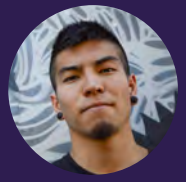
Criando webservice REST com NodeJS, NoSQL & Docker



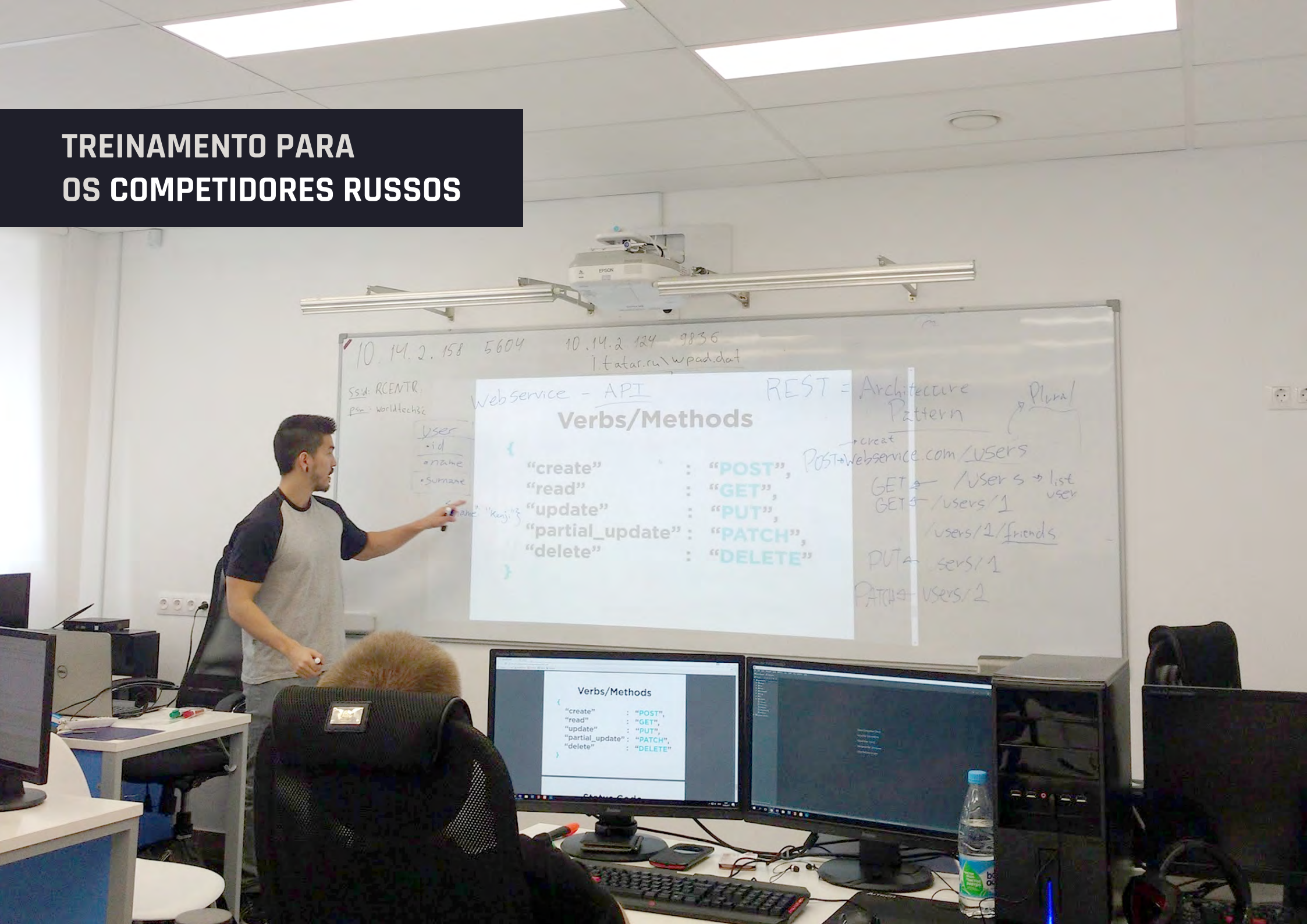
Giovanni Kenji Shiroma
DESENVOLVEDOR FULL-STACK
\ \\ SENAI DE INFORMÁTICA

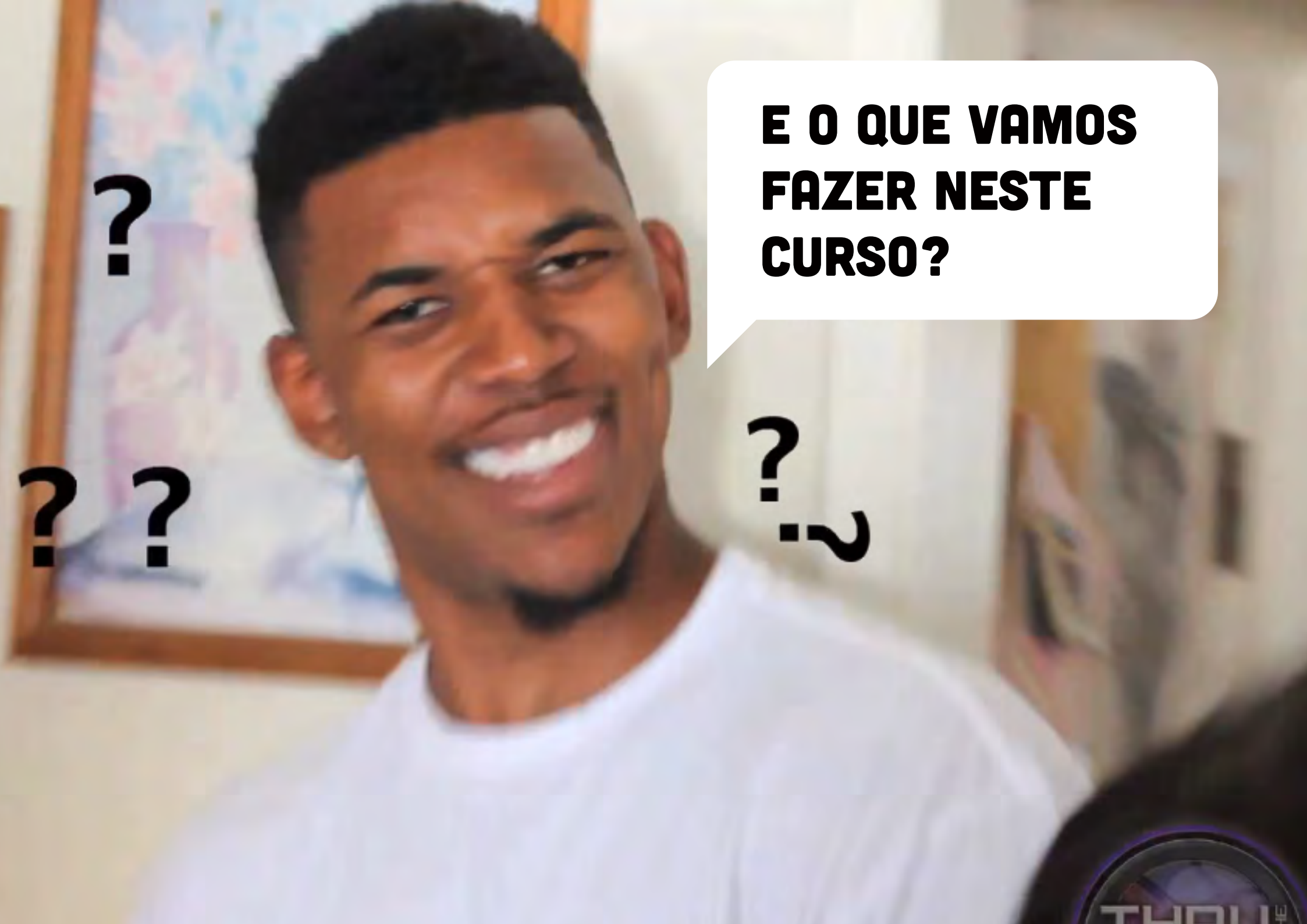

SENAI

Competidor da worldskills na modalidade de

WebDesign & Development 2015

TREINAMENTO PARA OS COMPETIDORES RUSSOS




E O QUE VAMOS FAZER NESTE CURSO?
?
? ?
? .~

BACK-END DE UMA TODOLIST!

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

Para comprar no mercado

- [x] 12 Maçãs
- [x] 12 batatas
- [ ] 5 pacotes de pipocas
- [ ] 2 óreos

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

Para comprar no mercado

- [x] 12 Maçãs
- [x] 12 batatas
- [ ] 5 pacotes de pipocas
- [ ] 2 óreos

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

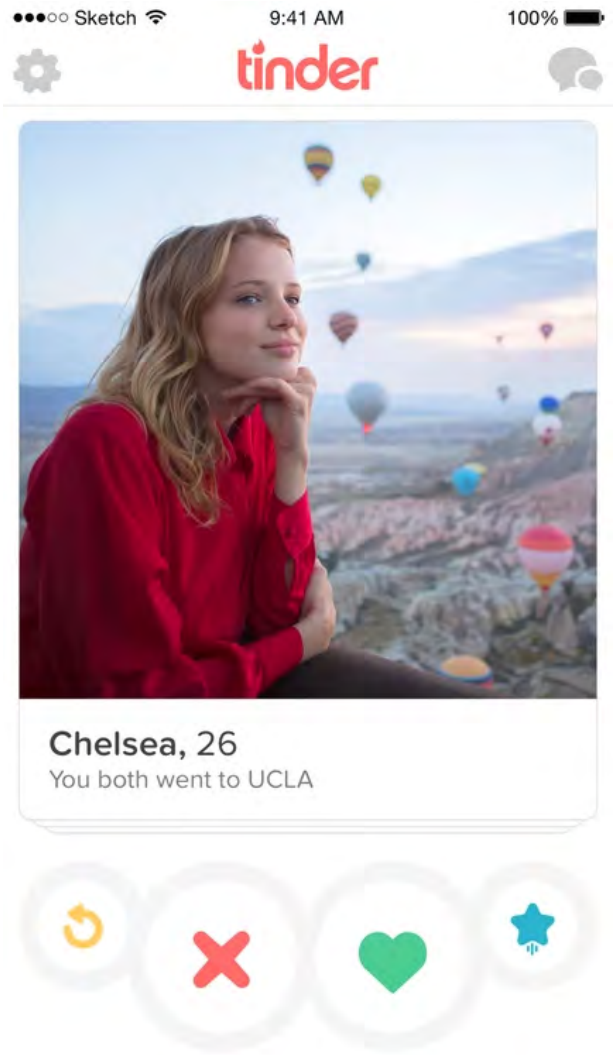
Mas primeiro...
Qual a diferença entre o front-end e back-end?

Front-End

Back-End

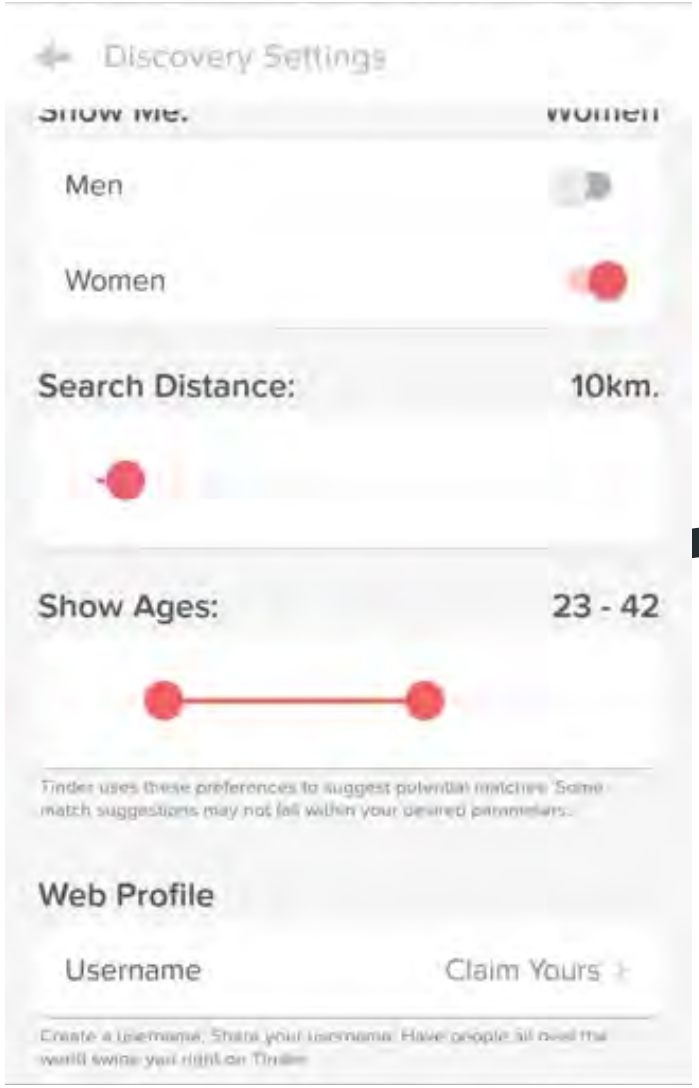
Front-End

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

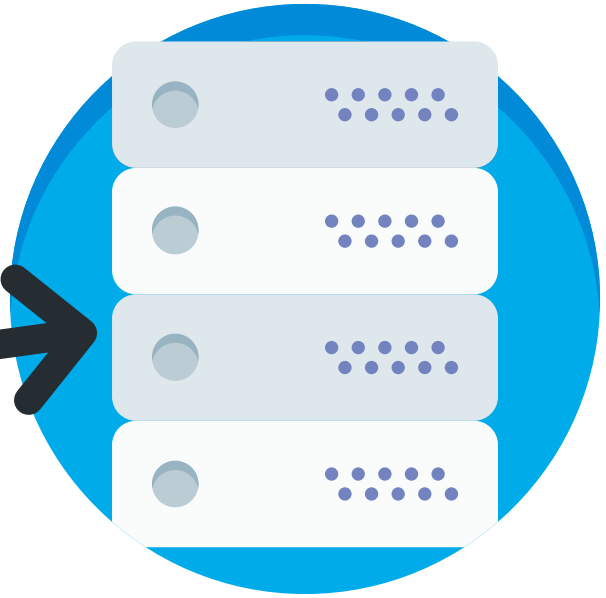


Back-End

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

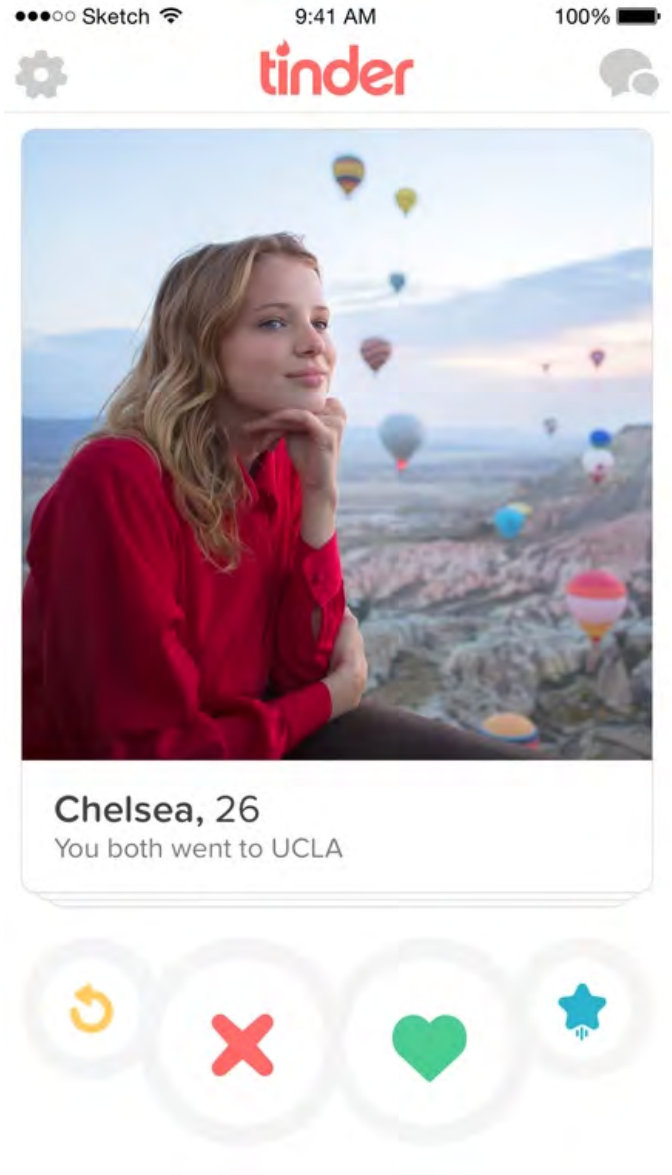


Eu quero **Mulheres** com máximo de **10 km** de distancia entre **23 a 42 anos**!

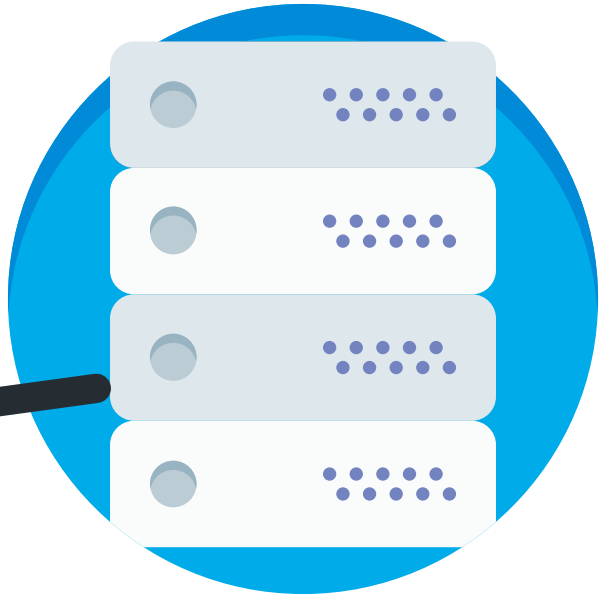


Back-End

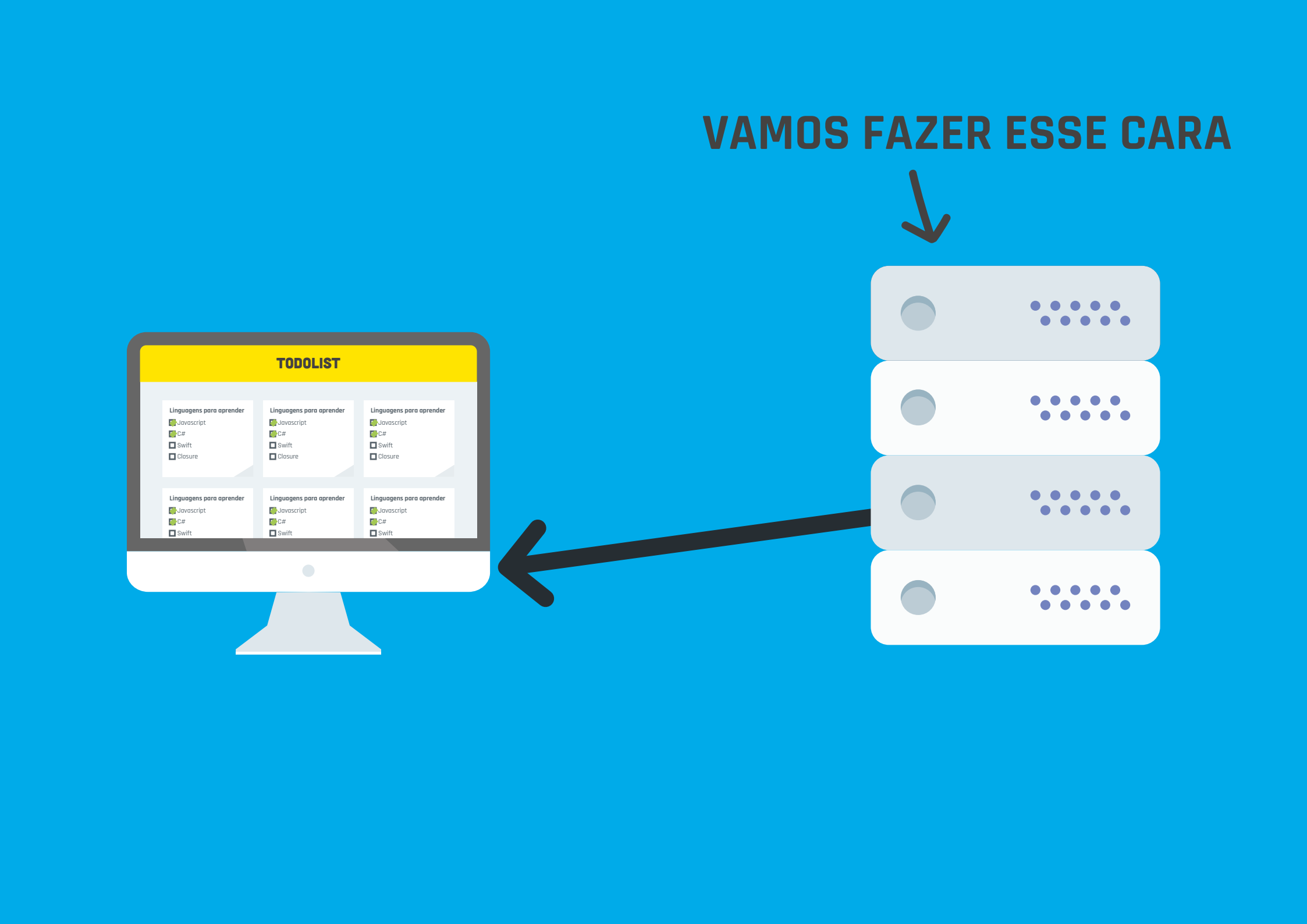
Recebe os pedidos do front-end, responsável pelo tratamento de dados e regras de negócios



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

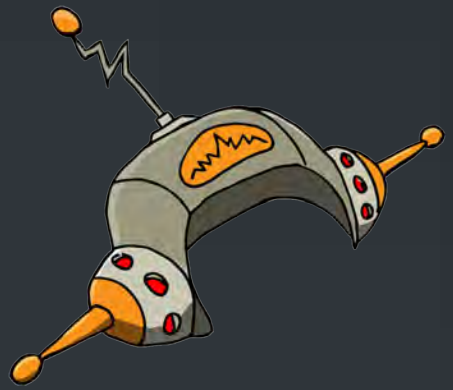


FRONT-END sem o BACK-END seria tipo:


VAMOS FAZER ESSE CARA
TODOLIST
Linguagens para aprender
Javascript
C#
Swift
Closure

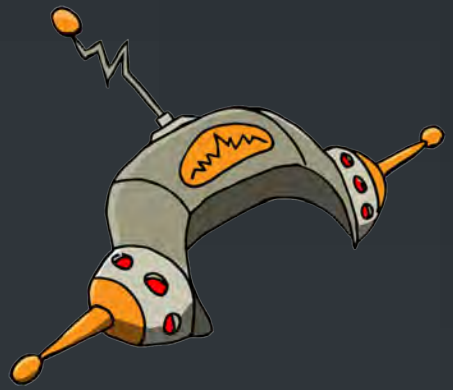

node
JS
NodeJS

NodeJS



Hapi.js

NodeJS



Hapi.js

Typescript

NodeJS

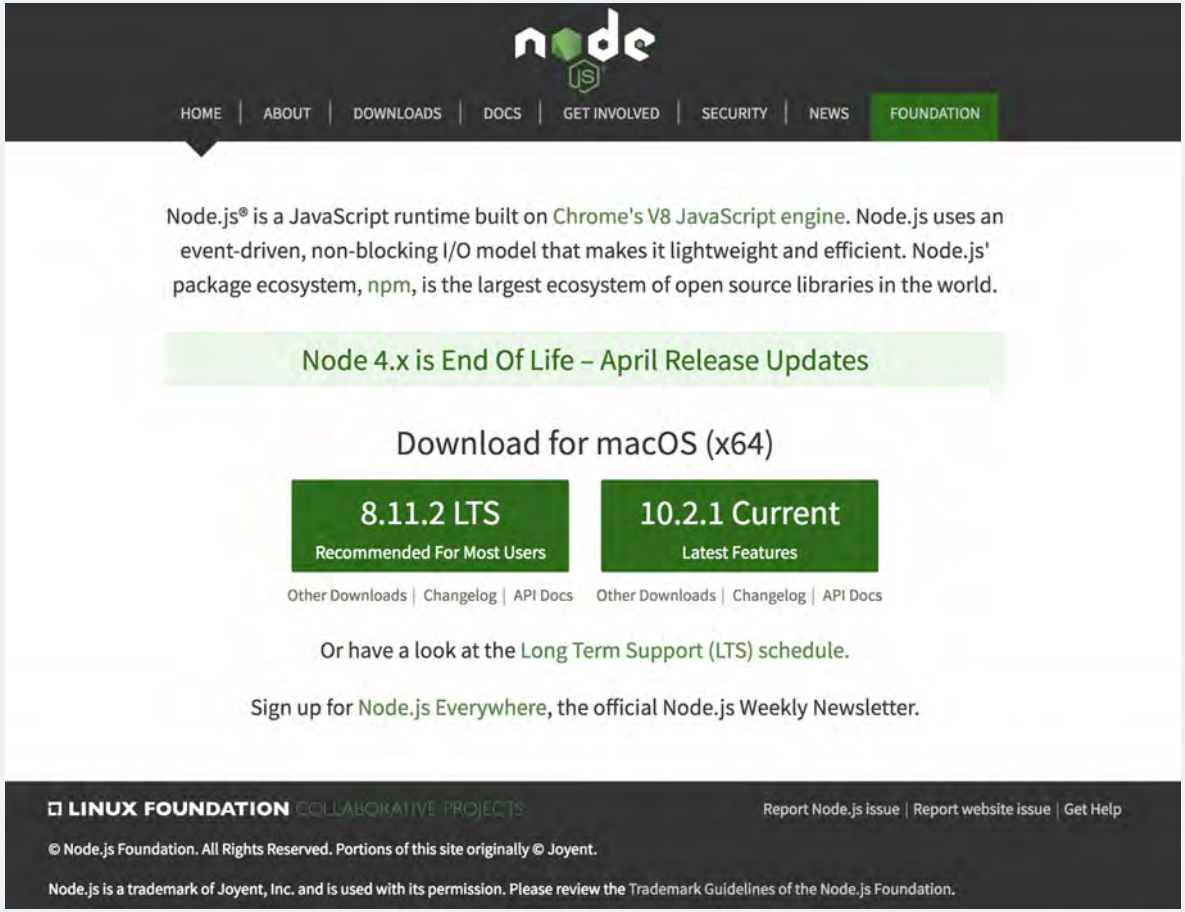
Hapi.js

Typescript

mongoDB®

PREPARAÇÕES PARA COMEÇAR A DESENVOLVER

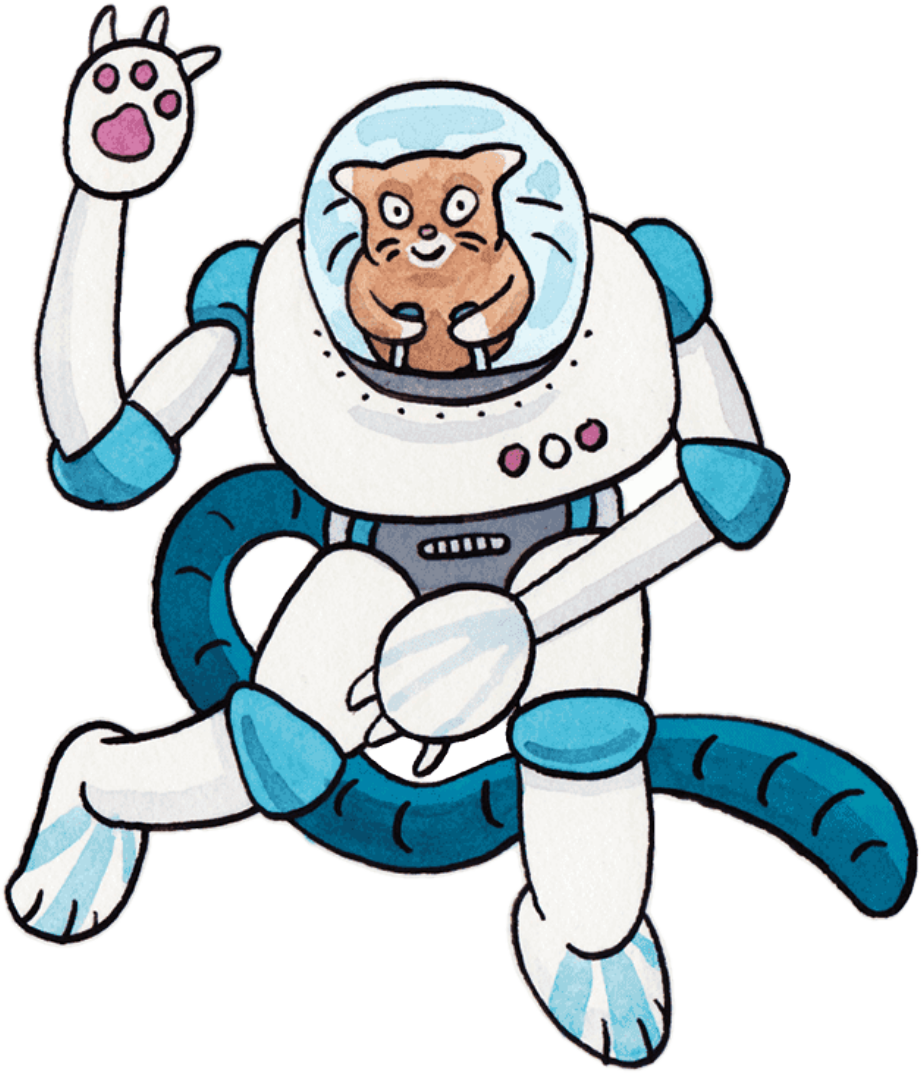
NodeJS SERVIDOR JAVASCRIPT



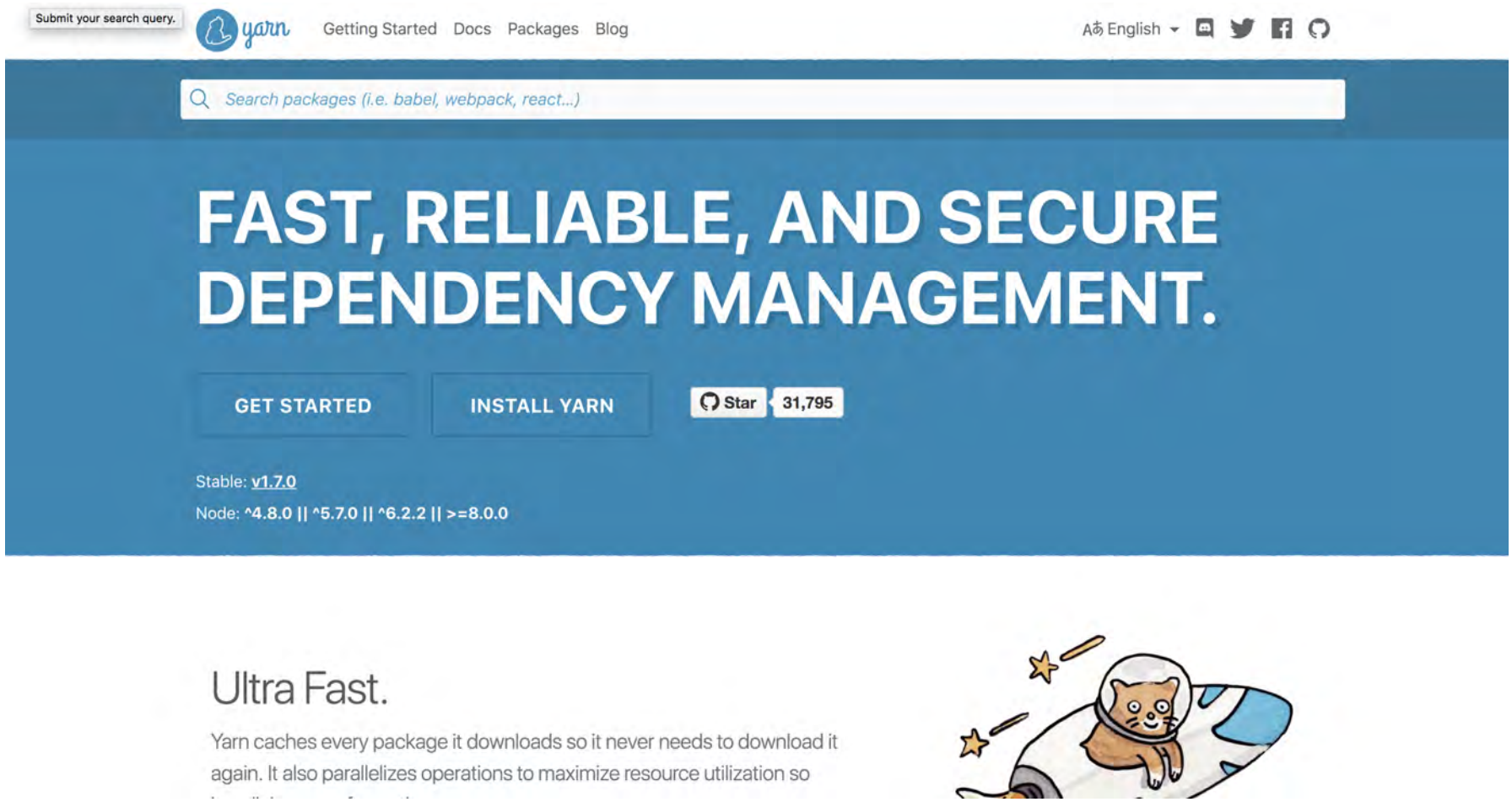
HTTPS://NODEJS.ORG/EN/

No MAC PODE instalar pelo BREW!

BREW INSTALL NODE



yarn



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!


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

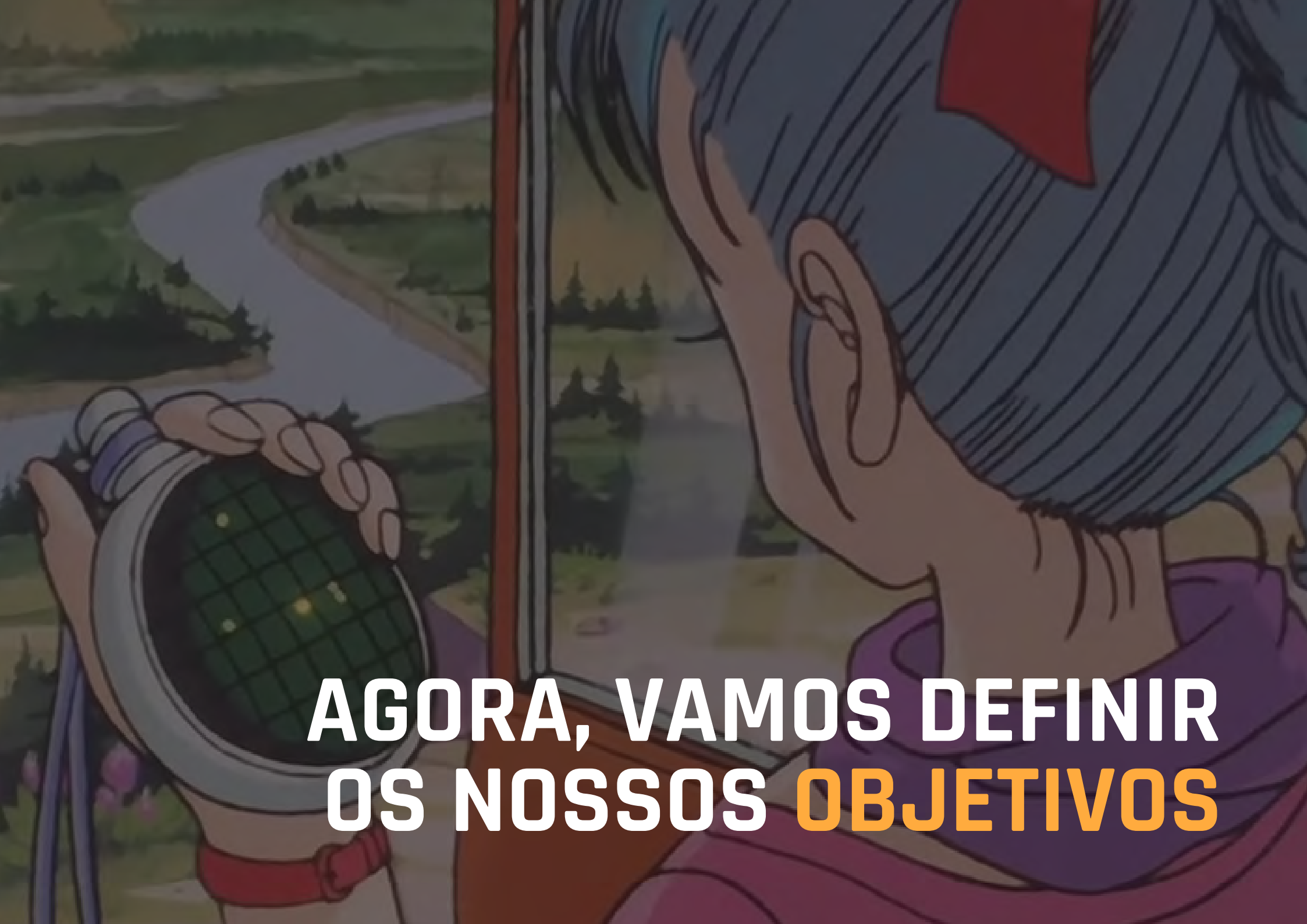
NOVAS FUNCIONA-LIDADES!

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

ES2016 ou
ES7

NOVAS FUNCIONA-LIDADES!

- Array.includes
- Exponentiation Operator
- ...



AGORA, VAMOS DEFINIR OS NOSSOS OBJETIVOS

AO PENSAR NA TODOLIST

Quais são as operações que são possíveis em uma todolist?

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

Para comprar no mercado

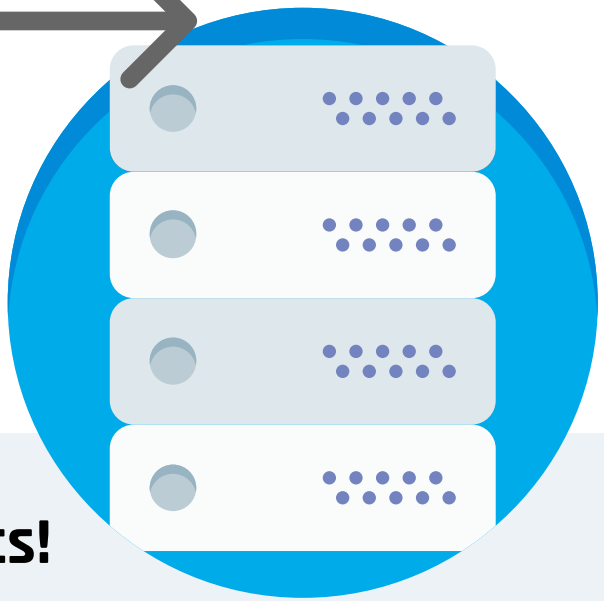
- [x] 12 Maçãs
- [x] 12 batatas
- [ ] 5 pacotes de pipocas
- [ ] 2 óreos

Linguagens para aprender

- [x] Javascript
- [x] 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

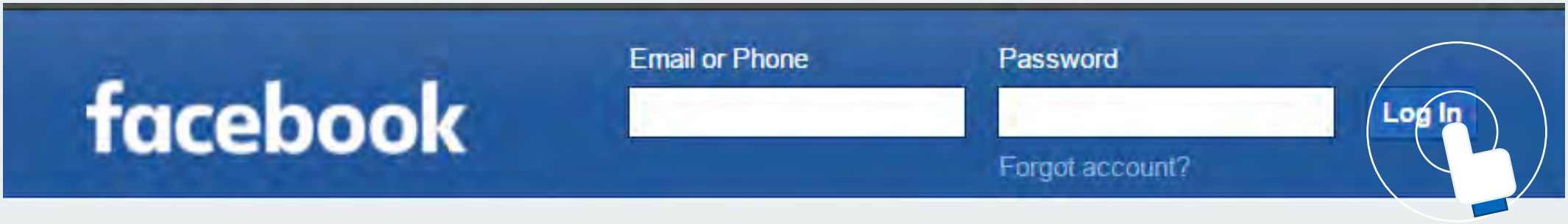


Pegaê todas as todolists!

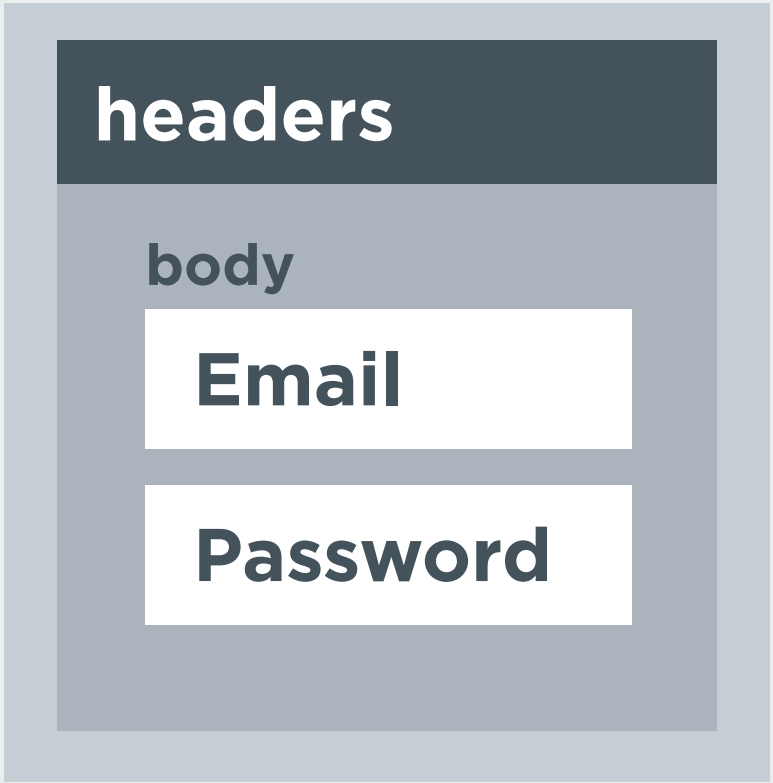
Front-End

Back-End

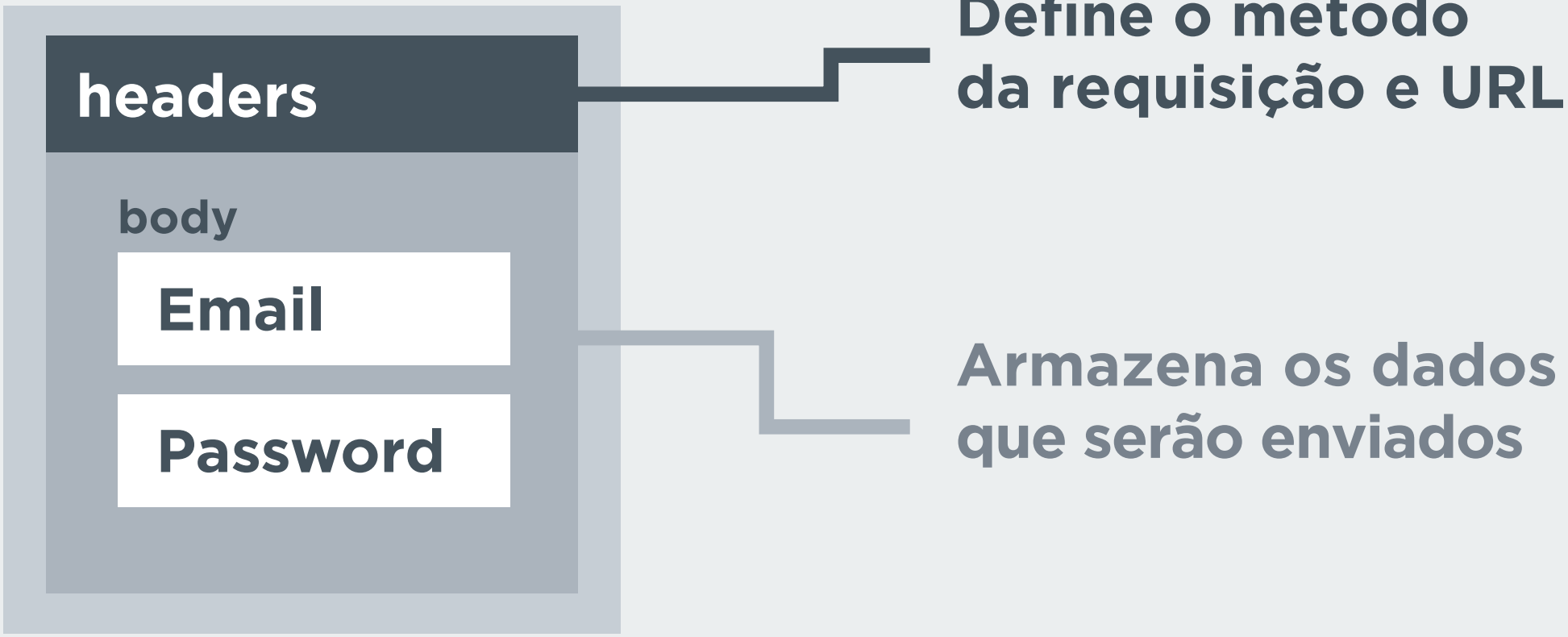
através de **requisições http!**
em alguma URL!


facebook
Email or Phone
Password
Log In
Forgot account?

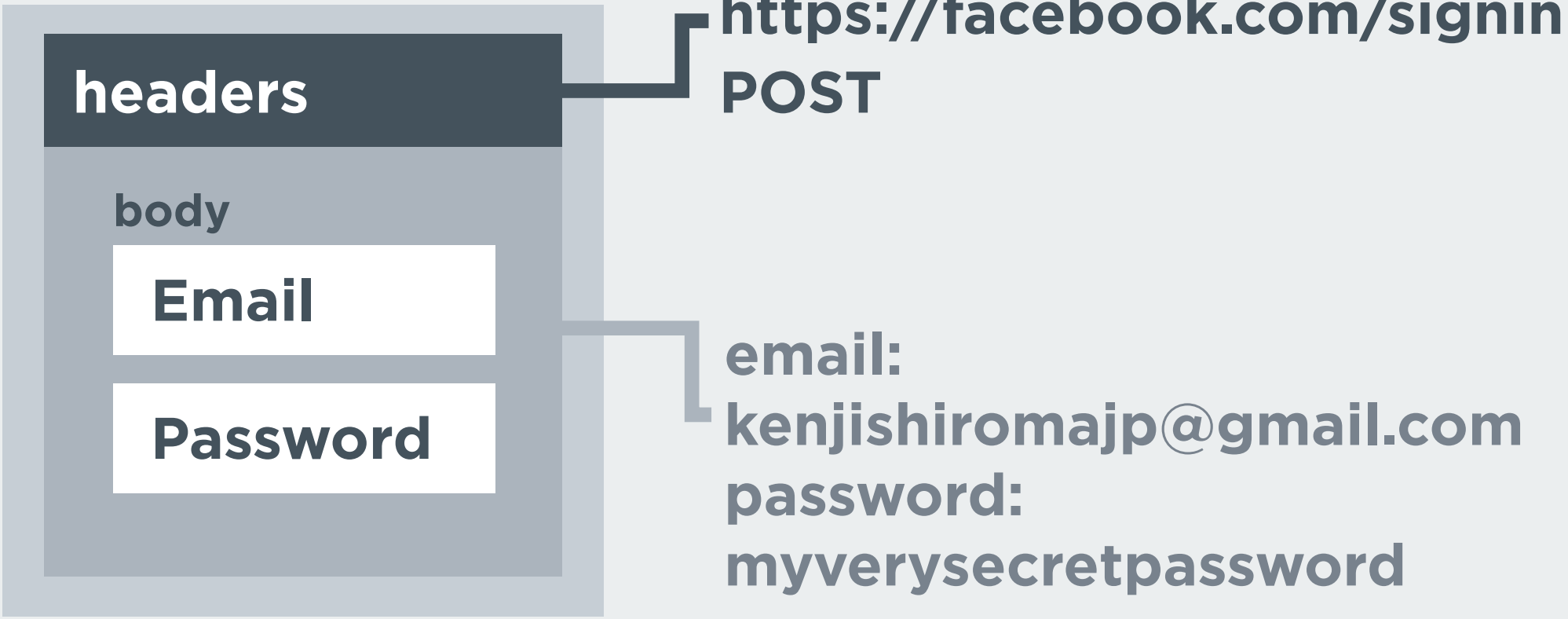
Estrutura de uma requisição HTTP



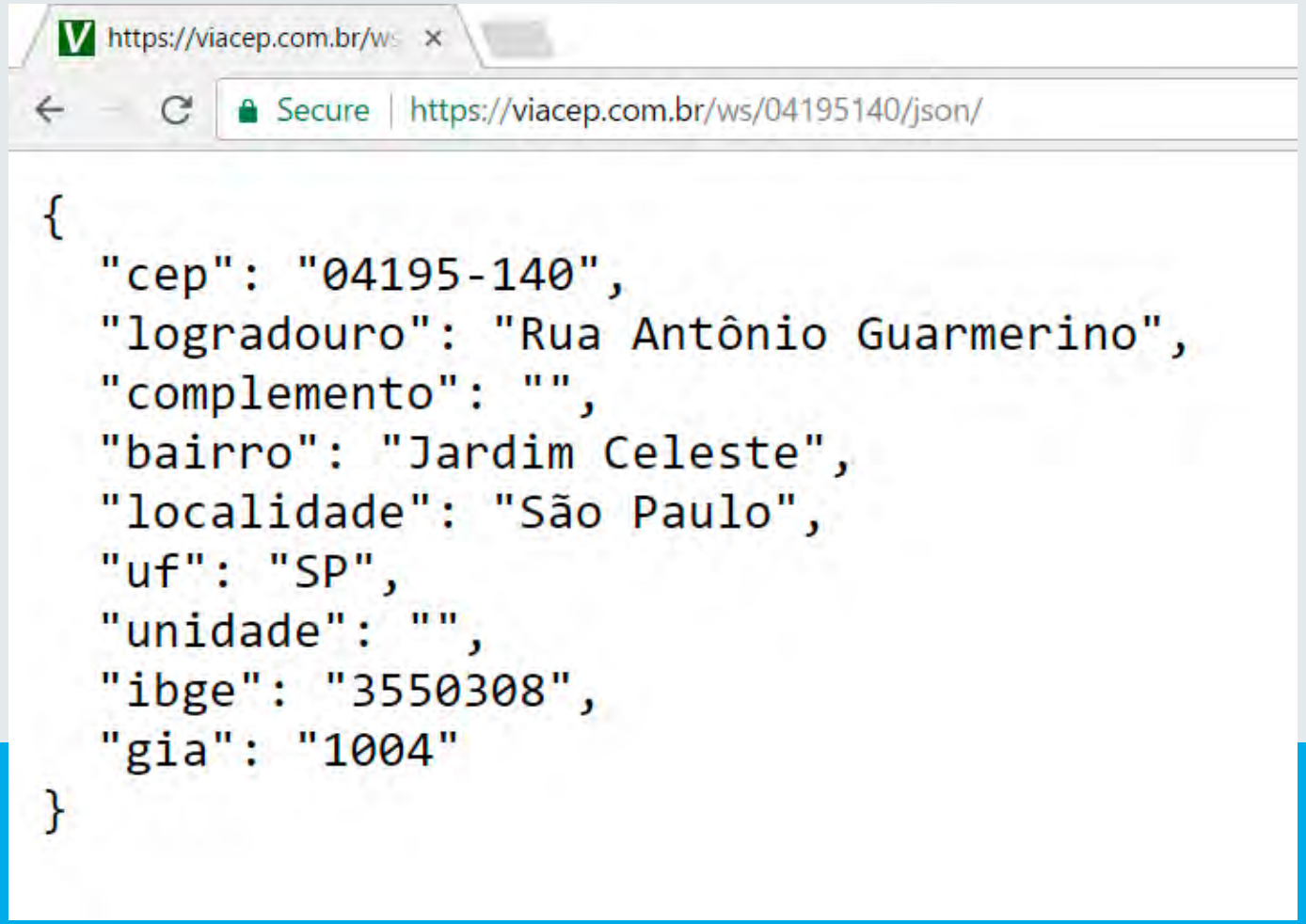
Estrutura de uma requisição



Request Structure



WebService


https://viacep.com.br/ws
Secure
https://viacep.com.br/ws/04195140/json/
{
"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
E COMO DECIDO ESSAS URLS?
?
POST - /DELETATODOLIST
GET - /CRIARTODOLISTS
PATCH - /PEGARTODOLISTS

REST VAI NOS DAR A RESPOSTA!

REST

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:

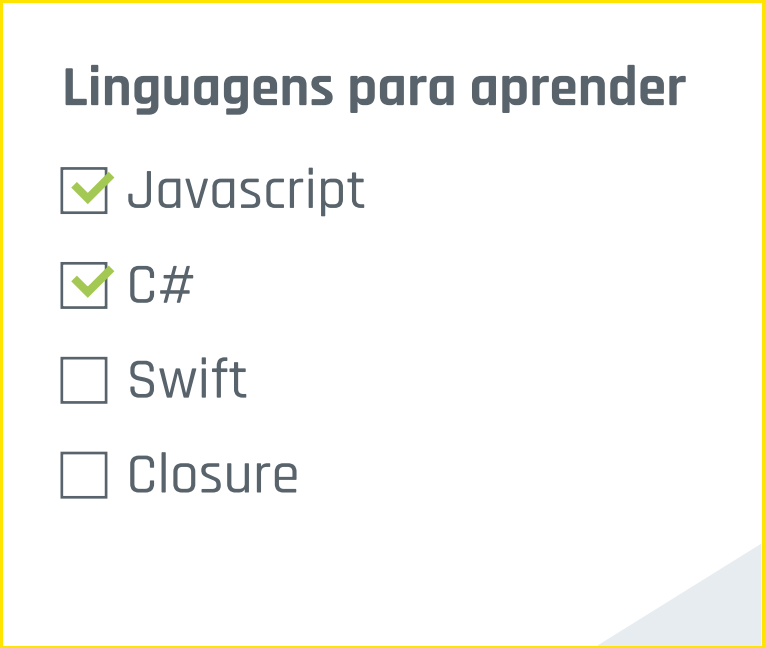
CRIAR TODOS ESTES ENDPOINTS!

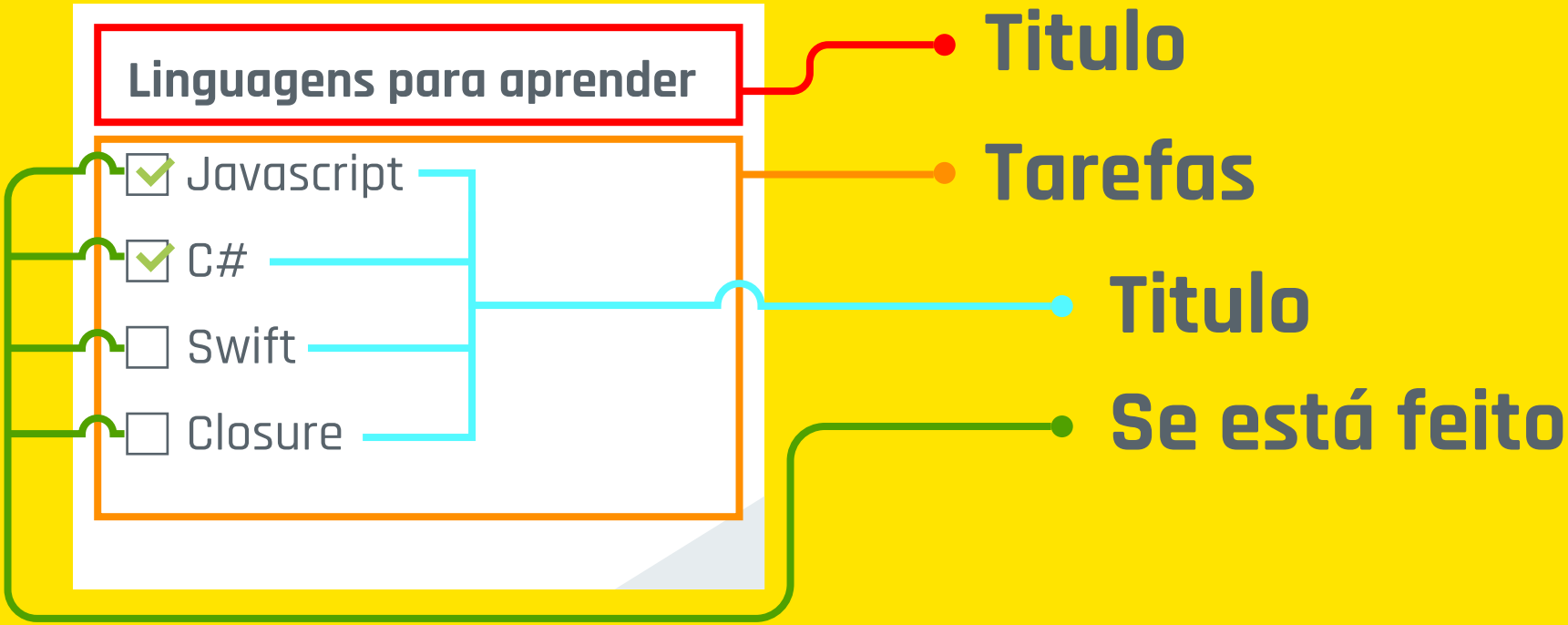
• Pegar todas as TodoLists	GET /todolists
• Criar uma TodoList	POST /todolists
• Criar uma Todo	POST /todos
• Marcar uma Todo como feito	PUT/PATCH /todos/{id}
• Atualizar uma TodoList	PUT/PATCH /todolists/{id}
• Deletar uma TodoList	DELETE /todolists/{id}
• Deletar uma Todo	DELETE /todos/{id}

Linguagens para aprender

- [x] Javascript
- [x] C#
- [ ] Swift
- [ ] Closure

Quais as propriedades você enxerga nesta todolist?




Linguagens para aprender
Javascript
C#
Swift
Closure
Titulo
Tarefas
Titulo
Se está feito

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!


REVISÃO

Hap.Server

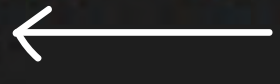
```
yarn add hapi
```

```
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,
    routes: {
        cors: {
            origin: ["*"]
        }
    }
});
```



Define a porta que vai subir o serviço http

Cross-Origin Resource Sharing
para aceitar de qualquer origem

Definindo Rota

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

url da rota
EX: "http://localhost/todolists"

Verbo/Método da Requisição

passa o corpo da response

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

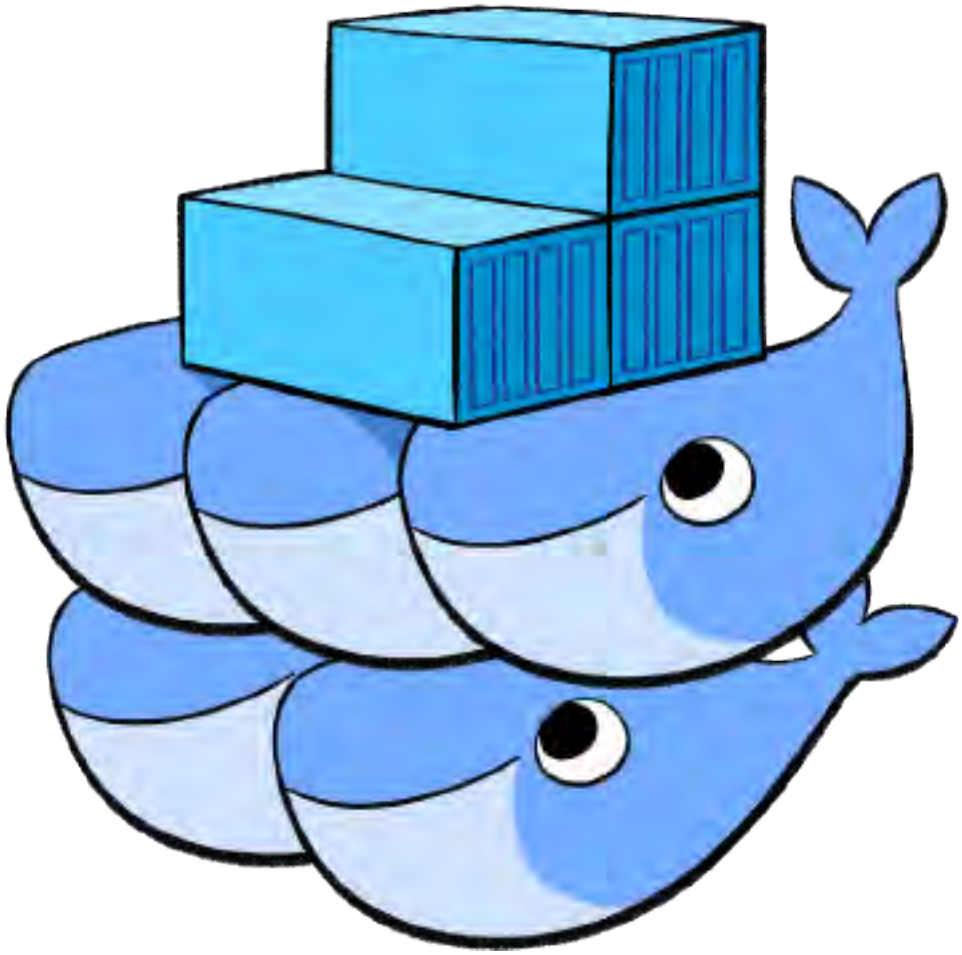
```
const _b = function(param){
    return "com retorno"
};
```

Versão com arrow function

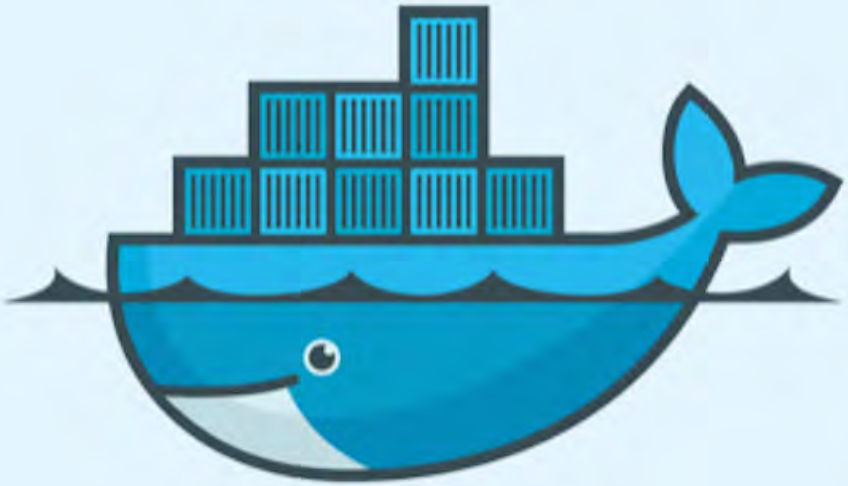
```
const b = (param) => "com retorno";
```

Agora vamos fazer o retorno dos dados com MongoDB

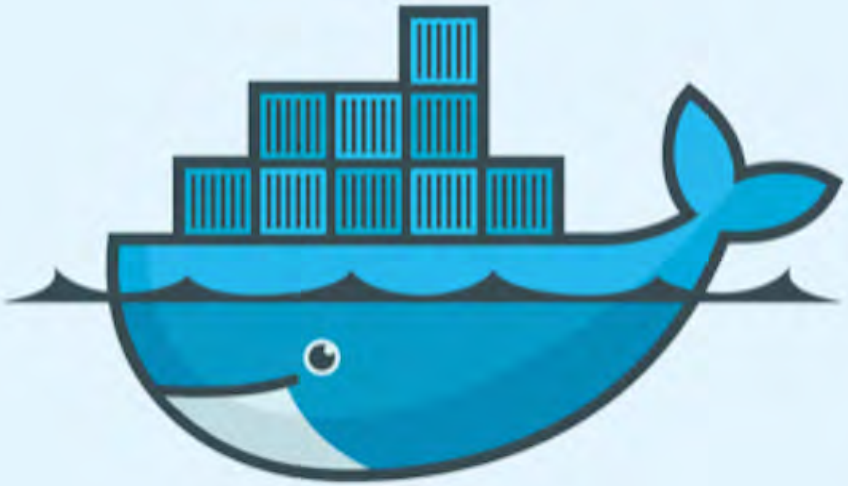
Para isso teria que instalar o mongo, baixar o client... (mais configuraçõ̃es)



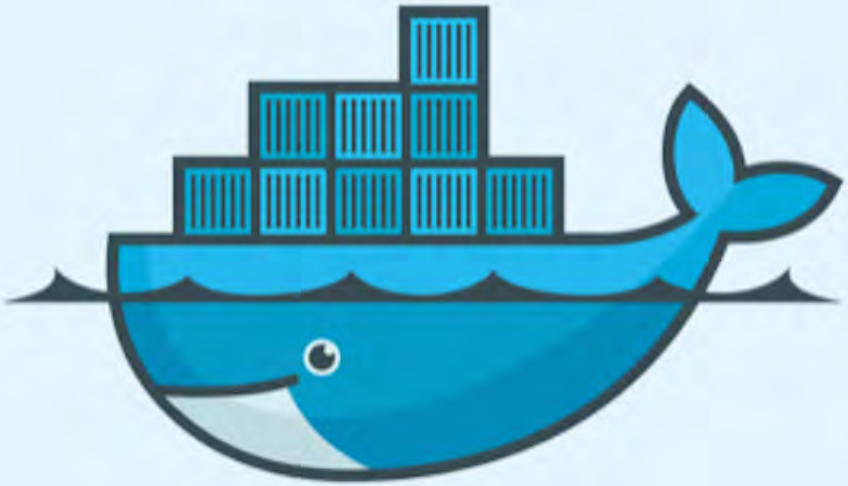
Por isso vamos usar o Docker compose!



O QUE É Docker?

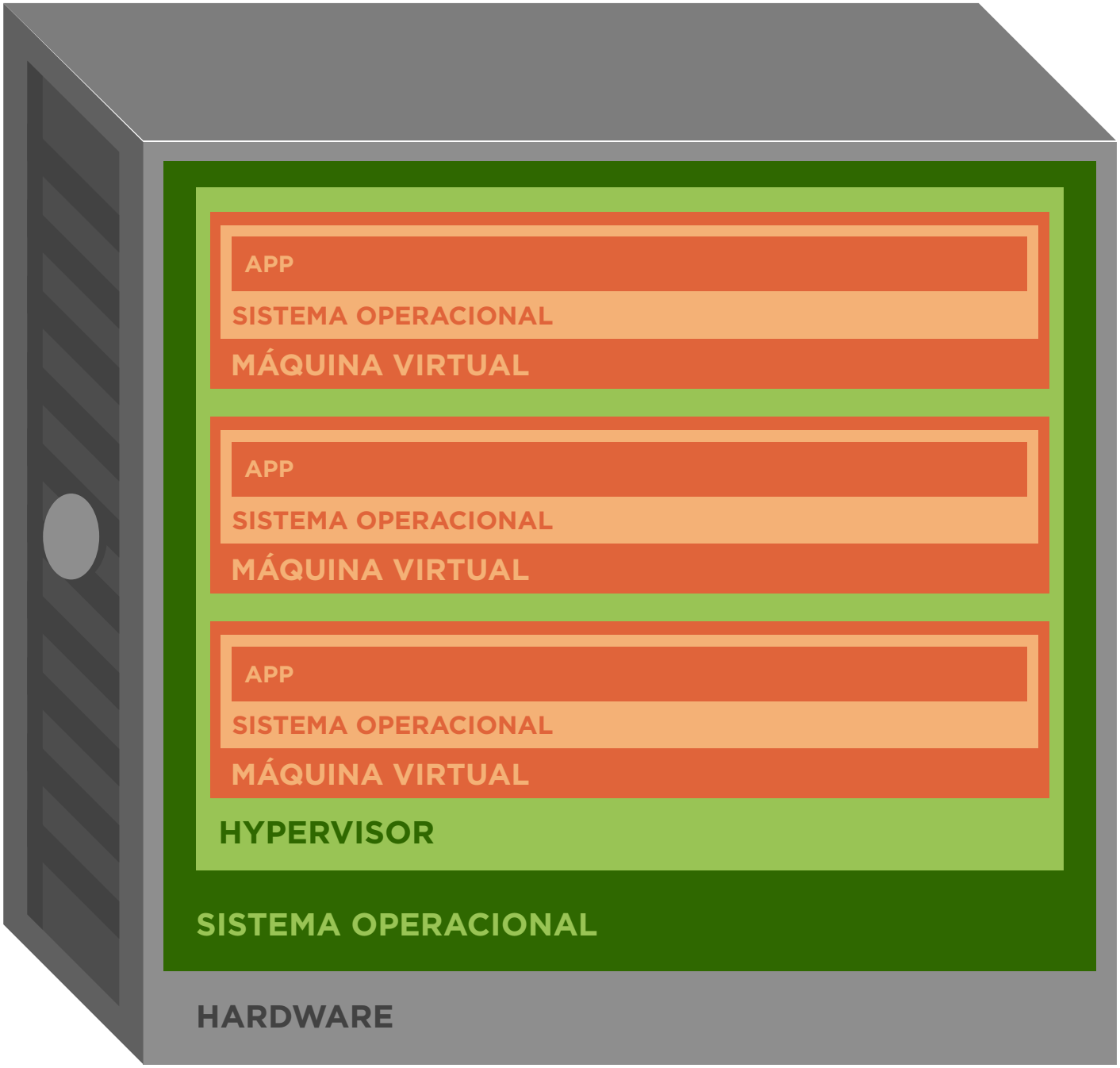


O QUE É Docker?

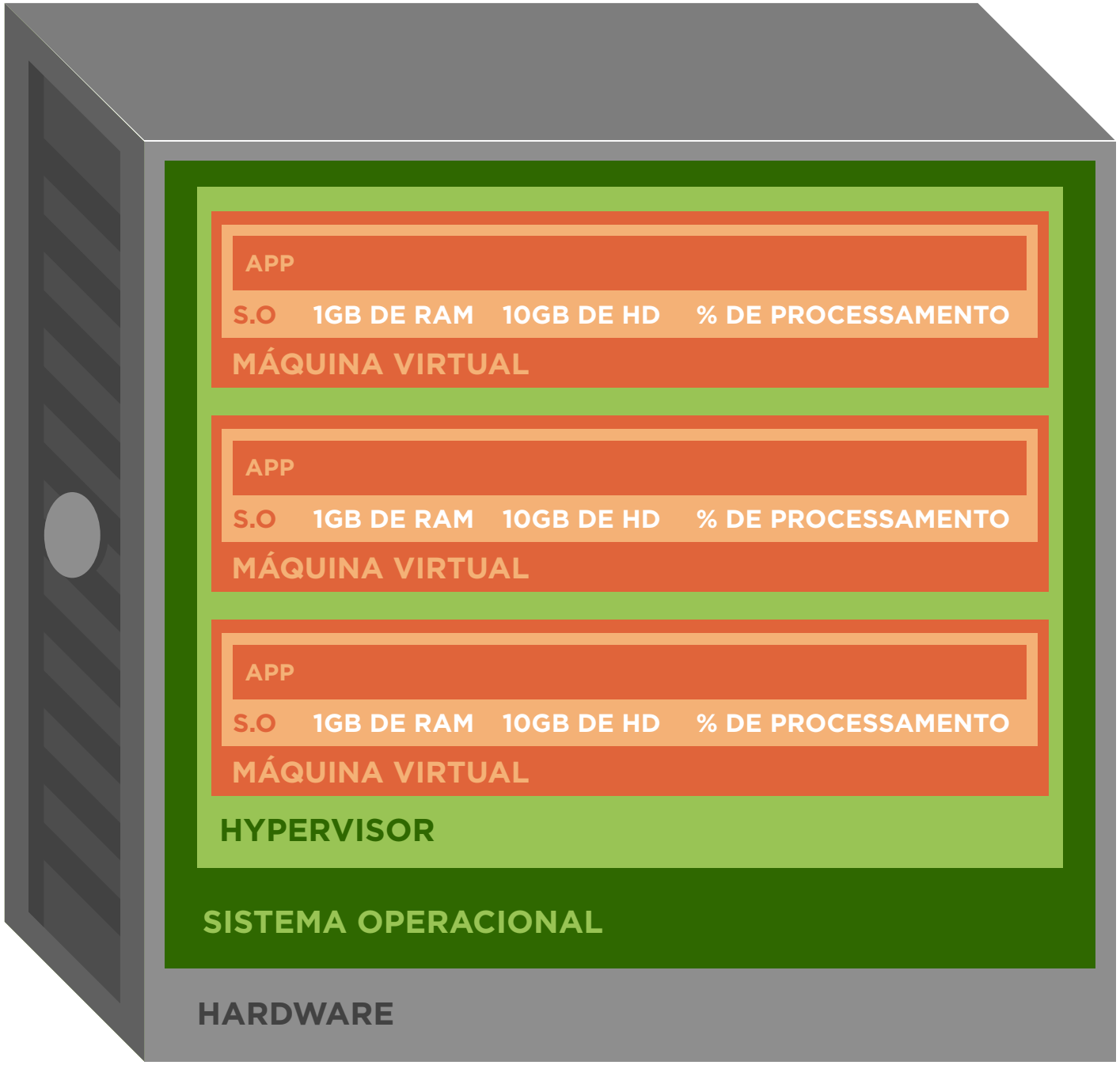


Docker é uma série de ferramentas que utiliza-se de **containers** para ajudar na criação de um **ambientes isolados**

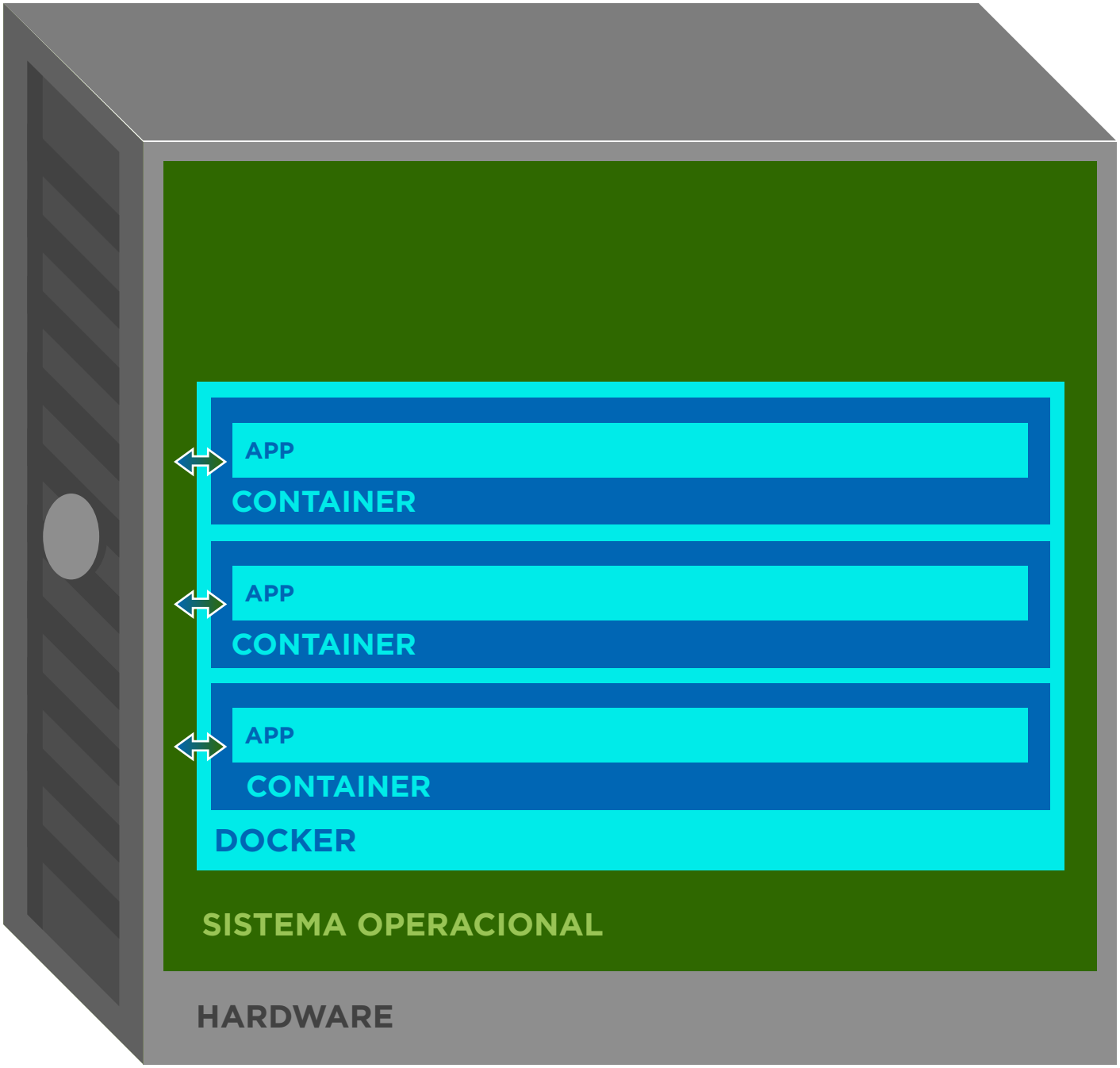
O que é Container? é tipo máquina virtual?



O que é Container? é tipo máquina virtual?



O que é Container? é tipo máquina virtual?



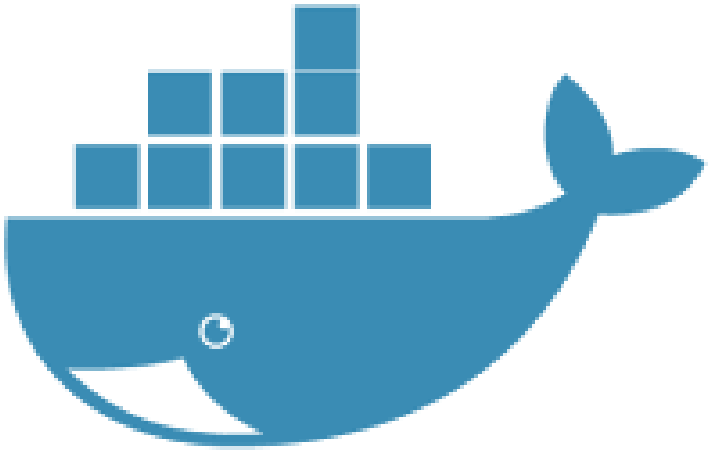
Vamos usar dois carinhas do Docker

Vamos usar dois carinhas do Docker

Docker Compose

Vamos usar dois carinhas do Docker

Docker Compose



Docker HUB

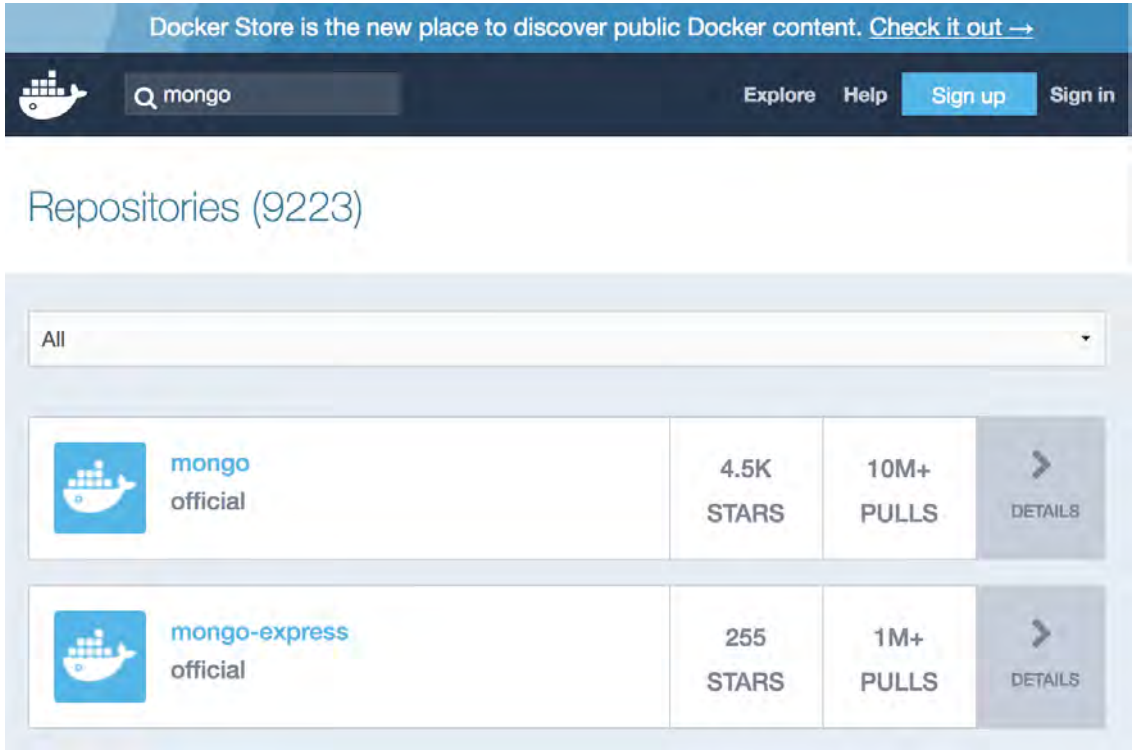
Vamos usar dois carinhas do Docker

Docker Compose

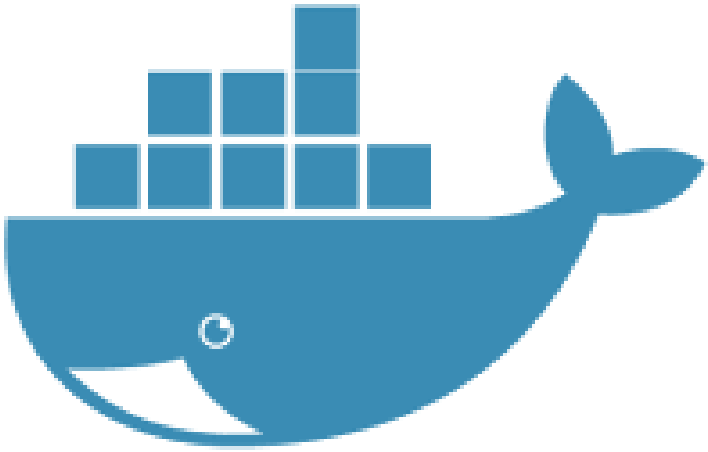
- Container com **MongoDb**
- Container com o **MongoDb Client**

versão interface para acessar o mongodb

Vamos usar dois carinhas do Docker



Repositório de imagens



Docker HUB

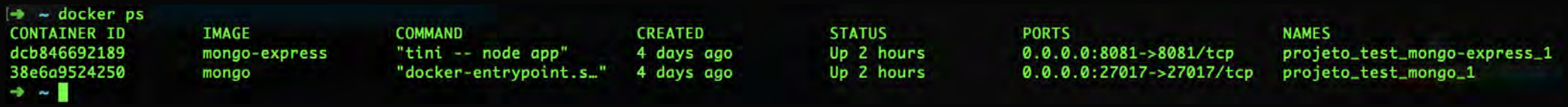
Vamos neste passo...

- **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!


REVISÃO

`docker ps` Lista todos os containers ligados



`docker ps -a` Lista todos os containers ligados e desligados

`docker stop <id>` Desliga o container com este id

```
docker container prune
```

Remove todos os container parados

```
docker-compose up
```

docker-compose.yml

```yaml
version: '3.1'

services:

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

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

imagem que achamos no docker hub

para reiniciar toda vez que monta

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

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

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",
    todoListSchema,
    "todoLists"
);
```

Nome do meu Modelo

Nome da minha collection

```
interface ITodo extends Mongoose.Document{
    mame: string;
    done: boolean;
}
interface ITodoList extends Mongoose.Document{
    mame: 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

await

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

```
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);
    }
});
```

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

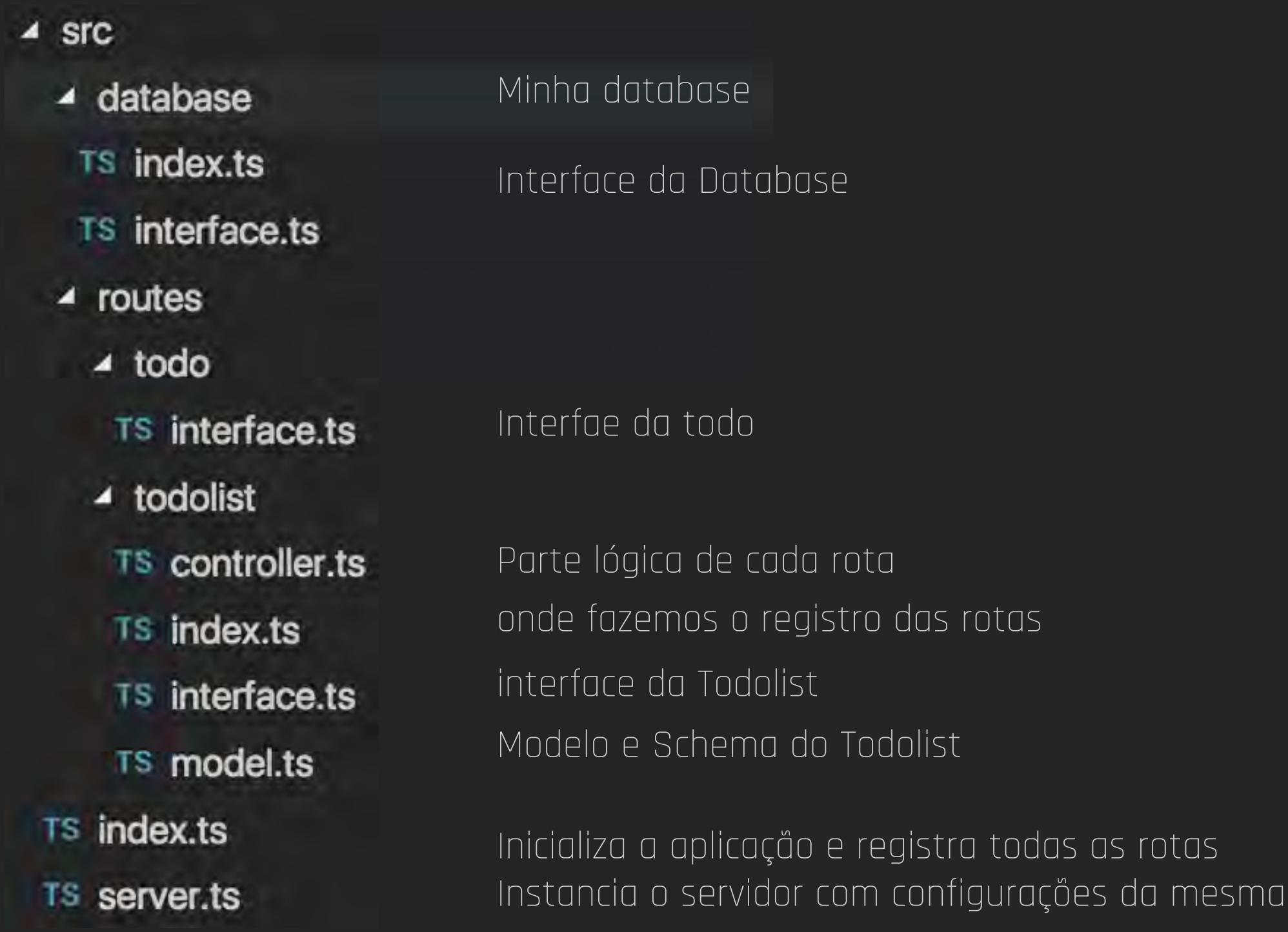
Want to improve your Postman skills?
Take the next step!

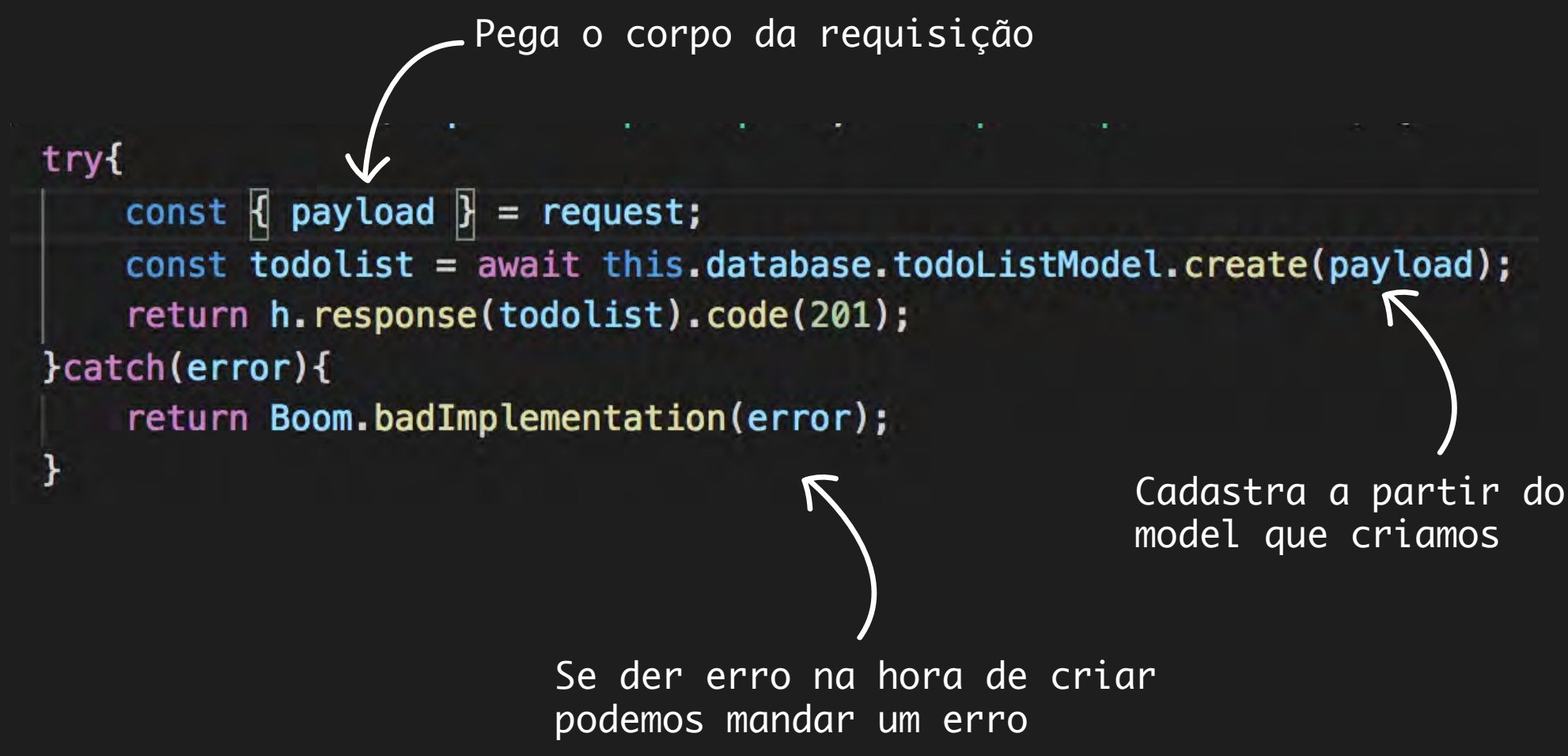
Vamos neste passo...

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

MÃO NA MASSA!


REVISÃO


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


Pega o corpo da requisição
try{
const { payload } = request;
const todolist = await this.database.todoListModel.create(payload);
return h.response(todolist).code(201);
}catch(error){
return Boom.badImplementation(error);
}
Cadastra a partir do model que criamos
Se der erro na hora de criar podemos mandar um erro