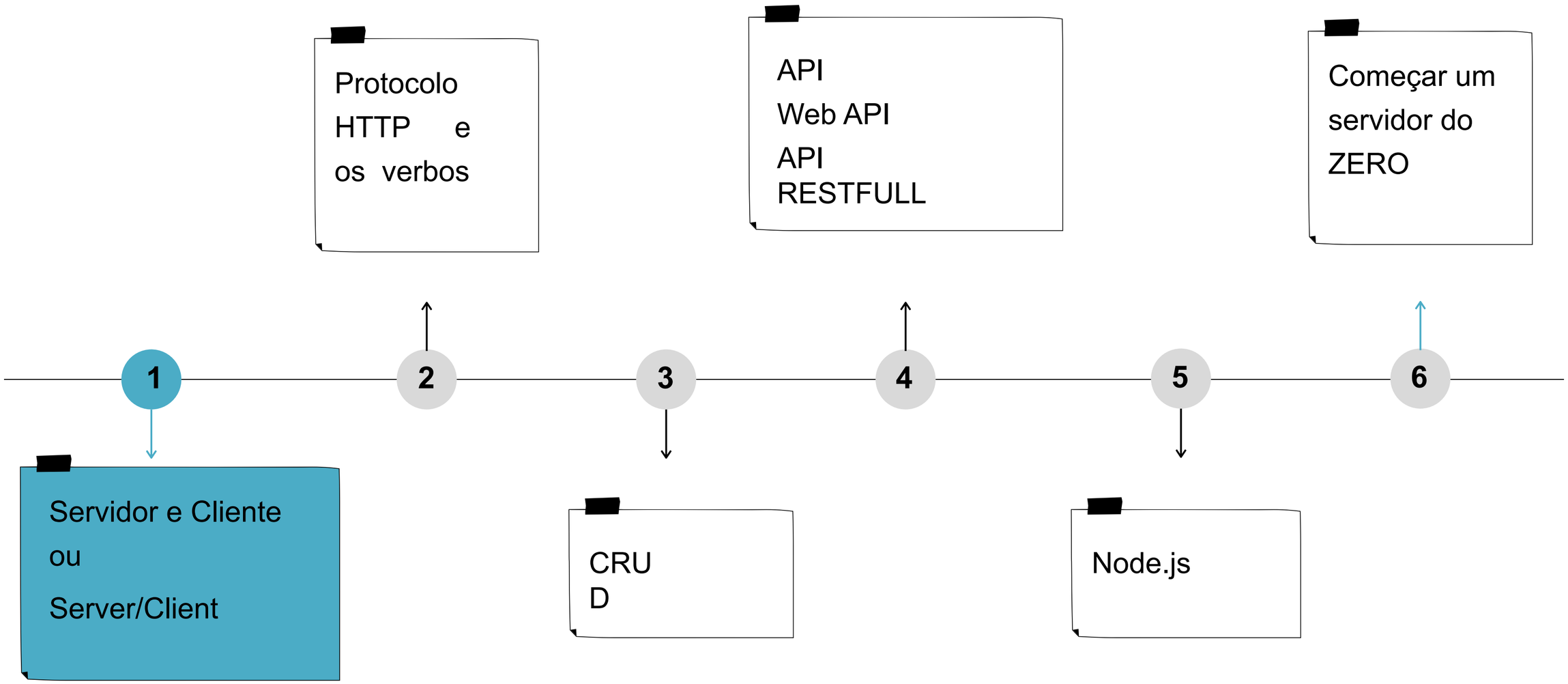
Projeto Guiado

{reprograma}flix

API - GET

{ reprograma }


Protocolo HTTP e os verbos
API
Web API
API RESTFULL
Começar um servidor do ZERO
1
2
3
4
5
6
Servidor e Cliente
ou
Server/Client
CRU
D
Node.js

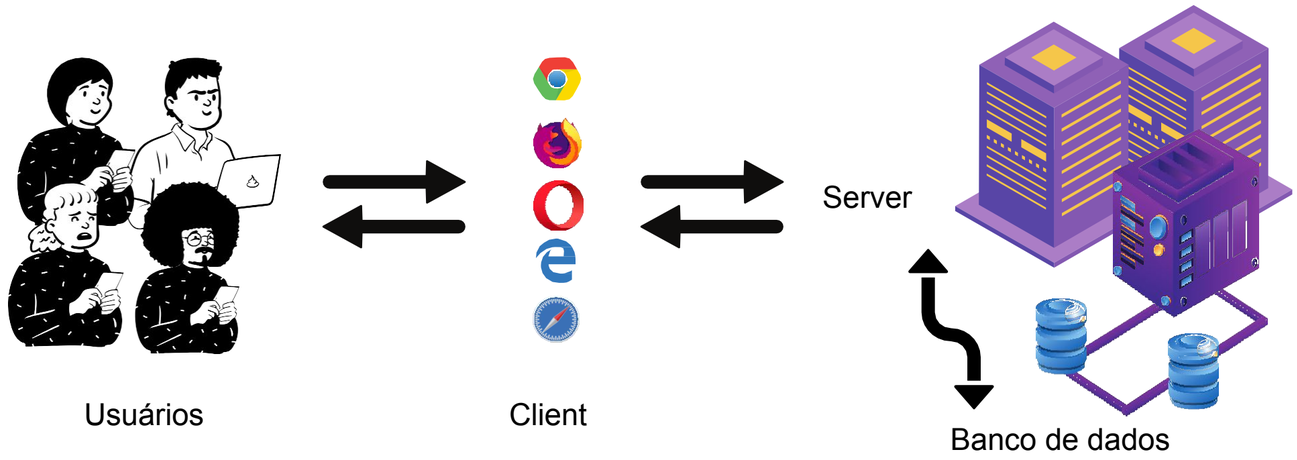
Modelo Server-Client

CLIENTE

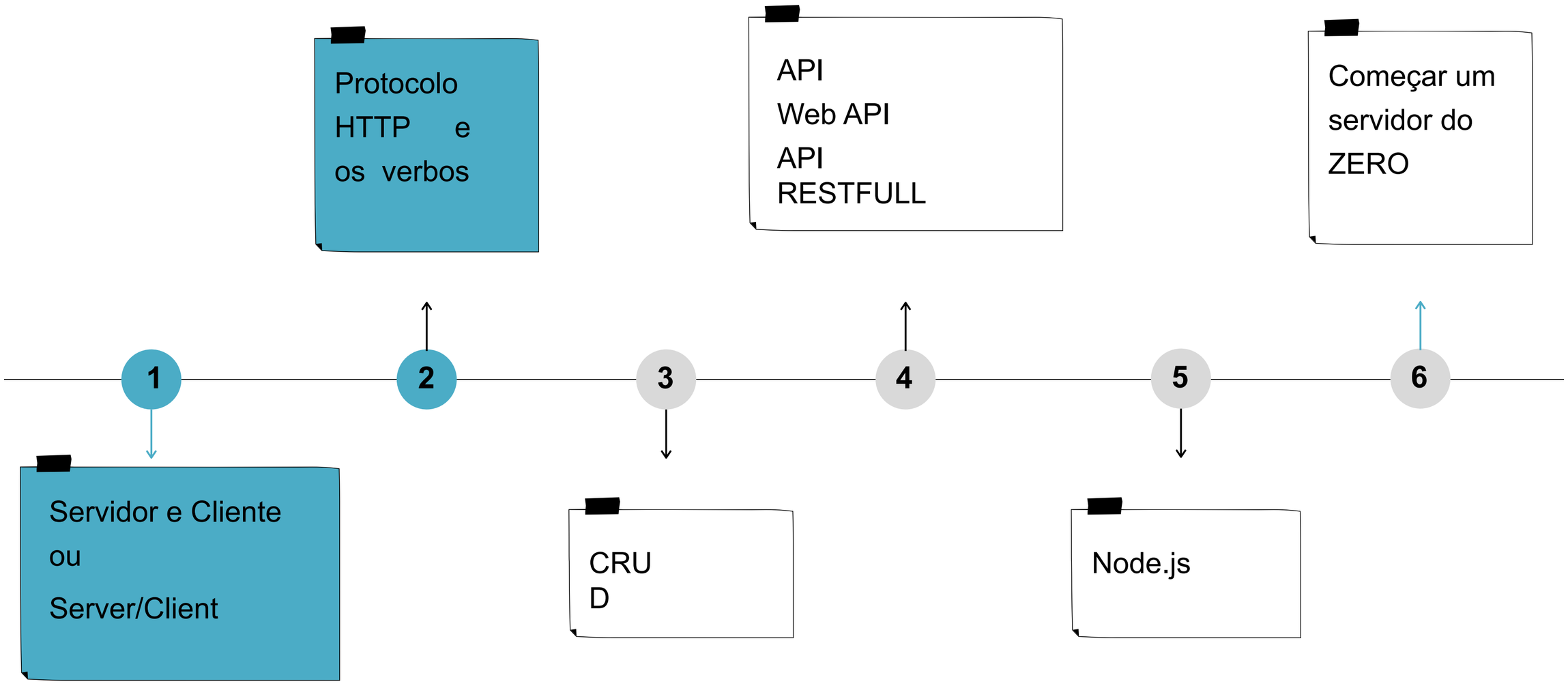
Entenda cliente como a interface que o usuário interage. É o Cliente que **solicita** serviços e informações de um ou mais servidores.

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.



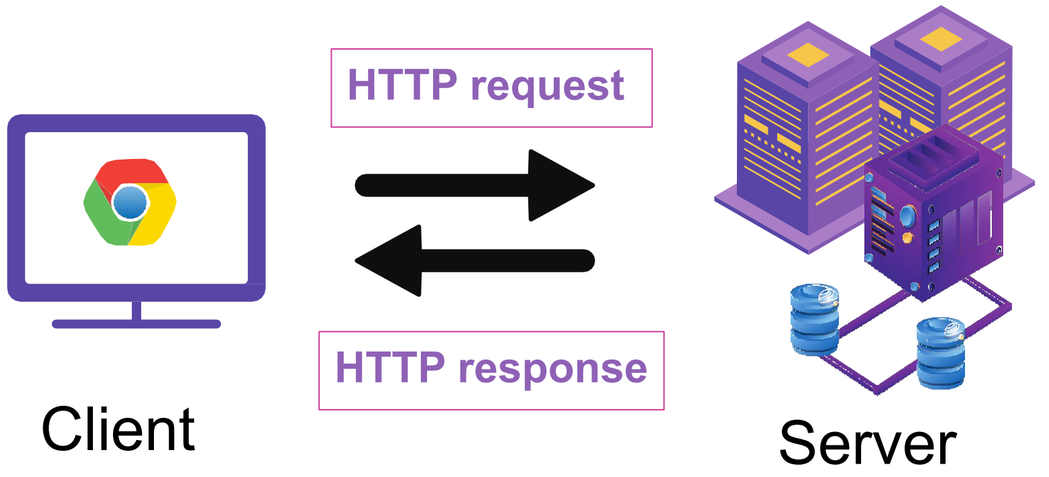
{ reprograma }


Protocolo HTTP e os verbos
API
Web API
API RESTFULL
Começar um servidor do ZERO
1
2
3
4
5
6
Servidor e Cliente
ou
Server/Client
CRU
D
Node.js

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.



http://	www.reprograma.com.br	/courses
<protocolo>	<endereço do servidor>	<recurso>

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:

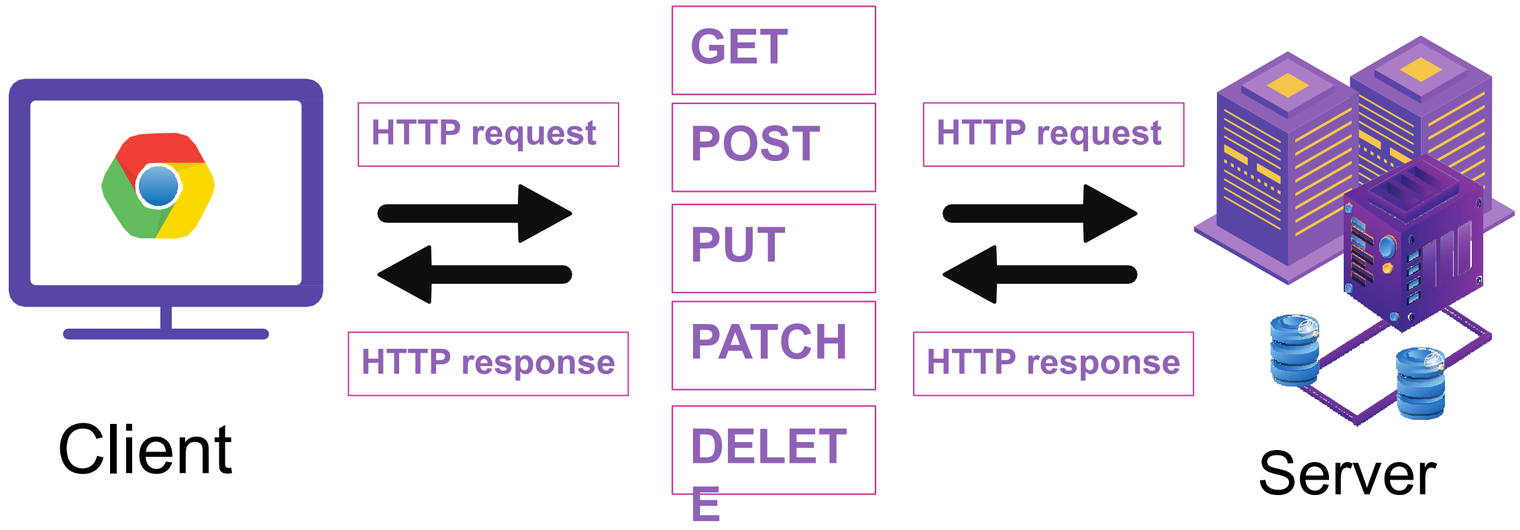
É a desenvolvedora Back end que coloca na construção do servidor quais serão as situações referentes a cada resposta.

código	tipo de resposta
100-199	informação
200-299	sucesso
300-399	redirecionamento
400-499	erro do cliente
500-599	erro de servidor

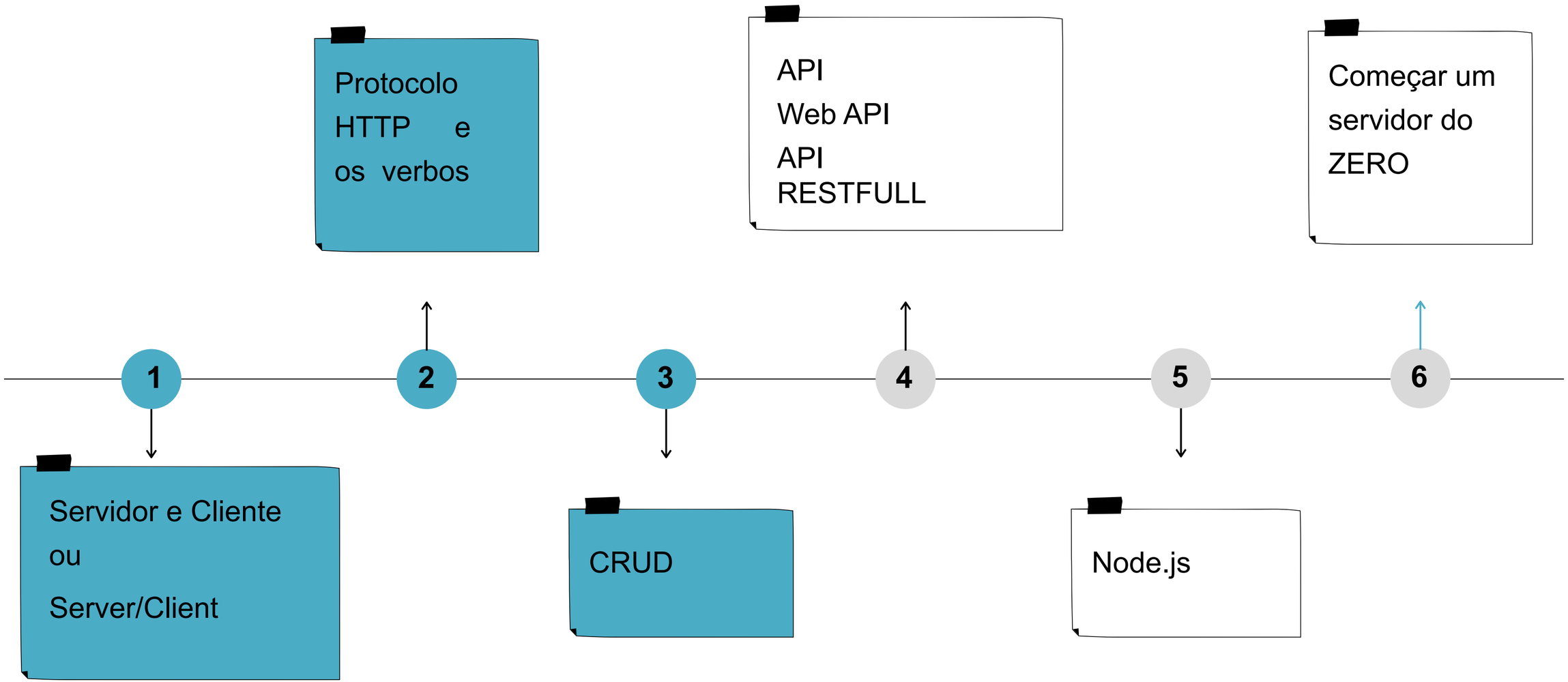
HTTP - Verbos

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 respondê-lo com um **response**.



{ reprograma }


Protocolo HTTP e os verbos
API
Web API
API RESTFULL
Começar um servidor do ZERO
1
2
3
4
5
6
Servidor e Cliente
ou
Server/Client
CRUD
Node.js

HTTP - CRUD

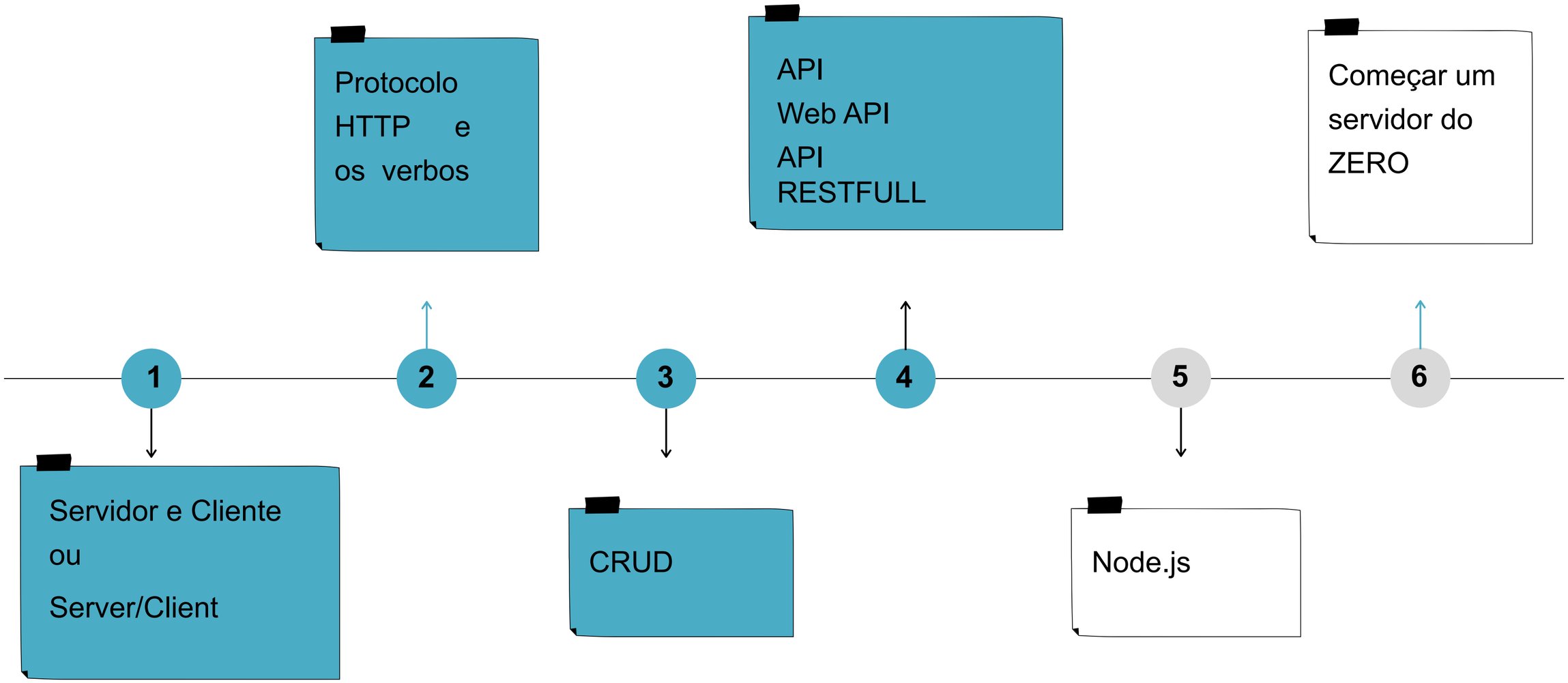
CRUD é a composição da primeira letra de 4 operações básicas de um banco de dados, e são o que a maioria das aplicação faz

✅ 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

Cada um deles corresponde a uma ação real no banco de dados.

GET	ler
POST	criar
PUT	substituir
PATCH	modificar
DELETE	excluir

{ reprograma }


Protocolo HTTP e os verbos
API
Web API
API RESTFULL
Começar um servidor do ZERO
1
2
3
4
5
6
Servidor e Cliente
ou
Server/Client
CRUD
Node.js

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."

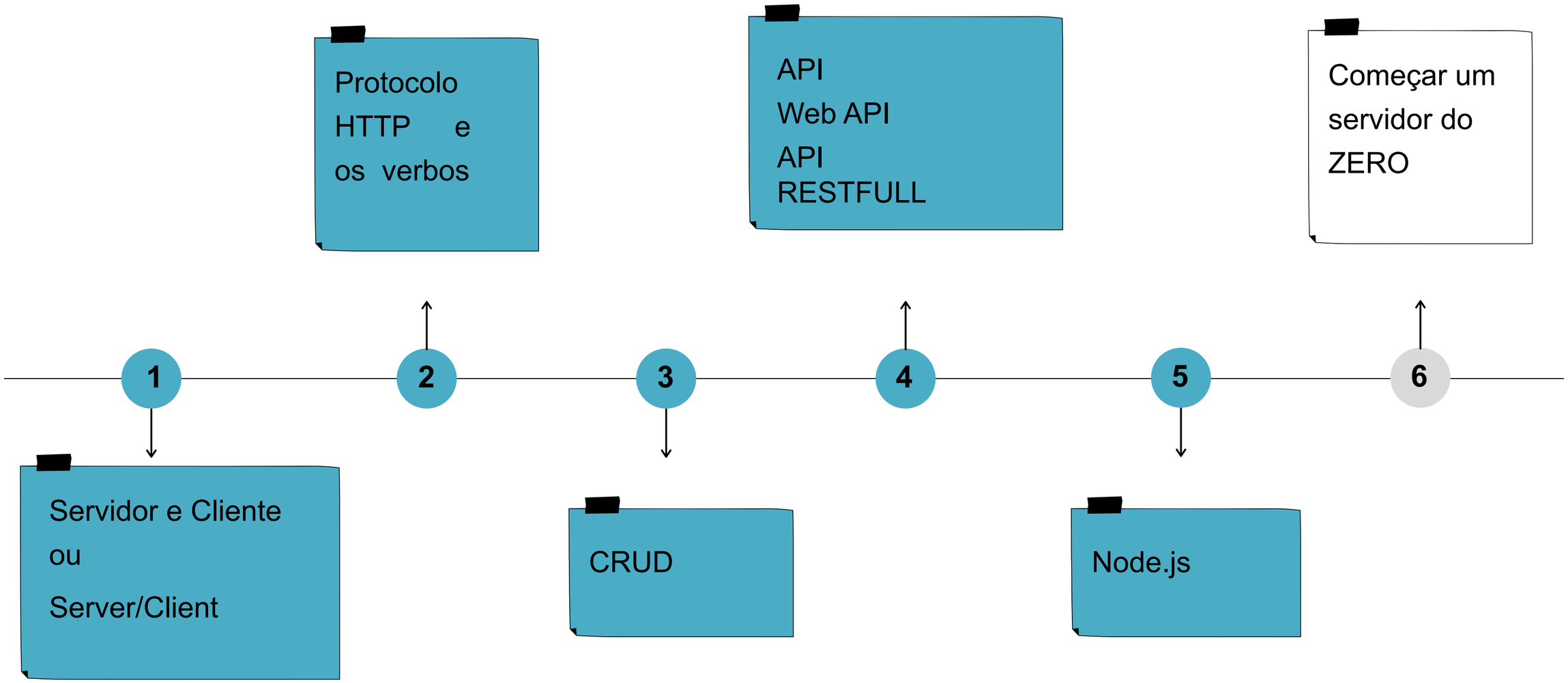
Ela não necessariamente está num link na Web, ela pode ser uma lib ou um framework, uma função já pronta em uma linguagem especifica, etc.

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.

{ reprograma }


Protocolo HTTP e os verbos
API
Web API
API RESTFULL
Começar um servidor do ZERO
1
2
3
4
5
6
Servidor e Cliente ou Server/Client
CRUD
Node.js

Node.js

O JavaScript do lado do servidor

Interpretador JavaScript que não precisa de navegador.

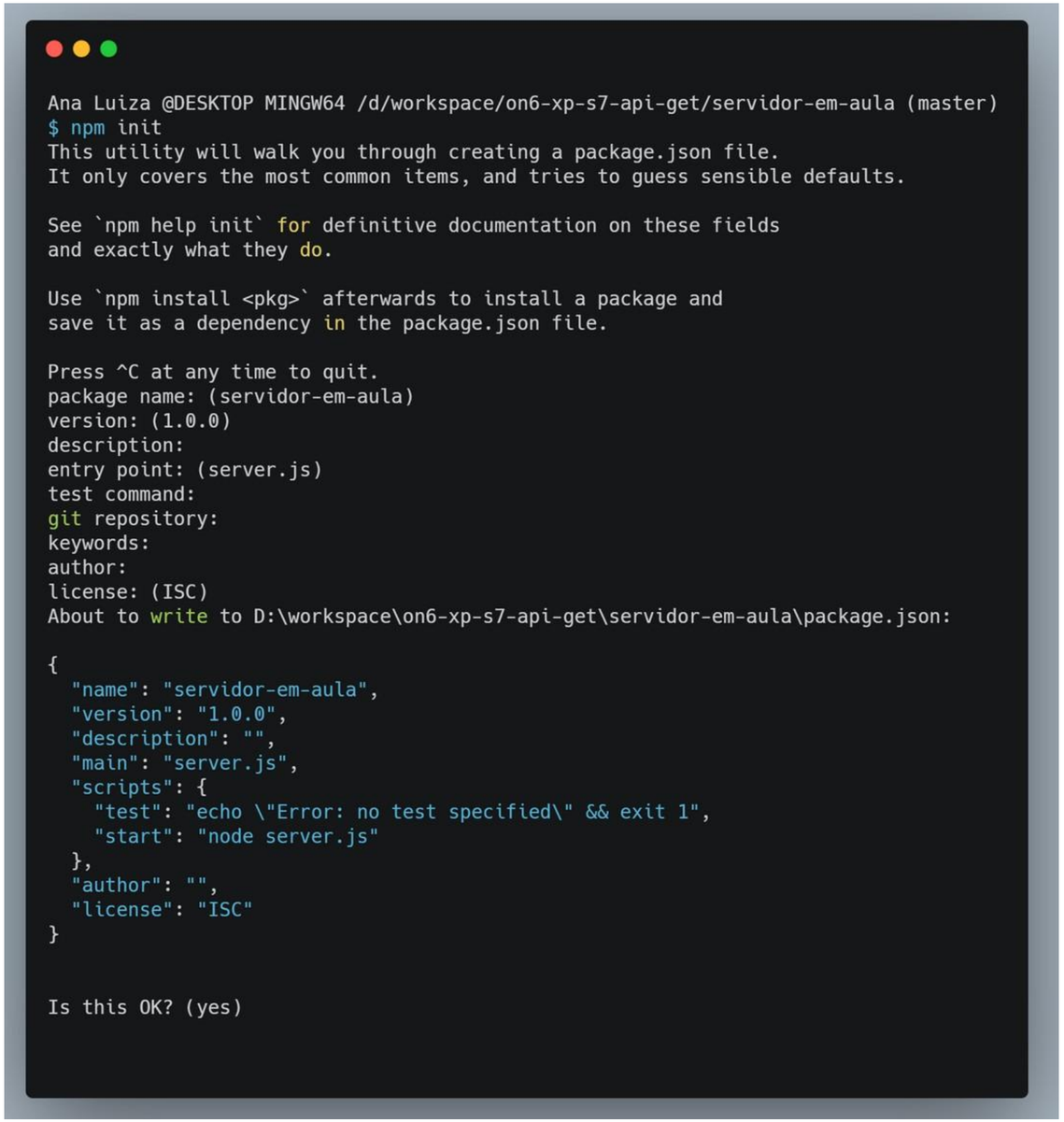
Ele pode:

Ler e escrever arquivos no seu computador
Conectar com um banco de dados

Se comportar como um servidor

init

npm init



{ reprograma }

Dependências

Só um pouquinho delas

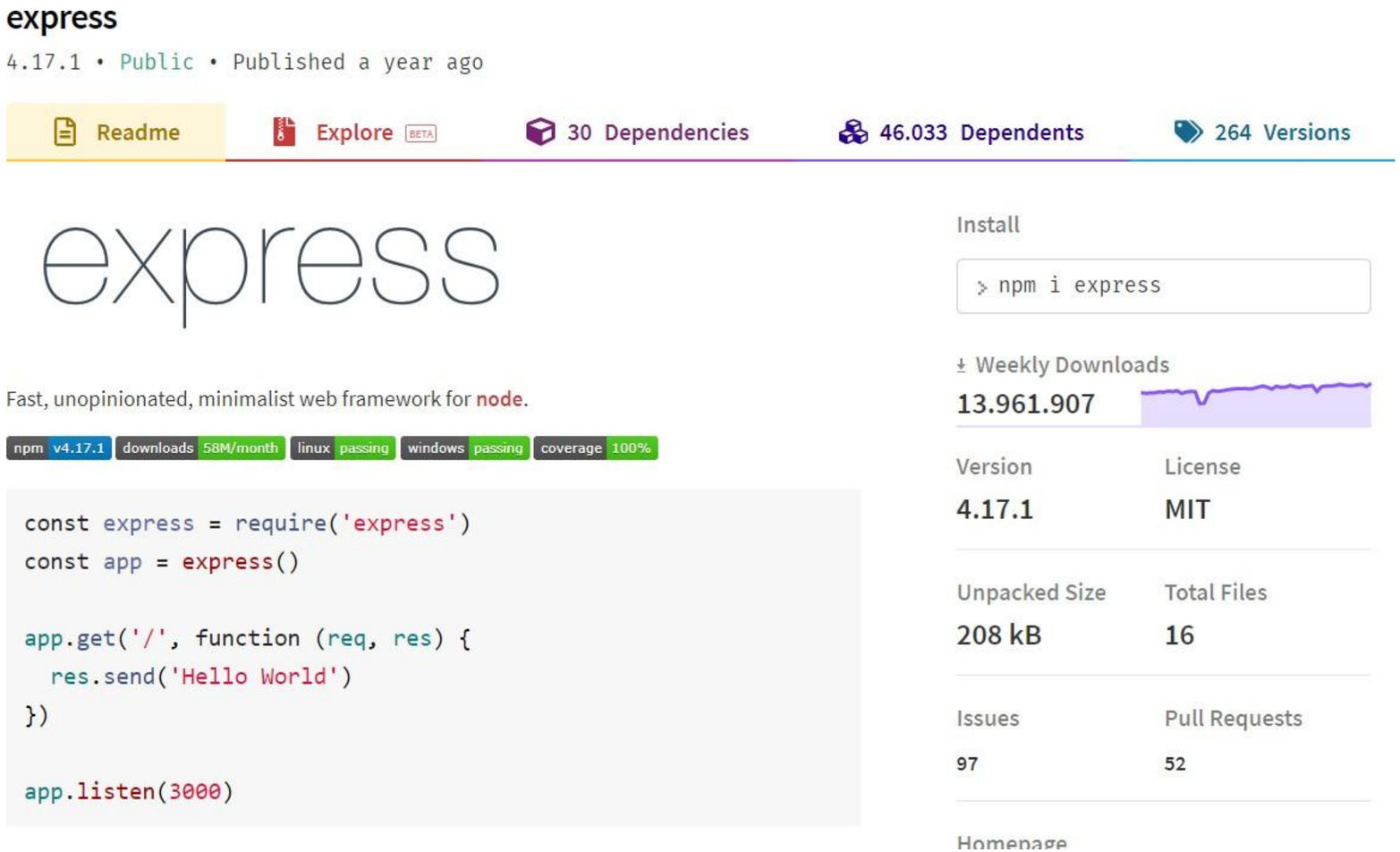
Package.json

O arquivo package.json é o ponto de partida de qualquer projeto NodeJS. Ele é responsável por:

- descrever o seu projeto
- informar a versão do node e do npm
- url do repositório
- versão do projeto
- dependências de produção e de desenvolvimento

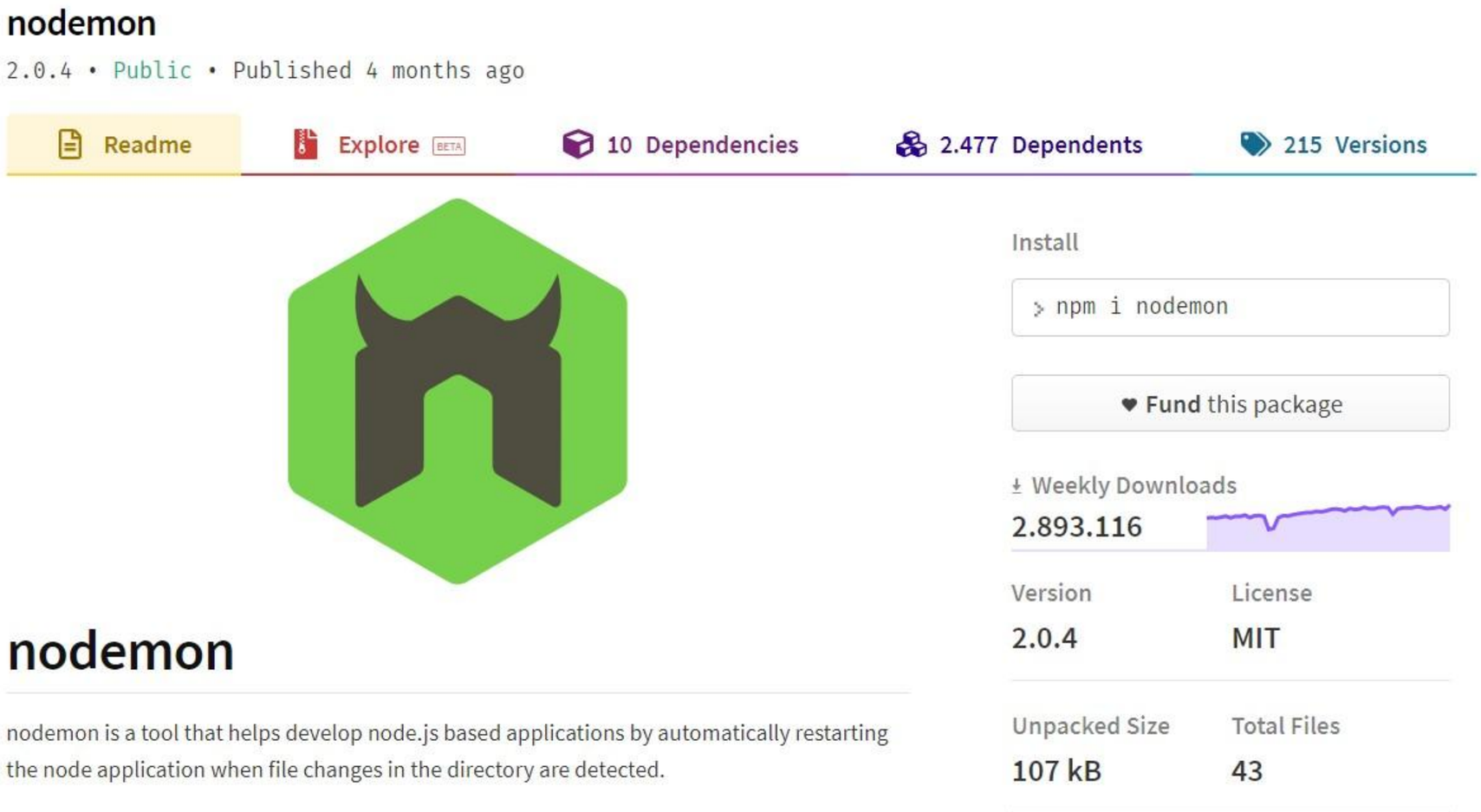
Express

npm install express

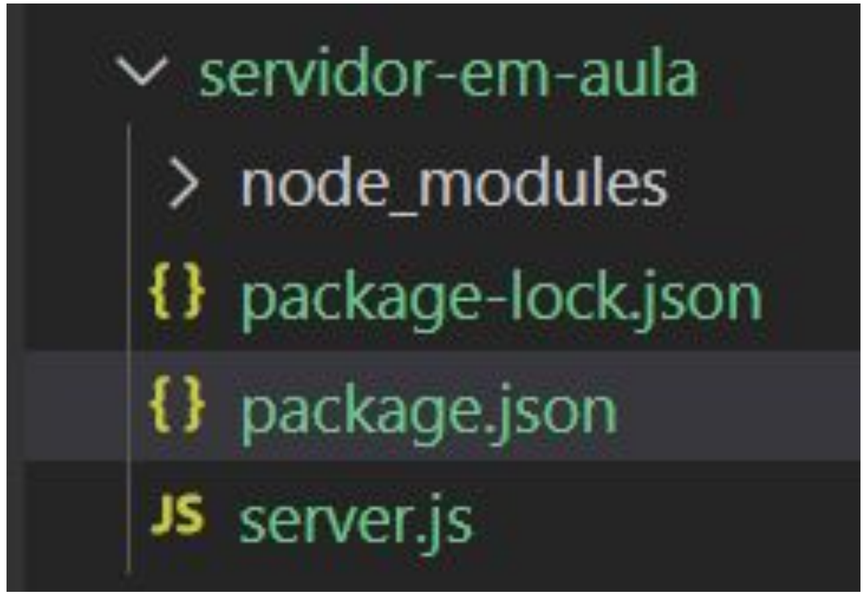


nodemon

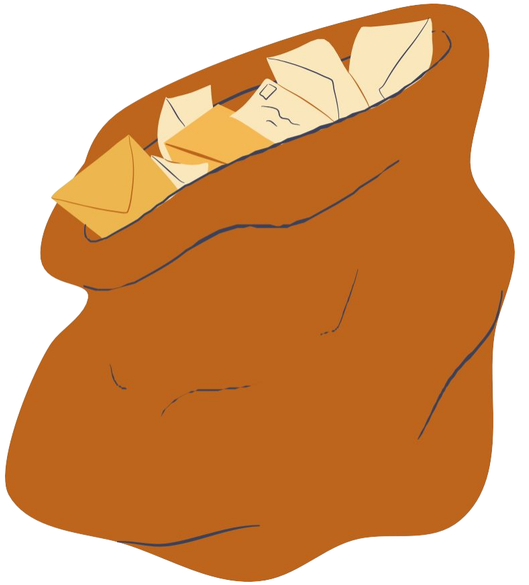
pra parar de parar



Pacotes

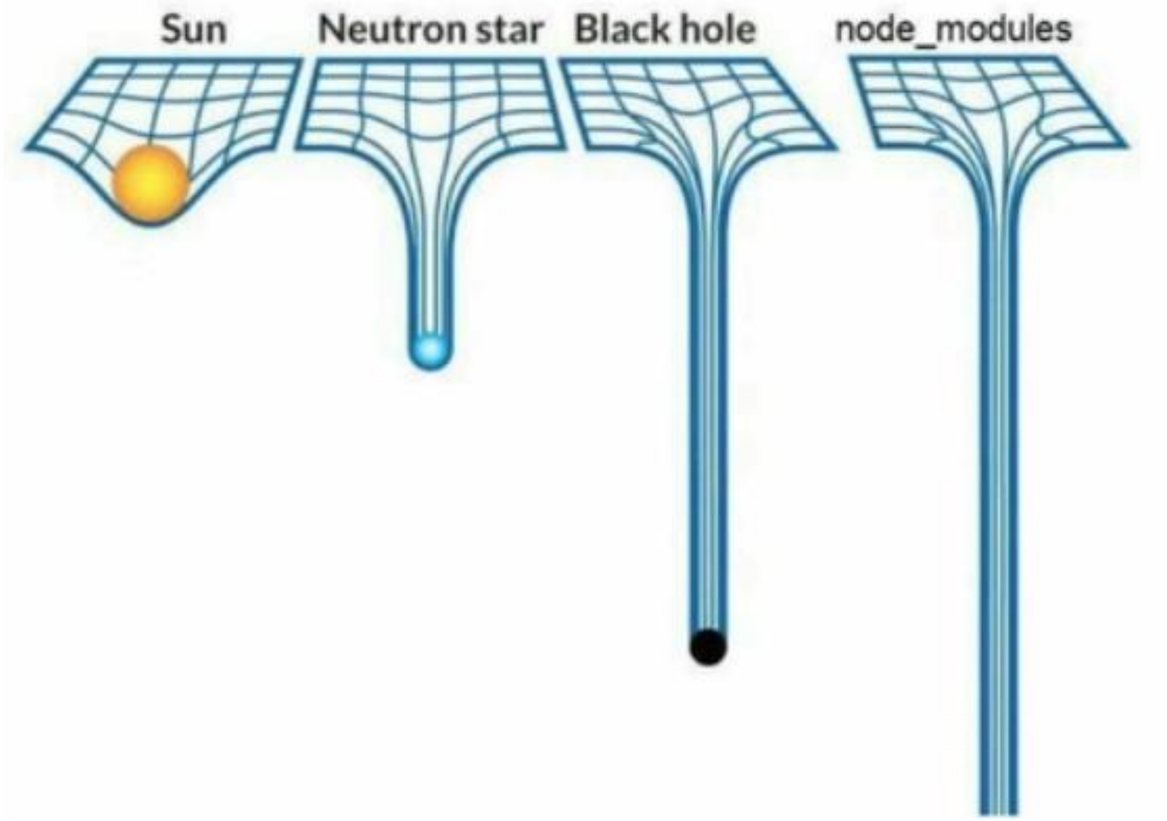


Sempre que você instalar um pacote do npm, ele será referenciado no **package-lock.json** e será instalado na pasta **node_modules**.



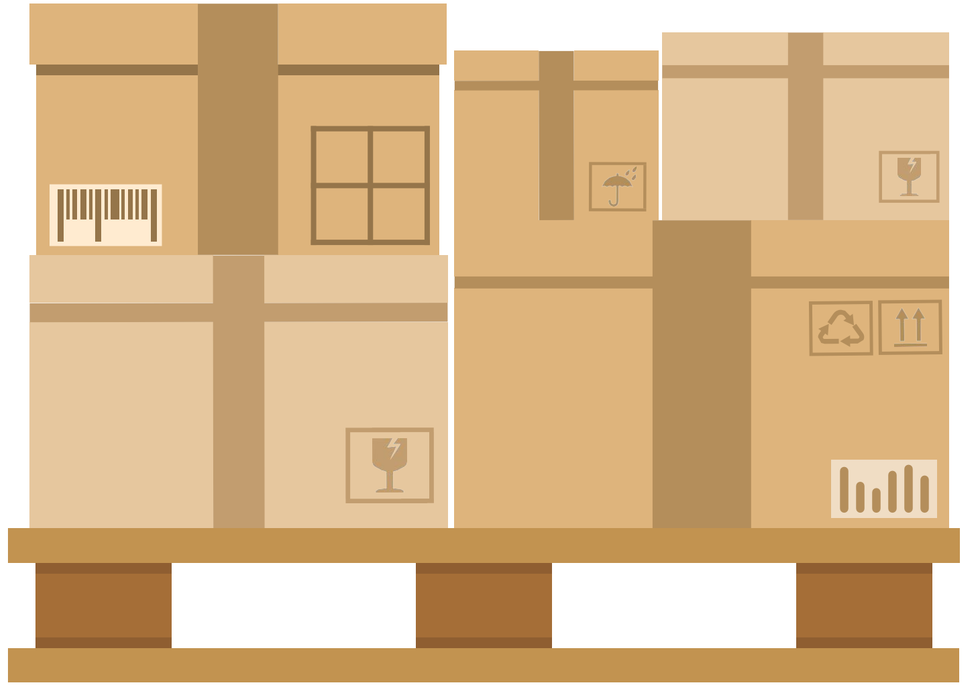
node_modules

Na node_modules estarão baixadas as dependências que o seus pacotes precisarão pra funcionar



Package-lock.json

O package-lock especifica a versão e suas dependências próprias, assim, a instalação criada será sempre a mesma, toda vez.



.gitignore

vamo ignorar ela, gente

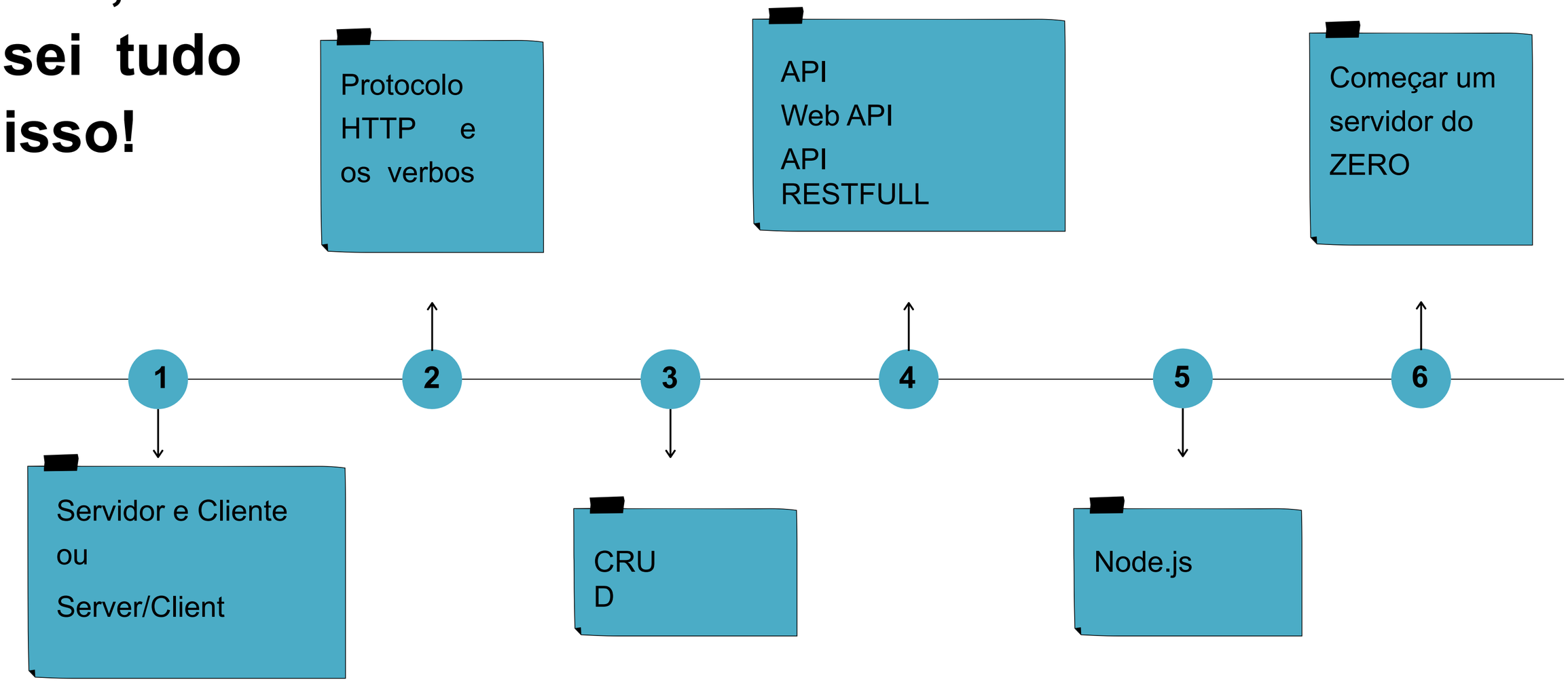
Nós ignoramos a node_modules pois nela estão todos os downloads de todas as dependências do projeto, fica MUITO pesado subir ela no git, por exemplo

Se apagarmos ela só precisamos dar o comando *npm install* que as dependências serão baixadas de novo e pasta node_modules reaparecerá.

O npm sabe quais dependências baixar pois elas estão referenciadas no package.json e no package-lock.json.

{ reprograma }

Sim, eu sei tudo isso!



{reprograma}flix

{ reprograma }

Demandas da API

Lá vem a galera de negócio....

- Quero duas rotas a **/filmes** e a **/series**
- **/filmes** deve retornar **todos os filmes**
- **/series** deve retornar **todos as series**
- Devo conseguir **filtrar** por **titulo**, **id** e **genero**
- se o usuário digitar errado o nome do filme quero retorno do erro

que???

Demandas da API

Lá vem a galera de negócio....

- **[GET] /filmes**
  - retorna todos os filmes
- **[GET] /filmes{id}**
  - retorna um filme pelo id
- **[GET] /filmes{titulo}**
  - retorna um filme pelo nome
- **[GET] /filmes{genero}**
  - retorna um filme pelo gênero
- se o usuário digitar errado quero retorno do erro

- **[GET] /series**
  - retorna todos os filmes
- **[GET] /series{id}**
  - retorna um filme pelo id
- **[GET] /series{titulo}**
  - retorna um filme pelo nome
- **[GET] /series{genero}**
  - retorna um filme pelo gênero
- se o usuário digitar errado quero retorno do erro

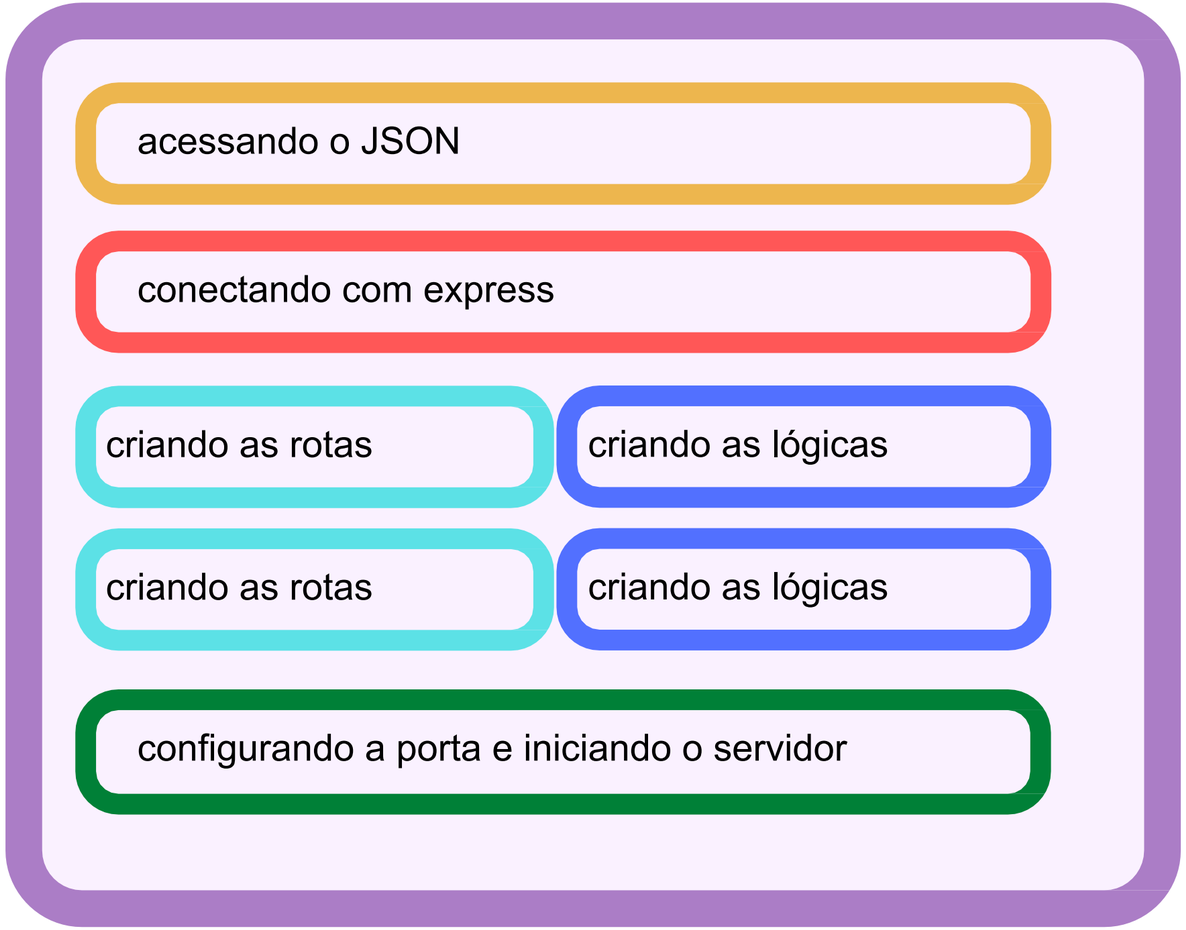
Vamos começar!

Almoço

Arquitetura

Vamos começar a organizar isso aqui

Até hoje nós estávamos fazendo tudo dentro de um arquivo só, o server.js



Arquitetura

Vamos começar a organizar isso aqui

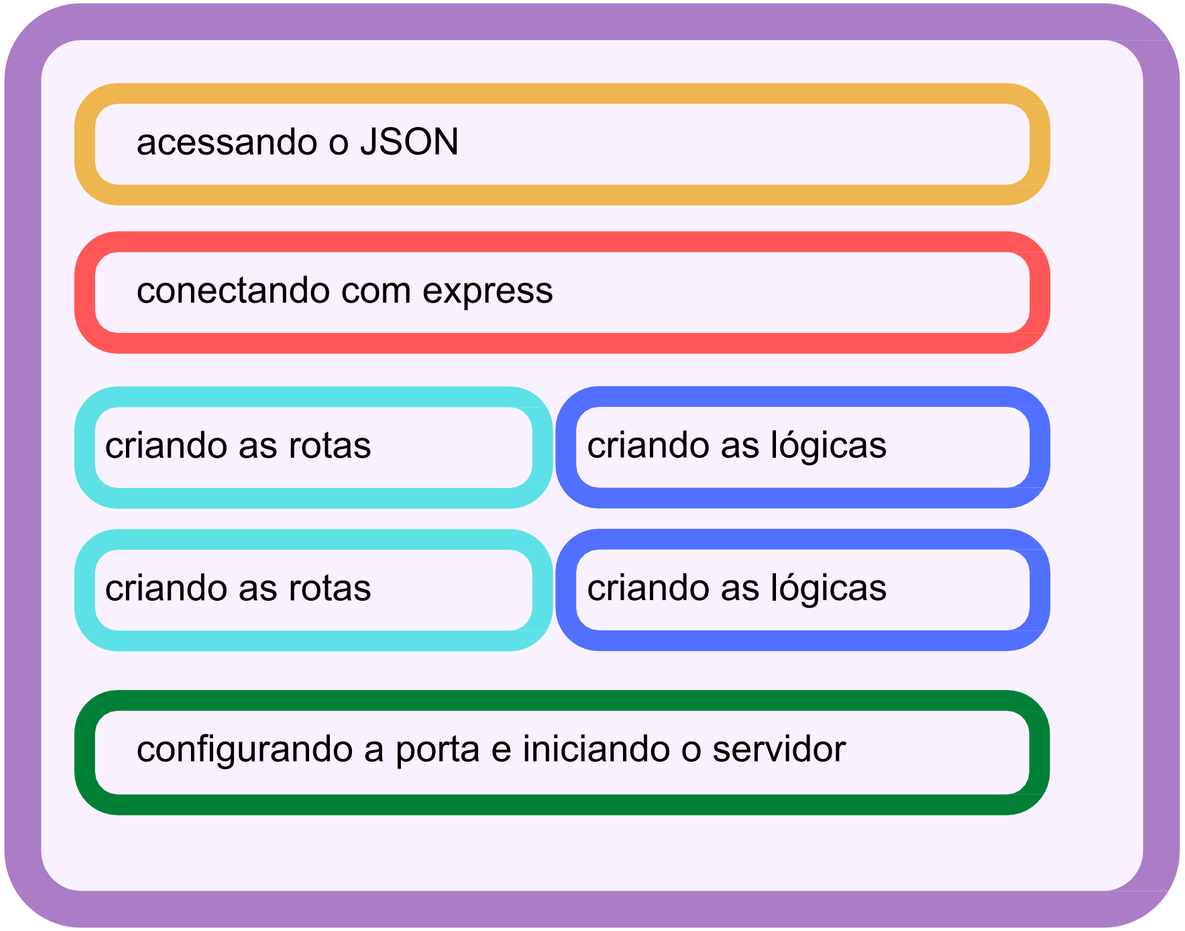
E tudo isso faz muito sentido se a gente tiver um projeto simples, mas... como a gente faz quando temos um projeto mais complexo?

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.



Arquitetura - MVC

server.js

configurando a porta e iniciando o servidor

app.js

indicação das rotas e requisições

filmesRoutes.js

seriesRoutes.js

rotas e verbos

pasta routes

filmesController.js

seriesController.js

lógica

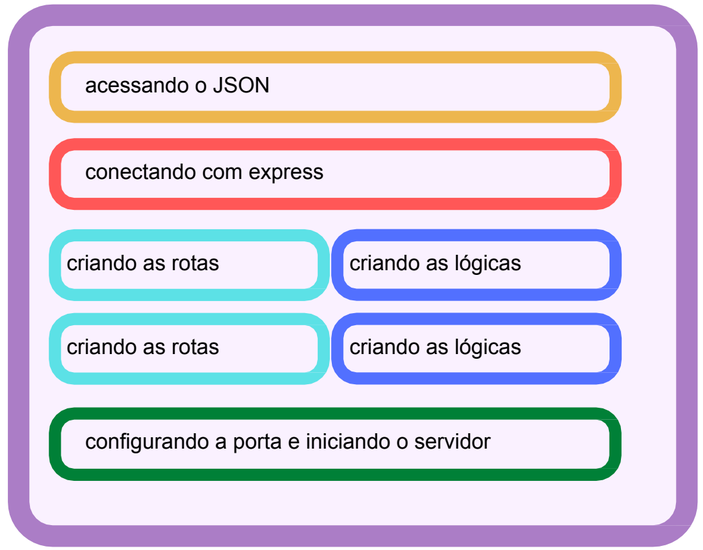
pasta controller

filmes.json

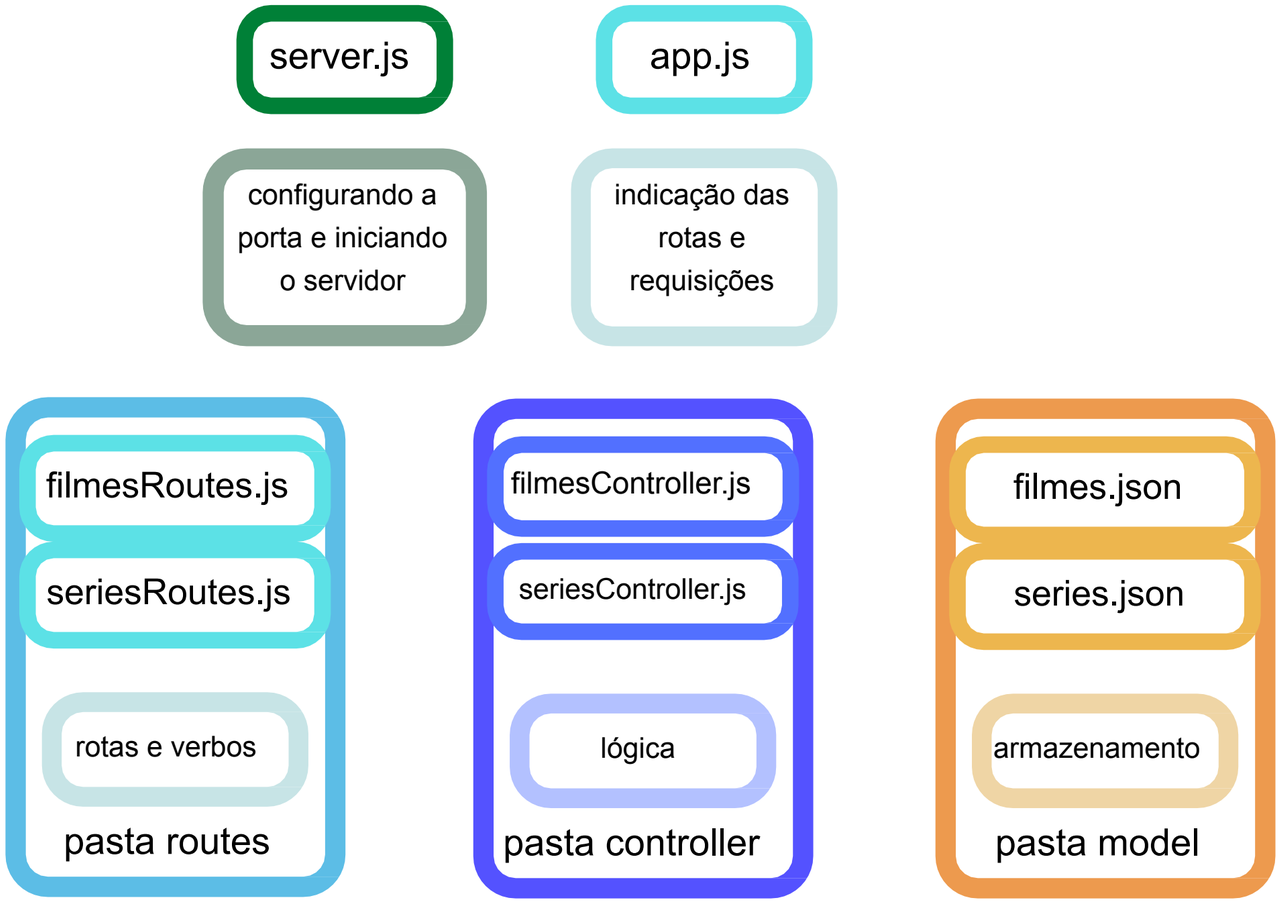
series.json

armazenamento

pasta model

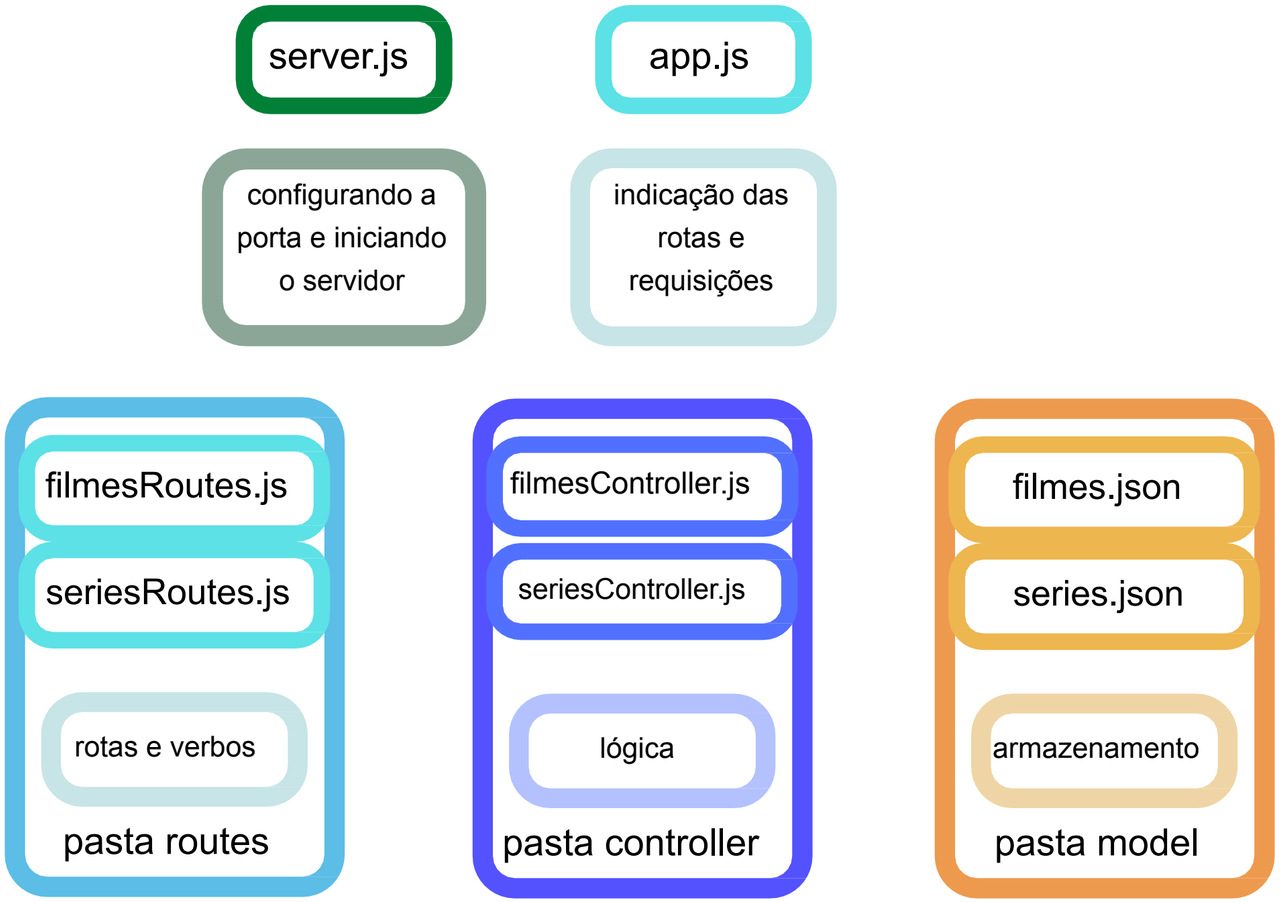


Arquitetura - MVC




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

Arquitetura - MVC




```
\-- 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
```

Parametros

Path params

- são aqueles que são adicionados diretamente na URL
- "/rota/:id"
- request.params.id bons,
- porem limitantes,

  por exemplo, se quisermos filtrar por uma string

Query params

- são aqueles que são adicionados a chave e o valor desejados
- "/rota?id=1234"
- request.query.id
- a forma mais efetiva de

  fazer requests com strings

  ou diversos valores

 Para casa 

Demandas da API

Lá vem a galera de negócio...

- **[GET] /filmes**
  - retorna todos os filmes
- **[GET] /filmes{id}**
  - retorna um filme pelo id
- **[GET] /filmes{titulo}**
  - retorna um filme pelo nome
- **[GET] /filmes{genero}**
  - retorna um filme pelo genero
- se o usuário digitar errado quero retorno do erro

- **[GET] /series**
  - retorna todos os filmes
- **[GET] /series{id}**
  - retorna um filme pelo id
- **[GET] /series{titulo}**
  - retorna um filme pelo nome
- **[GET] /series{genero}**
  - retorna um filme pelo genero
- se o usuário digitar errado quero retorno do erro