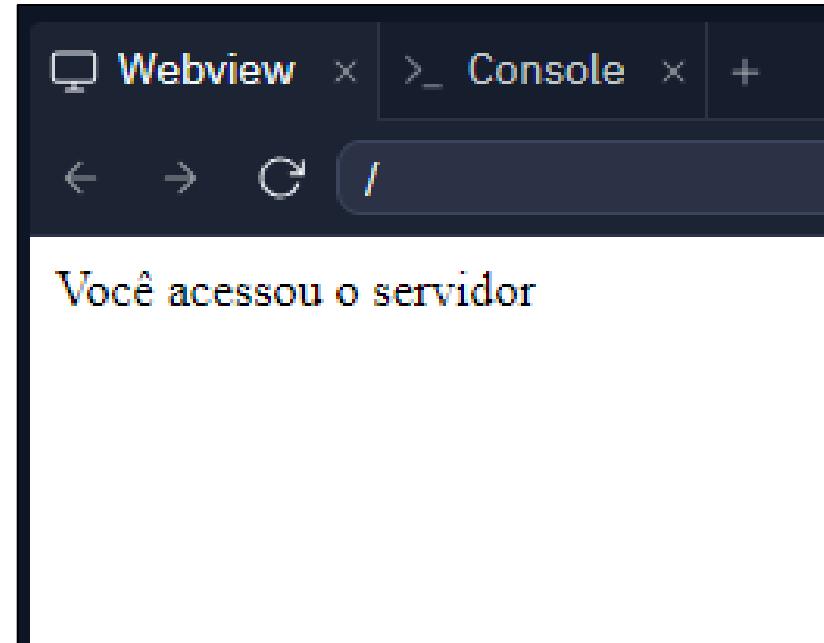
1 const express = require('express');
2 const app = express();
3
4 app.get('/', (request, response) => {
5   response.send('Você acessou o servidor');
6 }
7
8 // Indica a porta que o servidor irá rodar
9 // Para o VS Code, no navegador acessamos localhost:4000
10 // No Replit, é só contemplar o resultado após apertar Run
11 app.listen(4000);
```

Para executar

- No console do VS Code: **node .\src\servidor.js**
- No Replit, aperta **Run** e aguarda a instalação do Expresss



```
>_ Console  x | Shell  x | +  
10s on 16:28:46, 11/07 ✓  
✓ Packager  
--> npm install express  
added 62 packages, and audited 70 packages in 9s  
14 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
npm notice  
npm notice New major version of npm available! 9.6.7 -> 1  
0.2.3  
npm notice Changelog: <https://github.com/npm/cli/releases/tag/v10.2.3>  
npm notice Run `npm install -g npm@10.2.3` to update!  
npm notice
```



```
Webview  x | >_ Console  x | +  
← → ⌂ /  
  
Você acessou o servidor
```

Lembrando...

- Sempre que fizer alteração no VS Code, pare a aplicação, salve e rode novamente.
- Existem recursos para agilizar esse processo, mas tenham em mente que se a aplicação não recebe sua atualização no código, talvez seja preciso parar manualmente, salvar e executar para ver a alteração.

E se eu quiser mandar os dados de um usuário para o servidor?

```
js index.js

1 const express = require('express');
2 const app = express();
3
4 // Resolve possíveis problemas no uso do JSON
5 app.use(express.json());
6
7 // No Replit, se não funcionar, abram a aplicação numa nova aba
8 app.get('/', (request, response) => {
9   response.json({name : 'Walter', age: 42});
10 })
11
12 app.listen(4000);
```

```
{"name": "Walter", "age": 42}
```

Agora vamos testar uma rota POST com o Postman

Primeiro vamos testar com GET, depois iremos criar uma requisição com POST.

Abram o Postman

Clicando no + será aberta a aba para criar uma nova requisição.

No VS Code, usem o endereço localhost:4000

The image shows two screenshots of the Postman application. The left screenshot displays the main workspace interface with a red box highlighting the '+' button in the top navigation bar, indicating where to click to create a new request. The right screenshot shows a detailed view of a request configuration for 'localhost:4000' using a GET method. A red box highlights the URL field. Below it, the 'Body' tab is selected, showing a JSON response with the following data:

```
1  "name": "Walter",  
2  "age": 42  
3    
4  
```

No repl.it...

Coloquem o endereço que aparece quando abrimos a aplicação numa nova aba.

The screenshot shows a POSTMAN interface with the following details:

- Method: GET
- URL: <https://tragicsspecializedhypermedia.waltertravassos.repl.co/> (highlighted with a red box)
- Params tab is selected.
- Headers tab shows 6 items.
- Body tab is selected.
- Test Results tab is selected.
- Pretty tab is selected in the preview section.
- Raw tab is available in the preview section.
- Preview tab is available in the preview section.
- Visualize tab is available in the preview section.
- JSON dropdown is set to JSON.
- Raw dropdown is available in the preview section.
- Preview area displays the following JSON response:

```
1 {  
2   "name": "Walter",  
3   "age": 42  
4 }
```

**Agora vamos criar uma
requisição POST**

index.js (Replit) ou no servidor.js (VS Code)

```
JS index.js

1 const express = require('express');
2 const app = express();
3 app.use(express.json());
4
5 app.get('/', (request, response) => {
6   response.json({name : 'Walter', age: 42});
7 })
8
9 // Agora a rota será /userdata no final da URL
10 app.post('/userdata', (request, response) => {
11   // Vai imprimir o corpo da requisição do PostMan no console
12   console.log(request.body)
13   response.status(200).json({sucess: true})
14 })
15
16 app.listen(4000);
```

A linha 13 vai enviar um status de sucesso para o Postman.

Caso ela não exista, o Postman fica enviando em loop a requisição.

Requisição POST

POST

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

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

```
1
2   "name": "Walter",
3   "age": 42,
4   "city": "João Pessoa"
5
```

Body Cookies Headers (8) Test Results

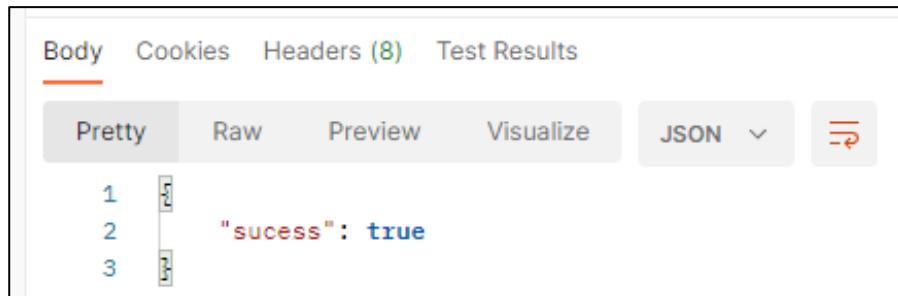
Pretty Raw Preview Visualize JSON

```
1
2   "success": true
3
```

- Primeiro coloca a URL com o **\userdata** no final. Se for no VS Code será: **localhost:4000\userdata**.
- Depois clica no raw, altera de text para JSON.
- Insere o JSON.

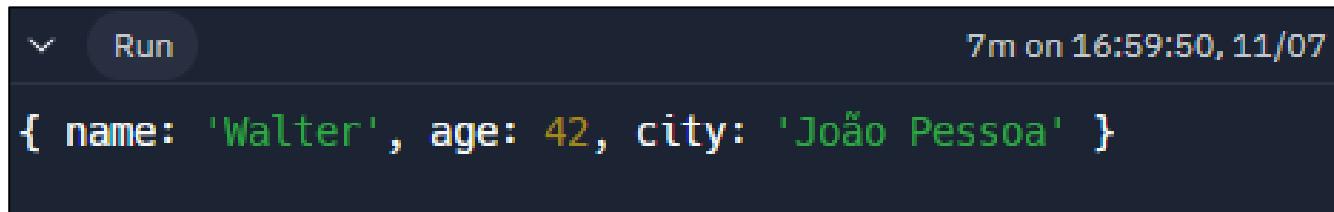
Atenção...

- Verifique se a aplicação no Replit ou VS Code está rodando antes de apertar o **SEND** no Postman.
- No Postman você deve receber a resposta de sucesso.



A screenshot of the Postman interface showing the 'Body' tab. The response is a JSON object with three lines of code:
1
2 "sucess": true
3

- E no console do Replit, o corpo do JSON que você inseriu no Postman.



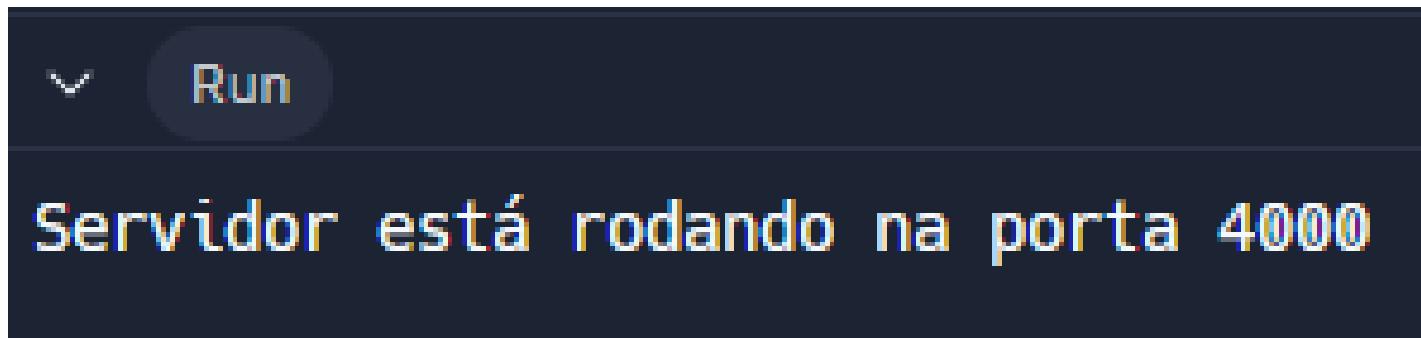
A screenshot of the Replit terminal window. At the top, there is a 'Run' button and a timestamp '7m on 16:59:50, 11/07'. Below the timestamp, the terminal shows the JSON object that was inserted:

```
{ name: 'Walter', age: 42, city: 'João Pessoa' }
```

Por fim...

Podemos deixar uma mensagem que a aplicação está rodando no console.

```
16 v app.listen(4000, () => {  
17   |   console.log('Servidor está rodando na porta 4000');  
18 });
```



DESIGN DE INTERFACE PARA WEB

AULA 12 – INTRODUÇÃO AO NODEJS



WALTER TRAVASSOS SARINHO

@WALTEROPROFESSOR

WALTER.TRAVASSOS@UNIPE.EDU.BR