

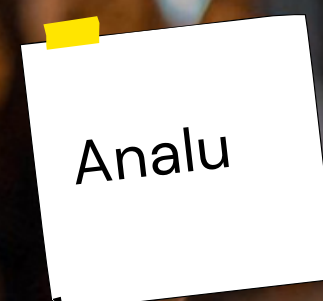
Projeto reprogramaFlix API – GET



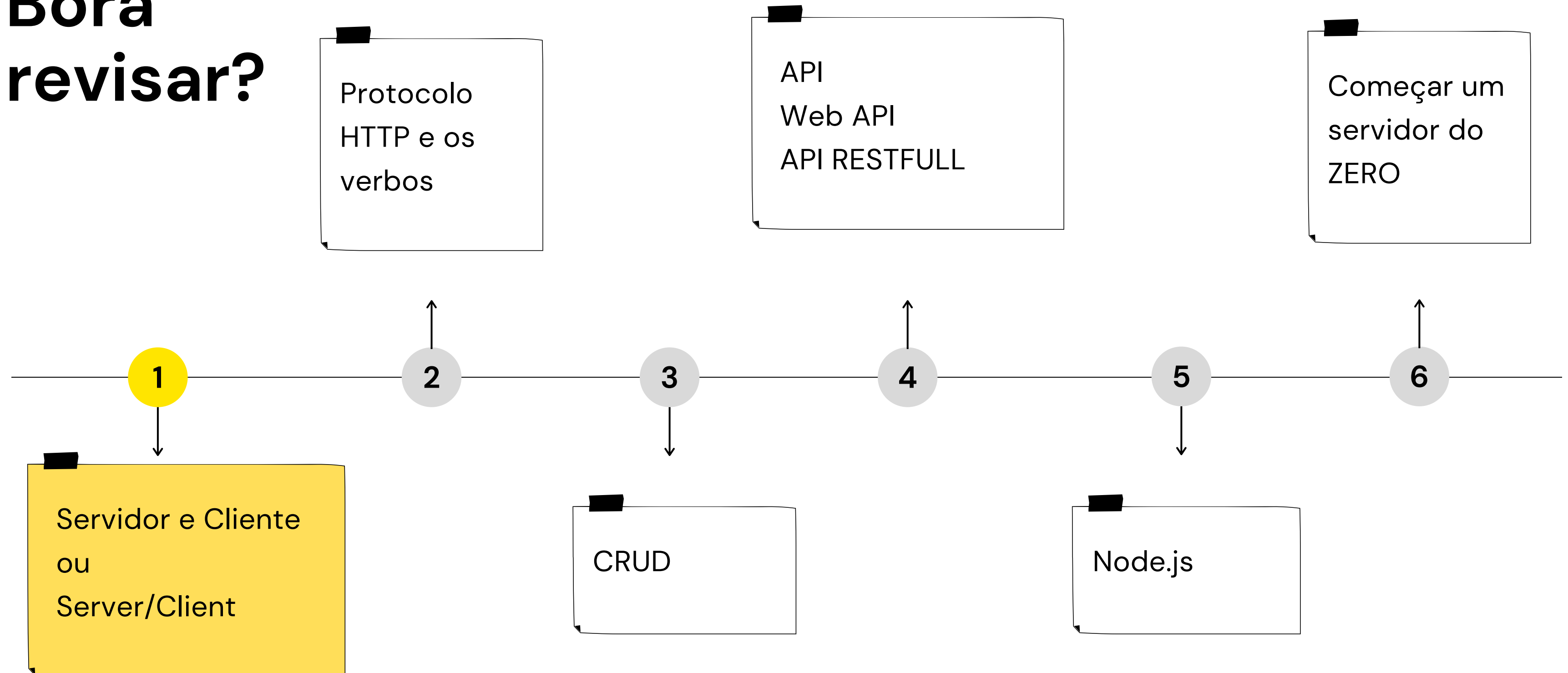
Roi, turma 9 né?

Eu sou desenvolvedora back-end.
Hoje sou Engenheira de Software trabalhando
principalmente com Java no Banco Itaú.

Pra me achar você pode procurar por
@analu.io no instagram ou por
@sampaioaanaluiza no linkedin e medium



Bora revisar?



Modelo Server-Client

CLIENTE

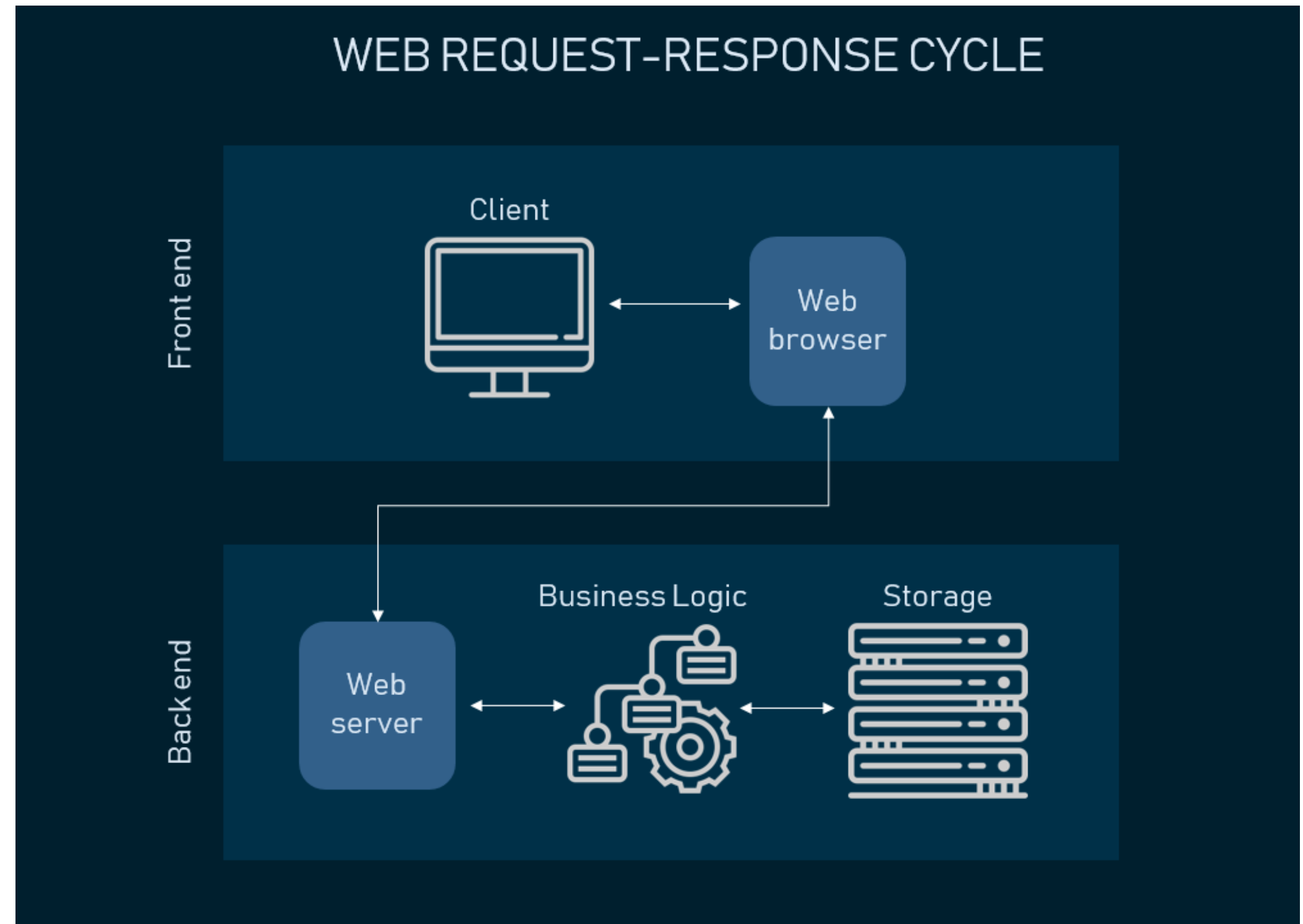
Entenda cliente como a interface que o usuário interage. É o Cliente que **solicita** serviços e informações de um ou mais servidores.

SERVIDOR

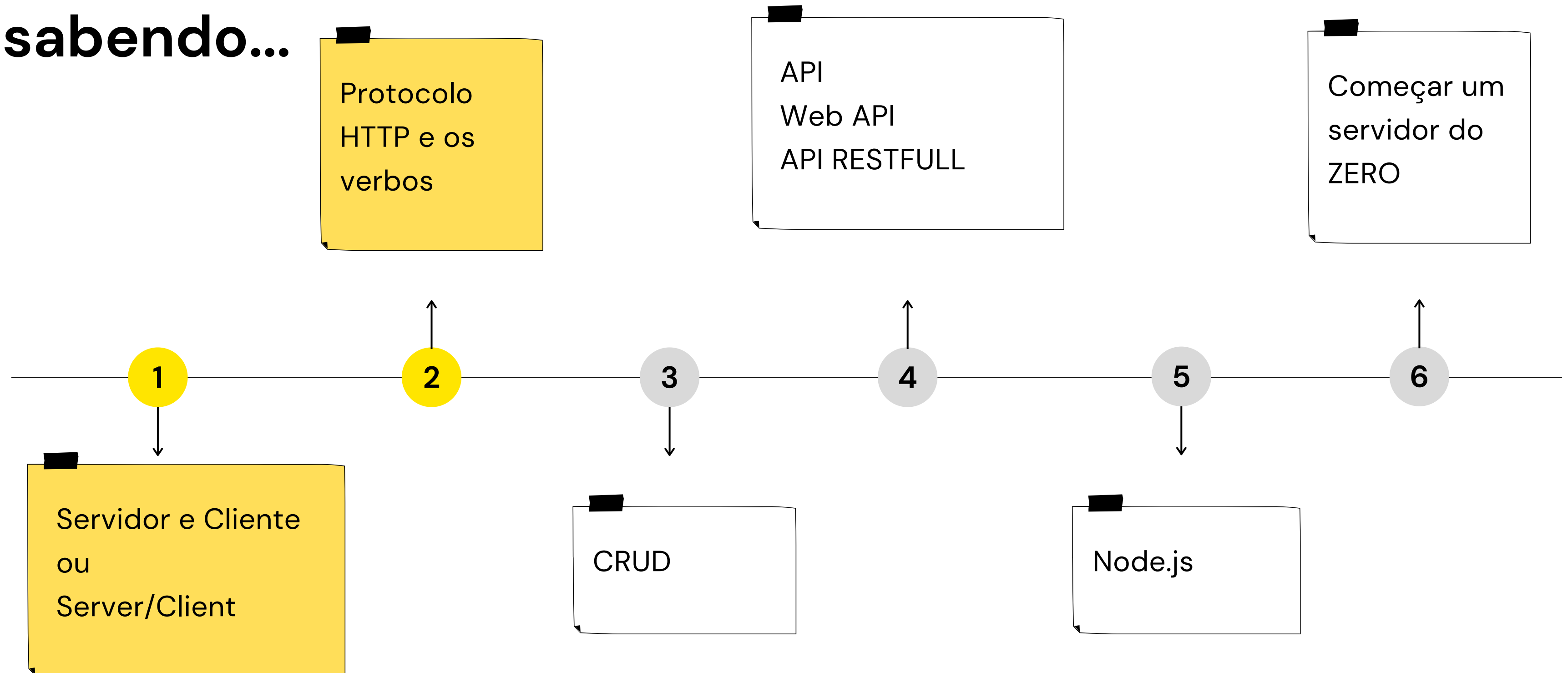
E o Servidor é o responsável pelo processo, organização e gerenciamento das informações. É ele que **responde** às solicitações feitas pelo usuário.

Ele é um processo reativo, disparado pela chegada de pedidos de seus clientes.

Revisão



Já tamo sabendo...



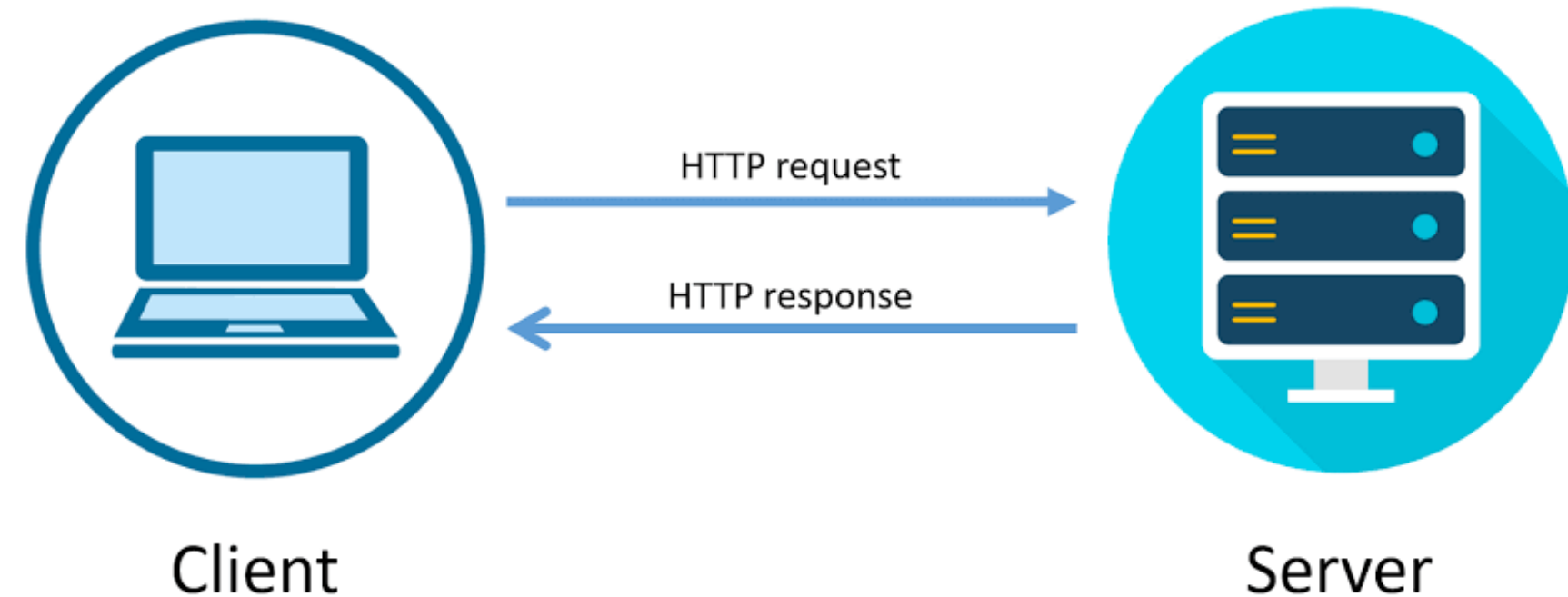
HTTP

Protocolo de Transferência de Hipertexto é um protocolo usado dentro do modelo Client/Server é baseado em pedidos (requests) e respostas (responses).

Ele é a forma em que o Cliente e o Servidor se comunicam.

Pensando em uniformizar a comunicação entre servidores e clientes foram criados **códigos** e **verbos** que são usados por ambas as partes, e essas requisições são feitas em **URLs** que possuem uma estrutura específica.

<protocolo>://<servidor>:<porta>/<recurso>



Revisão



HTTP – Status Code

Quando o Client faz uma requisição o Server responde com um código de status numérico também padronizado.

Os códigos de status das respostas HTTP indicam se uma requisição HTTP foi concluída. As respostas são agrupadas em cinco classes:

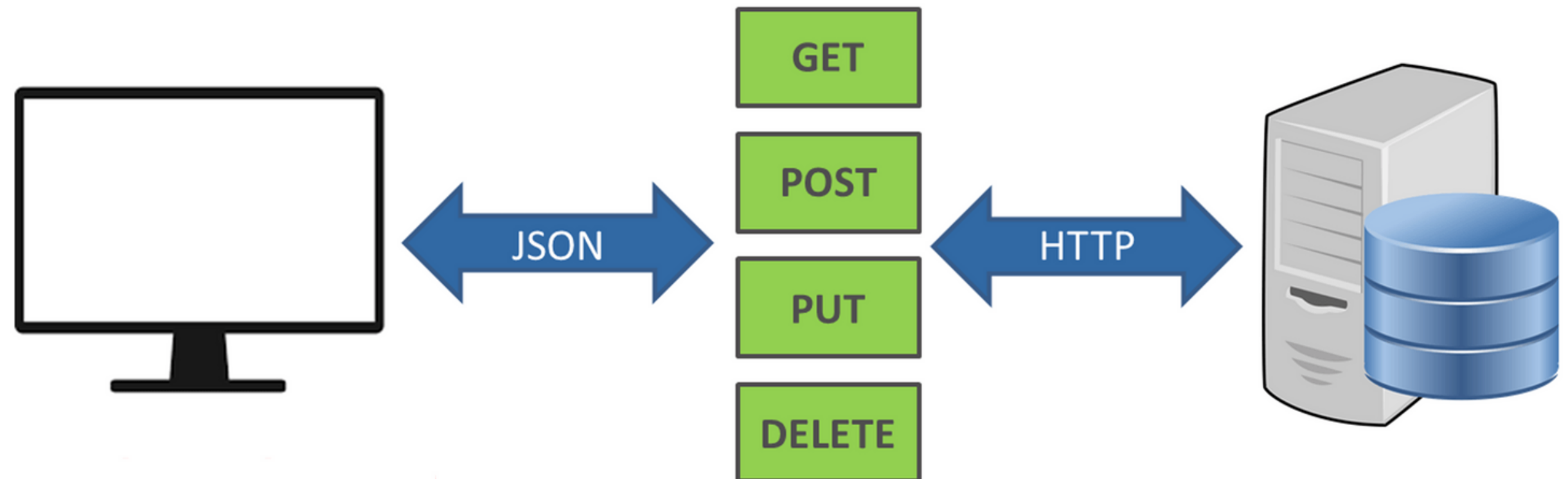
É a desenvolvedora Back end que coloca na construção do servidor quais serão as situações referentes a cada resposta.

Respostas de informação (100–199)
Respostas de sucesso (200–299)
Redirecionamentos (300–399)
Erros do cliente (400–499)
Erros do servidor (500–599)

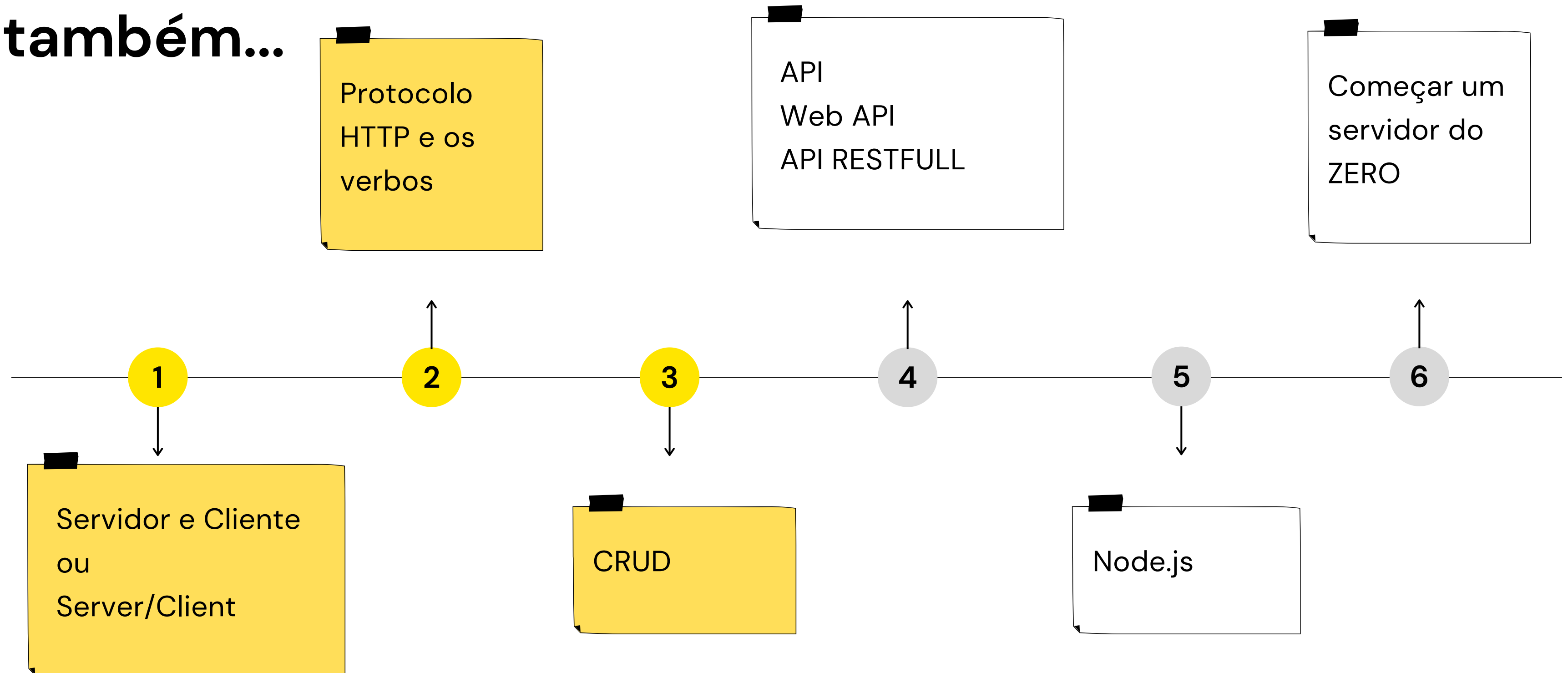
HTTP – Verbos

Os verbos HTTP são um conjunto de métodos de requisição responsáveis por indicar a ação a ser executada.

O Client manda um **request** solicitando um dos verbos e o Server deve estar preparado para receber e responde-lo com um **response**.



Lembrei também...



HTTP – CRUD

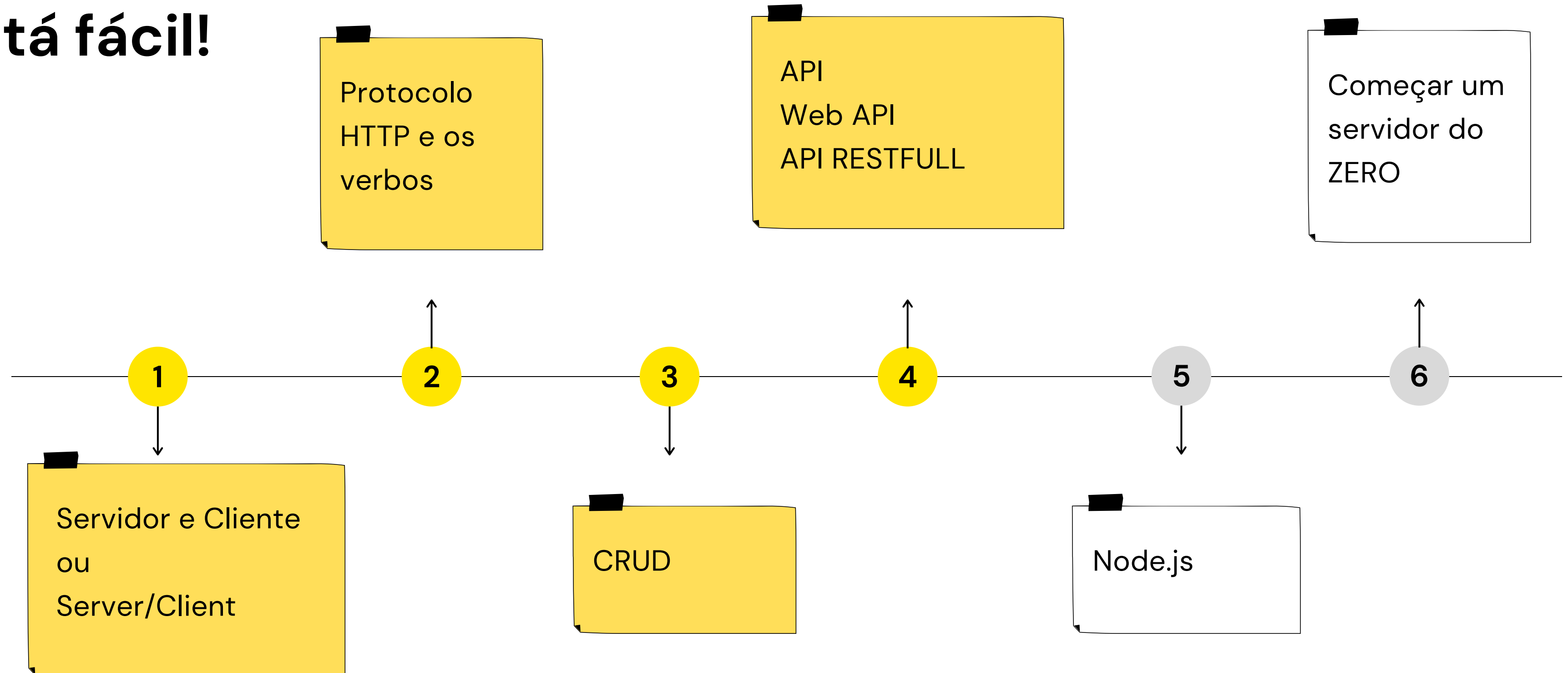
CRUD é a composição da primeira letra de 4 operações básicas de um banco de dados, e são o que a maioria das aplicação faz

- ✓ C: Create (criar) – criar um novo registro
- R: Read (ler) – exibir as informações de um registro
- ♻ U: Update (atualizar) – atualizar os dados do registro
- ✕ D: Delete (apagar) – apagar um registro

Operações CRUD com HTTP

Verbos HTTP	Operação CRUD
GET	Ler
POST	Criar
PUT	Substituir
PATCH	Modificar
DELETE	Excluir

Essa tá fácil!



API

Interface de Programação de Aplicativos

API busca criar formas e ferramentas de se usar uma funcionalidade ou uma informação sem realmente ter que "reinventar a tal função."

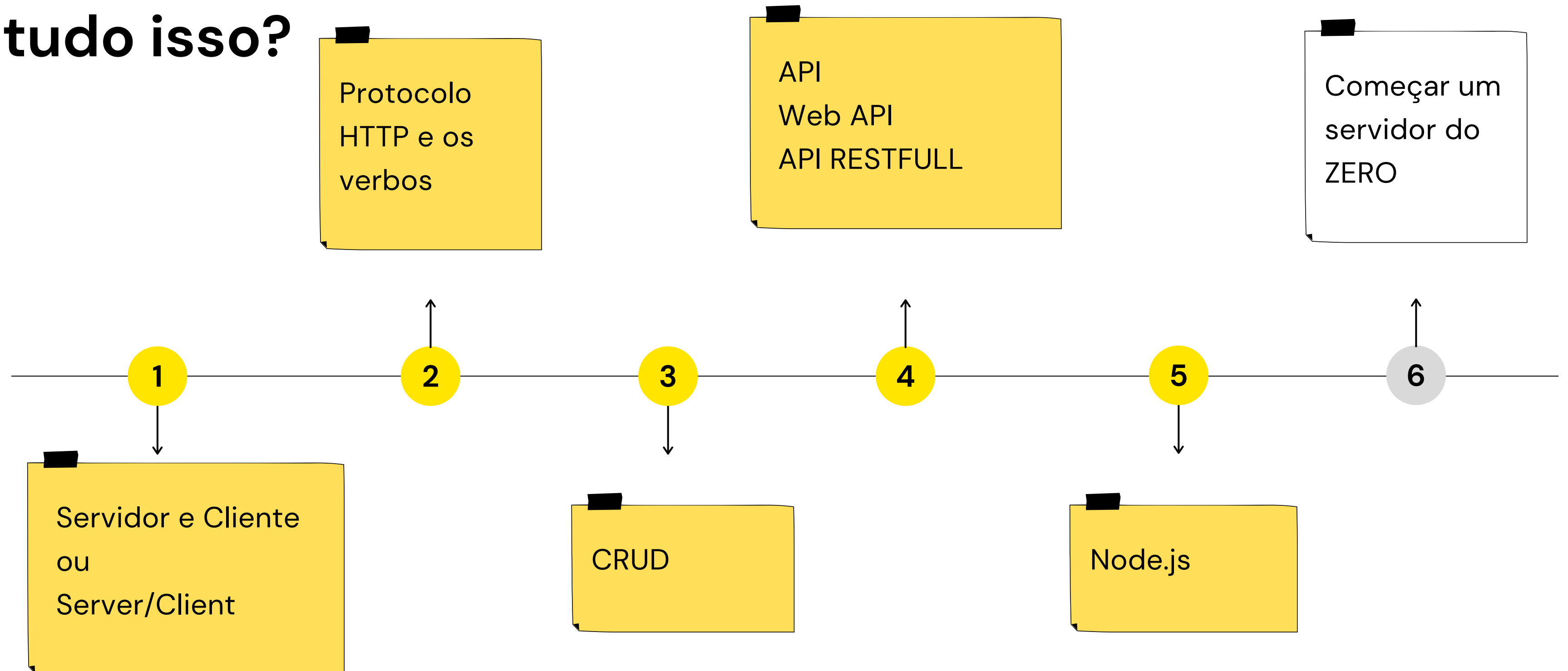
Ela não necessariamente está num link na Web, ela pode ser uma lib ou um framework, uma função já pronta em uma linguagem específica, etc.

Web API e API REST

Web API é uma interface que é disponibilizada de forma remota, pela web, que possibilita a programação aplicativos e softwares.

E as APIs RESTfull são aquelas que são capazes de fazer o REST. Que nada mais é uma API que usa os protocolos HTTP para comunicação entre o usuário e o servidor.

Eu sei tudo isso?



Node.js

O JavaScript pra servidor

Interpretador JavaScript que não precisa de navegador.

Ele pode:

Ler e escrever arquivos no seu computador

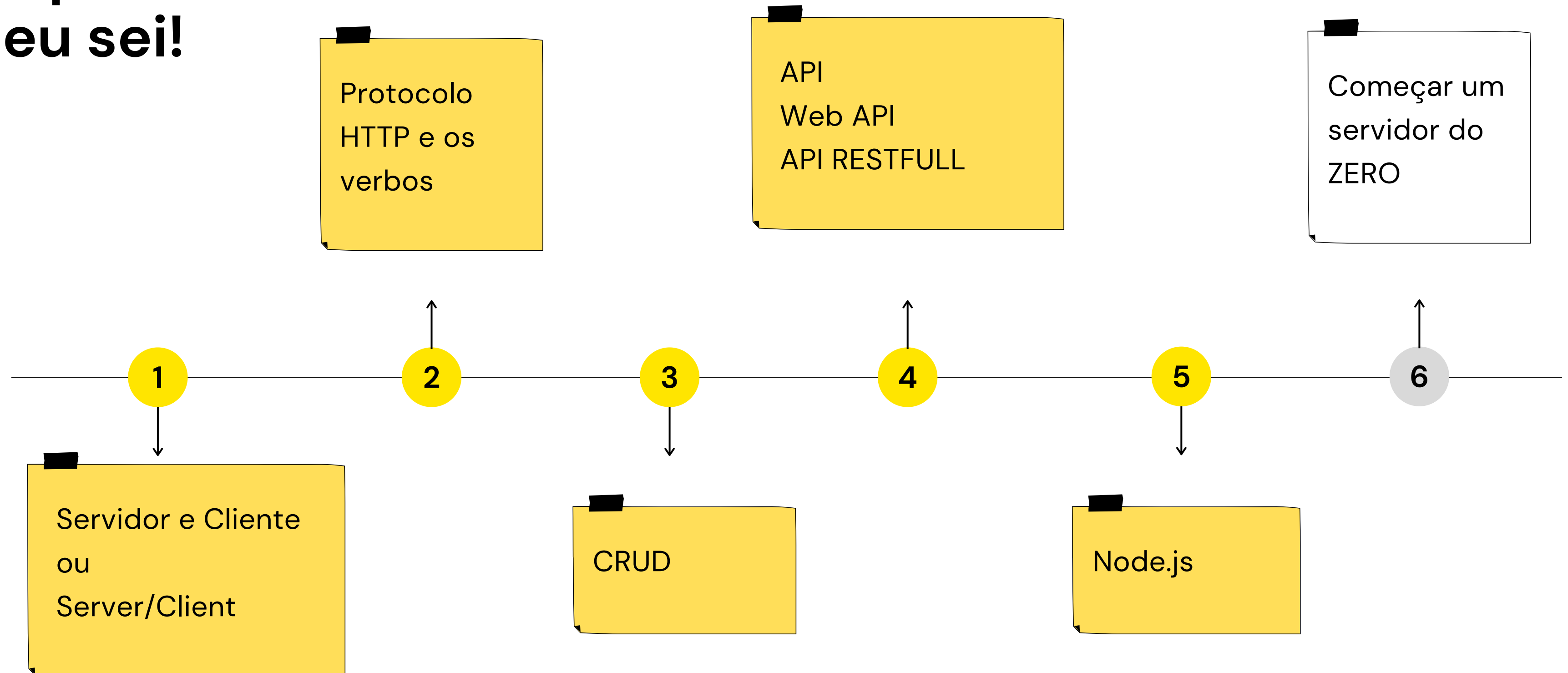
Conectar com um banco de dados

Se comportar como um servidor

Gerenciadores de pacotes

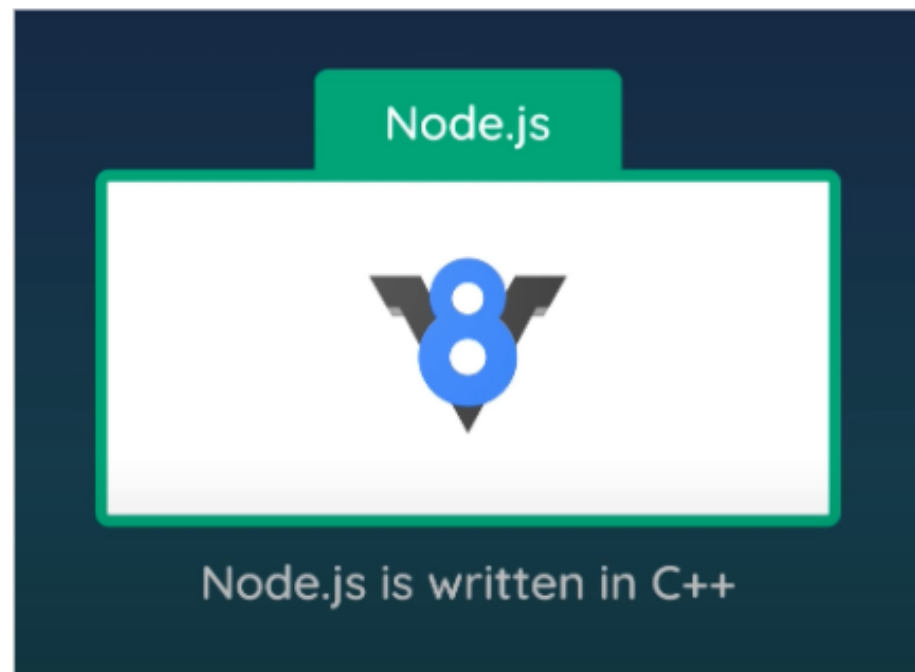


Opa, essa eu sei!



Node.js

Antes de tudo, um pouco de história!
Tudo começou em 2009...

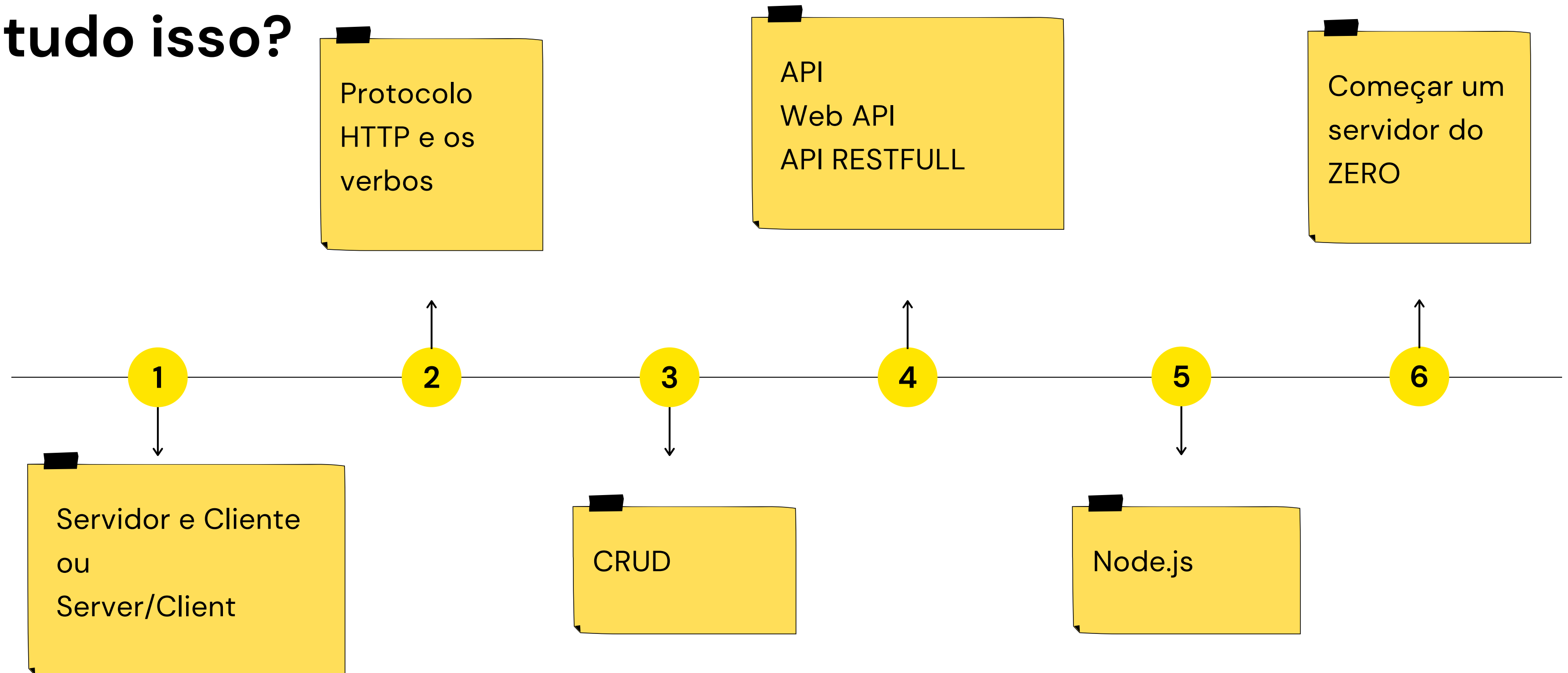


Interpretador JavaScript que não precisa de navegador.

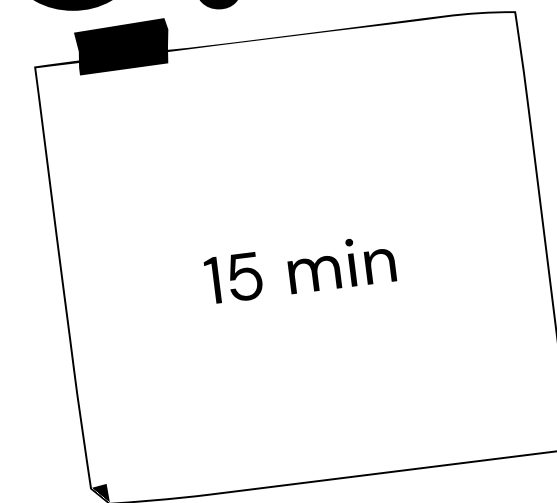
Ele pode:

- Ler e escrever arquivos no seu computador
- Conectar com um banco de dados
- Se comportar como um servidor

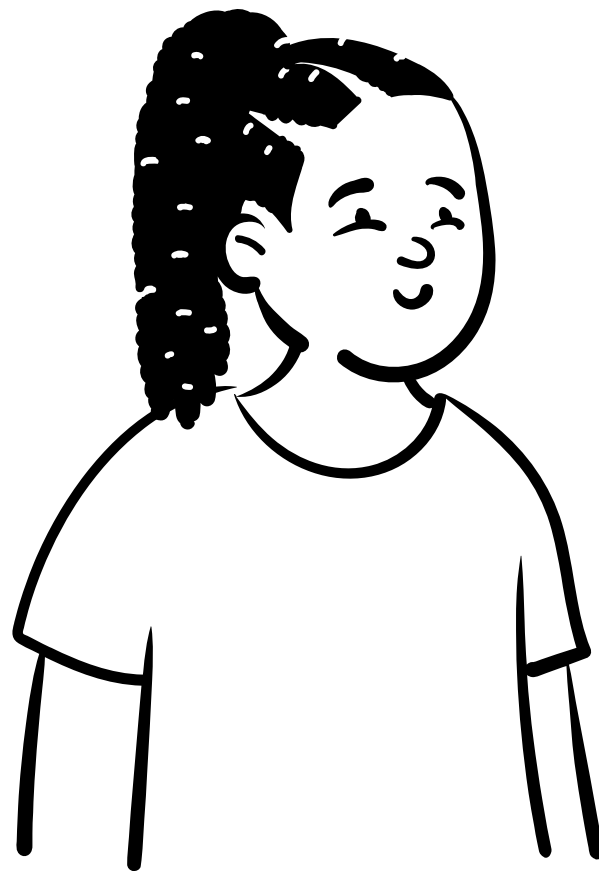
Eita, eu sei tudo isso?



**Ai... vamo para
um pouquinho?**



Criando nosso Server



UHUUUL

npm init

Server

```
Ana Luiza @DESKTOP MINGW64 /d/workspace/on6-xp-s7-api-get/servidor-em-aula (master)
$ npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.

See `npm help init` for definitive documentation on these fields
and exactly what they do.

Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.

Press ^C at any time to quit.
package name: (servidor-em-aula)
version: (1.0.0)
description:
entry point: (server.js)
test command:
git repository:
keywords:
author:
license: (ISC)
About to write to D:\workspace\on6-xp-s7-api-get\servidor-em-aula\package.json:

{
  "name": "servidor-em-aula",
  "version": "1.0.0",
  "description": "",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "",
  "license": "ISC"
}

Is this OK? (yes)
```

npm init

A screenshot of a code editor with three tabs at the top: 'script.js ...\02-exe-pokemon', 'script.js ...\01-exe-ghibli', and 'package.json X'. The 'package.json' tab is active, showing a JSON object for a project named 'servidor-em-aula'. The code is as follows:

```
servidor-em-aula > {} package.json > ...
1  {
2    "name": "servidor-em-aula",
3    "version": "1.0.0",
4    "description": "",
5    "main": "server.js",
6    "scripts": {
7      "test": "echo \"Error: no test specified\" && exit 1",
8      "start": "node server.js"
9    },
10   "author": "",
11   "license": "ISC"
12 }
13
```

Dependências


só um pouquinho delas



Express


express

4.17.1 • Public • Published a year ago

 [Readme](#)

 [Explore](#) BETA

 30 Dependencies

 46.033 Dependents

 264 Versions

express

Fast, unopinionated, minimalist web framework for **node**.

npm v4.17.1 downloads 58M/month linux passing windows passing coverage 100%

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

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

Install

```
> npm i express
```

± Weekly Downloads

13.961.907

Version

4.17.1

License

MIT

Unpacked Size

208 kB

Total Files

16

Issues

97

Pull Requests

52

[Homemage](#)


Server

nodemon

pra parar de ficar
atualizando


nodemon

2.0.4 • Public • Published 4 months ago

 [Readme](#)

 [Explore](#) BETA

 10 Dependencies

 2.477 Dependents

 215 Versions



nodemon

nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.

Install

```
> npm i nodemon
```

♥ [Fund this package](#)

Weekly Downloads

2.893.116



Version

2.0.4

License

MIT

Unpacked Size

107 kB

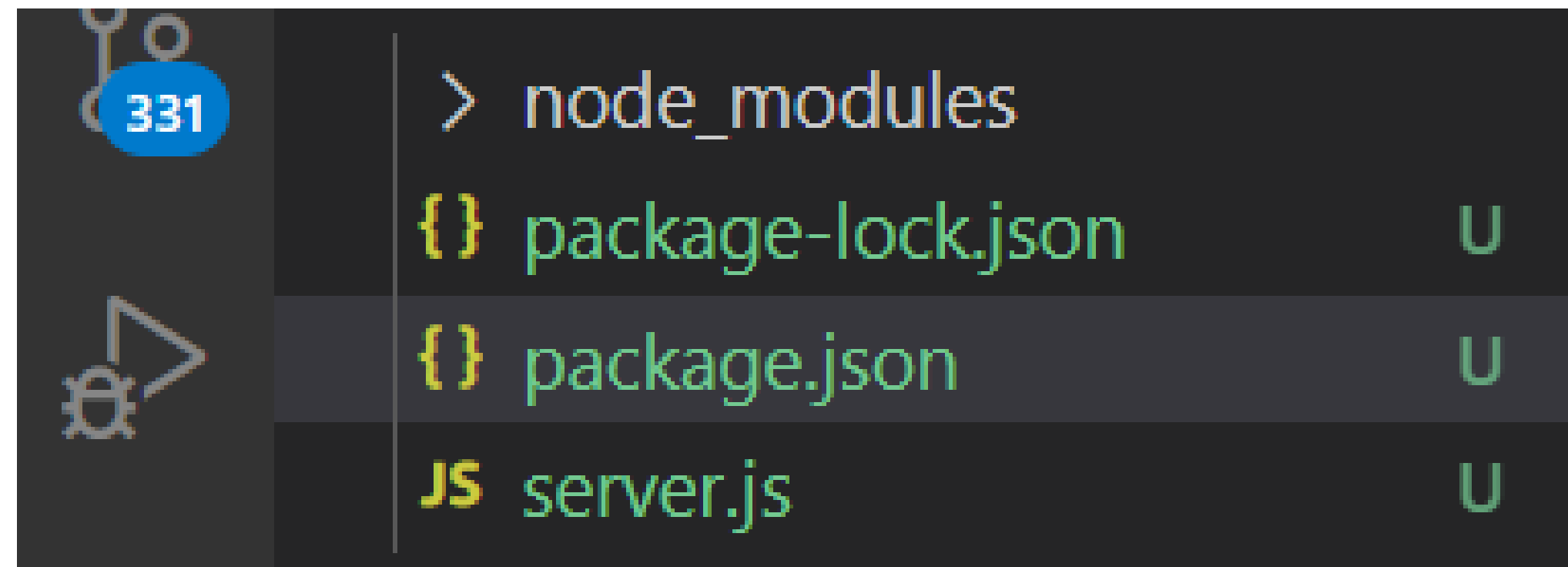
Total Files

43

Server

Package.json e package-lock

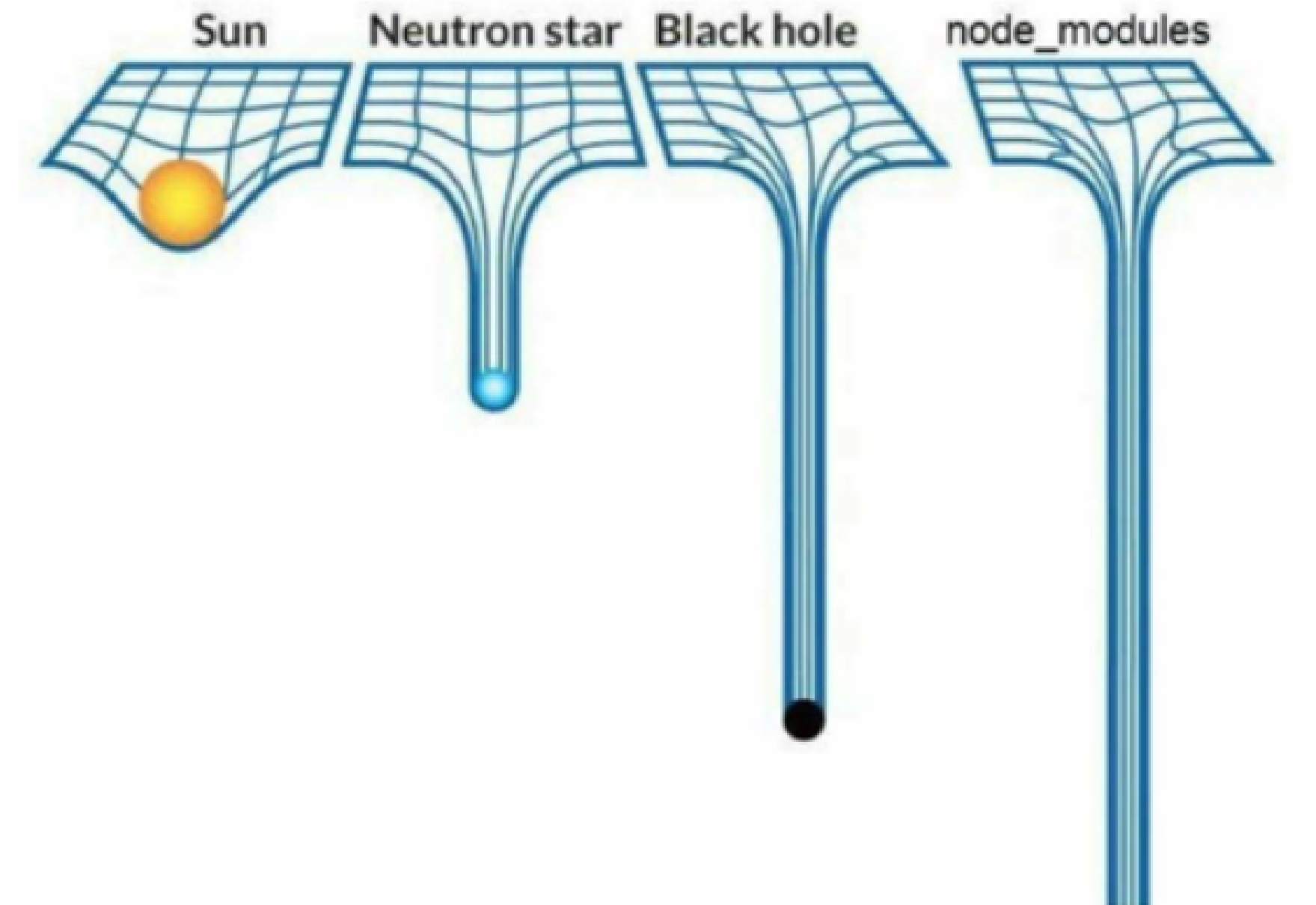
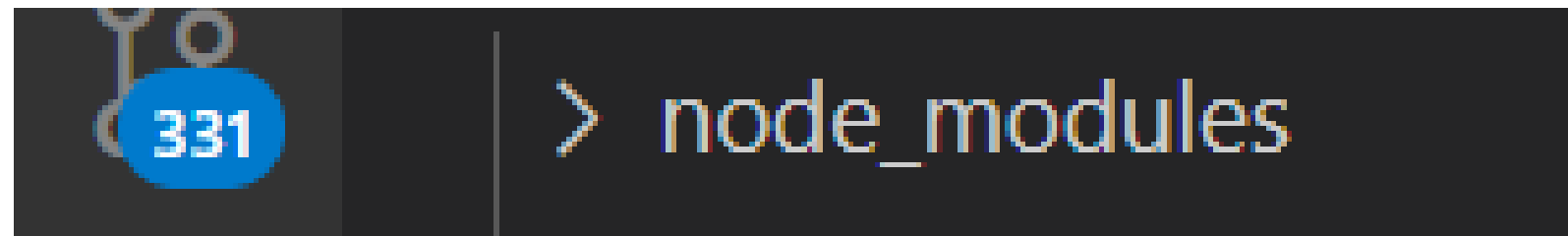
pacote o que?



```
> node_modules
{} package-lock.json      U
{} package.json           U
JS server.js              U
```

node_modules

a grande. Não, não! A GIGANTESCA!



.gitignore

vamo ignorar ela gente

Arquitetura e hierarquia de pastas

```
1 const express = require('express');
2 const app = express();
3 const porta = 3000;
4 const estudantes = require('./estudantes.json')
5
6 //@router GET /estudantes
7 //@desc Retorna todos os estudantes
8 //@access Public
9 app.get('/estudantes', function(requisicao, resposta){
10   resposta.json(estudantes);
11 });
12
13
14 app.use(express.json());
15 app.listen(porta, function(){
16   console.log("Servidor rodando na porta" + porta)
17 });
```

Arquitetura e hierarquia de pastas

Arquitetura

```
1 const express = require('express');
2 const app = express();
3 const porta = 3000;
4 const estudantes = require('./estudantes.json')
5 const professores = require('./professores.json')
6 const materias = require('./materias.json')
7
8 //@router GET /estudantes
9 //@desc Retorna todos os estudantes
10 //@access Public
11 app.get('/estudantes', function(requisicao, resposta){
12   resposta.json(estudantes);
13 });
14
15 //@router GET /professores
16 //@desc Retorna todos os professores
17 //@access Public
18 app.get('/professores', function(requisicao, resposta){
19   resposta.json(professores);
20 });
21
22 //@router GET /matérias
23 //@desc Retorna todas as matérias
24 //@access Public
25 app.get('/materias', function(requisicao, resposta){
26   resposta.json(materias);
27 });
28
29 app.use(express.json());
30 app.listen(porta, function(){
31   console.log("Servidor rodando na porta" + porta)
32 });
```


Arquitetura e hierarquia de pastas

Arquitetura

26 setembro de 2020

```
1 const express = require('express');
2 const app = express();
3 const porta = 3000;
4 const estudantes = require('./estudantes.json')
5 const professores = require('./professores.json')
6 const materias = require('./materias.json')
7
8 //@router GET /estudantes
9 //@desc Retorna todos os estudantes
10 //@access Public
11 app.get('/estudantes', function(requisicao, resposta){
12   resposta.json(estudantes);
13 });
14
15 //@router POST /estudantes
16 //@desc Retorna todos os estudantes
17 //@access Public
18 app.post('/estudantes', function(requisicao, resposta){
19   CODIGO CODIGO
20   resposta.json(estudantes);
21 });
22
23 //@router GET /professores
24 //@desc Retorna todos os professores
25 //@access Public
26 app.get('/professores', function(requisicao, resposta){
27   resposta.json(professores);
28 });
29
30 //@router POST /professores
31 //@desc Retorna todos os professores
32 //@access Public
33 app.post('/professores', function(requisicao, resposta){
34   CODIGO CODIGO
35   resposta.json(professores);
36 });
37
38 //@router GET /matérias
39 //@desc Retorna todas as matérias
40 //@access Public
41 app.get('/materias', function(requisicao, resposta){
42   resposta.json(materias);
43 });
44
45 app.use(express.json());
46 app.listen(porta, function(){
47   console.log("Servidor rodando na porta" + porta)
48 });
```

Arquitetura – MVC

MVC é um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário(view), a camada de manipulação dos **dados(model)** e a camada de **controle(controller)**

Já que estamos lidando com um projeto que tem somente back-end, não lidaremos com as views, porém lidaremos com as **rotas (routes)**.

O MVC nada mais é que uma forma de **organizar** o nosso código.

```
\--[] NOME-DO-SEU-SERVIDOR
|   server.js
|
|--[] src
|   app.js
|
|   |--controller
|       NOMEController.js
|
|   |--model
|       NOME.json
|
|   |--routes
|       NOMERoute.js
```

No final nosso projeto fica assim:

MVC é um padrão de arquitetura de software, separando sua aplicação em 3 camadas. A camada de interação do usuário(view), a camada de manipulação dos **dados(model)** e a camada de **controle(controller)**

Já que estamos lidando com um projeto que tem somente back-end, não lidaremos com as views, porém lidaremos com as **rotas (routes)**.

O MVC nada mais é que uma forma de **organizar** o nosso código.

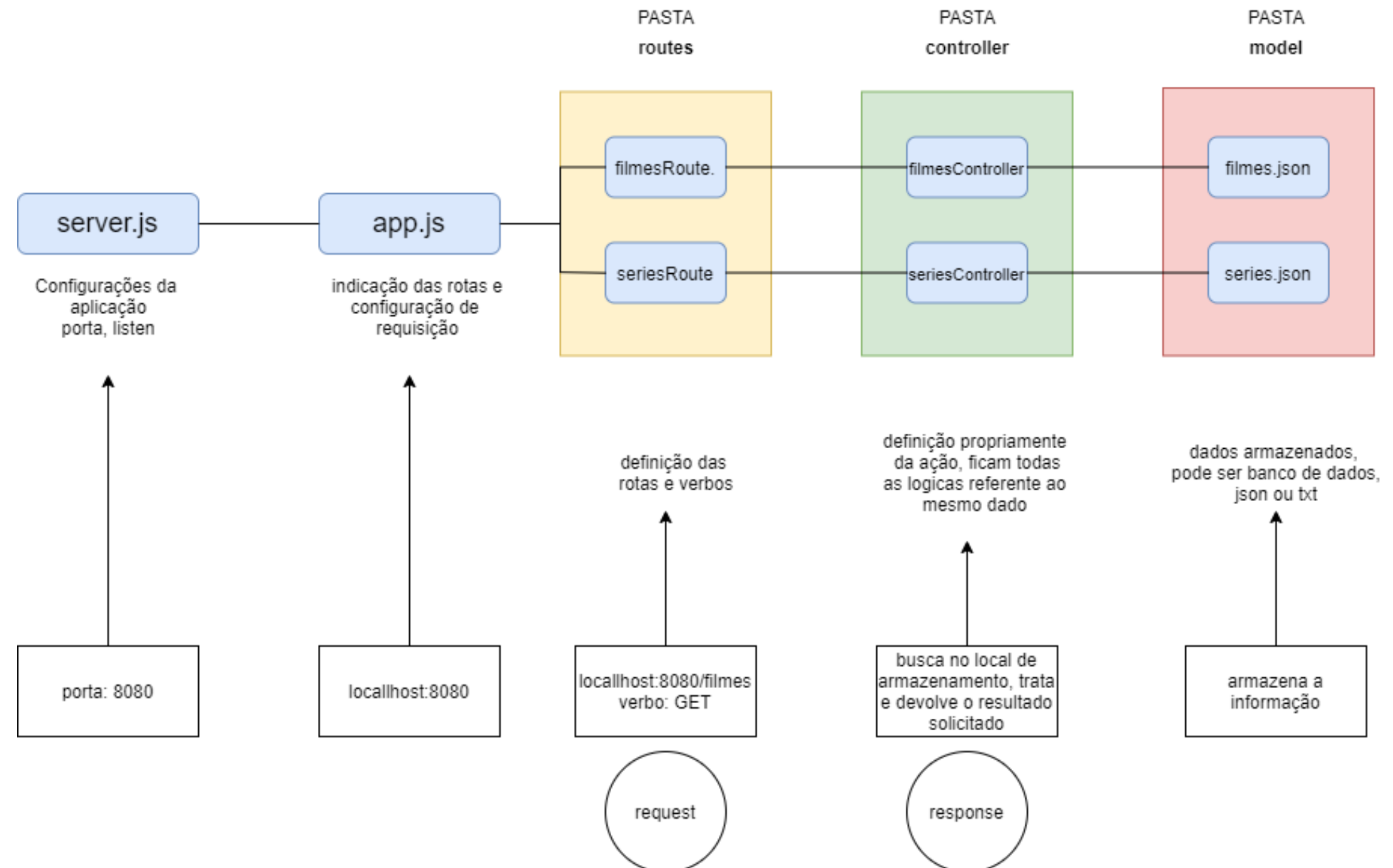
```
\--[] NOME-DO-SEU-SERVIDOR
| .gitignore
| package-lock.json
| package.json
| server.js
\--[] node_modules
\--[] src
| app.js
|
|---controller
|   NOMEController.js
|
|---model
|   NOME.json
|
|---routes
|   NOMERoute.js
```

MVC

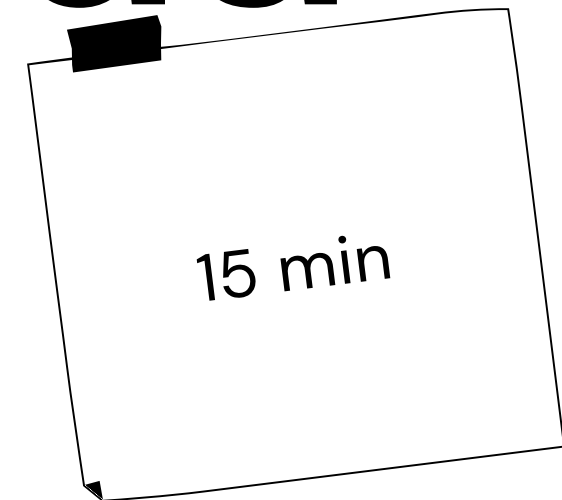
No final nosso projeto fica assim:

```
\--□ NOME-DO-SEU-SERVIDOR
| .gitignore
| package-lock.json
| package.json
| server.js
\--□ node_modules
\--□ src
| app.js
|
| ---controller
|   NOMEController.js
|
| ---model
|   NOME.json
|
| ---routes
|   NOMERoute.js
```

MVC



Calma, deixa eu beber uma água



Agora sim, o projeto de hoje

FINALMENTEEE

