

Criando webservice REST com NodeJS, NoSQL & Docker



Giovanni Kenji Shiroma
DESENVOLVEDOR FULL-STACK
\\ SENAI DE INFORMÁTICA



SENAI

SEE TO BE THE BEST

SE
Sociedade dos Engenheiros
do Brasil



 Competidor da worldskills na modalidade de
WebDesign & Development 2015

TREINAMENTO PARA OS COMPETIDORES RUSSOS



10.14.2.158 5604 10.14.2.124 9836
itatar.ru\wpad.dat

SSID: RCENTR
PSN: worldtech32

User
• id
• name
• Surname

WebService - API

Verbs/Methods

```
{  
    "create" : "POST",  
    "read"   : "GET",  
    "update" : "PUT",  
    "partial_update": "PATCH",  
    "delete" : "DELETE"  
}
```

REST = Architecture Pattern

Plural

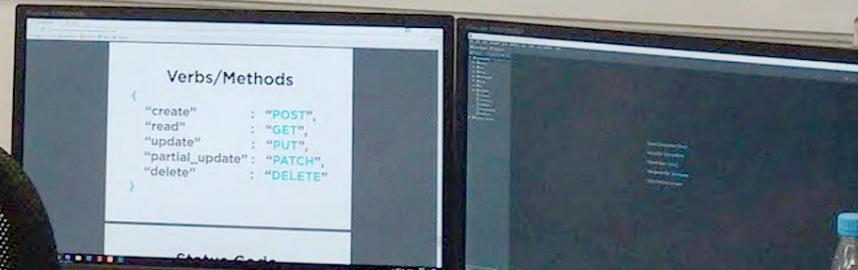
→ create
POST → WebService.com/Users

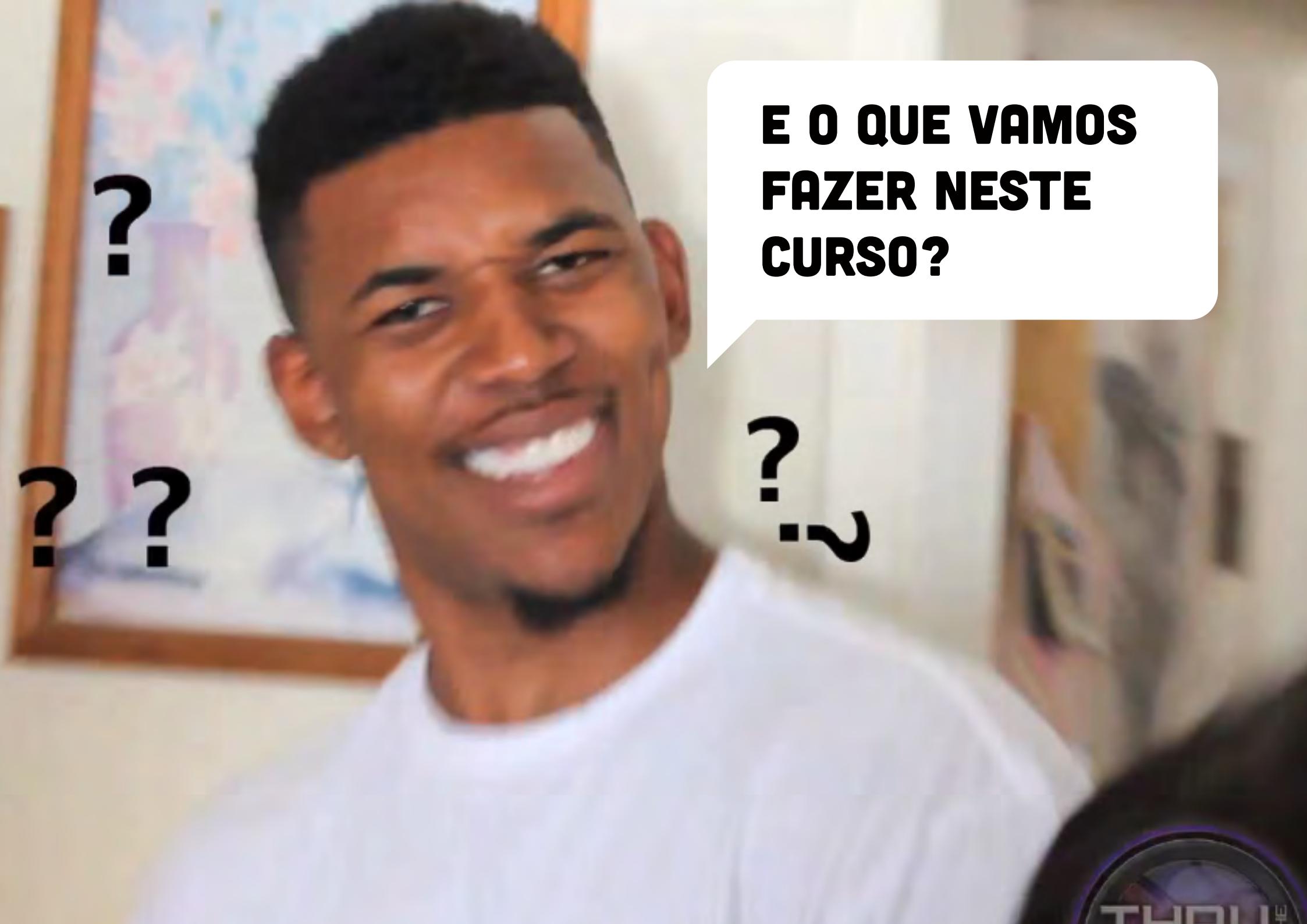
GET → /Users → list user

/users/1/friends

PUT → users/1

PATCH → users/1





**E O QUE VAMOS
FAZER NESTE
CURSO?**

? -~

BACK-END DE UMA TODOLIST!

Linguagens para aprender

- Javascript
- C#
- Swift
- Closure

Para comprar no mercado

- 12 Maçãs
- 12 batatas
- 5 pacotes de pipocas
- 2 óreos

Linguagens para aprender

- Javascript
- C#
- Swift
- Closure

Linguagens para aprender

- Javascript
- C#
- Swift
- Closure

Para comprar no mercado

- 12 Maçãs
- 12 batatas
- 5 pacotes de pipocas
- 2 óreos

Linguagens para aprender

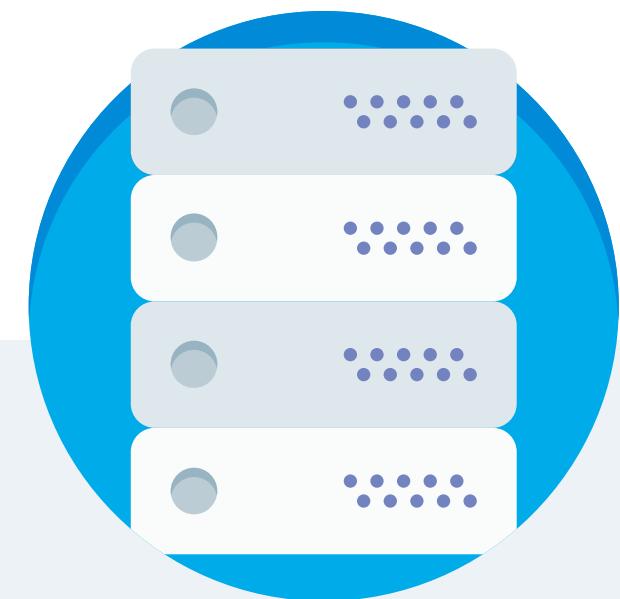
- Javascript
- C#
- Swift
- Closure

Mas primeiro...

Qual a diferença entre o front-end e back-end?



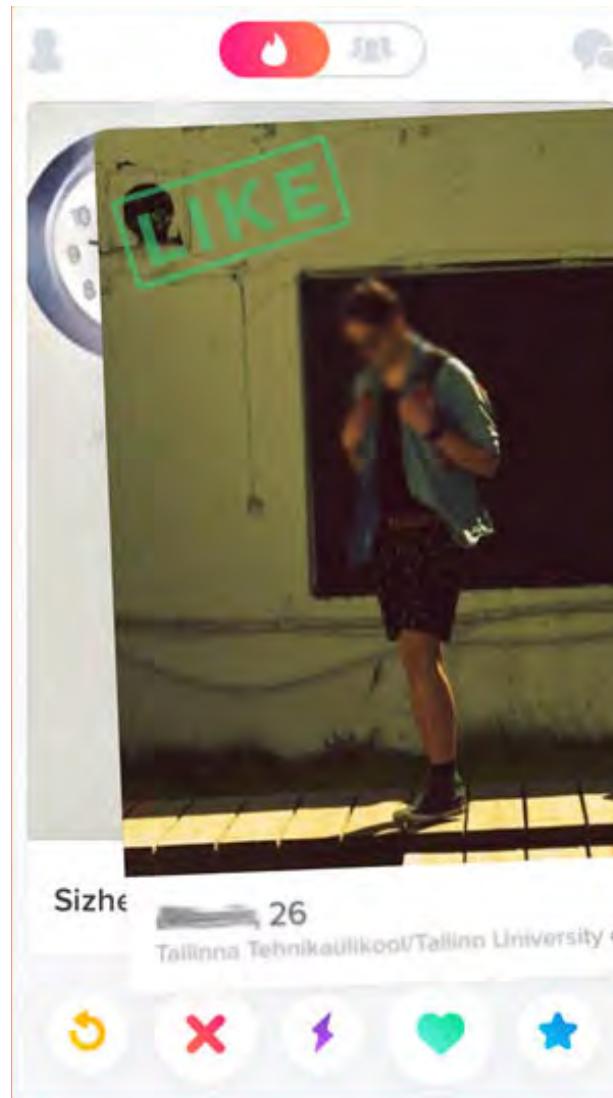
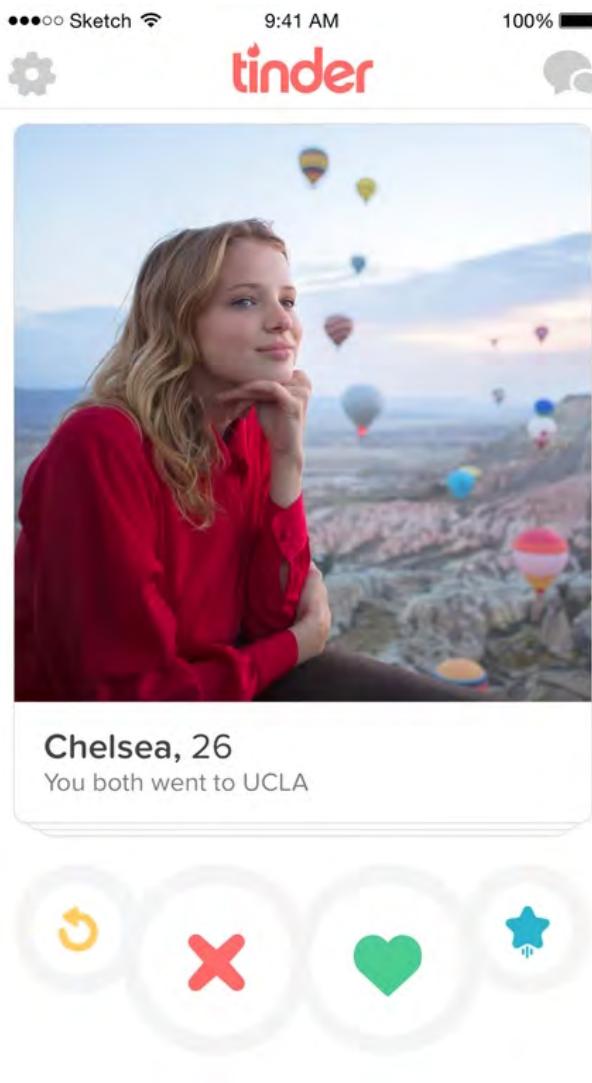
Front-End



Back-End

Front-End

Interface da sua aplicação, responsável por interações, faz os pedidos para o backend, transforma os dados que vem em forma de UI



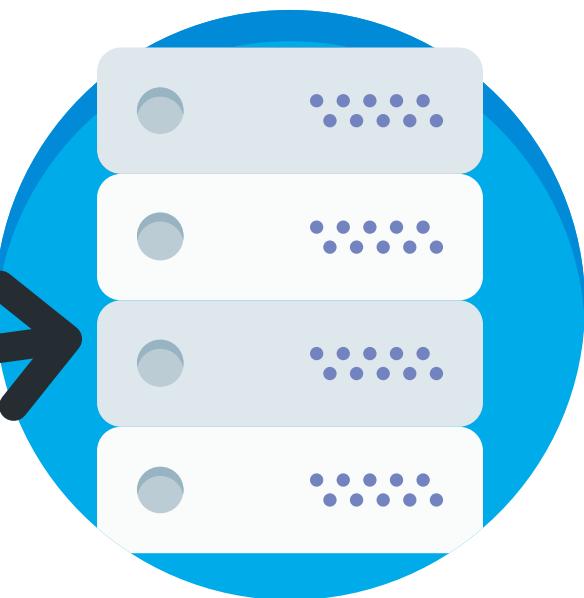
Back-End

Recebe os pedidos do front-end, responsável pelo tratamento de dados e regras de negócios

The screenshot shows the 'Discovery Settings' page on Tinder. It includes the following fields:

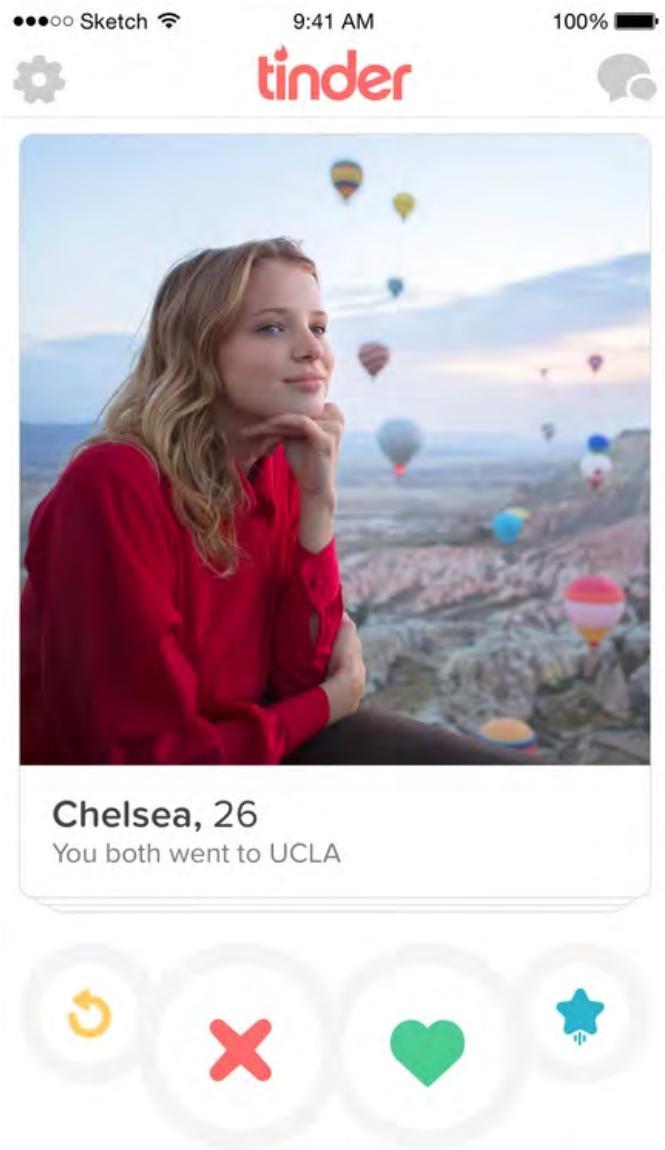
- Show me: Women (radio button selected)
- Search Distance: 10km.
- Show Ages: 23 - 42 (range slider set between 23 and 42)
- A note at the bottom states: "Tinder uses these preferences to suggest potential matches. Some match suggestions may not fall within your desired parameters."
- Web Profile section with 'Username' and 'Claim Yours' button.

Eu quero **Mulheres**
com máximo de **10 km**
de distância
entre **23 a 42 anos!**

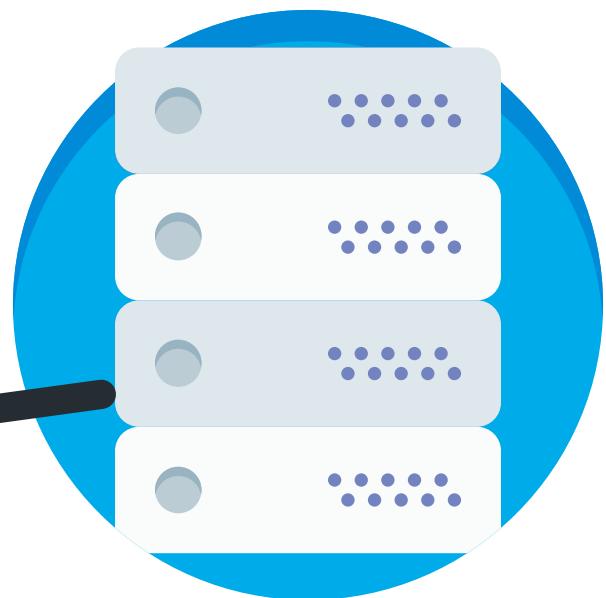


Back-End

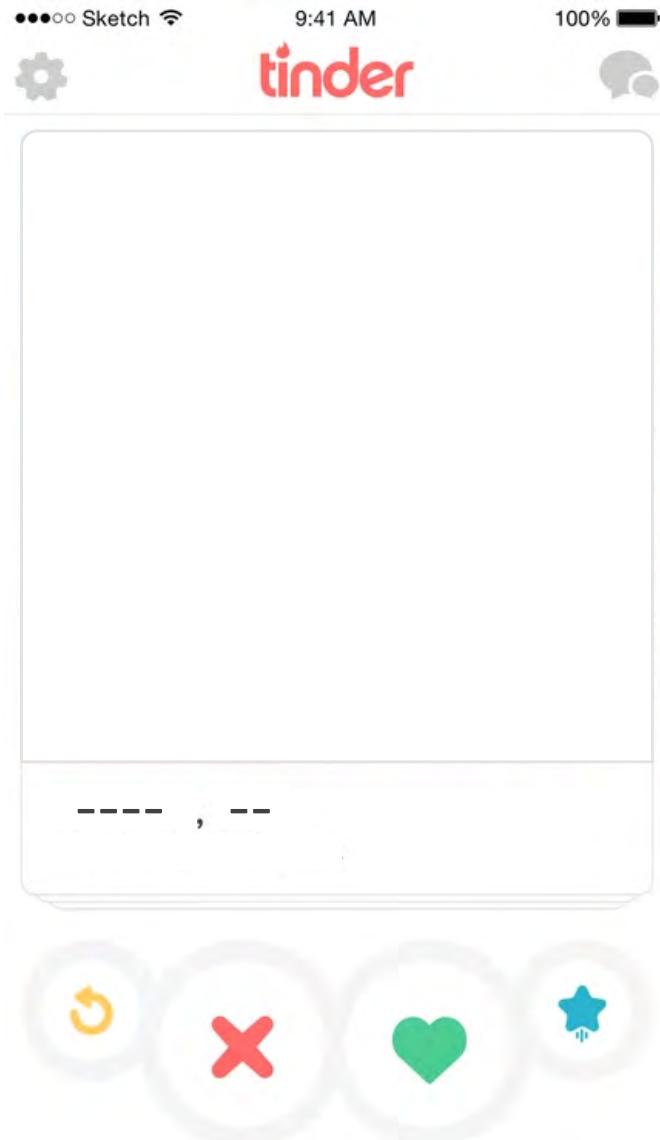
Recebe os pedidos do front-end, responsável pelo tratamento de dados e regras de negócios



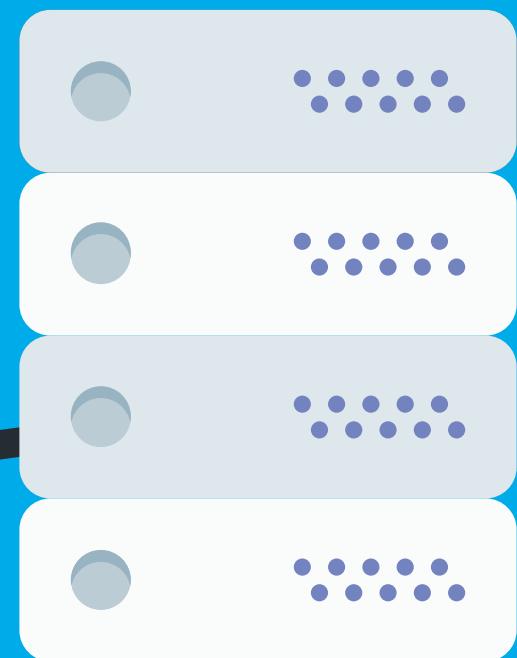
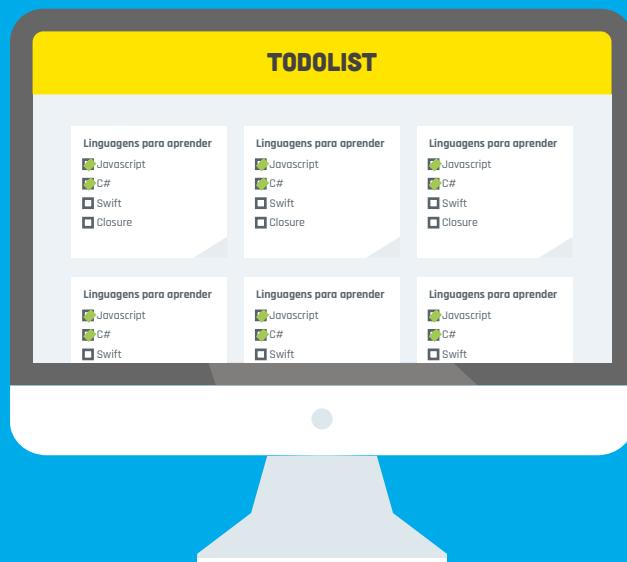
Olha ai as **Mulheres**
com máximo de **10 km**
de distancia
entre **23 a 42 anos!**



FRONT-END sem o BACK-END seria tipo:



VAMOS FAZER ESSE CARA

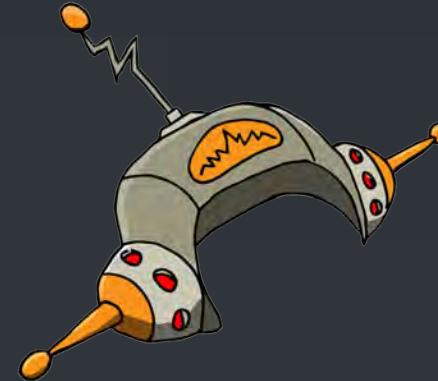




NodeJS



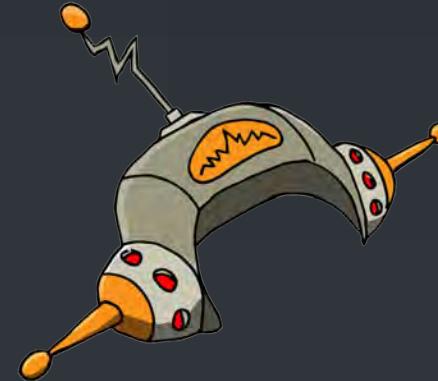
NodeJS



Hapi.js



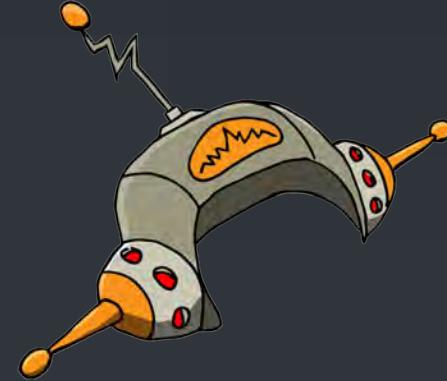
NodeJS



Hapi.js



TypeScript



Hapi.js



TypeScript



mongoDB®

PREPARAÇÕES
PARA COMEÇAR
A DESENVOLVER





NodeJS SERVIDOR JAVASCRIPT

Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient. Node.js' package ecosystem, npm, is the largest ecosystem of open source libraries in the world.

Node 4.x is End Of Life – April Release Updates

Download for macOS (x64)

8.11.2 LTS
Recommended For Most Users

10.2.1 Current
Latest Features

Other Downloads | Changelog | API Docs Other Downloads | Changelog | API Docs

Or have a look at the [Long Term Support \(LTS\) schedule](#).

Sign up for [Node.js Everywhere](#), the official Node.js Weekly Newsletter.

LINUX FOUNDATION COLLABORATIVE PROJECTS

Report Node.js issue | Report website issue | Get Help

© Node.js Foundation. All Rights Reserved. Portions of this site originally © Joyent.

Node.js is a trademark of Joyent, Inc. and is used with its permission. Please review the Trademark Guidelines of the Node.js Foundation.

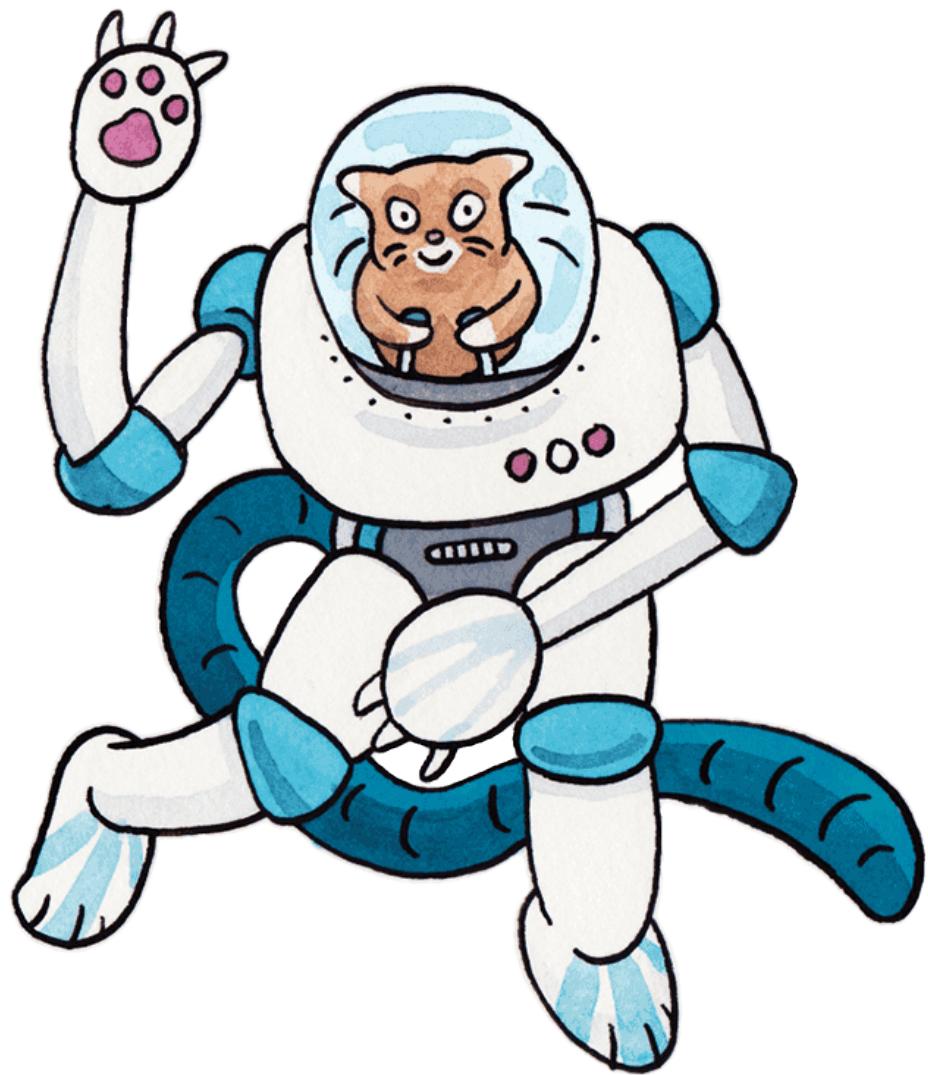
HTTPS://NODEJS.ORG/EN/



NodeJS SERVIDOR JAVASCRIPT

No MAC PODE instalar pelo BREW!

BREW INSTALL NODE



yarn

Submit your search query.



Getting Started Docs Packages Blog

American English ▾

Search packages (i.e. babel, webpack, react...)

FAST, RELIABLE, AND SECURE DEPENDENCY MANAGEMENT.

[GET STARTED](#)

[INSTALL YARN](#)

Star 31,795

Stable: [v1.7.0](#)

Node: ^4.8.0 || ^5.7.0 || ^6.2.2 || >=8.0.0

Ultra Fast.

Yarn caches every package it downloads so it never needs to download it again. It also parallelizes operations to maximize resource utilization so



HTTPS://YARNPKG.COM/EN/

Clona este projeto aqui para começar!

```
git clone https://github.com/kenjishiromajp/typescript_serverside_starterkit.git
```

Vamos neste passo...

- **Estrutura geral do projeto**
- **Configurar o Typescript**
- **Comandos básicos do node e yarn**
- **Como fazer o import de outros arquivos js**

MÃO NA MASSA!

A faint, semi-transparent watermark of a man's face and hands is visible against a bright yellow background. The man has a serious expression, looking directly at the viewer. His hands are clasped together in front of him. The overall effect is like a watermark or a stylized logo.

REVISÃO

Estrutura do package.json

```
{  
  "name": "nome_do_projeto",  
  "version": "1.0.0",  
  "description": "Descrição do nosso projeto",  
  "main": "src/meu-arquivo-principal.js",  
  "scripts": {  
    "alias": "meu-comando-do-terminal que-vou-executar-nesse-arquivo.js"  
  },  
  "author": "Kenji",  
  "license": "ISC",  
  "devDependencies": {  
    "nome-da-dependencia-de-dev": "^1.0.0"  
  },  
  "dependencies": {  
    "nome-da-dependencia": "^1.0.0"  
  }  
}
```

Para instalarmos dependencias

```
npm i <dependencia> -S
```

```
yarn add <dependencia>
```

Montando um ambiente para debug

1. Instalamos o nodemon

```
yarn add nodemon -D
```

2. executamos ele com uns parâmetros para utilizar o nodemon com ts-node

```
"scripts": {  
  "start": "ts-node src/index.ts",  
  "dev": "nodemon --inspect=5858 -r ts-node/register src/index.ts",  
  "build": "tsc"  
},|
```

3. Adicionamos uma configuração no debug no visual studio de attach



```
"configurations": [  
  {  
    "type": "node",  
    "request": "attach",  
    "name": "Attach",  
    "port": 5858  
  }]
```



4. Podemos executar o debug

Funcionamento dos imports

sem o `'.'` ele pega do **node_modules**
(dependencia que instalou com yarn add)

```
import * as Hapi from 'hapi';
import { server } from './server';
```

com o `'.'` ele pega um arquivo relativo
a pasta desse arquivo

**AS DEPENDÊNCIAS ESTÃO NA NUVEM,
PODEMOS VER AS DEPENDÊNCIAS
QUE EXISTEM NO SITE DA**



tsconfig.json

```
{  
  "compilerOptions": {  
    "module": "commonjs",  
    "target": "es6",  
    "outDir": "build"  
  },  
  "exclude": ["node_modules"]  
}
```

Define como serão feito os módulos

Define que utilizamos
as features do Ecma Script 6

Onde vamos gerar os arquivos
de build.

Array de pastas que devemos tirar da compilação.
no nosso caso o "node_modules"

ECMAScript

Especificação de uma linguagem
Onde o javascript é baseado

ES2015 ou **ES6**

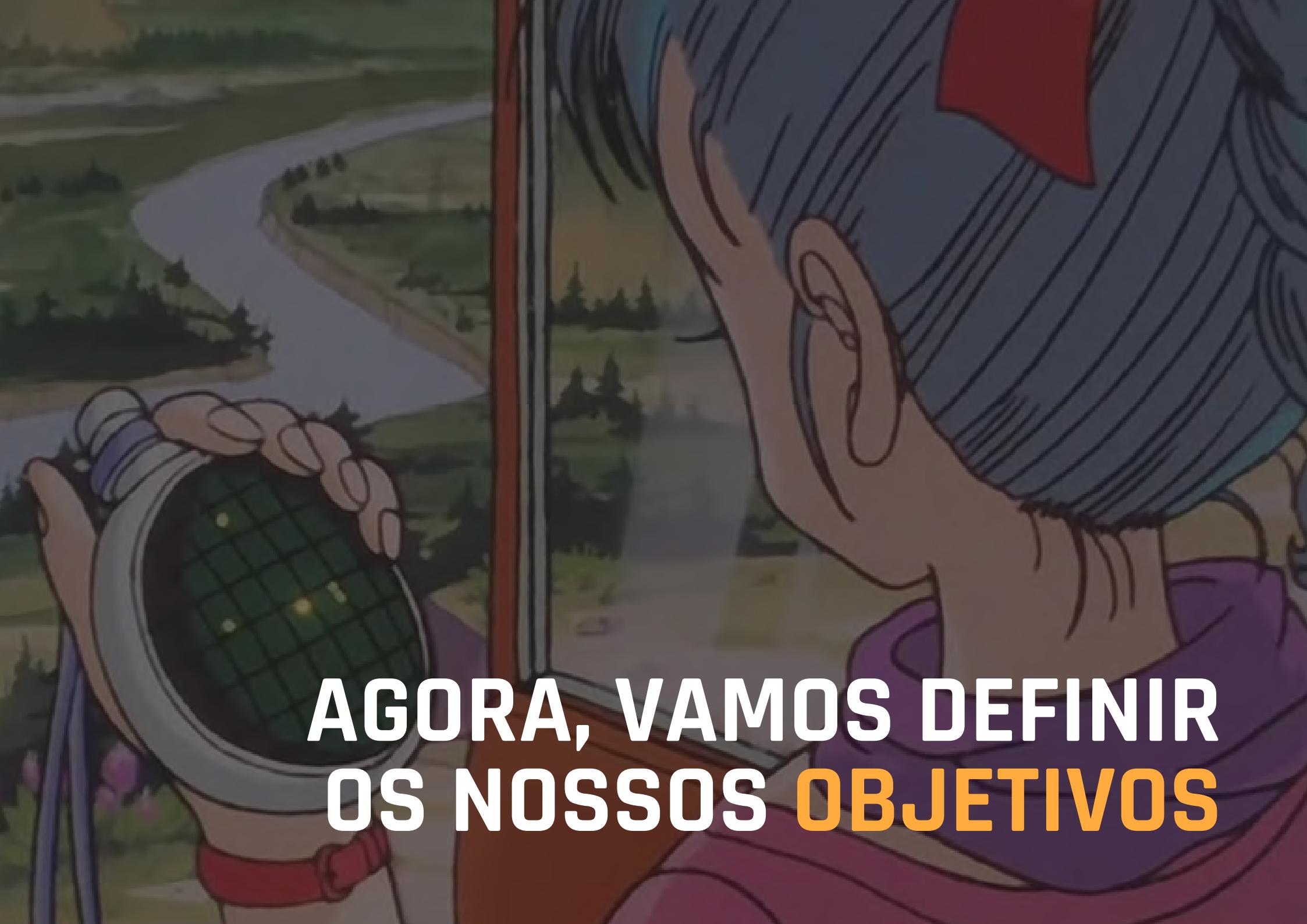
- Default parameters
- Arrow functions
- Template literal
- multiline string
- destructuring assignment
- Promises
- Classes
- ...



ES2016 ou **ES7**

- Array.includes
- Exponentiation Operator
- ...



A hand is shown from the right side of the frame, wearing a purple sleeve, holding a green and blue globe. The globe is tilted, showing a grid pattern. In the background, through a window, a road map of Brazil is visible, showing state borders and major cities like São Paulo and Rio de Janeiro.

AGORA, VAMOS DEFINIR
OS NOSSOS OBJETIVOS

AO PENSAR NA TODOLIST

Quais são as operações que
são possíveis em uma todolist?

Linguagens para aprender

- Javascript
- C#
- Swift
- Closure

Para comprar no mercado

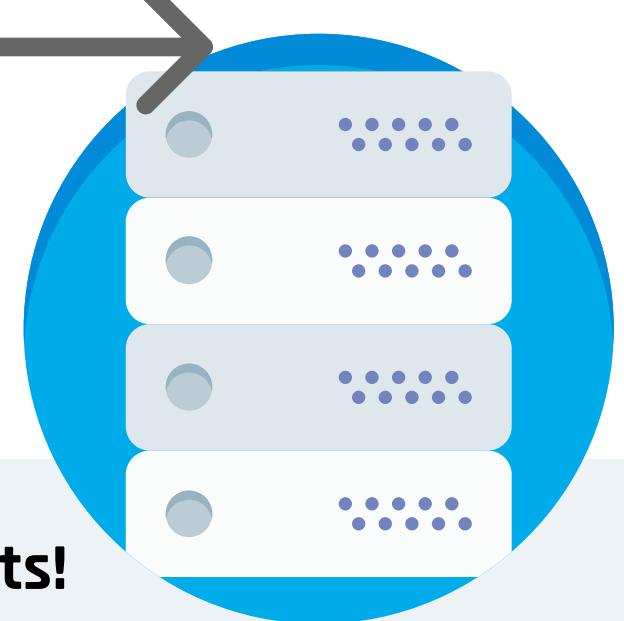
- 12 Maçãs
- 12 batatas
- 5 pacotes de pipocas
- 2 óreos

Linguagens para aprender

- Javascript
- C#
- Swift
- Closure

- Pegar todas as TodoLists
- Criar uma TodoList
- Criar uma Todo
- Marcar uma Todo como feito
- Atualizar uma TodoList
- Deletar uma TodoList
- Deletar uma Todo

quero pegar todas as todolists



Pegaê todas as todolists!

Front-End

Back-End

através de **requisições http!**
em alguma URL!

facebook

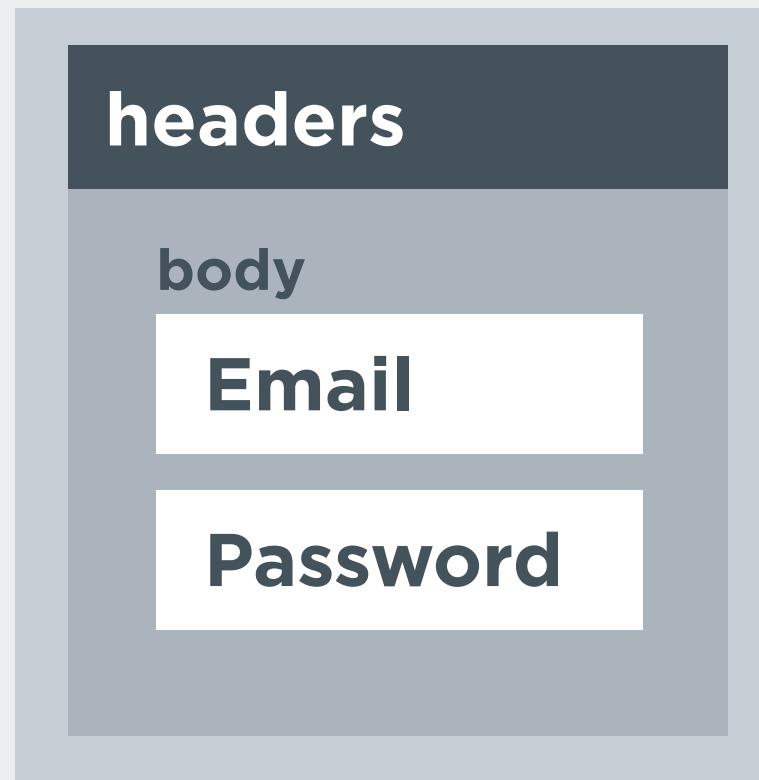
Email or Phone

Password

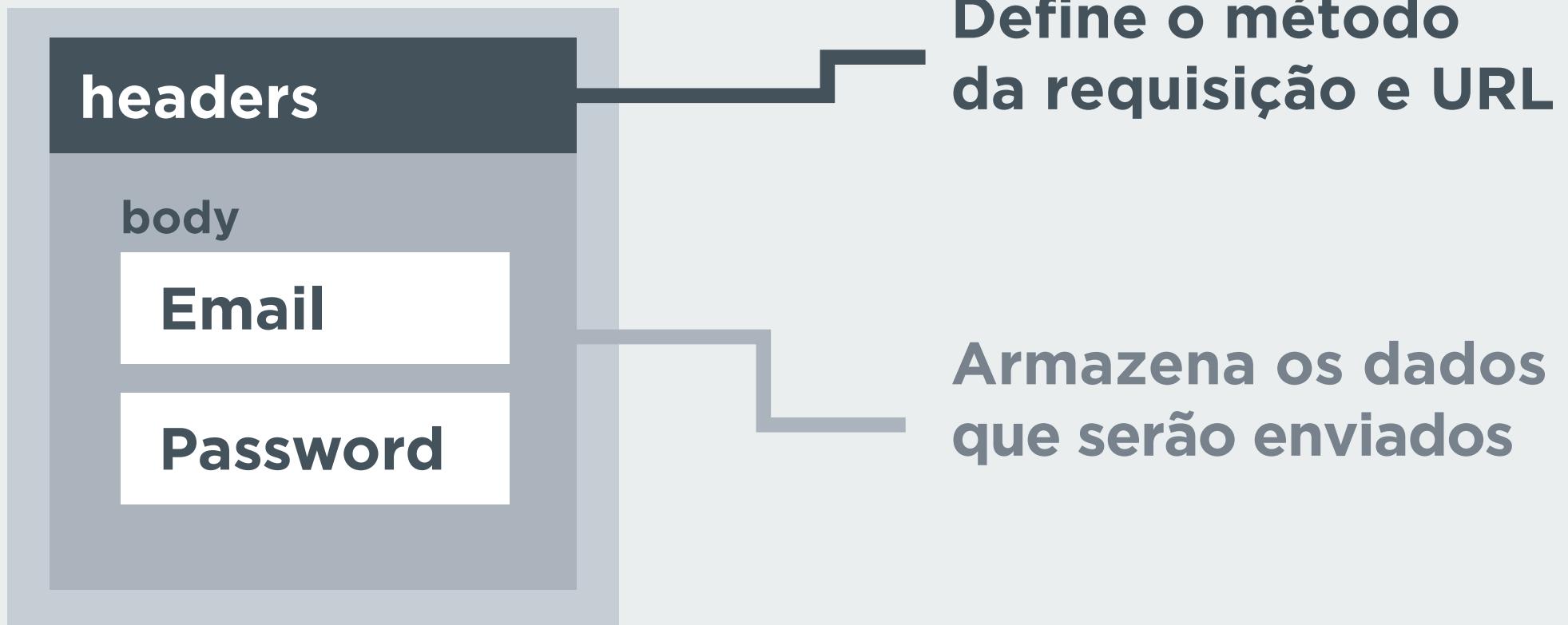
[Forgot account?](#)



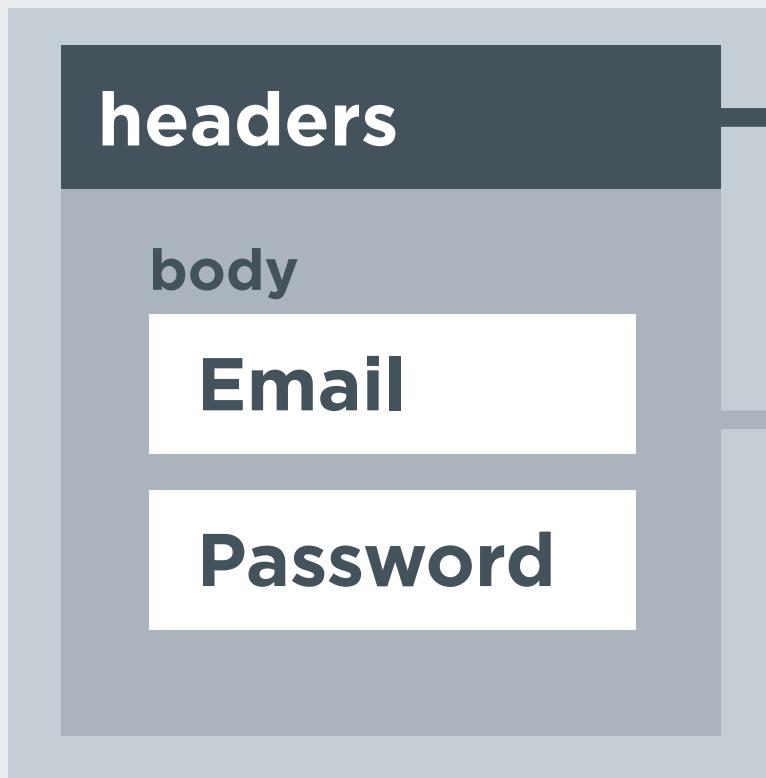
Estrutura de uma requisição HTTP



Estrutura de uma requisição



Request Structure



https://facebook.com/signin

POST

email:

kenjishiromajp@gmail.com

password:

myverysecretpassword

WebService



A screenshot of a web browser window displaying a JSON response. The URL in the address bar is `https://viacep.com.br/ws/04195140/json/`. The page content shows a single JSON object with the following fields and values:

```
{  
  "cep": "04195-140",  
  "logradouro": "Rua Antônio Guarmerino",  
  "complemento": "",  
  "bairro": "Jardim Celeste",  
  "localidade": "São Paulo",  
  "uf": "SP",  
  "unidade": "",  
  "ibge": "3550308",  
  "gia": "1004"  
}
```



POST - /LISTARTODOLISTS

?

? ?

GET - /BATATA

GET - /CRIARTODOLISTS

E COMO DECIDO
ESSAS URLs?

? -~

POST - /DELETATODOLIST

PATCH - /PEGARTODOLISTS

REST VAI NOS DAR A RESPOSTA!

RE^ST

Representational
State
Transfer

Verbo/Métodos

{

“create”	:	“POST” ,
“read”	:	“GET” ,
“update”	:	“PUT” ,
“partial_update”	:	“PATCH” ,
“delete”	:	“DELETE”

}

Status Code

```
{  
  "informational" : [100...199],  
  "successfull"  : [200...299],  
  "redirection"  : [300...399],  
  "clientErrors" : [400...499],  
  "serverErrors" : [500...599]  
}
```

COMO FICARIA NOSSOS ENDPOINTS?

- Pegar todas as TodoLists
- Criar uma TodoList
- Criar uma Todo
- Marcar uma Todo como feito
- Atualizar uma TodoList
- Deletar uma TodoList
- Deletar uma Todo

COMO FICARIA NOSSOS ENDPOINTS?

- Pegar todas as TodoLists GET /todolists
- Criar uma TodoList POST /todolists
- Criar uma Todo POST /todos
- Marcar uma Todo como feito PUT/PATCH /todos/{id}
- Atualizar uma TodoList PUT/PATCH /todolists/{id}
- Deletar uma TodoList DELETE /todolists/{id}
- Deletar uma Todo DELETE /todos/{id}

**AGORA TEMOS UM OBJETIVO
MAIS CONCRETO!**

OBJETIVO:

CRIAR TODOS ESTES ENDPOINTS!

- | | |
|------------------------------|---------------------------|
| · Pegar todas as TodoLists | GET /todolists |
| · Criar uma TodoList | POST /todolists |
| · Criar uma Todo | POST /todos |
| · Marcar uma Todo como feito | PUT/PATCH /todos/{id} |
| · Atualizar uma TodoList | PUT/PATCH /todolists/{id} |
| · Deletar uma TodoList | DELETE /todolists/{id} |
| · Deletar uma Todo | DELETE /todos/{id} |

Linguagens para aprender

Javascript

C#

Swift

Closure

Quais as propriedades você enxerga nesta todolist?

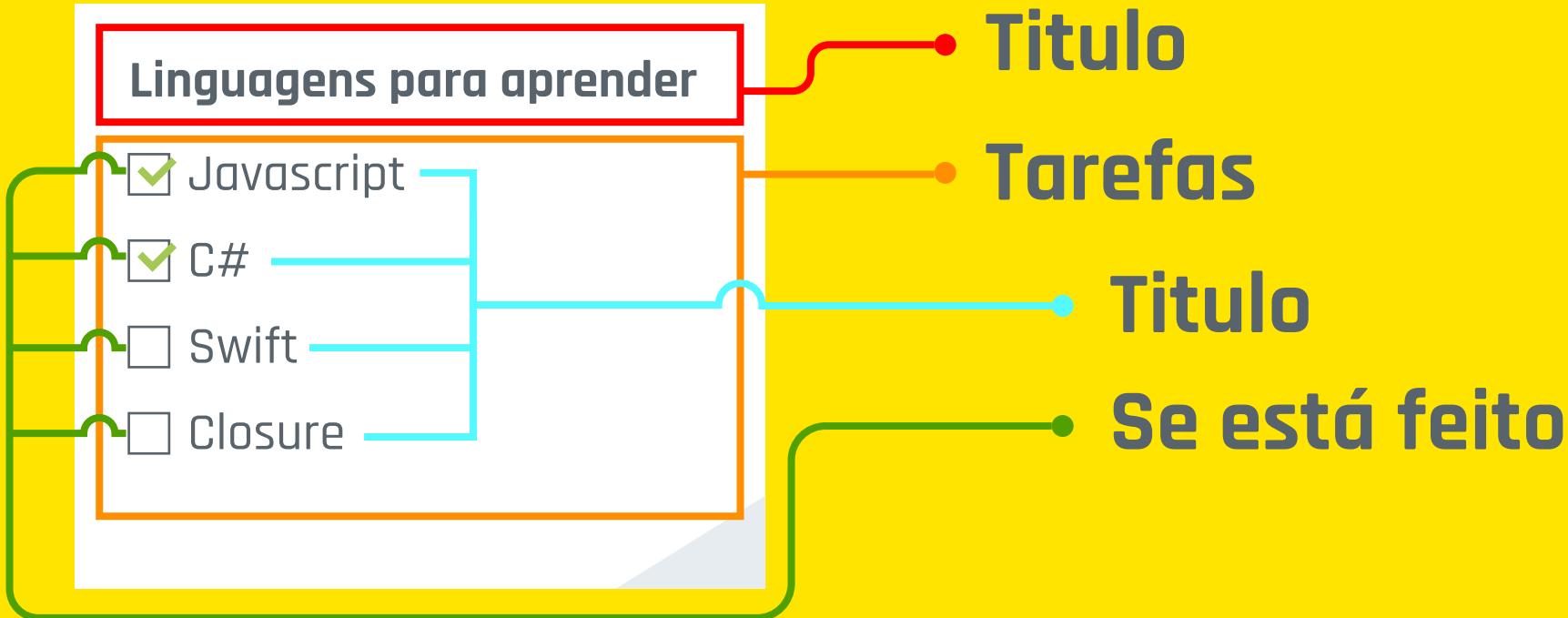
Linguagens para aprender

Javascript

C#

Swift

Closure



Vamos neste passo...

- **Levantar um serviço http**
- **Criar uma rota para nossa aplicação**
- **Definir tipos na aplicação**
- **Como mandar response para o client**

MÃO NA MASSA!

A faint, semi-transparent watermark of a man's face and hands is visible against a bright yellow background. The man has a serious expression, looking directly at the viewer. His hands are clasped together in front of him. The overall effect is like a watermark or a stylized logo.

REVISÃO

Hapi.Server

```
yarn add hapi
```

```
yarn add @types/hapi -D
```

Pacote para sabermos os tipos dos parâmetros e acesso a algumas documentações direto da IDE

```
import * as Hapi from 'hapi';
```

```
export const server = new Hapi.Server({
```

port: 8080, ← Define a porta que vai subir o serviço http

```
routes: {
```

```
    cors: {
```

```
        origin: ["*"]
```

Cross-Origin Resource Sharing
para aceitar de qualquer origem

```
}
```

```
});
```

Definindo Rota

```
server.route({  
    path: "/todolists", ← url da rota  
    method: "GET", ← Verbo/Método da Requisição  
    handler: (request: Hapi.Request, h: Hapi.ResponseToolkit) => {  
        const todolists = [ ...  
        ]  
        return h.response(todolists).code(200); ← passa o corpo da response  
    }  
});  
server.start(); ← passa o statusCode da response  
                  sobe o serviço http
```

Tipando os parâmetros

(request: Hapi.Request, h: Hapi.ResponseToolkit)



Esse é o parâmetro : Esse é o tipo

Arrow function

Versão
antiga

```
const _a = function(param){  
};
```

Versão
com arrow
function

```
const a = (param) => {  
}
```

Versão
antiga

```
const _b = function(param){  
    return "com retorno"  
};
```

Versão
com arrow
function

```
const b = (param) => "com retorno";
```

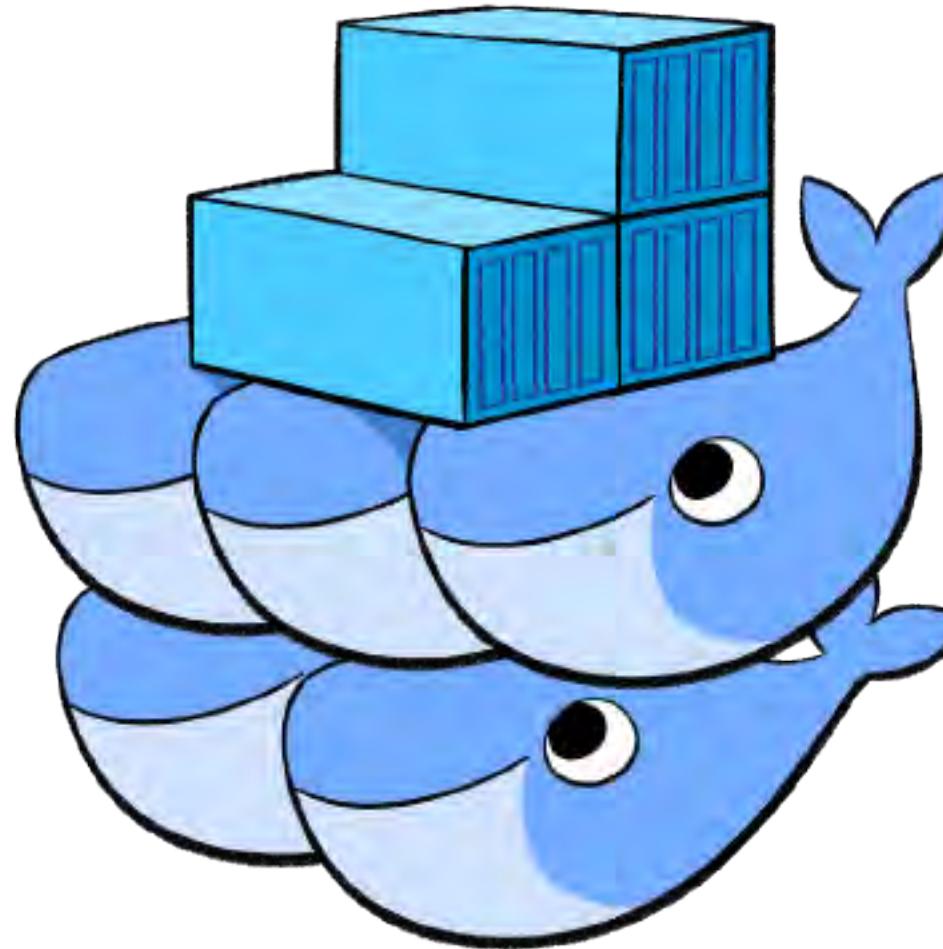


mongoDB[®]

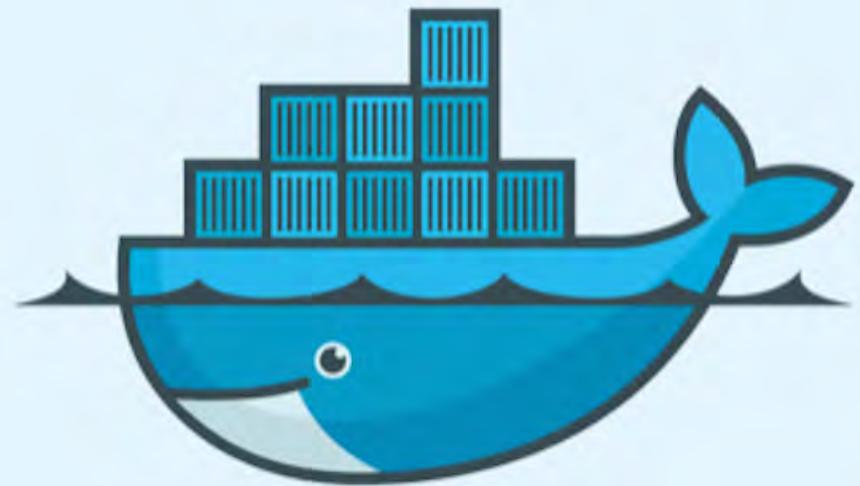
**Agora vamos fazer o retorno
dos dados com MongoDB**



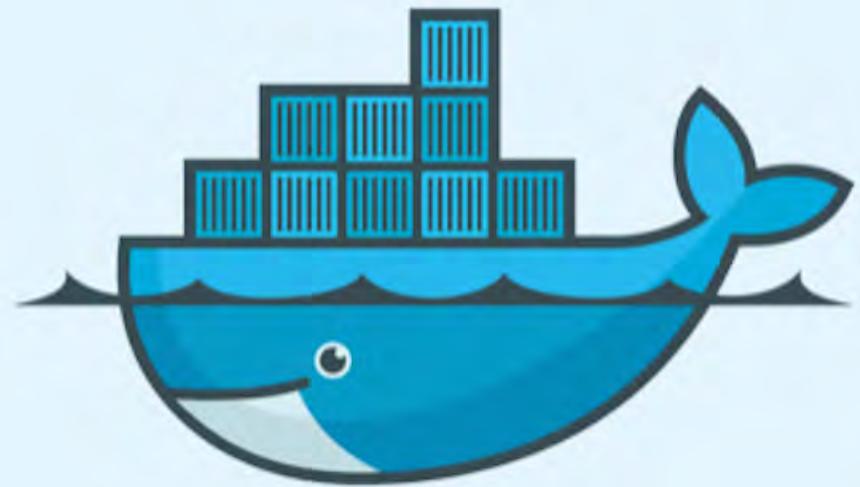
**Para isso teria que instalar o mongo,
baixar o client... (mais configurações)**



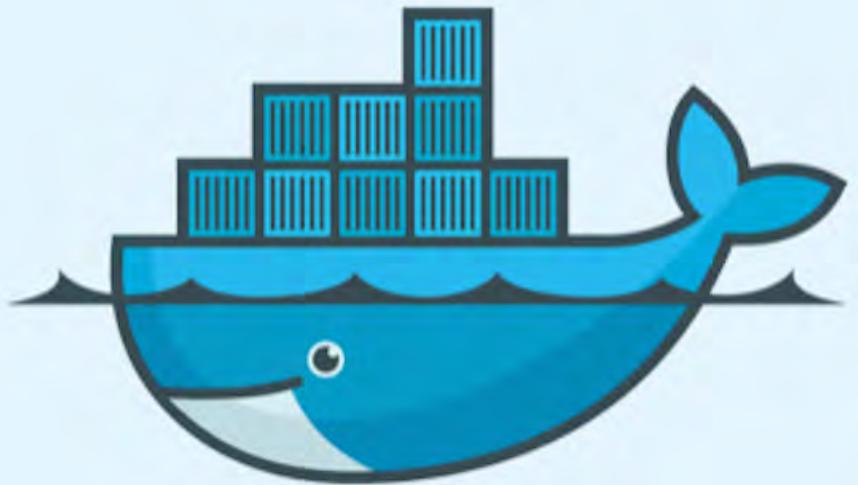
Por isso vamos usar o Docker compose!



O QUE É
Docker?

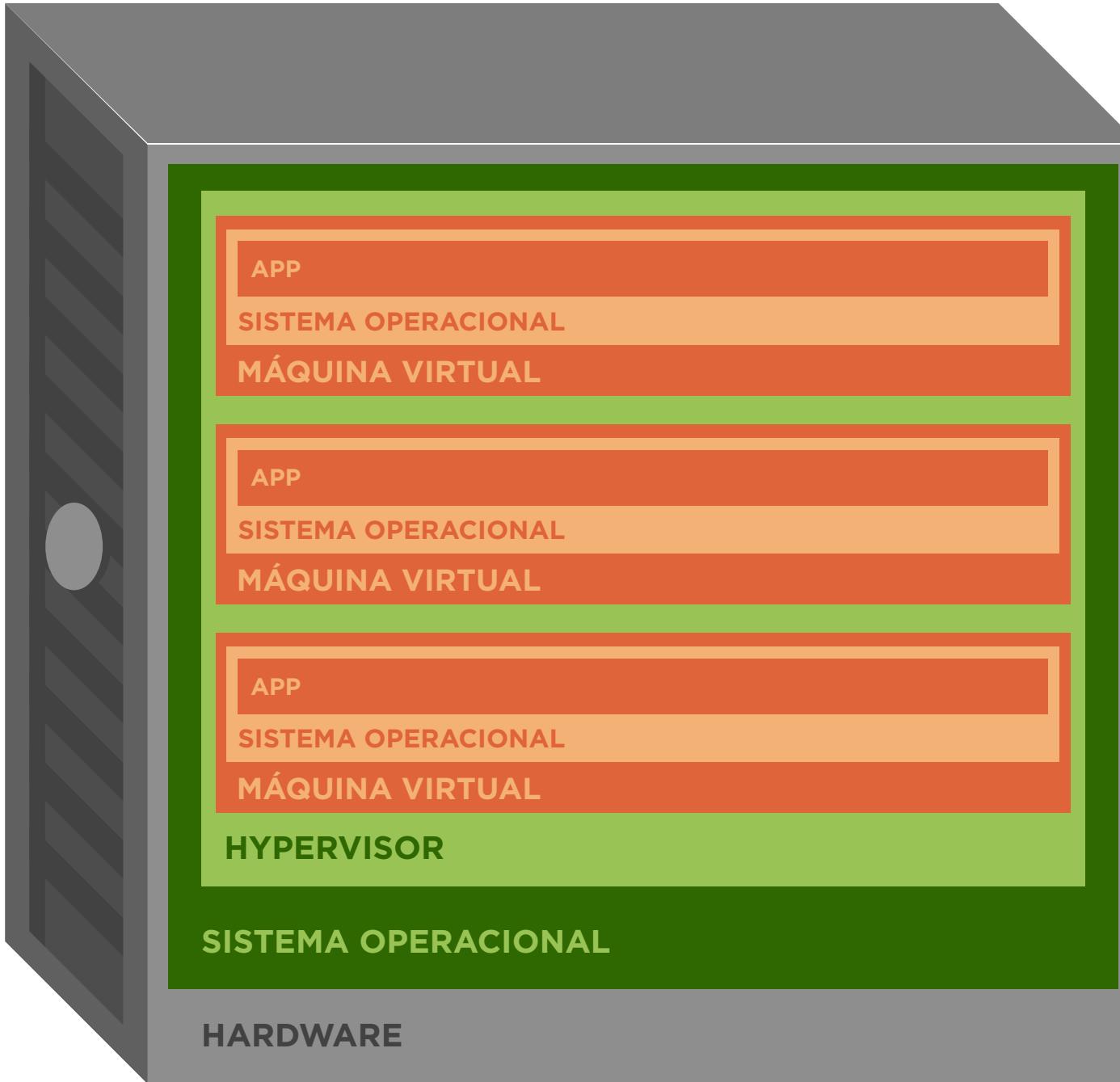


O QUE É
Docker?

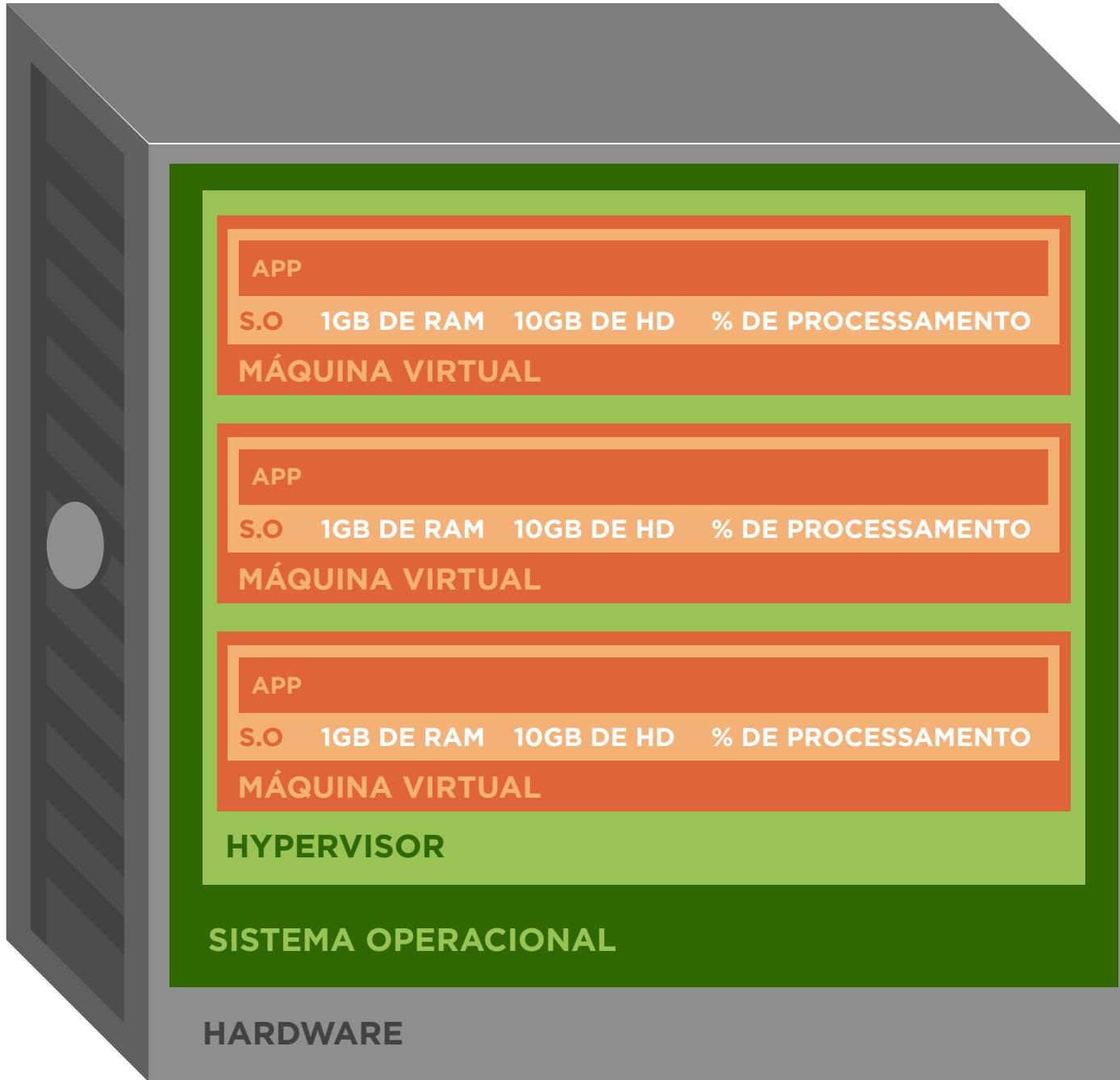


Docker é uma série de ferramentas que utiliza-se de **containers** para ajudar na criação de um **ambientes isolados**

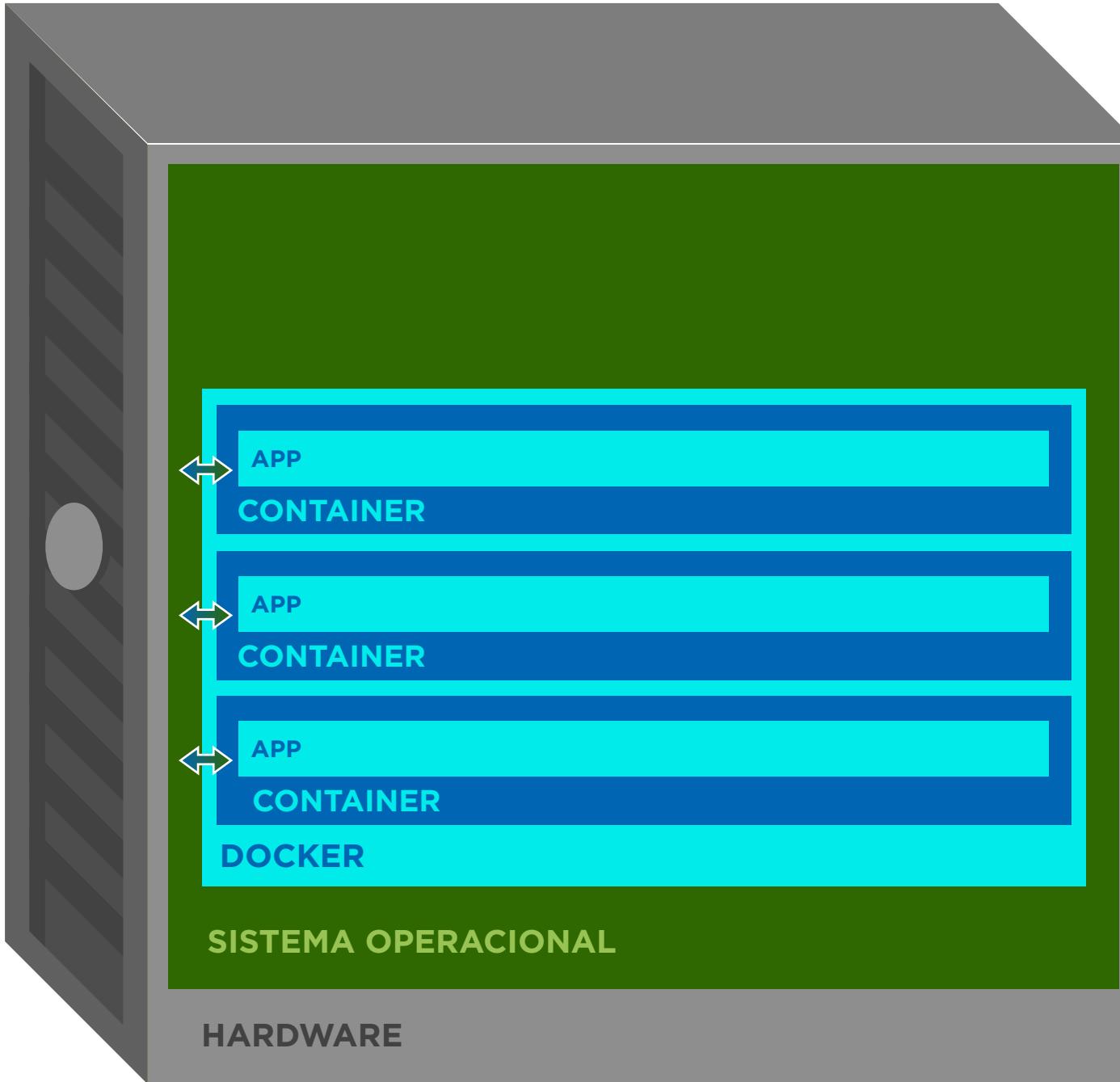
O que é Container? é tipo máquina virtual?



O que é Container? é tipo máquina virtual?



O que é Container? é tipo máquina virtual?



Vamos usar dois carinhos do Docker

Vamos usar dois carinhos do Docker

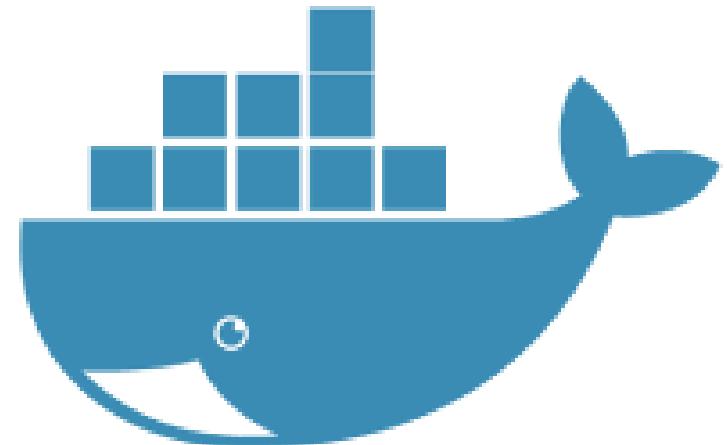


Docker Compose

Vamos usar dois carinhos do Docker



Docker Compose



Docker HUB

Vamos usar dois carinhos do Docker

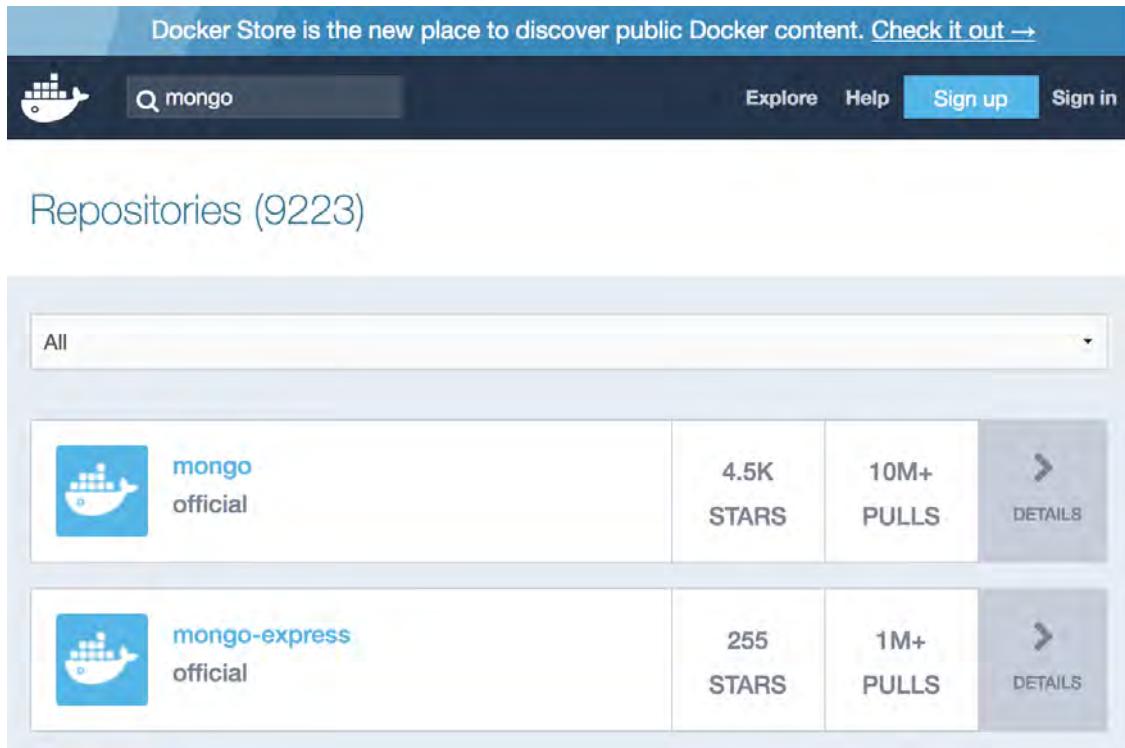


- Container com **MongoDb**
- Container com o **MongoDb Client**
versão interface para
acessar o mongodb

Docker Compose

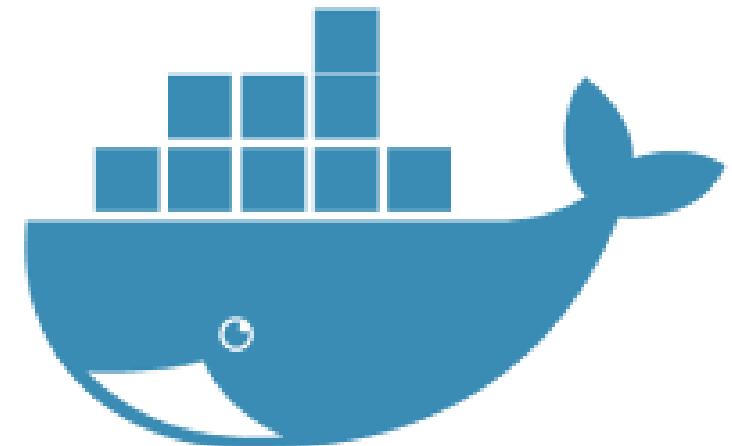
Vamos usar dois carinhos do Docker

Docker Store is the new place to discover public Docker content. [Check it out →](#)



The screenshot shows the Docker Hub interface. At the top, there's a search bar with the query "mongo". Below the search bar, there are navigation links for "Explore", "Help", "Sign up", and "Sign in". A blue banner at the top says "Docker Store is the new place to discover public Docker content. Check it out →". The main area is titled "Repositories (9223)". It shows two repository cards: "mongo" (official) with 4.5K stars and 10M+ pulls, and "mongo-express" (official) with 255 stars and 1M+ pulls. Each card has a "DETAILS" button.

Repositório de imagens



Docker HUB

Vamos neste passo...

- Como criar um container a partir de um docker-compose
- Como podemos utilizar o serviço dos containers
- Aprender o que é volume para que serve

MÃO NA MASSA!

A yellow-toned illustration of Doctor Strange in his traditional robes and mask, holding his staff.

REVISÃO

```
docker ps
```

Lista todos os containers ligados

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dcb846692189	mongo-express	"tini -- node app"	4 days ago	Up 2 hours	0.0.0.0:8081->8081/tcp	projeto_test_mongo-express_1
38e6a9524250	mongo	"docker-entrypoint.s..."	4 days ago	Up 2 hours	0.0.0.0:27017->27017/tcp	projeto_test_mongo_1

```
docker ps -a
```

Lista todos os containers ligados e desligados

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
dcb846692189	mongo-express	"tini -- node app"	4 days ago	Restarting (1) Less than a second ago		projeto_test_mongo-express_1
38e6a9524250	mongo	"docker-entrypoint.s..."	4 days ago	Exited (0) 28 seconds ago		projeto_test_mongo_1

```
docker stop <id>
```

Desliga o container com este id

`docker container prune`

Remove todos os container parados

```
docker-compose up
```

```
docker-compose.yml
```

```
version: '3.1'
```

```
services:
```

```
mongo:  
  image: mongo  
  restart: always  
  ports:  
    - 27017:27017  
  volumes:  
    - ./db:/data/db
```

mongo: imagem que achamos no docker hub

restart: always para reiniciar toda vez que monta

ports: Sincroniza a porta do container para a máquina host
[porta_do_host]:[porta_do_container]

volumes: Sincroniza a pasta db do host com a pasta /data/db do container
[pasta_do_host]:[pasta_do_container]

```
mongo-express:  
  image: mongo-express  
  restart: always  
  depends_on:  
    - mongo  
  ports:  
    - 8081:8081
```

Diz que só iniciará quando o mongo iniciar

**VAMOS MANIPULAR
OS DADOS DO BANCO!**

Vamos neste passo...

- **Vamos conectar com o banco mongodb**
- **Vamos criar os Schemas**
- **Vamos criar os modelos**
- **Pegar os dados que estão no banco e jogar na response**

MÃO NA MASSA!



REVISÃO

Mongoose.connect()

Conectando com o nosso banco

```
Mongoose.connect("mongodb://localhost:27017/todolists");
```

string de conexão conectando **localhost**
na porta **27017** no banco **todolists**

Mongoose.connect()

string de conexão conectando **localhost**
na porta **27017** no banco **todolists**

```
const todoListSchema = new Mongoose.Schema({  
  name: {type: String, required: true},  
  todos: [  
    {  
      name: { type: String, required: true },  
      done: { type: Boolean }  
    }]  
});
```

Model.find()

```
const todoListModel = Mongoose.model<ITodoList>(  
  "TodoList", ← Nome do meu Modelo  
  todoListSchema,  
  "todoLists" ← Nome da minha collection  
);
```

```
interface ITodo extends Mongoose.Document{  
  name: string;  
  done: boolean;  
}  
  
interface ITodoList extends Mongoose.Document{  
  name: string;  
  todos: ITodo[],  
}
```

Interface

Obriga o objeto ou classe que implementa a mesma ter essas propriedades com esses tipos

async

Indica que neste método possui algum método assíncrono sendo resolvido com await

```
server.route({
  path: "/todolists",
  method: "GET",
  handler: async (request: Hapi.Request, h: Hapi.ResponseToolkit) => {
    const todolists = await todoListModel.find();
    return h.response(todolists).code(200);
  }
});
```

await

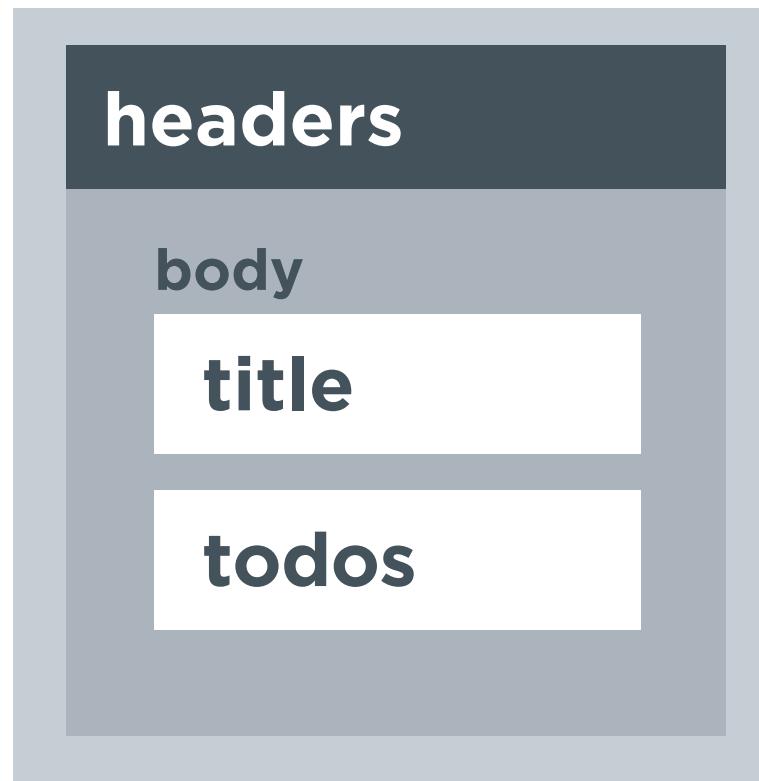
Indica que vamos esperar o método assíncrono completar para continuar a execução

OBJETIVO:

CRIAR TODOS ESTES ENDPOINTS!

• Pegar todas as TodoLists	GET /todolists
• Criar uma TodoList	POST /todolists
• Criar uma Todo	POST /todos
• Marcar uma Todo como feito	PUT/PATCH /todos/{id}
• Atualizar uma TodoList	PUT/PATCH /todolists/{id}
• Deletar uma TodoList	DELETE /todolists/{id}
• Deletar uma Todo	DELETE /todos/{id}

Estrutura de uma requisição HTTP





POSTMAN



Postman Makes API Development Simple.

Developers use Postman to build modern software for the API-first world.

[Download the App](#)

Want to improve your Postman skills?
[Take the next step!](#)



Vamos neste passo...

- Organizar o conteúdo
- Como cadastrar uma todolist
- Como pegar o corpo de um requisição

MÃO NA MASSA!

A yellow-toned illustration of Doctor Strange in his traditional robes and mask, holding his staff.

REVISÃO

src	
database	Minha database
TS index.ts	Interface da Database
TS interface.ts	
routes	
todo	
TS interface.ts	Interfae da todo
todolist	
TS controller.ts	Parte lógica de cada rota
TS index.ts	onde fazemos o registro das rotas
TS interface.ts	interface da Todolist
TS model.ts	Modelo e Schema do Todolist
TS index.ts	
TS server.ts	Inicializa a aplicação e registra todas as rotas Instancia o servidor com configurações da mesma

```
try{  
    const { payload } = request;  
    const todolist = await this.database.todoListModel.create(payload);  
    return h.response(todolist).code(201);  
}catch(error){  
    return Boom.badImplementation(error);  
}
```

Pega o corpo da requisição

Cadastra a partir do model que criamos

Se der erro na hora de criar podemos mandar um erro