



**Prof. David Fernandes de Oliveira  
Instituto de Computação  
UFAM**

# O que é um Framework Web

- Um **framework Web** é um conjunto de bibliotecas que visam facilitar o desenvolvimento de sistemas Web completos
- Desenvolver uma aplicação Web do zero é muitas vezes inviável e bastante trabalhoso
  - O uso de bibliotecas e frameworks de código aberto torna tudo mais fácil, rápido, confiável, e provavelmente mais seguro

METER

express

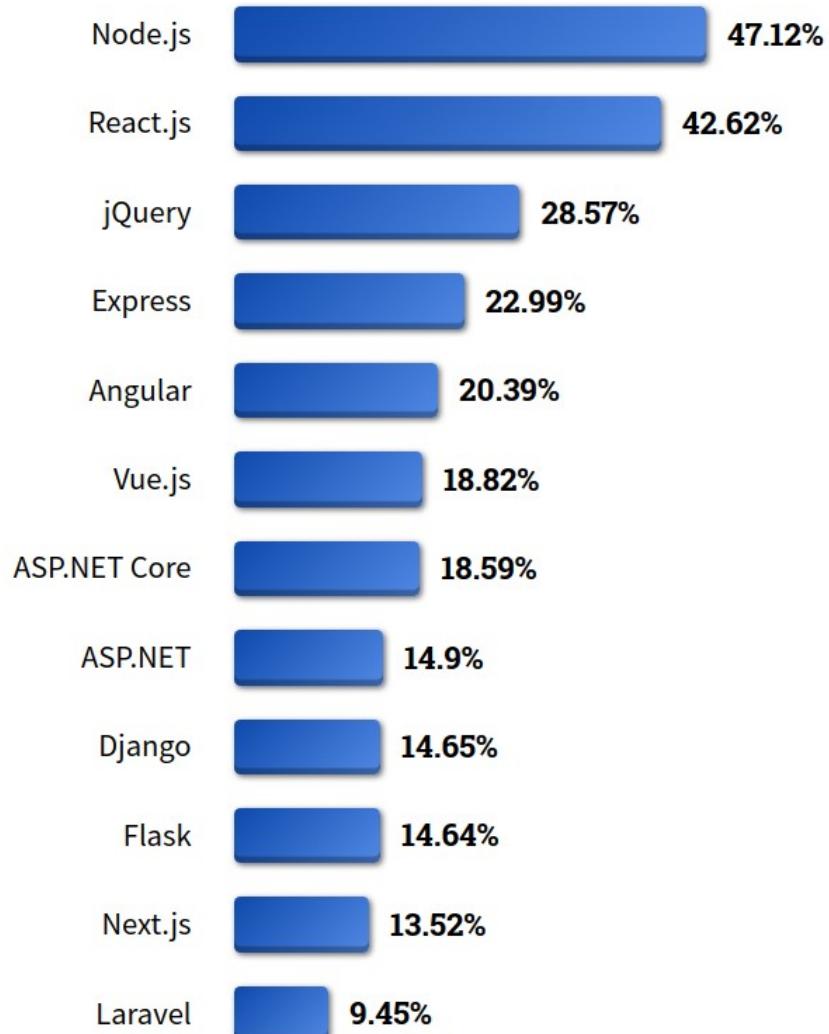
sails



django

laravel

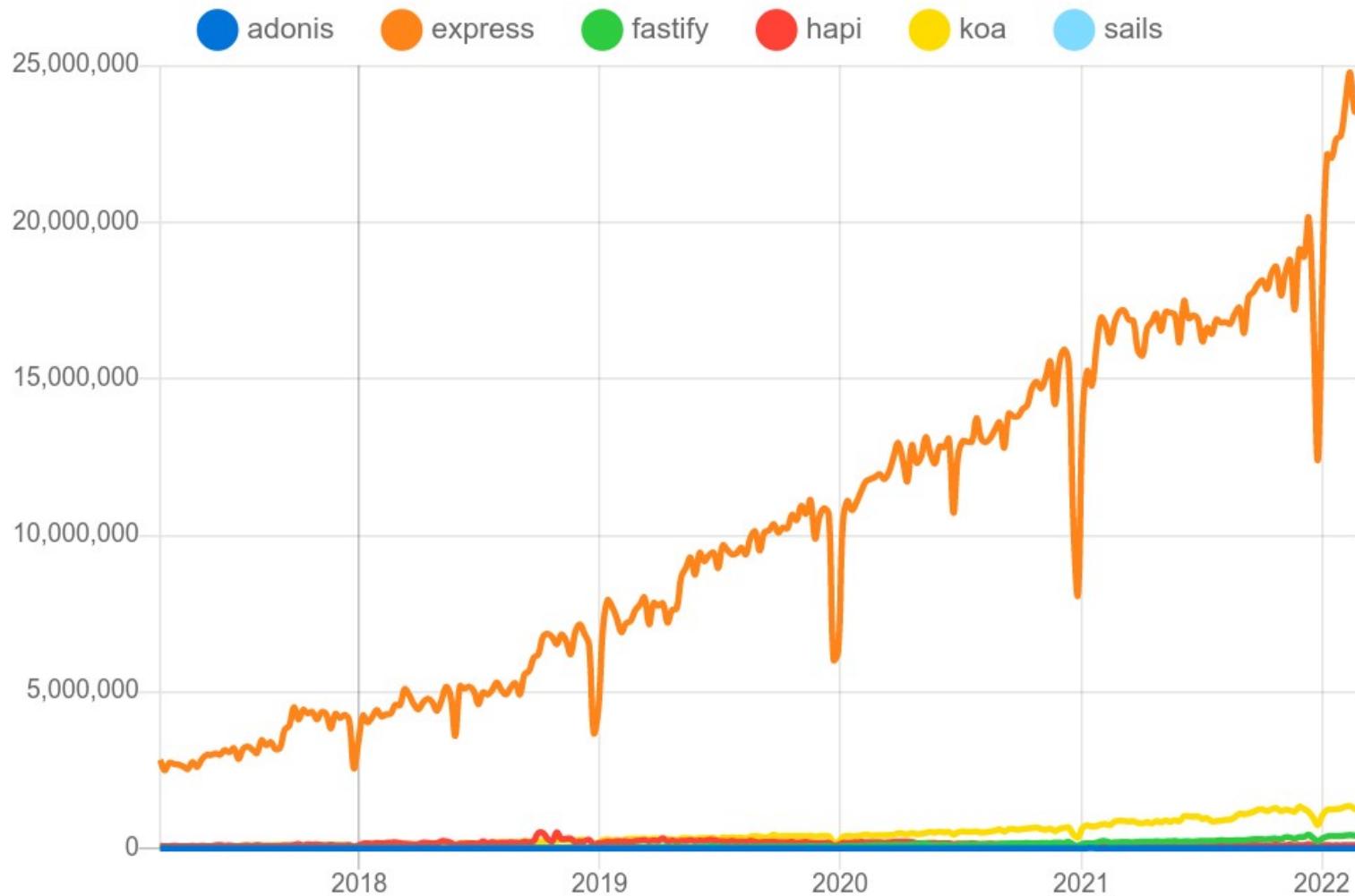
# StackOverflow Surveys



**Web Frameworks  
and technologies**

2022

# NPM Trends



express **JS**

# Um Framework não opinativo

- O Express é um framework Web baseado no Node.js que facilita o desenvolvimento de Apps e adiciona novos recursos
- O Express segue uma filosofia **não opinativa** e **minimalista**
  - **Não opinativa** significa que você precisará tomar muitas decisões sobre como organizar seu código dentro de sua aplicação
  - **Minimalista** significa que ele te dá total liberdade de escolher outros módulos para completar as necessidades de sua aplicação
- Essas características nos permitem usar o Express para desenvolver qualquer tipo de aplicação, de um hub de vídeos à um chat

ex Express - Node.js web app | +

Not secure | expressjs.com

Black Lives Matter.  
Support the Equal Justice Initiative.

Express  search Home Getting started Guide API reference Advanced topics Resources

# Express 4.18.1

Fast, unopinionated, minimalist web framework for **Node.js**

```
$ npm install express --save
```

**Express 5.0 beta documentation is now available.**  
The beta [API documentation](#) is a work in progress. For information on what's in the release, see the Express [release history](#).

Web Applications	APIs	Performance	Frameworks
Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.	With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.	Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.	Many <a href="#">popular frameworks</a> are based on Express.

express **JS**

# O Arquivo package.json

- O primeiro passo no desenvolvimento de uma aplicação node.js é a criação de um arquivo chamado **package.json**
  - A função desse arquivo é armazenar vários metadados sobre a aplicação, incluindo suas dependências

```
{  
  "name": "hello-world",  
  "author": "David Fernandes",  
  "private": true,  
  "version": "0.0.1",  
  "dependencies": {}  
}
```

# O Arquivo package.json

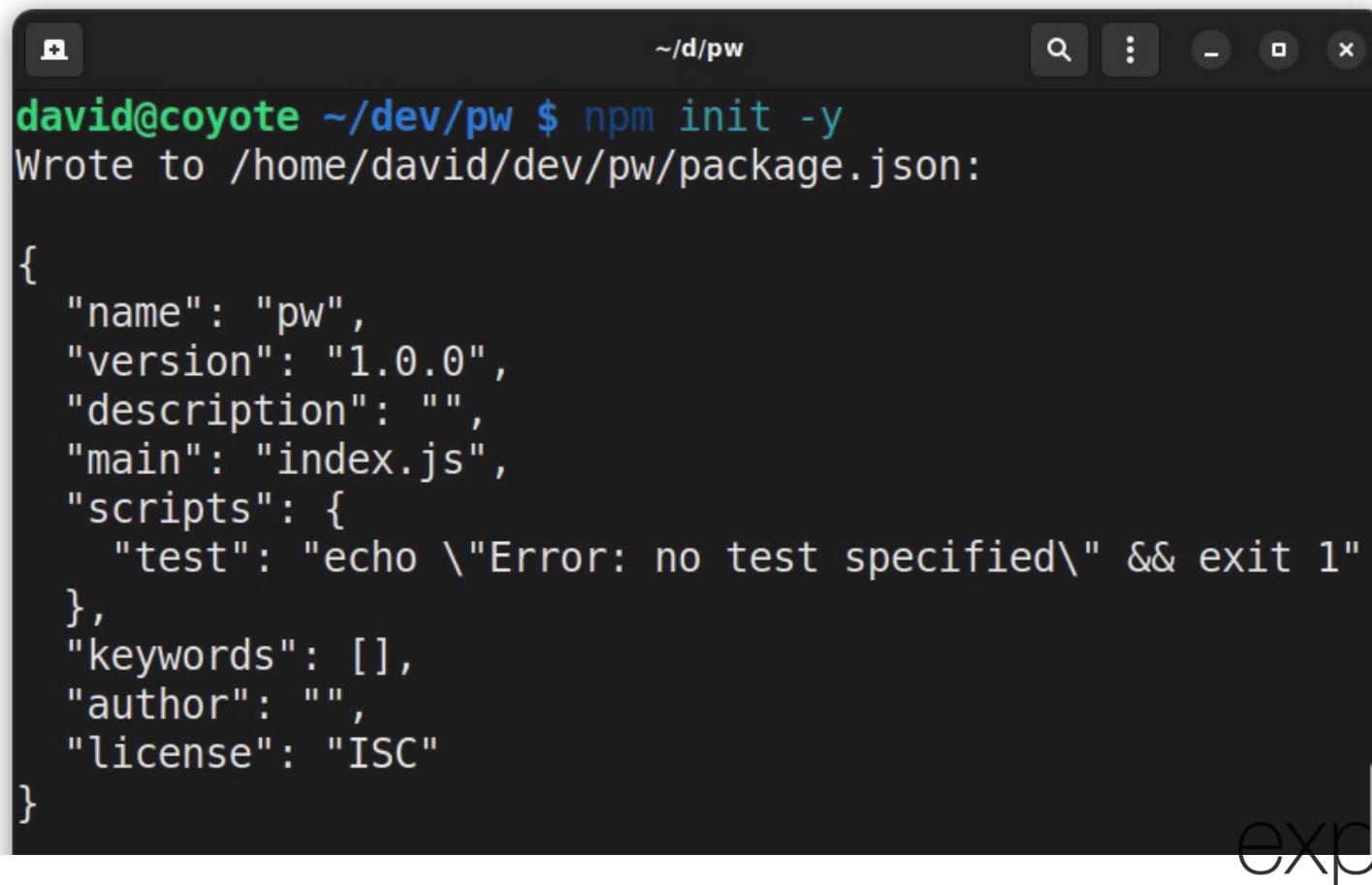
- O primeiro passo no desenvolvimento de uma aplicação node.js é a criação de um arquivo chamado **package.json**
  - A função desse arquivo é armazenar vários metadados sobre a aplicação, incluindo suas dependências

```
{  
  "name": "hello-world",  
  "author": "David Fernandes",  
  "private": true,  
  "version": "0.0.1",  
}
```

Para criar um novo arquivo **package.json** com os valores desejados, use o comando **npm init**. Esse comando fará algumas perguntas para o programador, e criará um arquivo **package.json** baseado nas suas respostas.

# O Arquivo package.json

- Outra opção é executar o comando **npm init** com a opção **-y**, que irá criar um package.json com valores iniciais

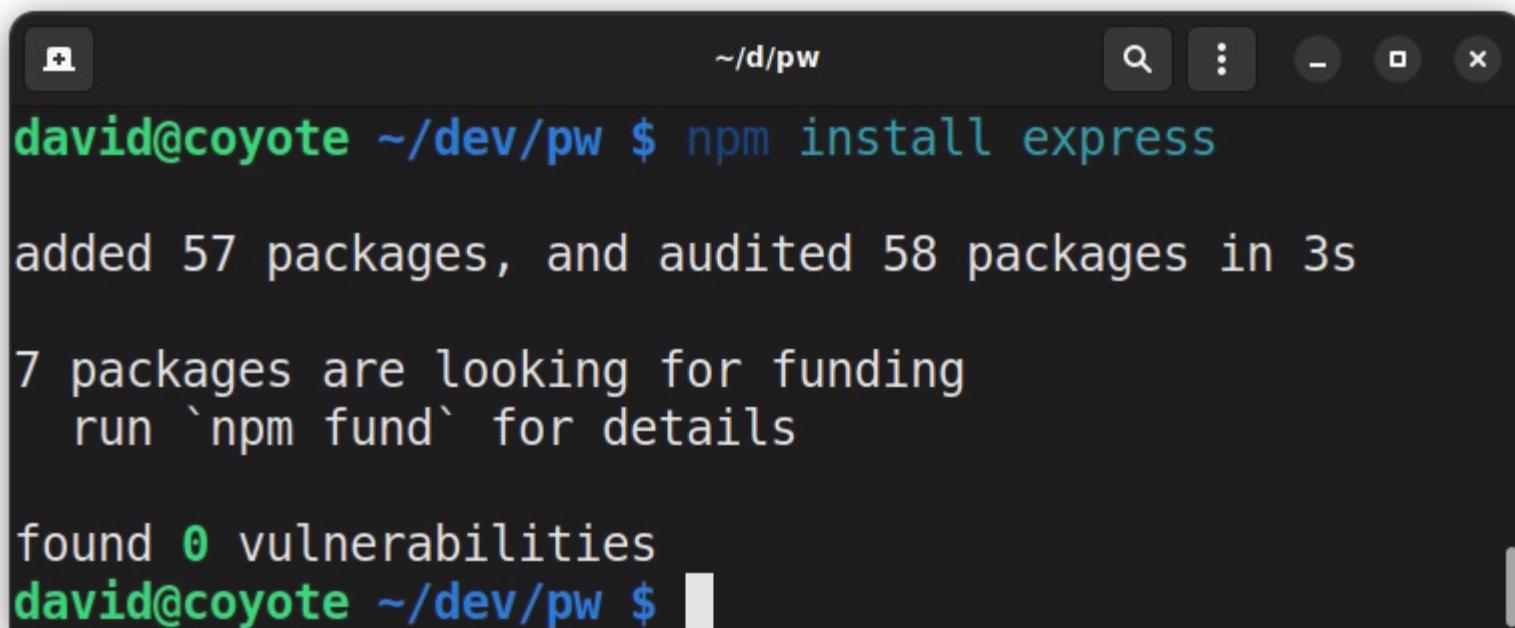


```
~d/pw
david@coyote ~/dev/pw $ npm init -y
Wrote to /home/david/dev/pw/package.json:

{
  "name": "pw",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \\\"Error: no test specified\\\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

# Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta executarmos o comando **npm install express**
  - O pacote express é adicionado automaticamente como uma dependência do projeto



```
~/d/pw
david@coyote ~/dev/pw $ npm install express
added 57 packages, and audited 58 packages in 3s
7 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
david@coyote ~/dev/pw $
```

# Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta executar:

```
david@coyote ~/dev/pw $ cat package.json
{
  "name": "pw",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2"
  }
}
david@coyote ~/dev/pw $
```

# Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta executar:

- O pacote express é adicionado automaticamente como uma dependência do projeto

```
{  
  "name": "pw",  
  "version": "1.0.0",  
  "description": "",  
  "main": "index.js",  
  "scripts": {
```

Observe que a definição de dependência do Express se refere à versão 4.18.2. A versão de um dado módulo é representada por três valores:

**Major version** (4), **Minor version** (18) e **Patch version** (2).

```
  "license": "ISC",  
  "dependencies": {  
    "express": "^4.18.2"  
  }  
david@coyote ~/dev/pw $
```

# Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta executar



```
~ /d/pw
```

Note também que a versão do Express é prefixada por um ^, indicando que ele pode ser atualizado caso seja lançado uma nova **minor version** do framework. Isto é, sua aplicacão é dependente do Express versão 4.\*.\*

Alguns módulos podem ser prefixados com um ~, indicando eles devem ser atualizados caso seja lançado um novo patch desses módulos.

Observe que a definição de dependência do Express se refere à versão 4.18.2. A versão de um dado módulo é representada por três valores: **Major version** (4), **Minor version** (18) e **Patch version** (2).

```
run "license": "ISC", details
  "dependencies": {
found 0 "express": "4.18.2"
david@coyote ~/dev/pw $ 
}
david@coyote ~/dev/pw $
```

# Hello World em Express

- Para fazer um primeiro teste com o Express, podemos criar um diretório **src** e arquivo **src/index.js** com o seguinte conteúdo:

```
const express = require("express");
require("dotenv").config();

const app = express();
const PORT = process.env.PORT || 3000

app.get("/", (req, res) => {
  res.send("Hello world!");
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}.`);
});
```

Envia a string  
Hello World  
para o browser

Inicializa o  
servidor HTTP  
na porta 3000

# Hello World em Express

- Para fazer um primeiro teste com o Express, podemos criar um diretório:

```
node src/index.js ~d/express
david@coyote ~/dev/express $ node src/index.js
Express app iniciada na porta 3333.
```

```
const app = require('express');
const PORT = process.env.PORT || 3333;

app.get('/', (req, res) => {
  res.send('Hello world!');
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}`);
});
```

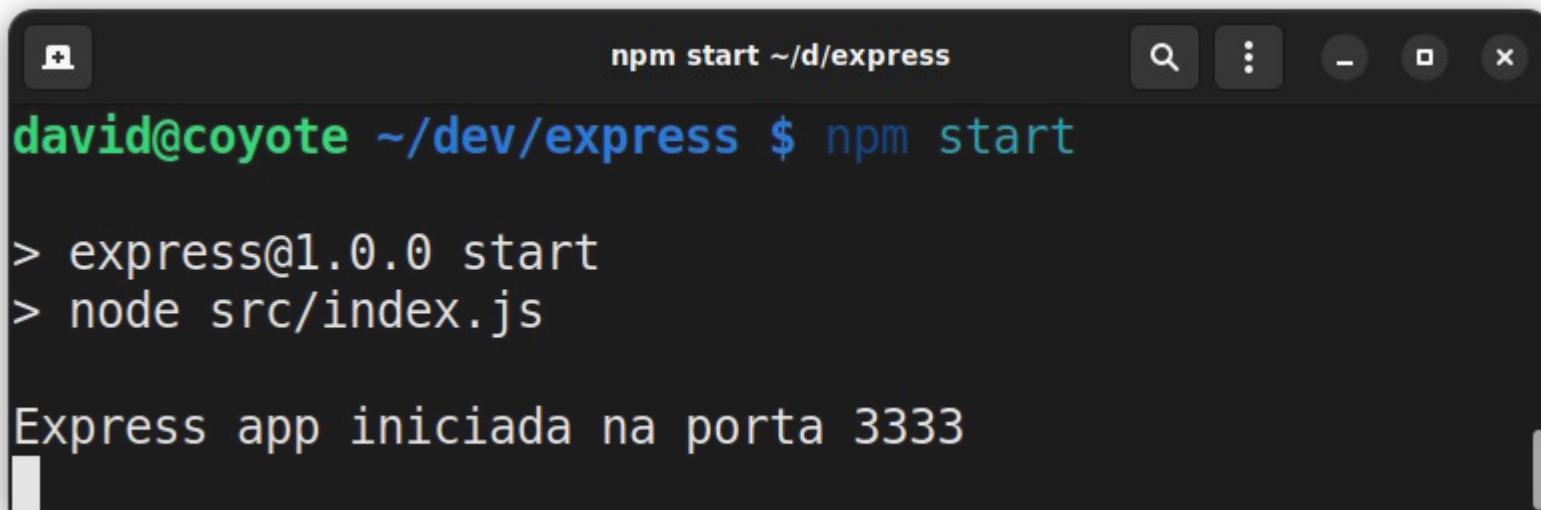
Hello world!

Inicializa o servidor HTTP na porta 3000

# Hello World em Express

- No arquivo **package.json**, podemos incluir um **script start** para inicializar a aplicação em um ambiente de desenvolvimento

```
"scripts": {  
  "start": "node src/index.js"  
},
```

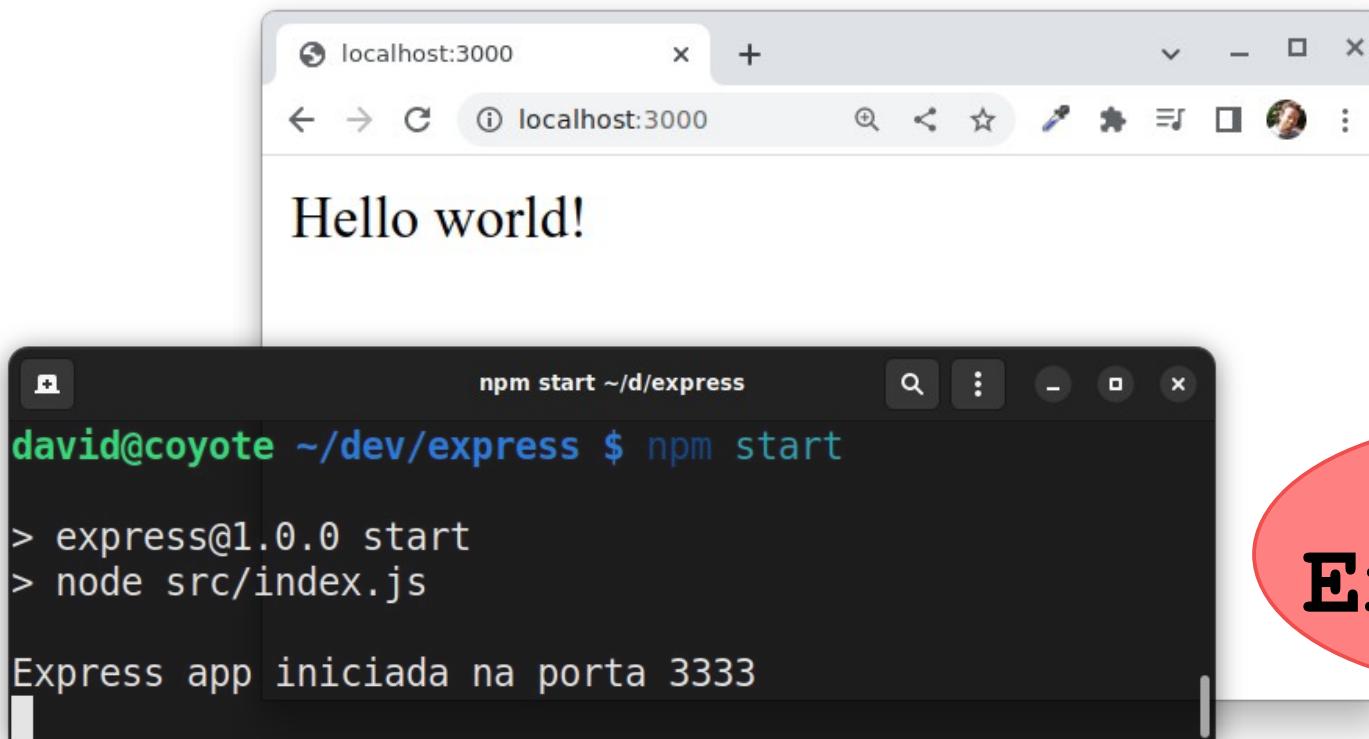


A screenshot of a terminal window titled "npm start ~/d/express". The terminal shows the command "david@coyote ~/dev/express \$ npm start" being run. The output shows the package.json script being executed, which runs the node command on the index.js file. The final message "Express app iniciada na porta 3333" is displayed at the bottom.

```
david@coyote ~/dev/express $ npm start  
  
> express@1.0.0 start  
> node src/index.js  
  
Express app iniciada na porta 3333
```

# Exercício 1

- Implemente o app Hello World apresentado nos slides anteriores. Além do express, adicione o **dotenv**, o **nodemon** e crie os scripts **start** e **start:prod** em sua aplicação



A screenshot showing a browser window and a terminal window. The browser window displays the URL 'localhost:3000' with the text 'Hello world!'. The terminal window shows the command 'npm start ~d/express' and the output: 'david@coyote ~/dev/express \$ npm start', followed by two lines starting with '> express@1.0.0 start' and '> node src/index.js', and finally 'Express app iniciada na porta 3333'.

github  
**Express**

express **JS**

# Instalação do TypeScript

- Após a instalação do Express, vamos instalar o **TypeScript** como dependência de desenvolvimento da aplicação
- Também serão instalados os pacotes de declaração **@types** para **Express** e **Node.js**, que fornecem definições de tipos
- Além disso, o pacote **ts-node** será útil para executar código Typescript

```
$ npm i -D typescript @types/express @types/node ts-node
```

- A opção **-D**, ou **--save-dev**, faz com que o npm instale esses pacotes como dependências de desenvolvimento

# Instalação do TypeScript

- Após a instalação do Express, vamos instalar o **TypeScript** como dependência de desenvolvimento da aplicação

```
~/d/express
david@coyote ~/dev/express $ ls -la ./node_modules/.bin/
total 8
drwxrwxr-x 2 david david 4096 mai 24 06:59 .
drwxrwxr-x 63 david david 4096 mai 24 06:59 ../
lrwxrwxrwx 1 david david 14 mai 24 06:51 mime -> ../../typescript/bin/tsc*
lrwxrwxrwx 1 david david 21 mai 24 06:59 tsc -> ../../typescript/bin/tsc*
lrwxrwxrwx 1 david david 26 mai 24 06:59 tsserver -> ../../typescript/bin/tsserver*
david@coyote ~/dev/express $
```

- A opção **-D**, ou **--save-dev**, faz com que o npm instale esses pacotes como dependências de desenvolvimento

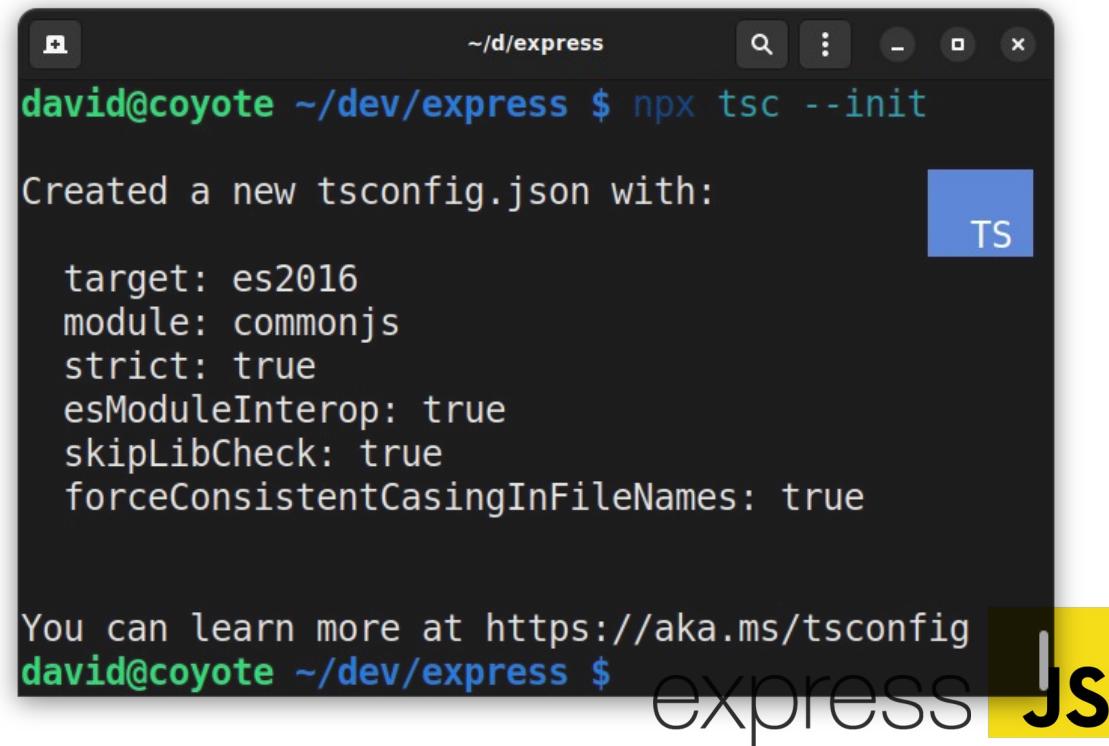
Compilador  
TypeScript

# Instalação do TypeScript

- Agora podemos iniciar as configurações do TypeScript, através do seguinte comando:

```
$ npx tsc --init
```

- O comando acima irá criar um arquivo **tsconfig.json**, que contém opções padrão do compilador TypeScript e nos permite ajustar ou personalizar essas opções



A screenshot of a terminal window titled 'david@coyote ~/dev/express \$'. The window shows the command 'npx tsc --init' being run, followed by the message 'Created a new tsconfig.json with:' and the contents of the generated tsconfig.json file. The file includes settings like target: es2016, module: commonjs, strict: true, esModuleInterop: true, skipLibCheck: true, and forceConsistentCasingInFileNames: true. A blue 'TS' icon is visible in the top right corner of the terminal window.

```
david@coyote ~/dev/express $ npx tsc --init
Created a new tsconfig.json with:
  target: es2016
  module: commonjs
  strict: true
  esModuleInterop: true
  skipLibCheck: true
  forceConsistentCasingInFileNames: true

You can learn more at https://aka.ms/tsconfig
david@coyote ~/dev/express $
```

express JS

# Instalação do TypeScript

- No arquivo **tsconfig.json** gerado, é preciso definir alguns paths manualmente

```
{  
  "compilerOptions": {  
    "rootDir": "src",  
    "outDir": "build",  
    ...  
  },  
  "include": ["src/**/*"],  
  "exclude": ["node_modules"]  
}
```

raiz dos  
fontes da  
aplicação

diretório do  
código  
compilado

diretórios  
incluídos na  
compilação

diretórios  
excluídos na  
compilação

# Hello World em Express/TypeScript

- Para fazer um primeiro teste com o Express/TypeScript, podemos renomear o arquivo **src/index.js** para **src/index.ts** e colocar o seguinte conteúdo:

```
import express, { Request, Response } from "express";
import dotenv from "dotenv";

dotenv.config();
const app = express();
const PORT = process.env.PORT || 3333;

app.get("/", (req: Request, res: Response) => {
  res.send("Hello world!");
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}.`);
});
```

# Hello World em Express/TypeScript

- Para fazer isso, podemos colocar

Hello world!

```
i  node build/index.js ~/d/express
i
david@coyote ~/dev/express $ npx tsc
david@coyote ~/dev/express $ node build/index.js
Express app iniciada na porta 3333.
const PORT = process.env.PORT || 3333,
```

```
app.get("/", (req: Request, res: Response) => {
  res.send("Hello world!");
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}.`);
});
```

Para rodar o código, primeiro temos que compilá-lo usando **npx tsc**, e depois podemos rodar o código gerado

# Hello World em Express/TypeScript

- Podemos inicializar a aplicação em ambiente de desenvolvimento usando o **nodemon**, de forma que os scripts em **package.json** ficam assim:

```
"scripts": {  
  "start": "nodemon src/index.ts",  
  "build": "npx tsc",  
  "start:prod": "node build/index.js"  
},
```

O nodemon  
executa código  
**TypeScript**  
através do pacote  
ts-node

# Exercício 2

- Faça uma cópia da pasta Express (do Exercício 1) e renomeie a nova pasta de **ExpTS**. Adapte o código dessa pasta para a linguagem TypeScript.

A screenshot of a computer interface showing a browser window and a terminal window. The browser window displays the text "Hello world!" at the address "localhost:3000". The terminal window shows the command "node build/index.js ~/d/express" being run, followed by the output of the TypeScript compiler ("npx tsc") and the execution of the application ("node build/index"). A red oval highlights the text "github ExpTS" in the bottom right corner of the terminal window. Below the terminal window, the words "express JS" are partially visible.

```
node build/index.js ~/d/express
david@coyote ~/dev/express $ npx tsc
david@coyote ~/dev/express $ node build/index
Express app iniciada na porta 3333.
```

github  
**ExpTS**

express JS

# Validando as variáveis de ambiente

- Esquecer de adicionar uma variável de ambiente ou não usar o tipo de dado certo nessas variáveis pode levar a erros inesperados que farão com que seu aplicativo funcione mal
- Para evitar isso, vamos usar o pacote **envalid** para validar as variáveis de ambiente

```
$ npm i envalid
```

# Validando as variáveis de ambiente

- Para usar o **envalid**, vamos criar um diretório utils no diretório src e adicionar um arquivo chamado validateEnv.ts



```
fish /home/david/dev/express/src
david@coiote ~/dev/express/src $ tree
.
└── index.ts
└── utils
    └── validateEnv.ts

1 directory, 2 files
david@coiote ~/dev/express/src $ 
```

A screenshot of a terminal window titled "fish /home/david/dev/express/src". The command "tree" is run, showing the directory structure: a root folder containing "index.ts" and a "utils" folder which contains "validateEnv.ts". Below the tree output, it says "1 directory, 2 files". The prompt "david@coiote ~/dev/express/src \$" is visible at the bottom.

# Validando as variáveis de ambiente

- No arquivo **validateEnv.ts**, definimos todas as variáveis de ambiente necessárias para rodar a aplicação
- O envalid lançará um erro caso esqueçamos de fornecer alguma das variáveis definidas ou se forem do tipo errado

```
import { cleanEnv, port, str } from 'envalid';

const validateEnv = () => {
  cleanEnv(process.env, {
    NODE_ENV: str(),
    PORT: port(),
  });
};

export default validateEnv;
```

A screenshot of a web browser window titled "envalid - npm". The URL in the address bar is "npmjs.com/package/envalid". The page content is visible, showing the title "Validator types" and some text about environment variable validation functions.

## Validator types

Node's `process.env` only stores strings, but sometimes you want to retrieve other types (booleans, numbers), or validate that an env var is in a specific format (JSON, url, email address).

To these ends, the following validation functions are available:

- `str()` - Passes string values through, will ensure an value is present unless a `default` value is given.  
Note that an empty string is considered a valid value - if this is undesirable you can easily create your own validator (see below)
- `bool()` - Parses env var strings `"1"`, `"0"`, `"true"`, `"false"`, `"t"`, `"f"` into booleans
- `num()` - Parses an env var (eg. `"42"`, `"0.23"`, `"1e5"`) into a Number
- `email()` - Ensures an env var is an email address
- `host()` - Ensures an env var is either a domain name or an ip address (v4 or v6)
- `port()` - Ensures an env var is a TCP port (1-65535)
- `url()` - Ensures an env var is a url with a protocol and hostname
- `json()` - Parses an env var with `JSON.parse`

Each validation function accepts an (optional) object with the following att

- `choices` - An Array that lists the admissible parsed values for the env var.

A página do envalid no npmjs tem uma documentação completa dos validadores disponíveis

# Validando as variáveis de ambiente

- Agora que definimos o `validEnv`, podemos usá-lo em nosso arquivo `index.ts`

```
import express, { Request, Response } from 'express';
import validateEnv from './utils/validateEnv';
import dotenv from 'dotenv';

dotenv.config();
validateEnv();

const app = express()
const PORT = process.env.PORT || 3333

app.get("/", (req: Request, res: Response) => {
  res.send("Hello world!");
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}.`);
});
```

# Validando as variáveis de ambiente

- Agora que definimos o validateEnv podemos usá-lo em nosso

```
david@coiote ~/dev/express $ npm start

> express@1.0.0 start { Request, Response } from 'express'
> nodemon src/index.ts
  Import validateEnv from './utils/validateEnv';
  import dotenv from 'dotenv';
[nodemon] 2.0.22
[nodemon] to restart at any time, enter `rs` (dotenv.config())
[nodemon] watching path(s): *.* (validateEnv)
[nodemon] watching extensions: ts,json
[nodemon] starting `ts-node src/index.ts`
=====
  const app = express();
  =====
  const PORT = process.env.PORT || 3333
Missing environment variables:
  NODE_ENV: undefined
  app.get('/', (req: Request, res: Response) => {
    res.send("Hello world!");
  });
Exiting with error code 1
[nodemon] app crashed - waiting for file changes before starting...
  app.listen(PORT, () => {
    console.log(`Express app iniciada na porta ${PORT}.`);
  });

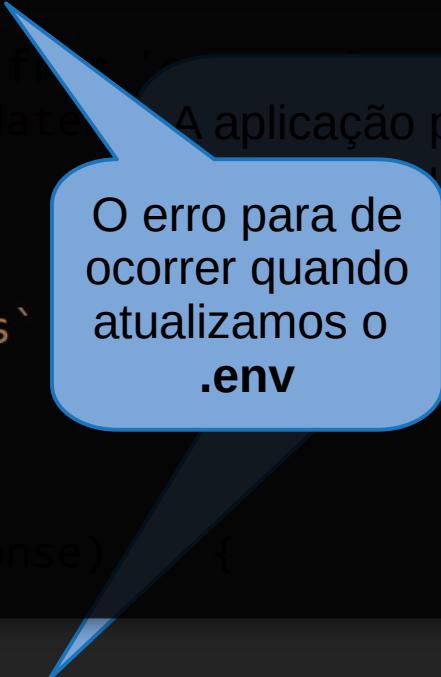
A aplicação passará a reportar um erro caso o arquivo .env não contenha todas as variáveis
```

# Validando as variáveis de ambiente

```
npm start /home/david/dev/express
david@coiote ~/dev/express $ echo "NODE_ENV=development" >> .env
david@coiote ~/dev/express $ npm start
> express@1.0.0 start { Request, Response } {
>   nodemon src/index.ts
>   > express@1.0.0 start [nodemon] 2.0.22
>   > [nodemon] to restart at any time, enter `rs` [nodemon] watching path(s): *.* [nodemon] watching extensions: ts,json [nodemon] starting `ts-node src/index.ts` [nodemon] app crashed - waiting for file changes before starting...
Express app iniciada na porta 3333.
  app.get('/', (req: Request, res: Response) => {
    res.send('Hello world!');
  });
}
Exiting with error code 1
[nodemon] app crashed - waiting for file changes before starting...
  app.listen(PORT, () => {
    console.log(`Express app iniciada na porta ${PORT}.`);
  });

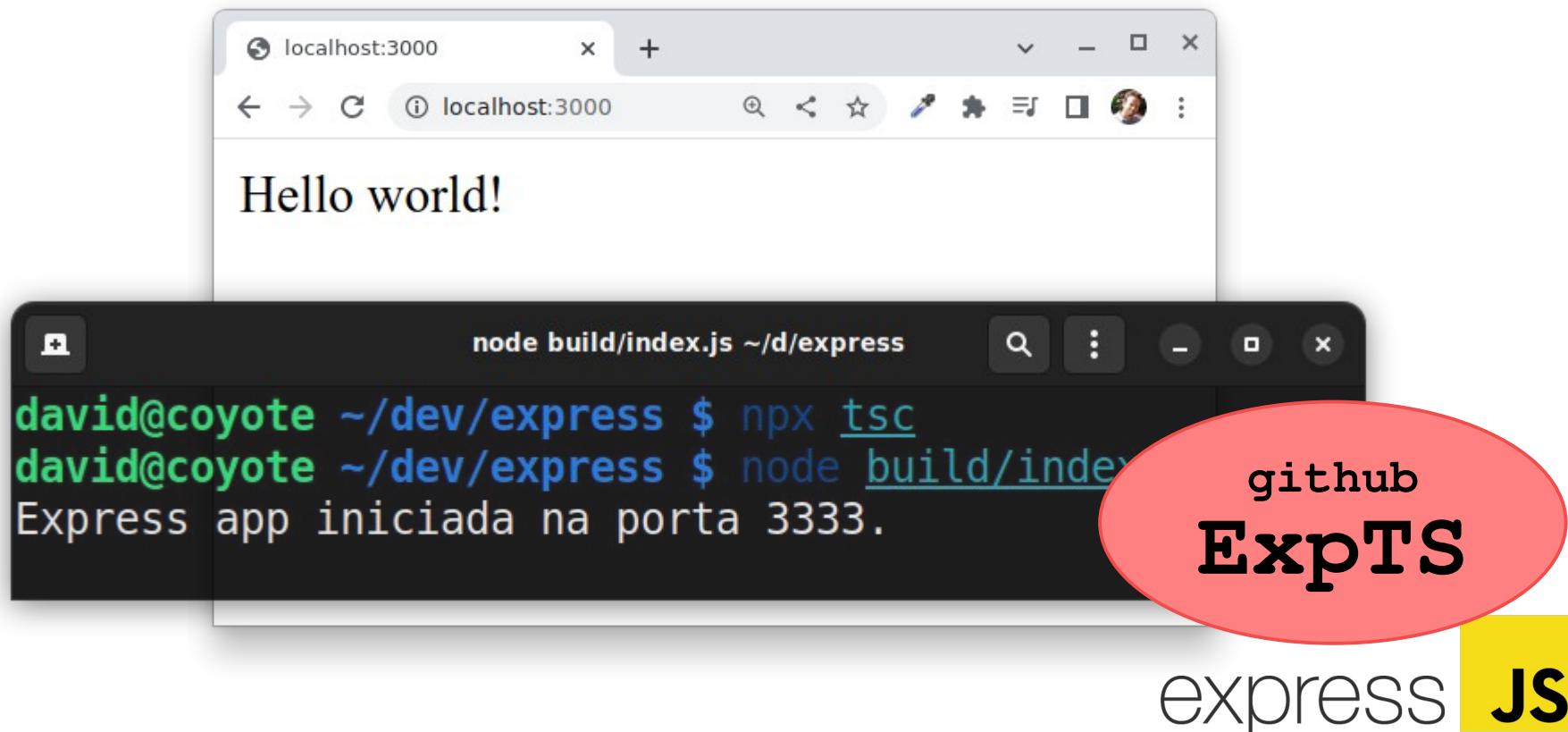
```

O erro para de ocorrer quando atualizamos o .env



# Exercício 3

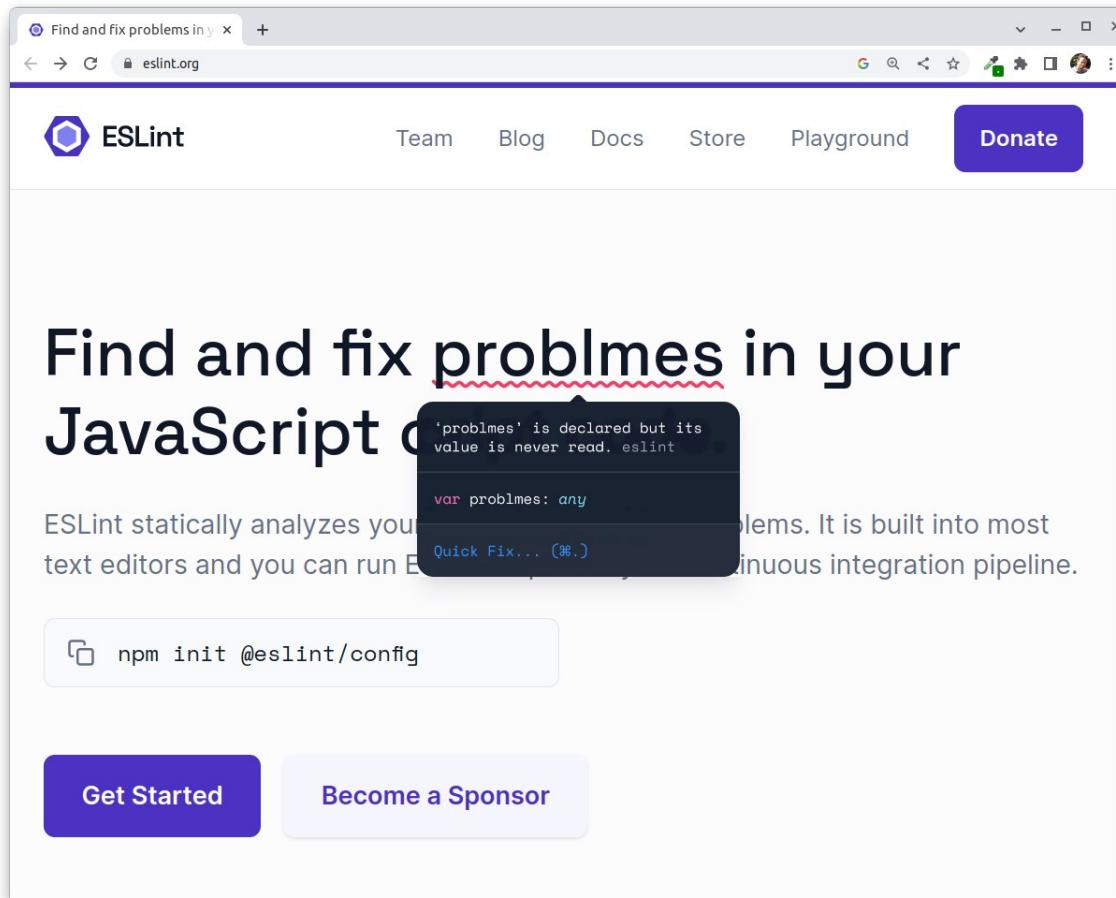
- Crie o arquivo **src/utils/validateEnv.ts** em seu projeto e use-o no **src/index.ts** para validar as variáveis de ambiente do arquivo **.env**



A screenshot of a terminal window and a browser window. The terminal window at the bottom shows the command `node build/index.js ~/d/express` being run, followed by the output: `david@coyote ~/dev/express $ npx tsc`, `david@coyote ~/dev/express $ node build/index`, and `Express app iniciada na porta 3333.`. A red oval on the right side of the terminal window contains the text "github" and "ExpTS". The browser window above shows a single line of text: "Hello world!".

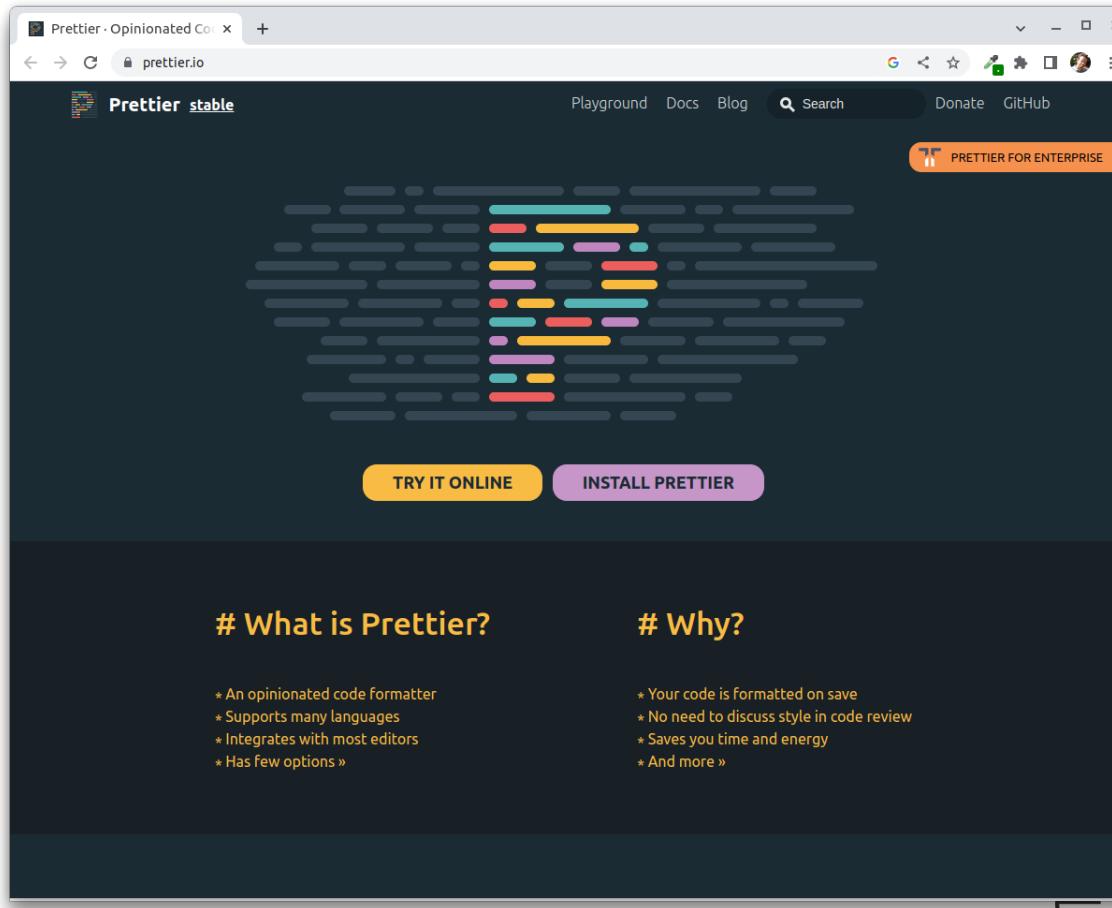
# Instalação do ESLint e Prettier

- **ESLint** é uma ferramenta de **lint**, que analisa o código em busca de erros de estilo e codificação que possam levar a bugs



# Instalação do ESLint e Prettier

- **Prettier** é uma ferramenta para **formatação** de código, que possibilita que os códigos sejam formatados automaticamente



# Instalação do ESLint e Prettier

- Para instalar o **ESLint** e o **Prettier** em nosso projeto, precisaremos adicionar os pacotes abaixo como dependências de desenvolvimento

```
$ npm i -D eslint @typescript-eslint/eslint-plugin  
$ npm i -D @typescript-eslint/parser prettier  
$ npm i -D eslint-config-prettier eslint-plugin-prettier
```

# Configuração do ESLint

- Depois de instalar todos os pacotes, precisamos configurá-lo com o arquivo `.eslintrc`, que poderá ter o seguinte código:

```
{  
  "parser": "@typescript-eslint/parser",  
  "extends": [  
    "eslint:recommended",  
    "plugin:@typescript-eslint/recommended"  
  ],  
  "parserOptions": {  
    "ecmaVersion": "latest",  
    "sourceType": "module"  
  },  
  "rules": {  
    "@typescript-eslint/explicit-function-return-type": 0,  
    "@typescript-eslint/explicit-module-boundary-types": 0,  
    "@typescript-eslint/no-explicit-any": "error",  
    "@typescript-eslint/no-unused-vars": "error",  
    "@typescript-eslint/no-var-requires": "error"  
  }  
}
```

Overview | typescript-eslint x +

typescript-eslint.io/rules/

typescript-eslint Docs Rules Blog v5.59.7 Playground Search CTRL K

Overview

TypeScript Rules

- adjacent-overload-signatures
- array-type
- await-thenable
- ban-ts-comment
- ban-tslint-comment
- ban-types
- class-literal-property-style
- consistent-generic-constructors
- consistent-indexed-object-style
- consistent-type-assertions
- consistent-type-definitions
- consistent-type-exports
- consistent-type-imports
- explicit-function-return-type
- explicit-member-accessibility
- explicit-module-boundary-types
- member-ordering
- method-signature-chaining

Supported Rules

Extension Rules

# Overview

`@typescript-eslint/eslint-plugin` includes over 100 rules that detect best practice violations, bugs, and/or stylistic issues specifically for TypeScript code. See [Configs](#) for how to enable recommended rules using configs.

## Supported Rules

recommended    strict    fixable    has suggestions    requires type information

Rule	<input checked="" type="checkbox"/>	<input type="lock"/>	<input type="lightbulb"/>	<input type="cloud"/>
<code>@typescript-eslint/adjacent-overload-signatures</code> Require that function overload signatures be consecutive	<input checked="" type="checkbox"/>			
<code>@typescript-eslint/array-type</code> Require consistently using either <code>T[]</code> or <code>Array&lt;T&gt;</code> for arrays		<input type="lock"/>	<input type="lightbulb"/>	
<code>@typescript-eslint/await-thenable</code>	<input checked="" type="checkbox"/>			<input type="cloud"/>

Em <https://typescript-eslint.io/rules/> é possível ver todas as rules do Eslint disponíveis para TypeScript

JS

# Configuração do ESLint

- Também podemos adicionar um arquivo `.eslintignore`, onde definimos todos os arquivos que o ESLint deverá ignorar
- Esse arquivo é especialmente importante ao usar Typescript, pois através dele podemos ignorar o diretório **build**

build

# Configuração do Prettier

- Para configurarmos o Prettier, podemos criar um arquivo **.prettierrc** com as seguintes diretrizes de formatação

```
{  
  "printWidth": 80,  
  "tabWidth": 2,  
  "singleQuote": true,  
  "trailingComma": "all",  
  "semi": true,  
  "arrowParens": "always"  
}
```

- Também podemos adicionar um arquivo **.prettierignore**

```
build
```

Options · Prettier

prettier.io/docs/en/options.html

## Prettier stable

Playground Docs Blog Donate GitHub

Configuring Prettier

# Options

Prettier ships with a handful of format options.

To learn more about Prettier's stance on options – see the [Option Philosophy](#).

If you change any options, it's recommended to do it via a [configuration file](#). This way the Prettier CLI, [editor integrations](#) and other tooling knows what options you use.

## Print Width

---

Specify the line length that the printer will wrap on.

**For readability we recommend against using more than 80 characters:**

In code styleguides, maximum line length rules are often set to 100 or 120. However, when humans write code, they don't strive to fit all the code on one line. It's better to have multiple lines for readability.

Em <https://prettier.io/docs/en/options.html> é apresentado a lista de opções do pacote Prettier

EXPRESS JS

# Configuração do Prettier

- E agora podemos adicionar os scripts lint e lint:fix para rodar o eslint

```
"scripts": {  
  "start": "nodemon src/index.ts",  
  "build": "npx tsc",  
  "start:prod": "node build/index.js",  
  "lint": "eslint --ext .ts src/",  
  "lint:fix": "eslint --fix --ext .ts src/",  
  "format": "npx prettier --write src/"  
},
```

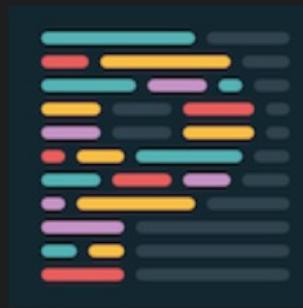


json

{ package.json

≡ Extension: Prettier - Code formatter X

≡ Settings



# Prettier - Code formatter

Prettier [prettier.io](#)

32,564,963



Code formatter using prettier

[Disable](#)[Uninstall](#)

This extension is enabled globally.

[DETAILS](#)[FEATURE CONTRIBUTIONS](#)[CHANGELOG](#)[RUNTIME](#)

## Prettier Formatter for Visual Studio Code

Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.



JavaScript · TypeScript · Flow · JSX · JSON

Podemos usar a extensão Prettier do VSCode para formatar o código automaticamente a cada salvamento

[Resources](#)[Marketplace](#)[Repository](#)[License](#)[Prettier](#)

0



0

{..} : 7

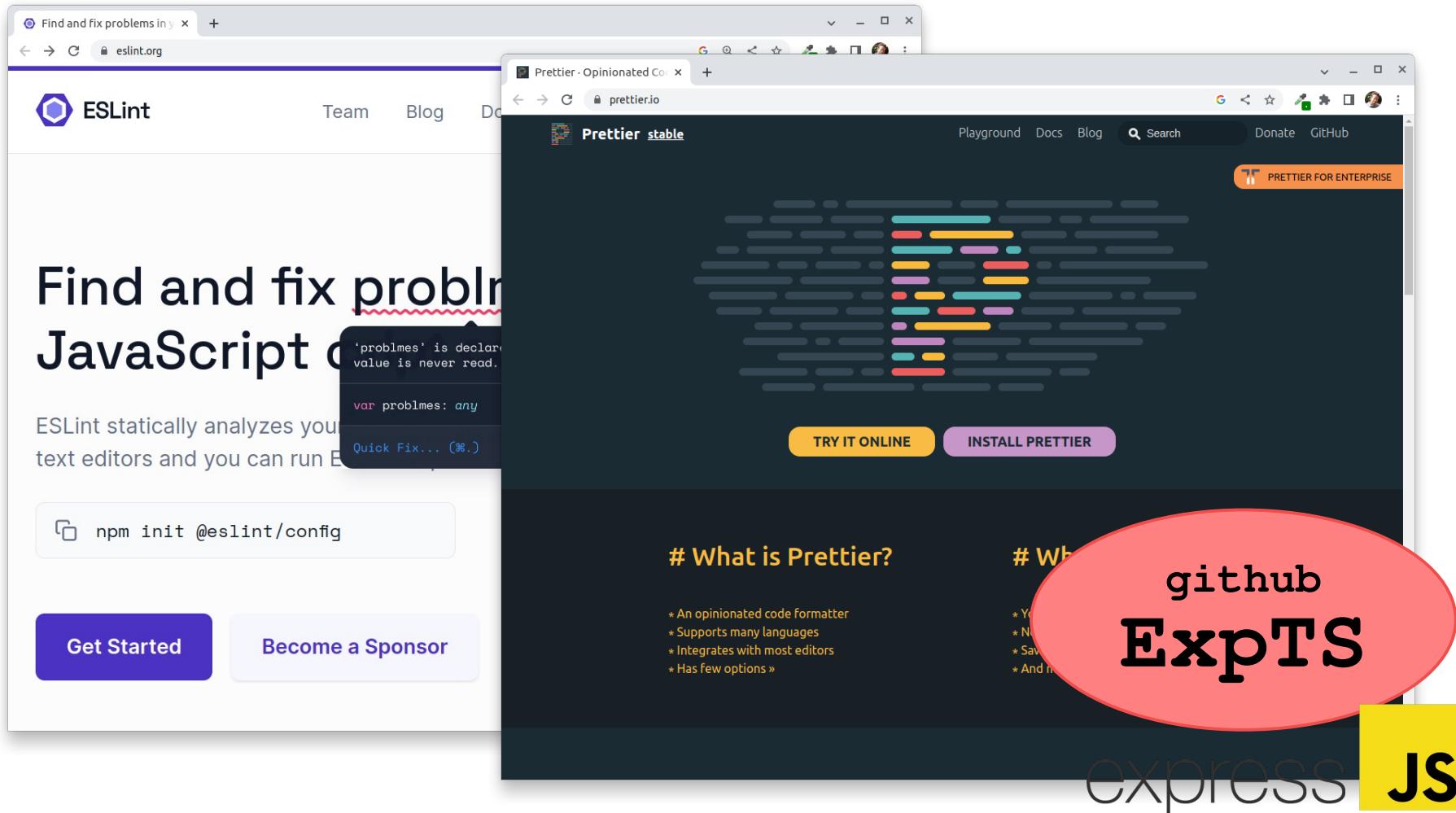


Go Live



# Exercício 4

- Adicione e configure o **eslint** e o **prettier** em seu projeto



# Tipos de Request Handles

- As **requisições GET** podem ser tratadas através de **app.get**

```
app.get("/", (req: Request, res: Response) => {  
    res.send("Hello world!");  
});
```

- As **requisições POST** podem ser tratadas através de **app.post**

```
app.post("/", (req: Request, res: Response) => {  
    console.log("Requisição POST no /");  
});
```

- A função **app.use** é executada em toda requisição

```
app.use((req: Request, res: Response) => {  
    console.log("Executado por toda requisição");  
});
```

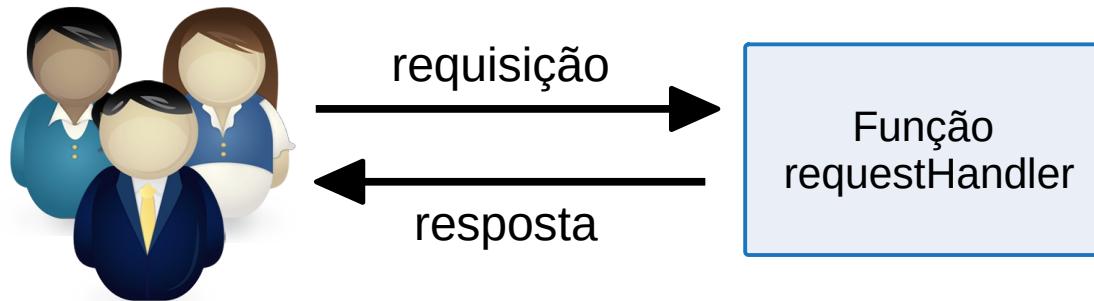
# Os Fundamentos do Express

- O framework express provê 4 funcionalidades principais, que serão vistas ao longo dos próximos slides:
  - **Middleware** – São um conjunto de funções que tem acesso ao objeto de solicitação (req) e ao o objeto de resposta (res)
  - **Routing** – As rotas definem como o aplicativo irá responder às solicitações (URI + HTTP Method) dos clientes
  - **Extensões aos objetos request e response** – Express estende os objetos request e response com novos métodos e propriedades
  - **Views** – As views permitem criar conteúdo HTML de forma dinâmica

# Middleware

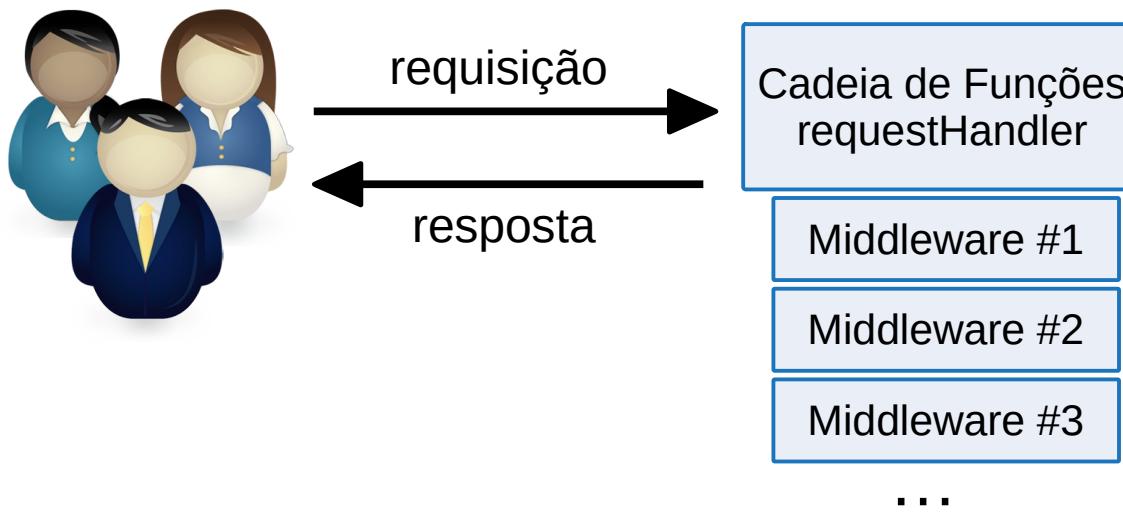
- Usando Node.js puro, usamos o callback de **http.createServer** para responder as requisições dos usuários

```
http.createServer((req, res) => {
  console.log(`Requição do usuário: ${req.url}`);
  res.end("Instituto de Computação!");
});
```



# Middleware

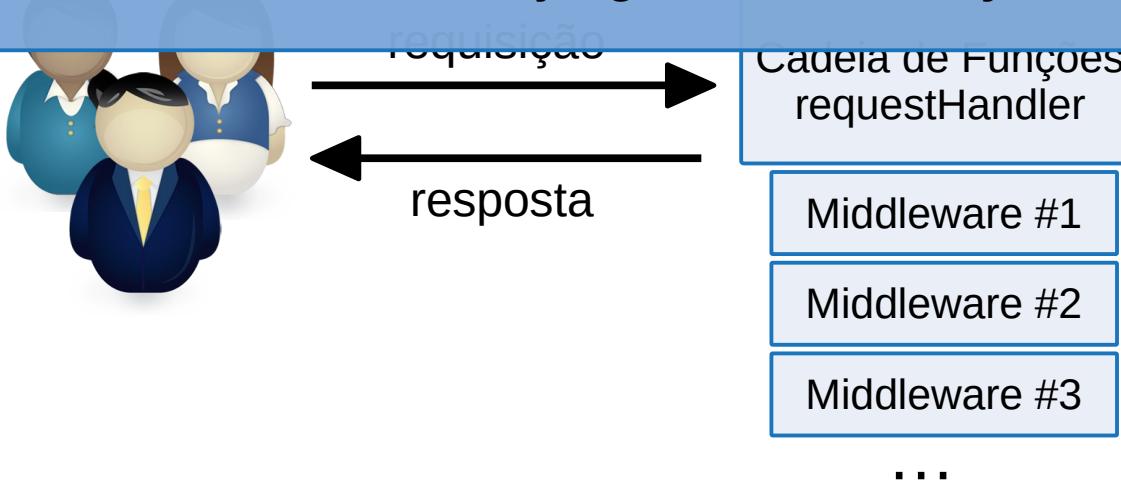
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental: ao invés de um callback, podemos criar vários em sequência



# Middleware

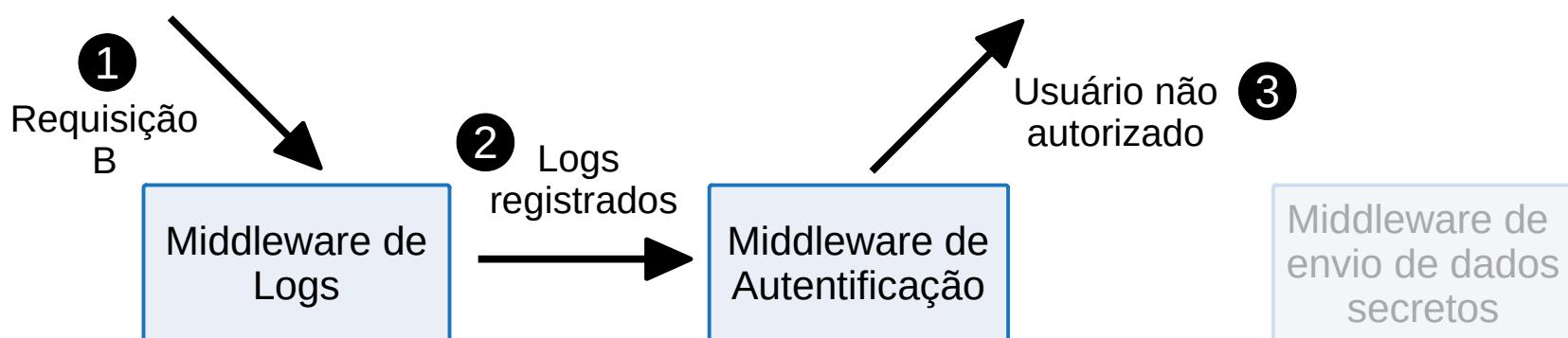
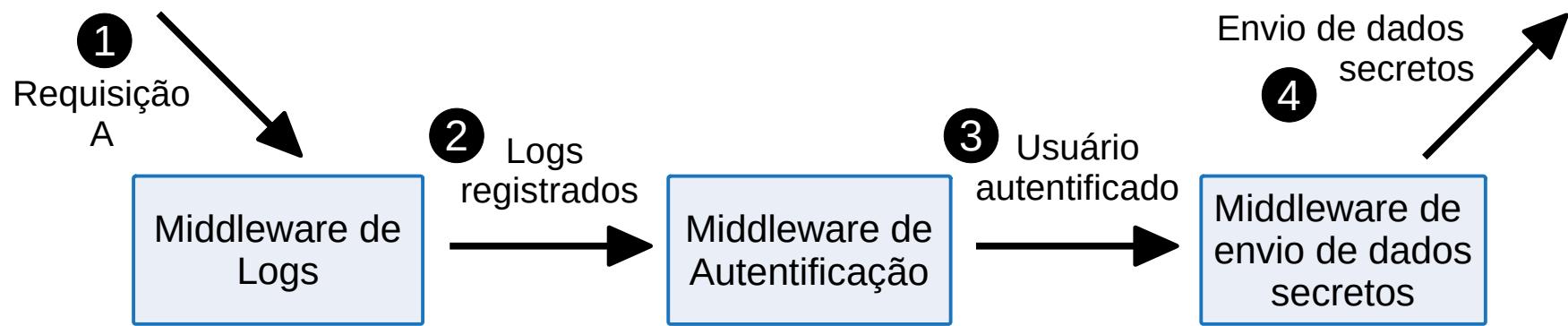
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental:  
ao invés de um callback, podemos criar vários em sequência

Os middlewares não são uma invenção do Express, e estão presentes em vários outros frameworks: **Django**, **Laravel**, **Ruby on Rails**, etc.



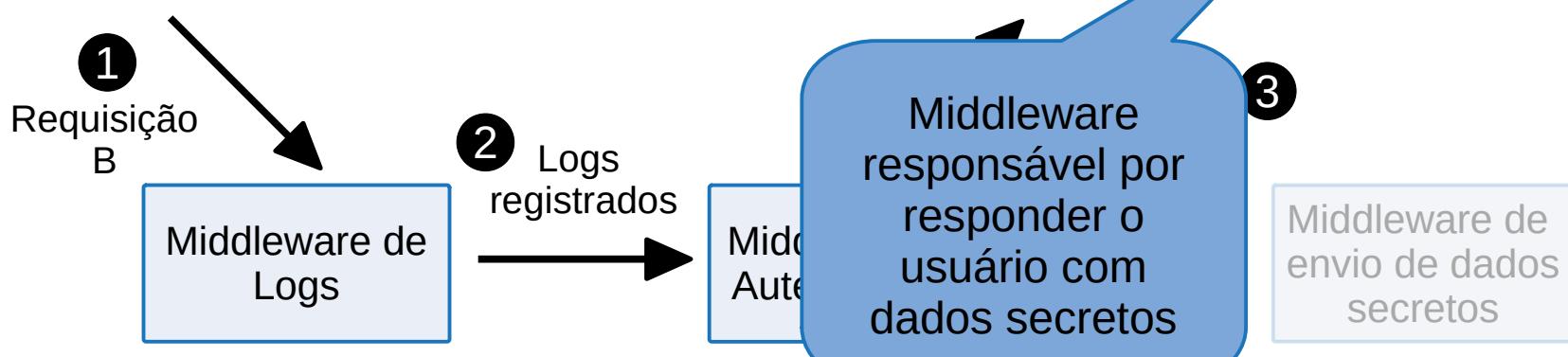
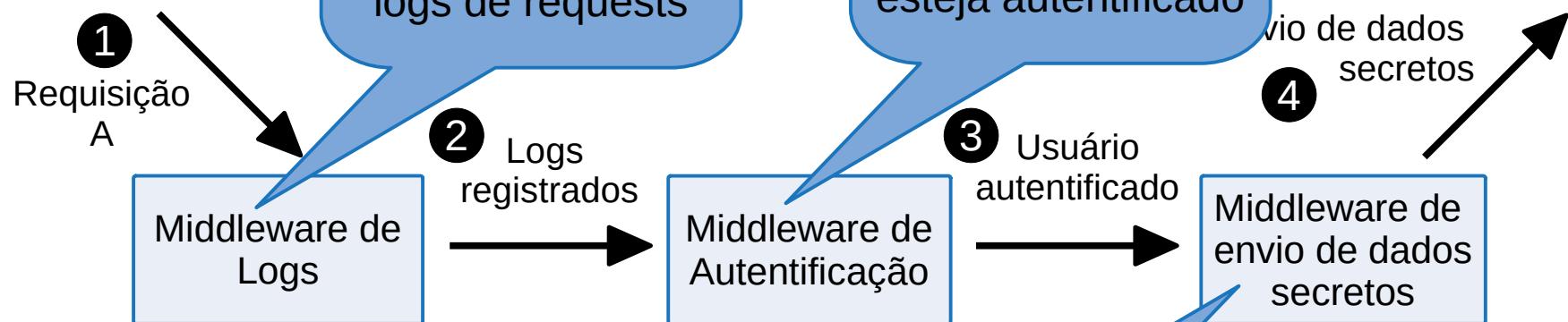
# Middleware

- A Figura abaixo mostra um exemplo de app que autentifica os usuários e retorna dados sigilosos para usuários autorizados



# Middleware

- A Figura mostra que os usuários devem ser autenticados para enviar dados secretos.



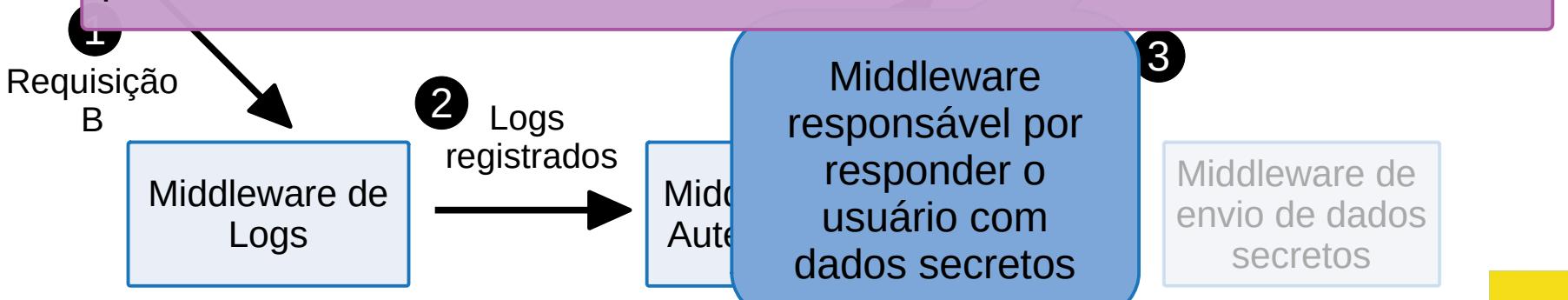
# Middleware

- A Figura mostra que os usuários enviam requisições para o servidor.



Os middlewares são muito úteis porque permitem dividir a lógica da aplicação em várias partes menores

Além disso, muitos módulos de terceiros são projetados para se encaixarem perfeitamente no formato dos middlewares



# Middleware

- Além dos parâmetros **Request (req)** e **Response (res)**, os middlewares também recebem o parâmetro **next**, que pode ser usado para chamar o próximo middleware da cadeia

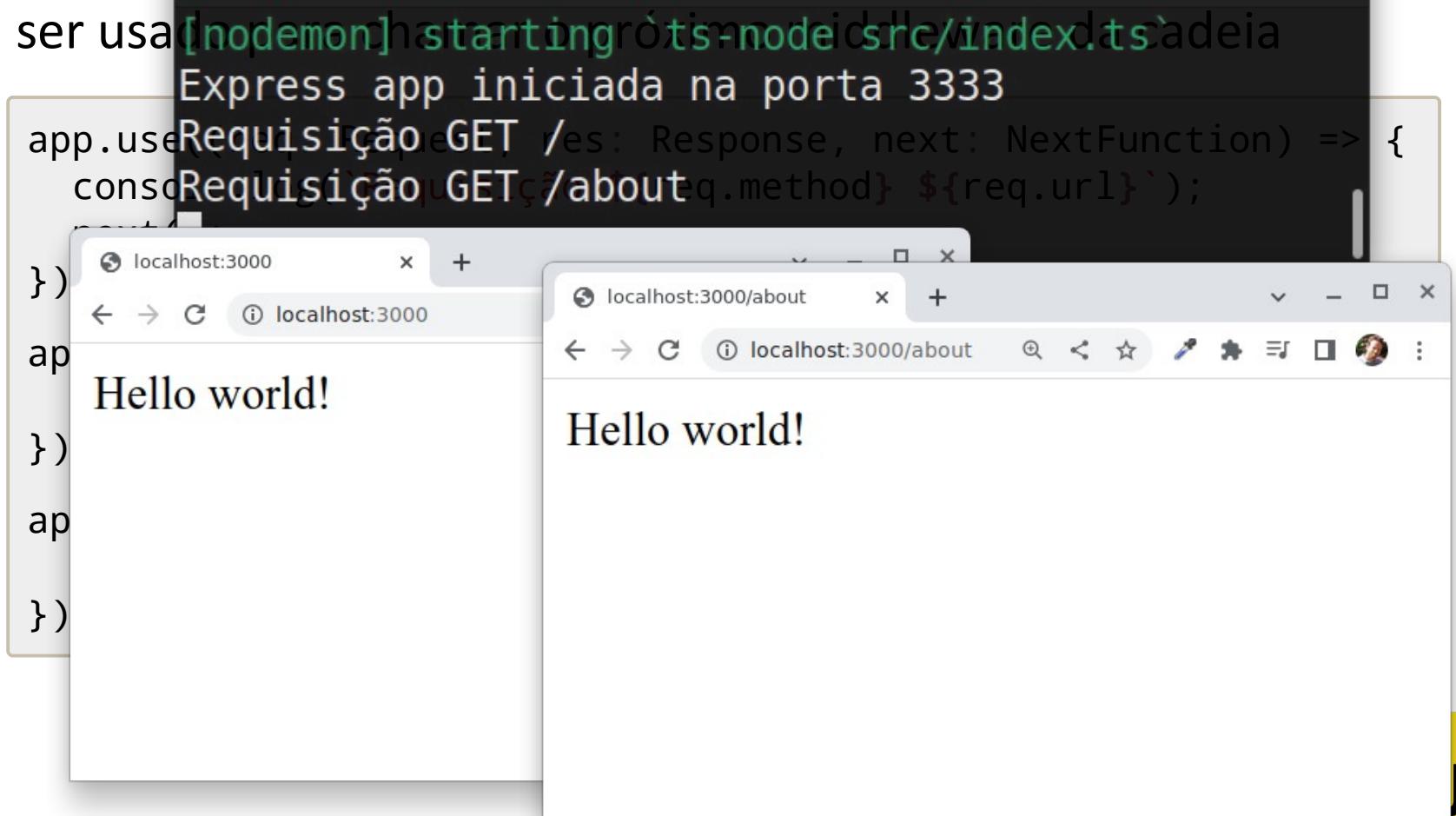
```
app.use((req: Request, res: Response, next: NextFunction) => {
  console.log(`Requisição ${req.method} ${req.url}`);
  next();
});

app.get("/", (req, res) => {
  res.send("Hello world!");
});

app.listen(PORT, () => {
  console.log(`Express app iniciada na porta ${PORT}.`);
});
```

# Middleware

- Além dos parâmetros **Request (req)** e **Response (res)**, os middleware



The screenshot shows a terminal window with the command `npm start ~/d/express` running, outputting the message "[nodemon] starting rotisserie-node [src/index.js] cadeia". Below the terminal, two browser windows are open. The left window is titled "localhost:3000" and displays the text "Hello world!". The right window is titled "localhost:3000/about" and also displays the text "Hello world!". A yellow "JS" logo is visible in the bottom right corner.

```
npm start ~/d/express
[nodemon] starting rotisserie-node [src/index.js] cadeia
Express app iniciada na porta 3333
app.use((req, res, next) => {
  const url = req.url;
  if (url === '/about') {
    res.end(`Hello world!`);
  } else {
    res.end(`Hello ${url}!`);
  }
});
```

localhost:3000

Hello world!

localhost:3000/about

Hello world!

JS

# Middleware

- Dentro de cada middleware, também é possível alterar propriedades dos objetos **request (req)** e **response (res)**

```
app.use((req, res, next) => {
  console.log(`Requisição ${req.method} ${req.url}`);
  next();
});

app.use((req, res, next) => {
  if (user.checkAuth(req)) {
    next();
  } else {
    res.statusCode = 403;
    res.end("Não autorizado");
  }
});

app.use((req, res) => {
  res.end('Dados secretos: {código, 156234}');
});
```

Código HTTP  
para  
**Forbidden**

# Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar middlewares disponíveis no repositório do NPM
- Existem milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O primeiro passo para usar o Morgan é instalá-lo

```
$ npm i morgan  
$ npm i -D @types/morgan
```

# Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar middlewares disponíveis no repositório do NPM
- Existem milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O prime

```
$ npm i  
$ npm i
```

```
import express from 'express';
import morgan from 'morgan';
import dotenv from 'dotenv';

dotenv.config();
const PORT = process.env.PORT ?? 3001;

const app = express();

app.use(morgan('short'));
```

**morgan("short")**  
retorna uma  
função  
middleware

# Bibliotecas Middleware

- Ao invés de programar todos os middlewares de sua aplicação, podemos usar o NPM
- Existem muitos middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs
- O primeiro exemplo:

```
import express from 'express';
$ [nodemon] 2.0.22
$ [nodemon] to restart at any time, enter `rs`  

$ [nodemon] watching path(s): *.*  

[nodemon] watching extensions: ts,json  

[nodemon] starting ts-node src/index.ts
Express app iniciada na porta 3333
::1 - GET / HTTP/1.1 200 2 - 1.459 ms
::1 - GET /favicon.ico HTTP/1.1 404 150 - 1.099 ms
app.use(morgan('short'));
```

morgan("short")  
retorna uma  
função  
middleware

# Exercício 5

- Implemente um **middleware** para salvar os dados de acesso dos usuários em um arquivo de log, dentro de uma pasta informada no arquivo `.env`. O middleware receberá um parâmetro indicando um dos seguintes formatos de logs:
  - **simples**: hora de acesso, a url acessada na aplicação, e o método HTTP

```
new Date().toISOString(), req.url, req.method
```
  - **completo**: hora de acesso, a url acessada na aplicação, o método HTTP, a versão do HTTP, e o User-Agent do browser

```
new Date().toISOString(), req.url, req.method,  
req.httpVersion, req.get('User-Agent')
```

- Coloque a pasta de logs no arquivo `.gitignore`

github  
**ExptS**

express JS

# Bibliotecas Middleware

- Outro exemplo de middleware amplamente utilizado é o **middleware de arquivos estáticos** do Express
- Esse middleware é usado para envio de arquivos de imagens, de estilos CSS e código JavaScript para o browser
- A maioria das aplicações possui um diretório específico para esse tipo de arquivo, normalmente chamado de **public**



A screenshot of a terminal window titled 'david@coyote ~/dev/express/public'. The window shows the command 'ls' being run twice. The first 'ls' command shows subdirectories 'css', 'html', 'img', and 'js'. The second 'ls' command shows files 'img/' and 'codebench.png\*'. The terminal has a dark theme with light-colored text.

```
david@coyote ~/dev/express/public $ ls
css/ html/ img/ js/
david@coyote ~/dev/express/public $ ls img/
codebench.png*
david@coyote ~/dev/express/public $
```

# Bibliotecas Middleware

- Através do código abaixo, quaisquer arquivos **css**, **js**, ou **imagem**, disponíveis em suas respectivas pastas de **/public**, podem ser acessados através das rotas **/css**, **/js** e **/img**, respectivamente

```
import express, { Response, Request } from 'express';
import dotenv from 'dotenv';

dotenv.config();
const PORT = process.env.PORT ?? 3333;
const publicPath = `${process.cwd()}/public`;

const app = express();

app.use('/css', express.static(`${publicPath}/public/css`));
app.use('/js', express.static(`${publicPath}/public/js`));
app.use('/img', express.static(`${publicPath}/public/img`));

app.listen(PORT, () => {
  console.log(`Express server listening on port ${PORT}`);
});
```

Executa **next**  
caso o arquivo  
não seja  
encontrado

Executado em  
toda requisição  
que comence  
com /img

# Bibliotecas Middleware

- Através do código abaixo, quaisquer arquivos **css**, **js**, ou **imagem**, disponíveis em suas respectivas pastas de **/public**, podem ser acessados através das rotas **/css**, **/js** e **/img**, respectivamente

```
import express, { Response, Request } from 'express';
import dotenv from 'dotenv';

codebench.png (951×208) x + next
localhost:3333/img/codebench.png


The browser window shows the URL localhost:3333/img/codebench.png. The page content includes a cartoon boy's head on the left and the word "CODEBENCH" in large, bold, black letters on the right.


```

```
app.listen(PORT, () =>
  console.log(`Express server listening at http://localhost:${PORT}`));
});
```

Executado em  
toda requisição  
que comece  
com /img

express **JS**

# Bibliotecas Middleware

- Existem várias outras bibliotecas middleware muito usadas, dentre as quais podemos destacar:
  - **Connect-ratelimit** – Permite limitar o número de requisições por hora de alguém que esteja tentando derrubar seu servidor
  - **Compression** – comprime os dados enviados para o cliente (browser) usando gzip
  - **Body-parser** – converte a propriedade body da requisição do usuário (req.body) para vários formatos, incluindo JSON
  - **Response-time** – registra o tempo de resposta das requisições dos usuários, sendo útil para debugar a performance da aplicação

# Roteamento

- Roteamento é usado para definir quais middlewares serão usados para responder quais tipos de URLs

```
app.get("/", (req, res) => {
  res.end("Bem-vindo ao meu site!");
});
```

Executado só na requisição **GET /**

```
app.get("/sobre", (req, res) => {
  res.end("Bem-vindo à página sobre!");
});
```

Executado só na requisição **GET /sobre**

```
app.use((req, res) => {
  res.statusCode = 404;
  res.end("404!");
});
```

Executado se a rota não for **GET /** nem **GET /sobre**

# Roteamento

The image shows three separate browser windows side-by-side, each displaying a different page from a web application running on localhost:3333.

- Top Window:** The URL is `localhost:3333`. The content says "Bem-vindo ao meu site!"
- Middle Window:** The URL is `localhost:3333/sobre`. The content says "Bem-vindo à página sobre!"
- Bottom Window:** The URL is `localhost:3333/oi`. The content says "404!"

On the left side of the image, there is some code visible, likely the Node.js server code for the application:

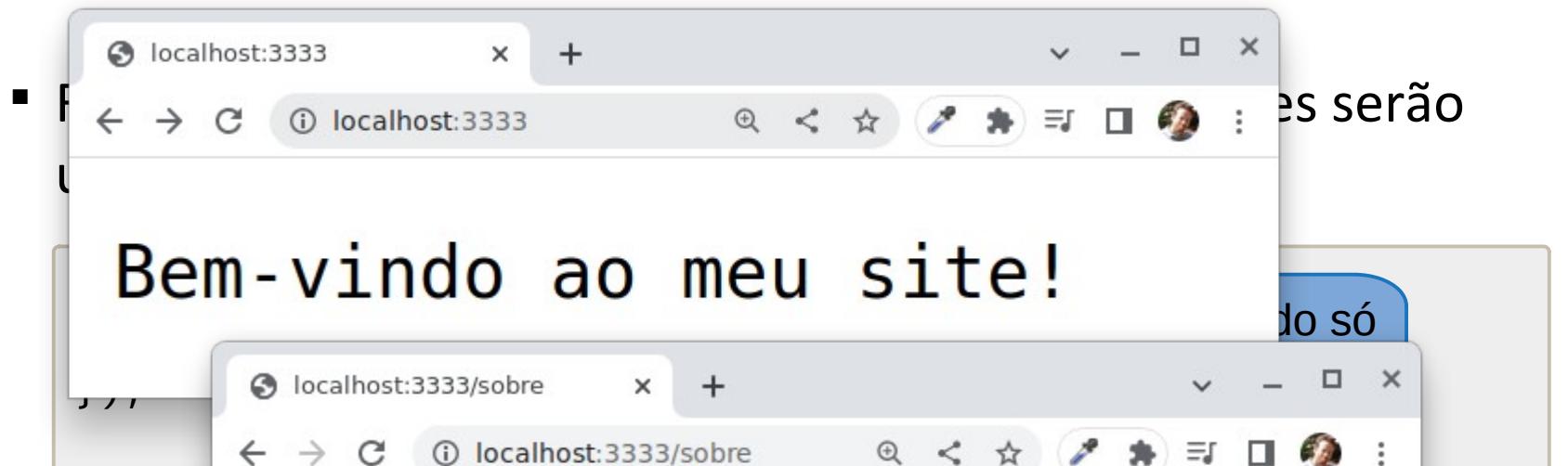
```
app.get('/', function (req, res) {
  res.send('Bem-vindo ao meu site!');
});

app.get('/sobre', function (req, res) {
  res.send('Bem-vindo à página sobre!');
});

app.use(function (req, res, next) {
  res.status(404);
  res.end("404!");
});
```

A yellow box in the bottom right corner contains the letters "JS".

# Roteamento



Além de `app.get()`, também existe o método `app.post()`, que é usado para responder às requisições do tipo POST

```
app.us  
res.statu  
res.end("'  
});
```

404!

JS

# Roteamento

- Além dos nomes de rotas fixos, o Express aceita o uso de rotas mais complexas, incluindo **expressões regulares**

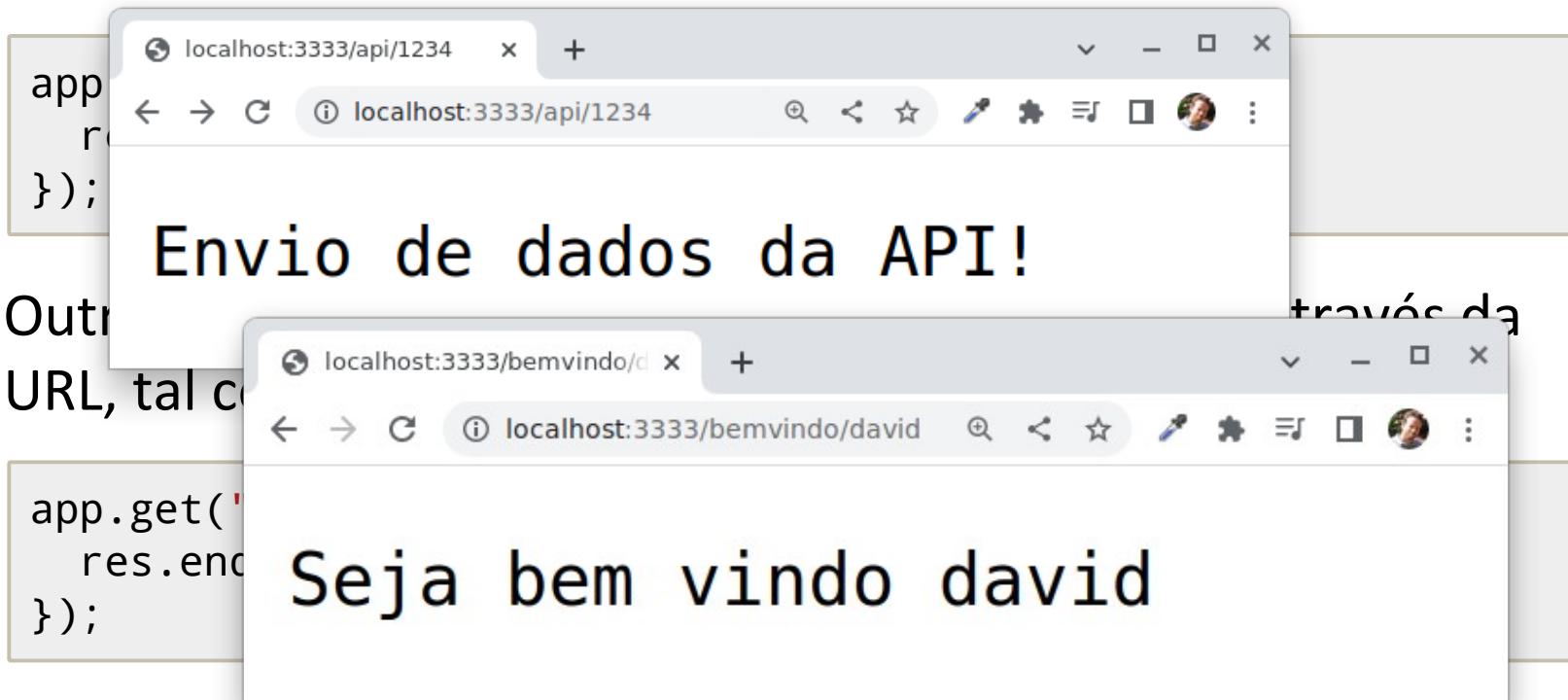
```
app.get( /^\/(api|rest)\/.+$/, (req, res) => {
  res.send("Envio de dados da API!");
});
```

- Outra possibilidade é a **passagem de parâmetros** através da URL, tal como demonstrado no exemplo a seguir

```
app.get("/bemvindo/:nome", (req, res) => {
  res.end(`Seja bem vindo ${req.params.username}`);
});
```

# Roteamento

- Além dos nomes de rotas fixos, o Express aceita o uso de rotas mais complexas, incluindo **expressões regulares**



# Arquivo Separado de Rotas

- Ao invés de definir todas as rotas no arquivo principal da aplicação, é possível criar um arquivo separado para elas
- Para isso, podemos usar a classe **express.Router**, que retorna um middleware com um sistema de roteamento completo

```
// Módulo de rotas, arquivo /router/router.ts

Import { Router } from 'express';
const router = Router();

router.get('/', (req, res) => {
  res.send('Página principal do site');
});

router.get('/sobre', (req, res) => {
  res.send('Página sobre');
});

export default router;
```

# Arquivo Separado de Rotas

- Em seguida, podemos carregar o módulo de roteamento no arquivo principal da app

```
// Módulo principal, arquivo /src/index.ts

import express from 'express';
import router from './router/router';

const app = express();
app.use(router);

app.listen(3000, () => {
  console.log("Express app iniciada na porta 3000.");
});
```

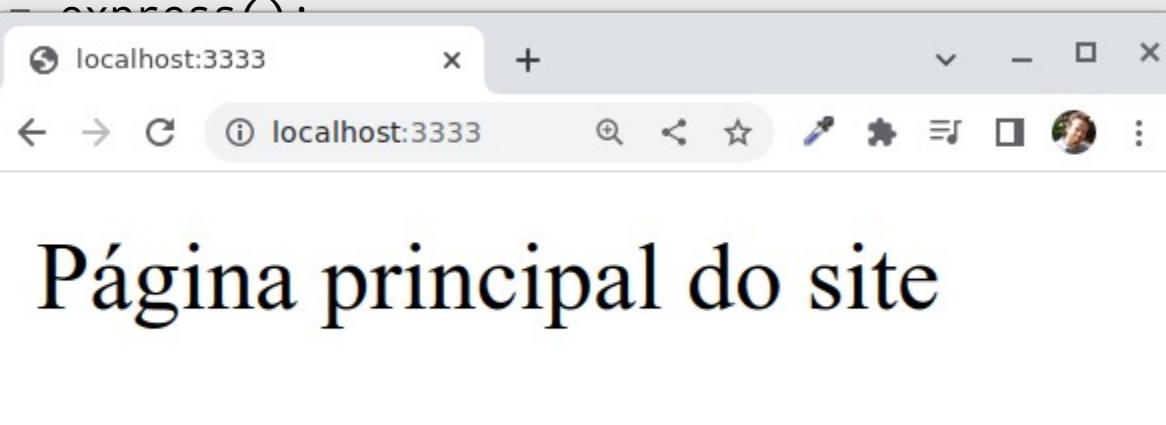
# Arquivo Separado de Rotas

- Em seguida, podemos carregar o módulo de roteamento no arquivo principal da app

```
// Módulo principal, arquivo /src/index.ts

import express from 'express';
import router from './router/router';

const app = express();
app.use(router);
app.listen(3333, () => {
  console.log('Servidor rodando na porta 3333');
});
```



# Arquivo Separado de Rotas

- Em seguida, podemos carregar o módulo de roteamento no arquivo principal da app

```
// Módulo principal, arquivo /src/index.ts  
  
import express from 'express';  
import router from './router/router';
```

O **Router** surgiu na versão 4.0 do **Express**, e é uma alternativa ao método tradicional de criação das rotas. A função do Router é permitir a definição das rotas em um módulo separado.

```
});
```

Página principal do site

# Extensões de Req/Res

- **Request** (req) e **response** (res) são objetos passados para todas as funcões que tratam as requisições de usuários
- Eles já existiam no Node.js puro, mas o Express adicionou novas funcionalidades para esses dois objetos
- Por exemplo, o método **redirect()** foi adicionado pelo Express

```
res.redirect("/hello/world");
res.redirect("http://expressjs.com");
```

Redireciona  
o usuário para  
a nova URL

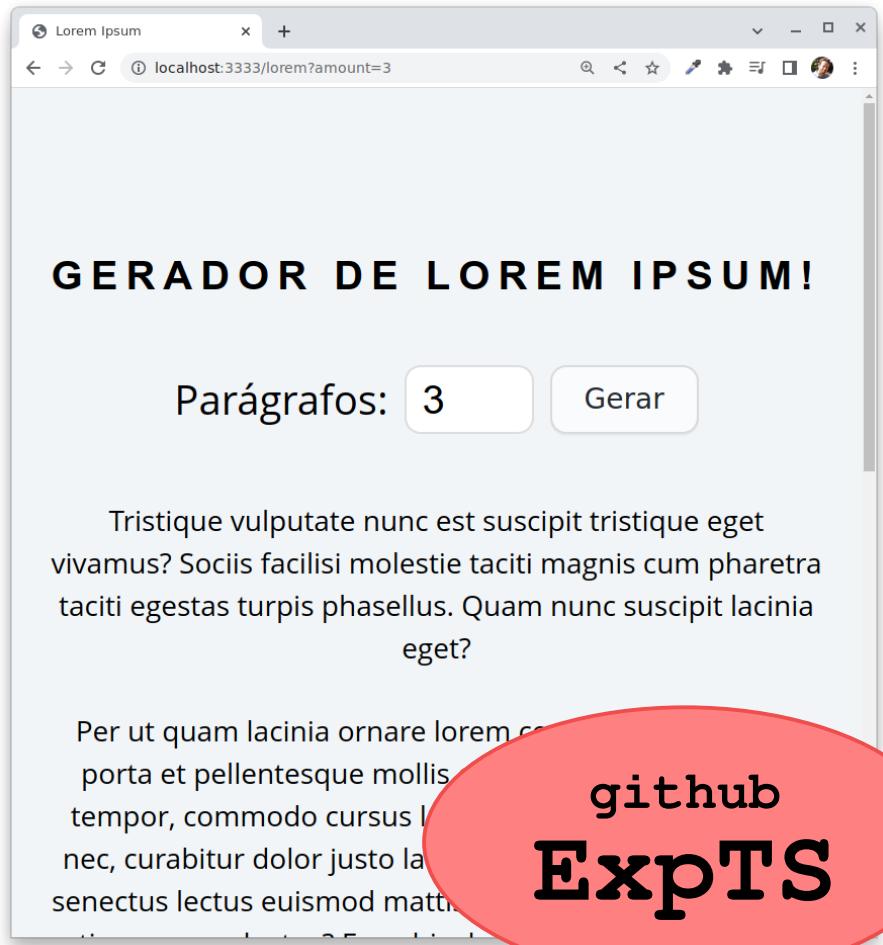
- Outro exemplo é o método **sendFile()**

```
res.sendFile("/path/to/song.mp3");
```

Envia o  
arquivo para  
o browser

# Exercício 6

Reimplemente o exercício Lorem Ipsum dos slides sobre NodeJS, mas desta vez usando o **middleware express.static** para compartilhar o conteúdo html, css e javascript com o usuário. As rotas da aplicação devem estar em um **arquivo separado**, e a página de geração de parágrafos Lorem Ipsum deverá ser carregada para a **rota /lorem**.



# Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
  - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express, dentre as quais destaca-se: EJS, Handlebars, Pug e Mustache



# Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
  - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express,

Conforme já dito, o Express é um **framework não opinativo** e requer uma série de acessórios de terceiros para construir uma aplicação completa

Desta forma, como o Express não possui uma **engine de views** própria, é necessário escolher uma dentre as engines disponíveis para node.js

handlebars



# Views

- As **views** são responsáveis por gerar automaticamente o código HTML que é enviado pelo usuário a cada requisição
  - Fazem parte do modelo **MVC – Model, View, Controller**
- Existem muitas engines de views disponíveis para o Express,

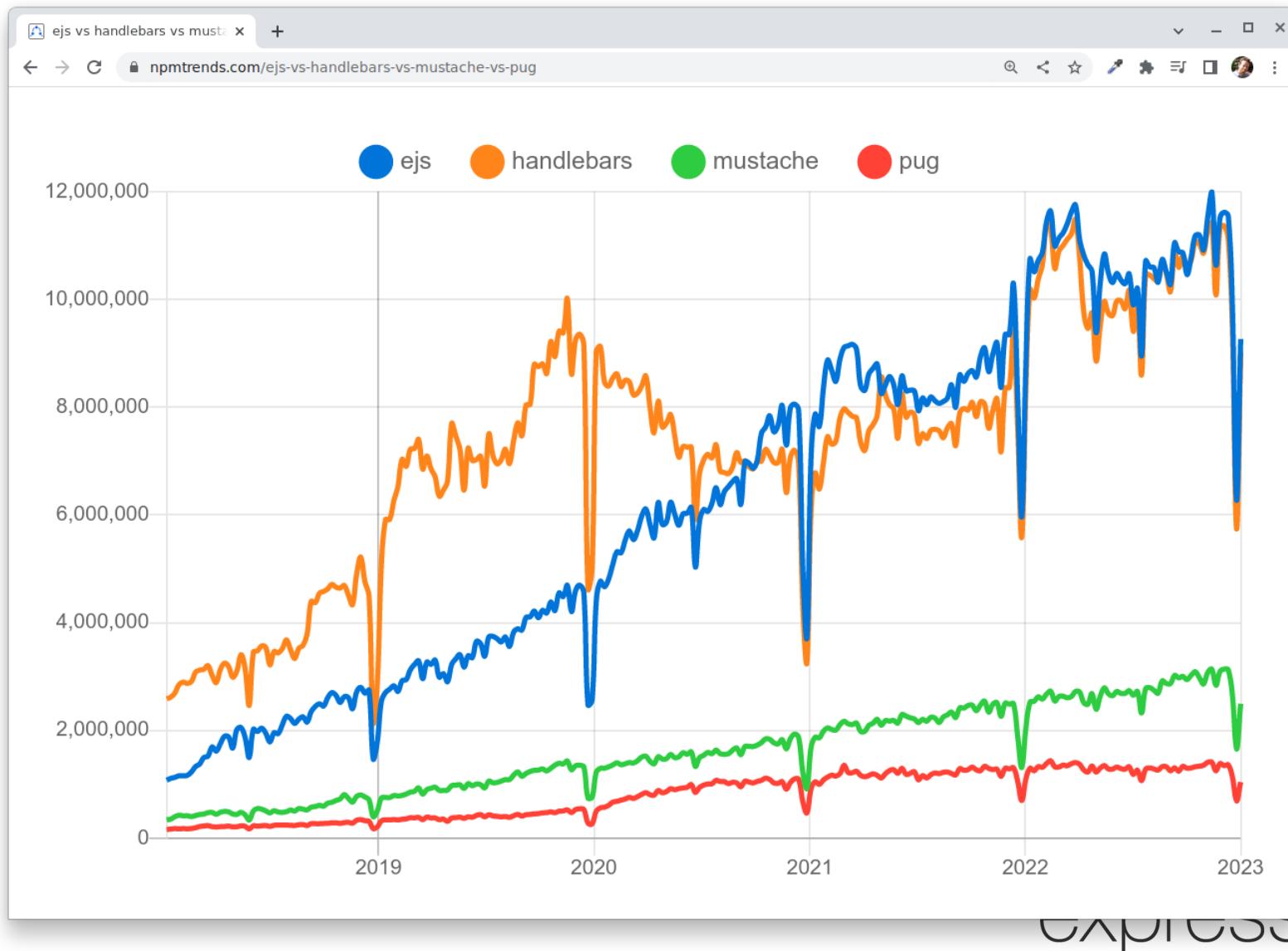
Conforme já dito, o Express é um **framework não opinativo** e requer uma série de acessórios de terceiros para construir uma aplicação completa

O que parece ser uma limitação é na realidade uma grande vantagem, pois sempre que formos desenvolver uma nova aplicação, podemos escolher as melhores bibliotecas disponíveis para node.js naquele momento

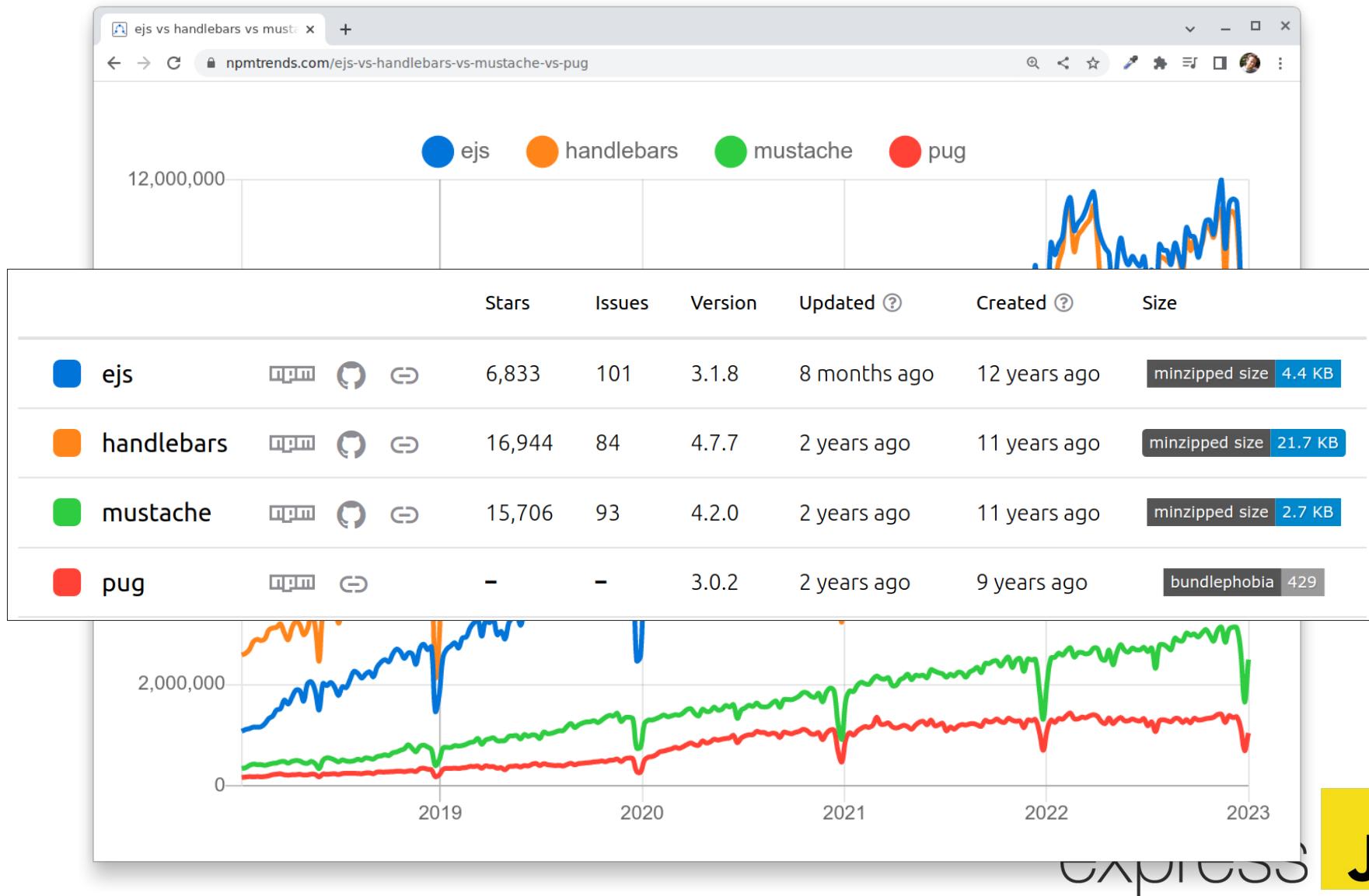
handlebars



# Views



# Views



# Views



# Engine Handlebars

- Para usar o **Handlebars**, é necessário instalá-lo através do NPM

```
$ npm install express-handlebars  
$ npm install -D @types/express-handlebars
```

- Após isso, é preciso informar o Express sobre a escolha do **Handlebars** como engine de views

```
import { engine } from 'express-handlebars';  
  
app.engine("handlebars", engine());  
app.set("view engine", "handlebars");  
app.set("views", `${__dirname}/views`);
```

Informa a localização do diretório de views

# Engine Handlebars

- O Express adiciona o método **render()** ao objeto **response (res)**, usado para renderizar o conteúdo de uma view

```
app.get('/hb1', (req, res) => {
  res.render('hb1', {
    mensagem: 'Olá, você está aprendendo Express + HBS!',
    layout: false,
  });
});
```

Objeto com dados para a view

```
<!DOCTYPE html>
<html>
  <head>
    <title>Express + HBS!</title>
  </head>
  <body>
    {{mensagem}}
  </body>
</html>
```

Arquivo views/index.handlebars



# Engine Handlebars

- Também é possível usar estruturas **if** com o **Handlebars**

```
app.get('/hb2', (req, res) => {
  res.render('hb2', {
    poweredByNodejs: true,
    name: 'Express',
    type: 'Framework',
    layout: false,
  });
});
```

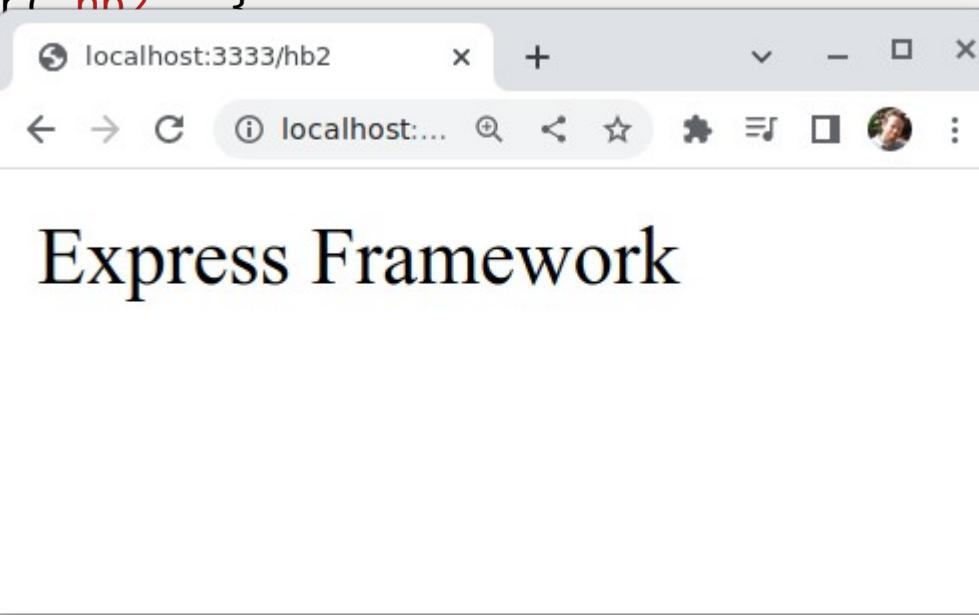
```
<div class='entry'>
  {{#if poweredByNodejs}}
    <span>{{name}} {{type}}</span>
  {{/if}}
</div>
```

# Engine Handlebars

- Também é possível usar estruturas **if** com o **Handlebars**

```
app.get('/hb2', (req, res) => {
  res.render('hb2', {
    powered: true,
    name: 'Node.js',
    type: 'Express Framework',
    layout: 'main'
  });
});
```

```
<div class="header">
  {{#if powered}}
    <span>{{name}} - {{type}}
  {{/if}}
</div>
```



# Engine Handlebars

- Também é possível usar percorrer um vetor com **#each**

```
app.get('/hb3', (req, res) => {
  const profes = [
    { nome: 'David Fernandes', sala: 1238 },
    { nome: 'Horácio Fernandes', sala: 1233 },
    { nome: 'Edleno Moura', sala: 1236 },
    { nome: 'Elaine Harada', sala: 1231 }
  ];
  res.render('hb/hb3', { profes, layout: false });
});
```

```
<div class="container">
  <p>Alguns professores do Icomp/UFAM</p>
  <ul>
    {{#each profes}}
      <li>{{nome}} - sala {{sala}}</li>
    {{/each}}
  </ul>
</div>
```

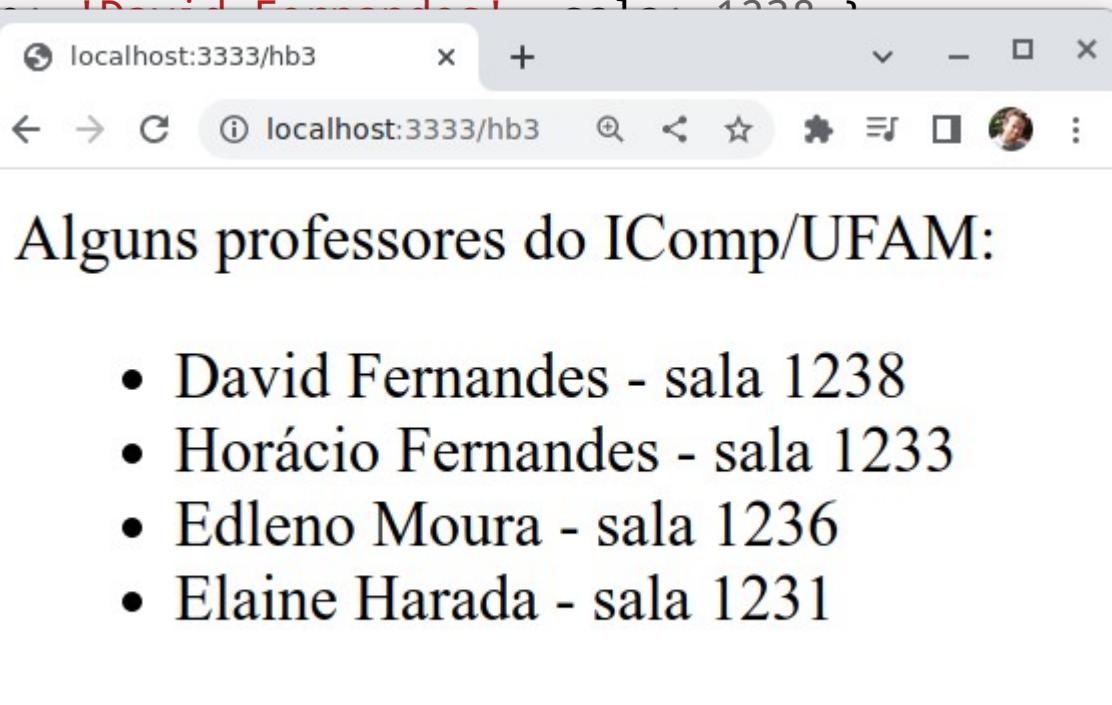
# Engine Handlebars

- Também é possível usar percorrer um vetor com **#each**

```
app.get('/hb3', (req, res) => {
  const profes = [
    { nome: 'David Fernandes', sala: 1238 },
    { nome: 'Horácio Fernandes', sala: 1233 },
    { nome: 'Edleno Moura', sala: 1236 },
    { nome: 'Elaine Harada', sala: 1231 }
  ];
  res.render('hb3', { profes });
});
```

```
<div class="list">
  <p>Alguns professores do IComp/UFAM:</p>
  <ul>
    {{#each profes}}
      <li>
        {{/each}}
      </li>
    {{/each}}
  </ul>
</div>
```

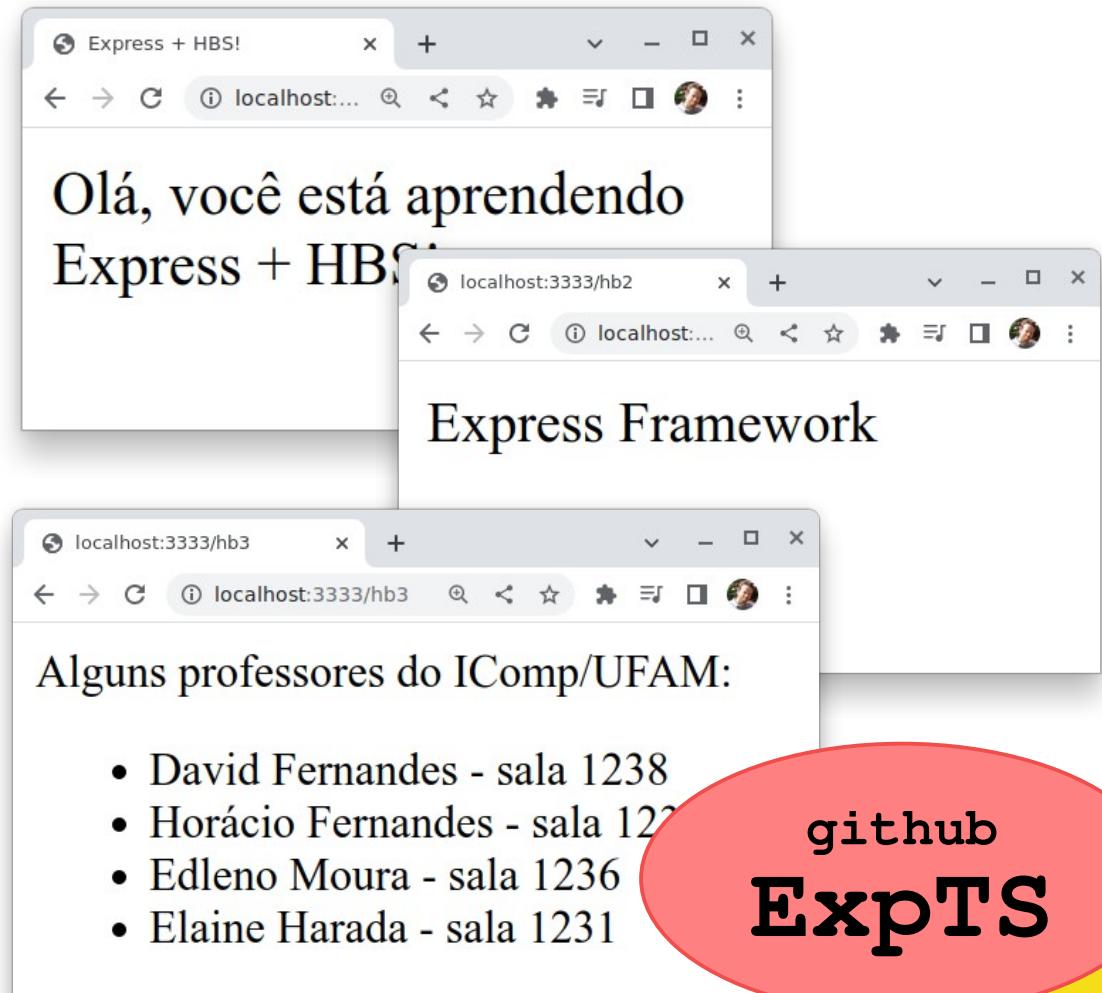
localhost:3333/hb3



- David Fernandes - sala 1238
- Horácio Fernandes - sala 1233
- Edleno Moura - sala 1236
- Elaine Harada - sala 1231

# Exercício 7

- Implemente as páginas das rotas **/hb1**, **/hb2** e **/hb1** mostradas nos slides anteriores. Pode-se mudar o conteúdo das páginas, mas não os recursos utilizados do **Handlebars**. Coloque todas as rotas no arquivo externo de rotas.



github  
**Expts**

express **JS**

# Helpers Handlebars

- Um dos recursos mais importantes do Handlebars é a possibilidade de criar **helpers customizados**

```
app.engine("handlebars", handlebars.engine({
  helpers: require(`__dirname/views/helpers/helpers.ts`)
}));
```

```
// Arquivo /views/helpers/helpers.ts

import { Prof } from './helpersTypes';

export function listProfs(profs: Prof[]) {
  const list = profs.map((p)=>`<li>${p.nome}-${p.sala}</li>`);
  return `<ul>${list.join('')}</ul>`;
}
```

```
// Arquivo /views/helpers/helpersTypes.ts

export interface Prof {
  nome: string;
  sala: string;
}
```

# Helpers Handlebars

- A partir deste momento, a função **listProfs** pode ser usada nas views Handlebars

```
app.get('/hb4', function (req, res) {
  const profes = [
    { nome: 'David Fernandes', sala: 1238 },
    { nome: 'Horácio Fernandes', sala: 1233 },
    { nome: 'Edleno Moura', sala: 1236 },
    { nome: 'Elaine Harada', sala: 1231 },
  ];
  res.render('hb/hb4', { profes, layout: false });
});
```

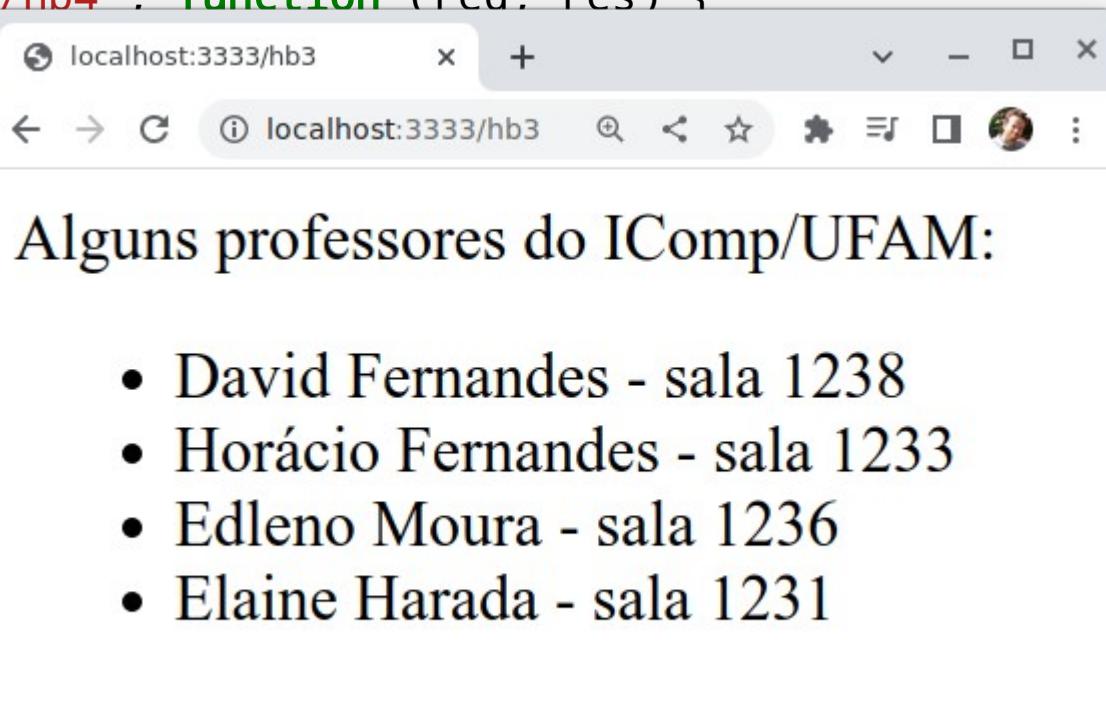
```
<div class='container'>
  Alguns professores do IComp/UFAM:
  {{{listProfs profes}}}
</div>
```

# Helpers Handlebars

- A partir deste momento, a função **listProfs** pode ser usada nas views Handlebars

```
app.get('/hb4', function (req, res) {  
  const profs = [  
    { nome: 'David Fernandes', sala: 1238 },  
    { nome: 'Horácio Fernandes', sala: 1233 },  
    { nome: 'Edleno Moura', sala: 1236 },  
    { nome: 'Elaine Harada', sala: 1231 }  
  ];  
  res.render('list', { profs });  
});
```

```
<div class="list">  
  Alguns professores do IComp/UFAM:  
  {{list}}  
</div>
```



# Helpers Handlebars

- A partir deste momento, a função **listProfs** pode ser usada nas views Handlebars

```
app.get('/hb4', function (req, res) {
```

O Handlebars possui uma série de pacotes de helpers que podem ser instalados via NPM, tal como o **handlebars-helpers**

```
    var profs = [
      { nome: "David Fernandes", sala: 1238 },
      { nome: "Horácio Fernandes", sala: 1233 },
      { nome: "Edleno Moura", sala: 1236 },
      { nome: "Elaine Harada", sala: 1231 }
    ];
    res.render('list', { profs: profs });
});
```

```
<div class="list">
  Alguns professores do IComp/UFAM:
  {{list}}
</div>
```

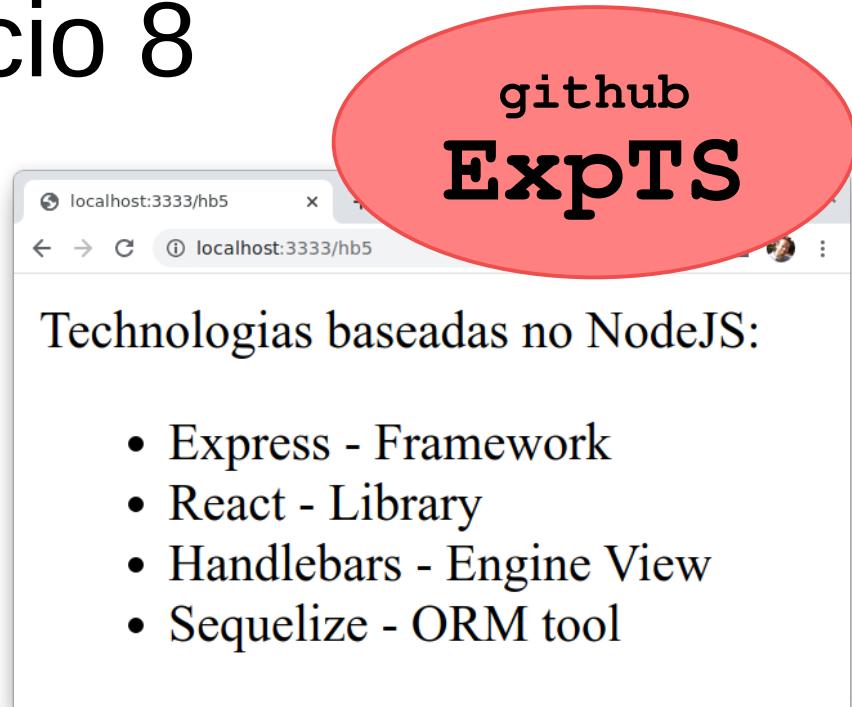
Alguns professores do IComp/UFAM:

- David Fernandes - sala 1238
- Horácio Fernandes - sala 1233
- Edleno Moura - sala 1236
- Elaine Harada - sala 1231

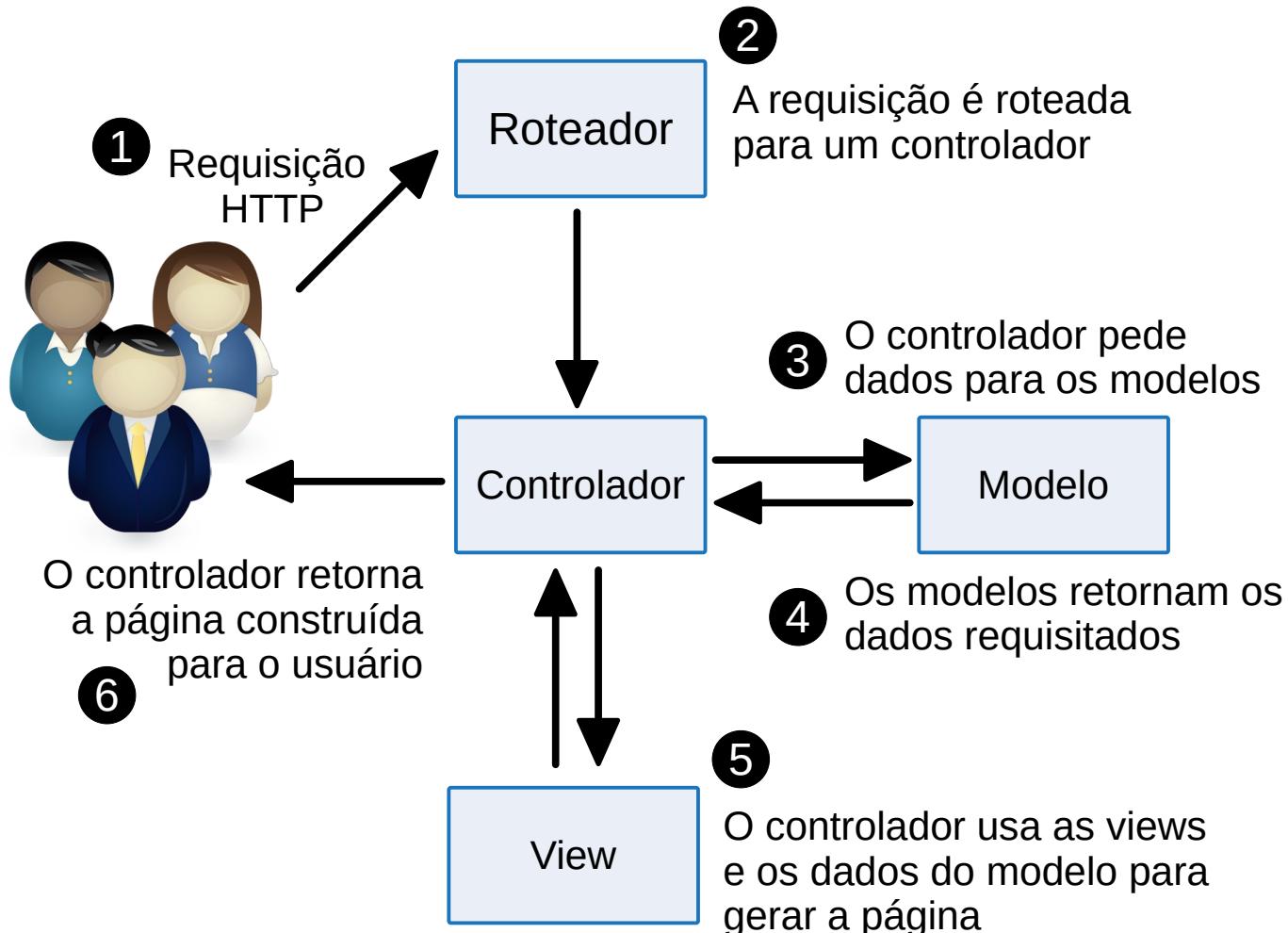
# Exercício 8

Crie uma rota `/hb5` que usa um **helper handlebars** capaz de receber um vetor de tecnologias, com os campos **name (string)**, **type (string)** e **poweredByNodejs (boolean)** e imprima a lista de tecnologias desenvolvidas com Node.JS

```
const technologies = [
  { name: 'Express', type: 'Framework', poweredByNodejs: true },
  { name: 'Laravel', type: 'Framework', poweredByNodejs: false },
  { name: 'React', type: 'Library', poweredByNodejs: true },
  { name: 'Handlebars', type: 'Engine View', poweredByNodejs: true },
  { name: 'Django', type: 'Framework', poweredByNodejs: false },
  { name: 'Docker', type: 'Virtualization', poweredByNodejs: false },
  { name: 'Sequelize', type: 'ORM tool', poweredByNodejs: true },
];
```

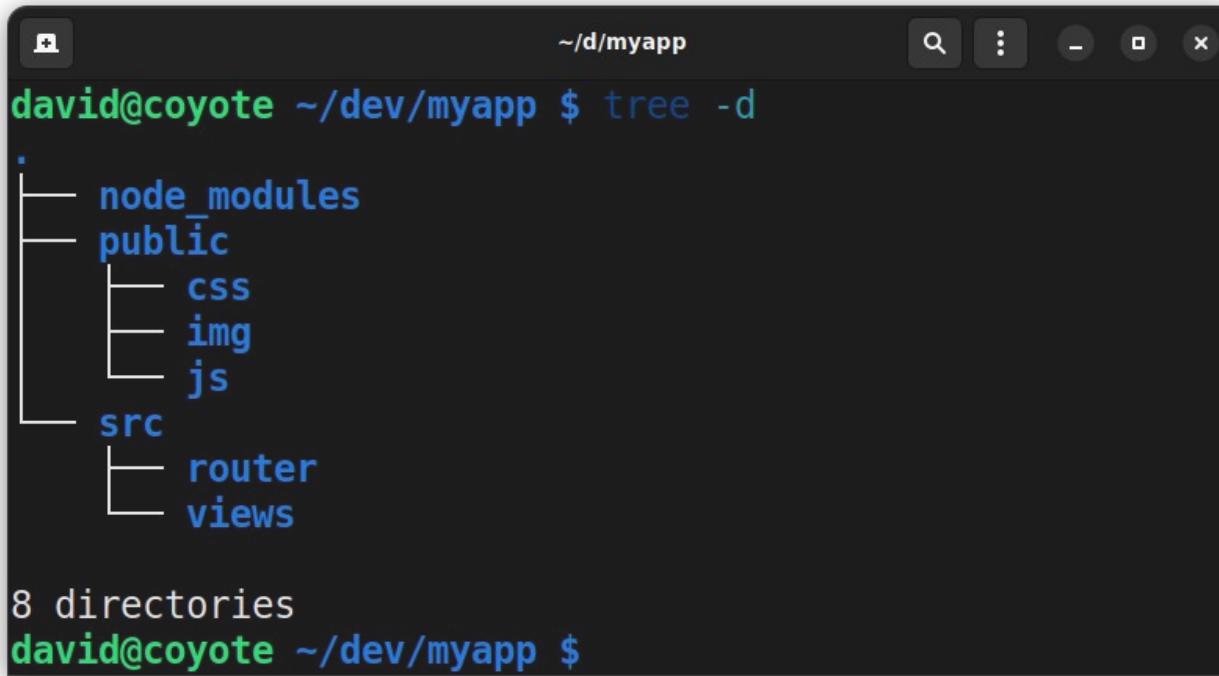


# Movendo o Express para o MVC



# Movendo o Express para o MVC

- Até este momento, nossa aplicação possui os diretórios **src/routes** (com as rotas), **src/views** e **public** (para os assets)
- No entanto, ela ainda não possui nenhum diretório específico para **controladores** e **modelos**

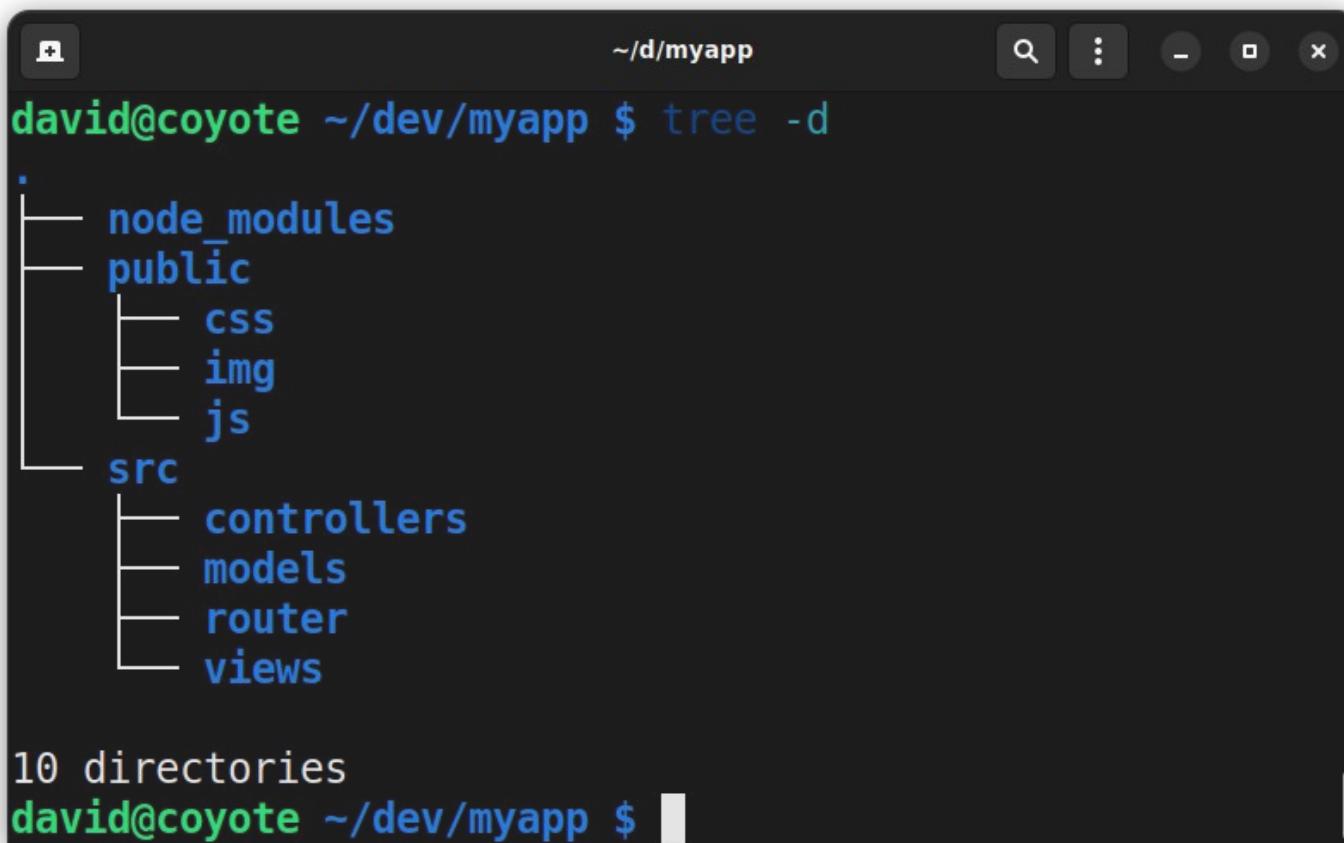


```
david@coyote ~/dev/myapp $ tree -d
.
├── node_modules
└── public
    ├── css
    ├── img
    └── js
└── src
    ├── router
    └── views

8 directories
david@coyote ~/dev/myapp $
```

# Movendo o Express para o MVC

- Criando os diretórios **models** e **controllers**, nossa aplicação passa a ter a seguinte estrutura de diretórios



```
david@coyote ~/dev/myapp $ tree -d
.
├── node_modules
└── public
    ├── css
    ├── img
    └── js
└── src
    ├── controllers
    ├── models
    ├── router
    └── views

10 directories
```

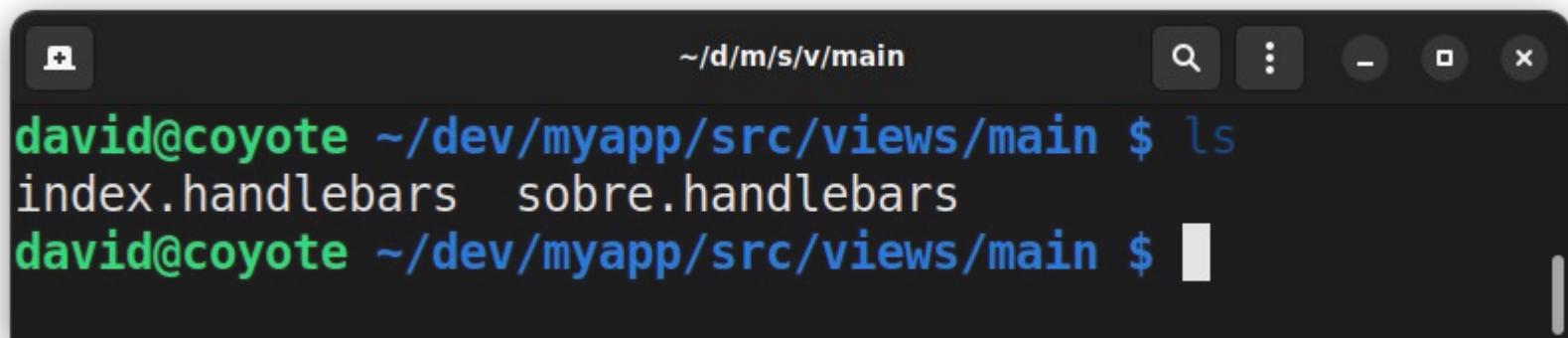
The terminal window shows the command `tree -d` being run in the directory `~/dev/myapp`. The output displays the directory structure:

- Root level:
  - `node_modules`
  - `public`: contains `css`, `img`, and `js`
  - `src`: contains `controllers`, `models`, `router`, and `views`

At the bottom of the terminal, it indicates there are 10 directories.

# Movendo o Express para o MVC

- Uma aplicação pode conter vários controladores, mas por enquanto vamos criar apenas um controlador chamado **main**, que irá responder pelas rotas / e /sobre
- Primando pela organização do código, criamos um diretório **src/views/main** que irá conter as views do controlador main



A screenshot of a terminal window with a dark theme. The title bar shows the path `~/d/m/s/v/main`. The terminal prompt is `david@coyote ~/dev/myapp/src/views/main $`. The user has run the command `ls` to list the contents of the directory, which are `index.handlebars` and `sobre.handlebars`. The terminal window has standard OS X-style controls at the top right.

```
david@coyote ~/dev/myapp/src/views/main $ ls
index.handlebars sobre.handlebars
david@coyote ~/dev/myapp/src/views/main $
```

# Movendo o Express para o MVC

- Uma aplicação pode conter vários controladores, mas por enquanto vamos criar apenas um controlador chamado **main**, que irá responder pelas rotas `/` e `/sobre`.  
Em um primeiro momento, vamos adotar
- Primeiramente, vamos adicionar o código abaixo para as views **src/views/main/index.handlebars** e **src/views/main/sobre.handlebars**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
</head>
<body>
  <div>{{conteudo}}</div>
</body>
</html>
```

# Movendo o Express para o MVC

- O **controlador main** será construído no formato de módulo, e irá conter as funções **index** e **sobre**

```
// Arquivo src/controllers/main.js
const index = (req, res) => {
  const conteudo = 'Página principal da aplicação';
```

Note que um controlador é um conjunto de funções, também conhecidas como **actions**. Cada action é responsável por atender uma dada requisição dos usuários. No nosso exemplo, **index** e **sobre** são **actions** do **controlador main**.

```
const sobre = (req, res) => {
  const conteudo = 'Página sobre a aplicação';
  res.render('main/sobre', {
    conteudo,
    layout: false
  });
};

module.exports = { index, sobre }
```

# Movendo o Express para o MVC

- O próximo passo é preparar no arquivo **src/router** para referenciar as **actions** dos controladores

```
const express = require('express');
const router = express.Router();
const mainController = require('../controllers/main');

router.get('/', mainController.index);
router.get('/sobre', mainController.sobre);

module.exports = router;
```

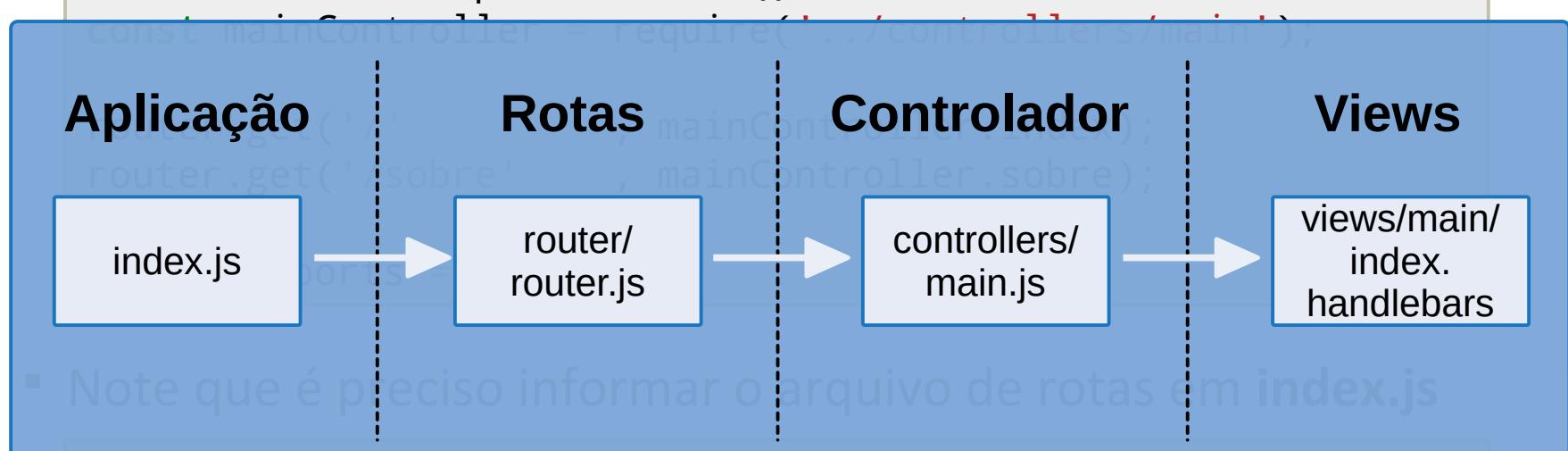
- Note que é preciso informar o arquivo de rotas em **index.js**

```
const router = require("./router/router");
app.use(router);
```

# Movendo o Express para o MVC

- O próximo passo é preparar no arquivo `src/router` para referenciar as **actions** dos controladores

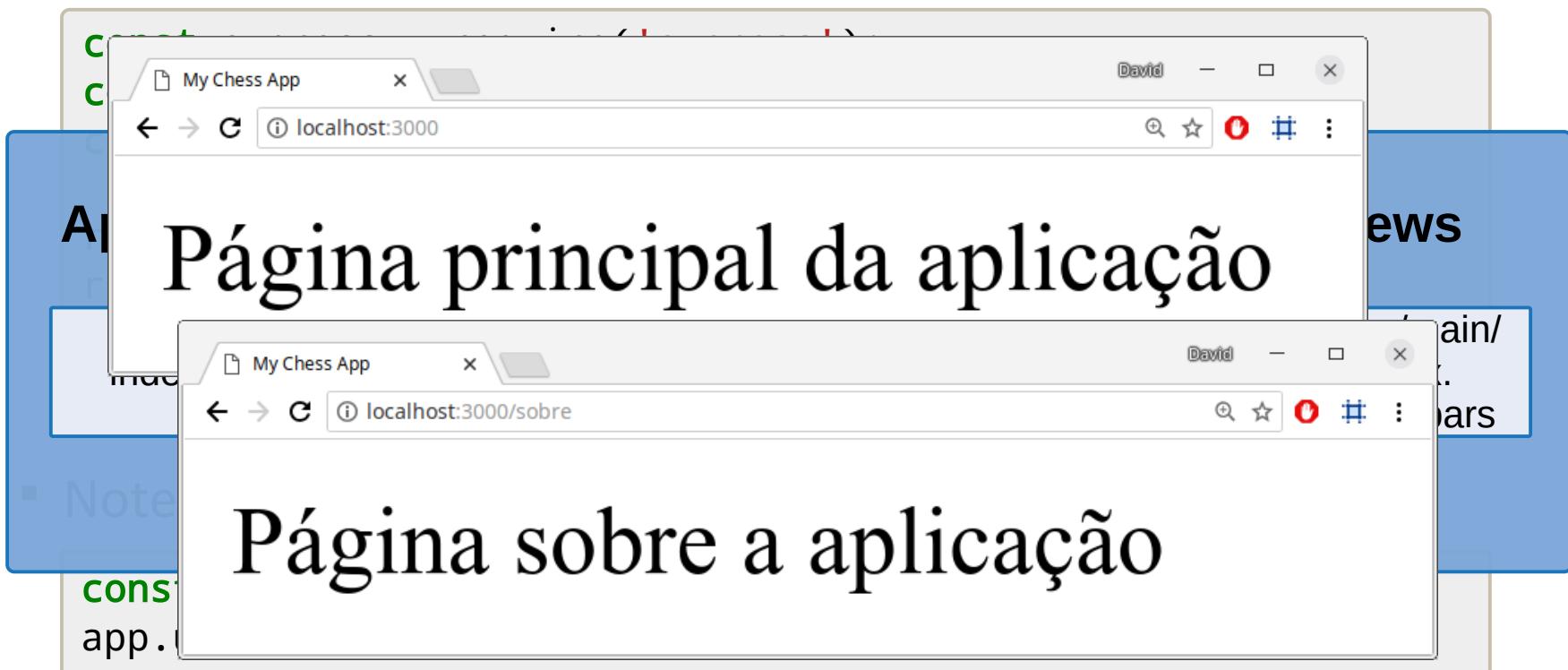
```
const express = require('express');
const router = express.Router();
```



```
const router = require("./router/router");
app.use(router);
```

# Movendo o Express para o MVC

- O próximo passo é preparar no arquivo `src/router` para referenciar as **actions** dos controladores



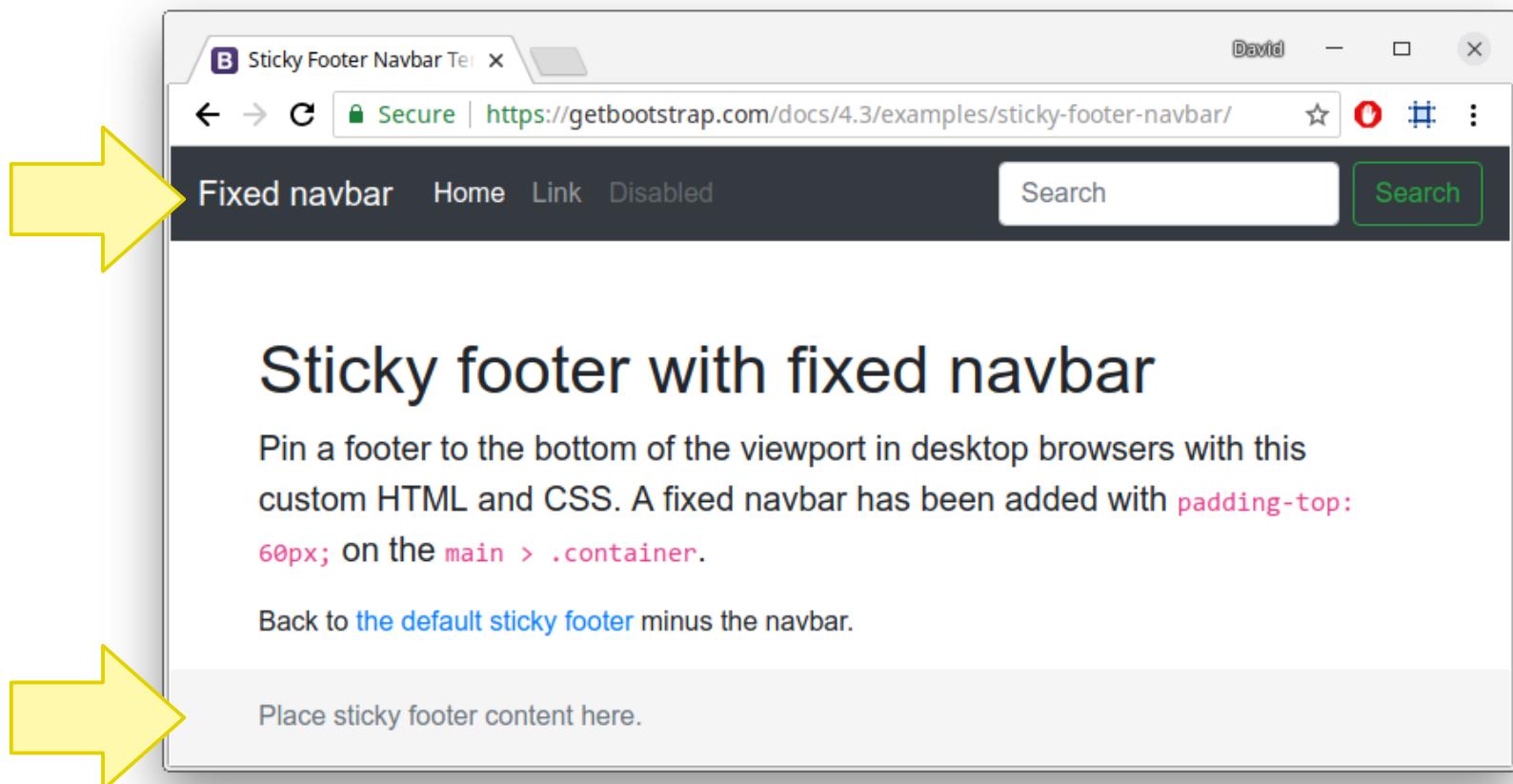
# Exercício 10

- Mova a aplicação para o MVC



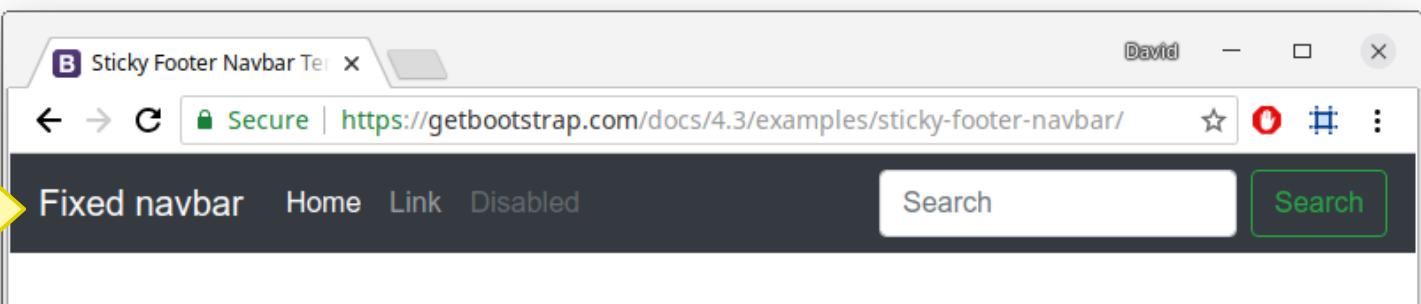
# Layout X Views

- Normalmente, as páginas de uma mesma app possuem muito código HTML em comum, como **menus, headers e footers**



# Layout X Views

- Normalmente, as páginas de uma mesma app possuem muito código HTML em comum, como **menus, headers e footers**



A screenshot of a web browser window titled "Sticky Footer Navbar Test". The address bar shows a secure connection to "https://getbootstrap.com/docs/4.3/examples/sticky-footer-navbar/". The page content includes a fixed navbar with links for "Fixed navbar", "Home", "Link", and "Disabled", along with a search bar. A yellow arrow points from the left towards the top of the page. Below the browser window, a blue callout box contains the text: "Esse conteúdo comum, que está presente na maioria das páginas da aplicação, é o que chamamos de **layout**." A second yellow arrow points from the bottom left towards the bottom of the page.

custom HTML and CSS. A fixed navbar has been added with `padding-top: 60px;` on the `main > .container`.

Back to [the default sticky footer](#) minus the navbar.

Place sticky footer content here.

# Layout X Views

- Por padrão, o **layout** da aplicação deverá ser definido no arquivo **views/layouts/main.handlebars**
- No entanto, é possível mudar o arquivo de layout padrão através da função `app.engine`

```
app.engine('handlebars', handlebars({
  layoutsDir: `__dirname/views/diretorio_layouts`,
  defaultLayout: 'main2',
}))
```

- Também é possível definir layouts específicos para determinadas actions

```
const index = (req, res) => {
  res.render('main/index', { layout: 'main2' });
};
```

# Layout X Views

- Os layouts geralmente contém as tags `<html>`, `<head>`, `<body>`, `<style>` e `<meta>` da aplicacão
- Para informar o local onde o código das views será inserido dentro do layout, usa-se o código `{{{body}}}`
- Em outras palavras, o Express irá substituir o código acima pelo conteúdo da view no momento da geracão da página

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
</head>
<body>
  {{{body}}}
</body>
</html>
```

Local onde as views serão carregadas

# Layout X Views

- Os layouts geralmente contém as tags `<html>`, `<head>`,

```
<!DOCTYPE html>
<html lang="en">
<head>
```

A partir deste momento, as views não precisarão mais ter o código HTML já incluído no layout

- Para informar o local onde o conteúdo da view será inserido, adicione o diretiva `locals` no layout
- No nosso exemplo, o código das views passa a ser somente

```
<div>{{conteúdo}}</div>
```

código `{{{body}}}`

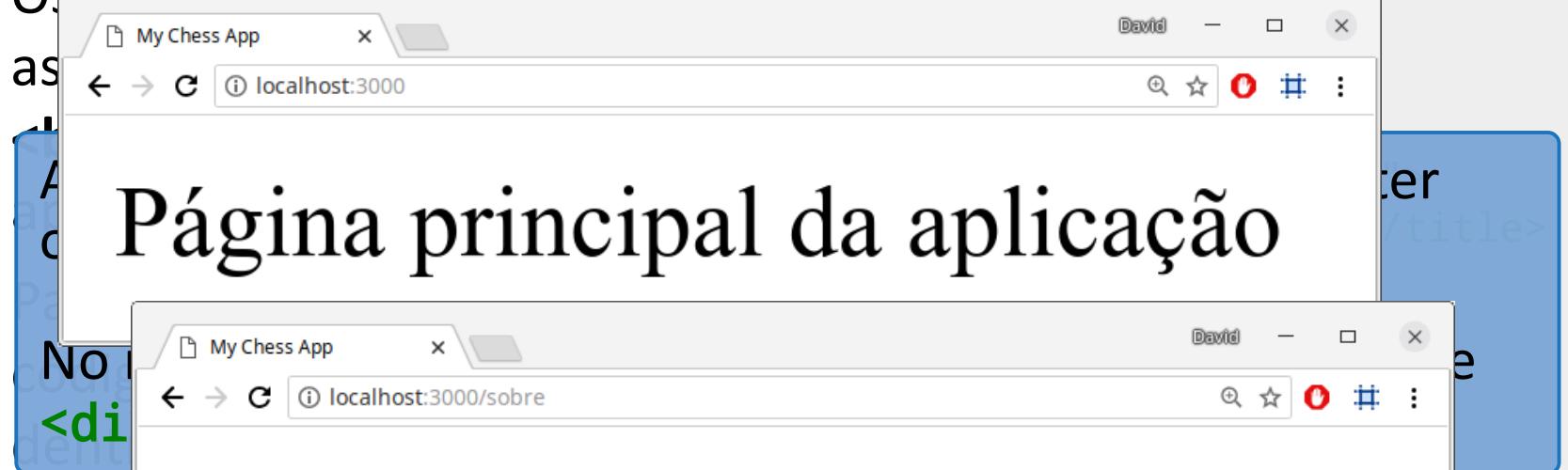
```
</body>
</html>
```

- Em outras palavras, o Express irá substituir o código acima pelo conteúdo da view no momento da geração da página

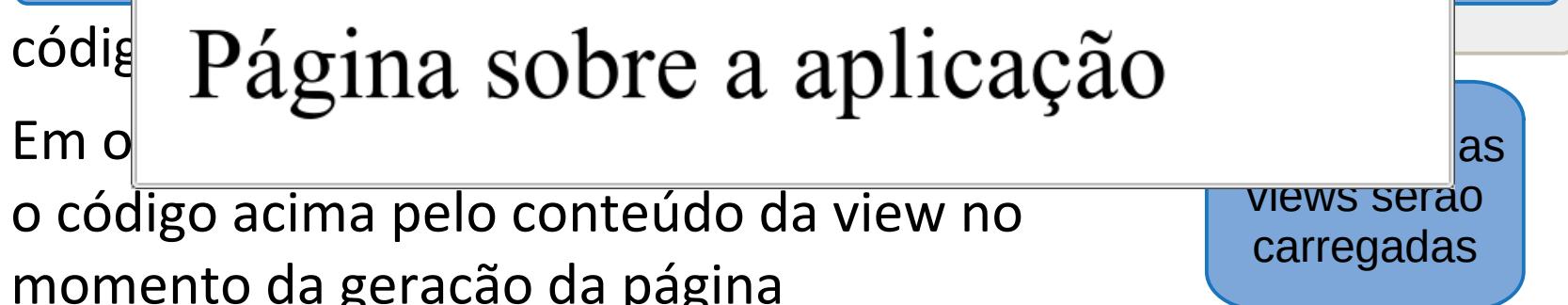
Local onde as views serão carregadas

# Layout X Views

- Os layouts geralmente contêm



- as views



- Em outras palavras, o código acima pelo conteúdo da view no momento da geração da página

views serão carregadas

# Exercício 11

- Até então cada view possui seu próprio template (<html><header><body>). Mova esse conteúdo para um template separado, de forma que cada view contenha só o conteúdo que é específico da página que ela precisa renderizar.

github

**ExptS**

express **JS**

# Assets

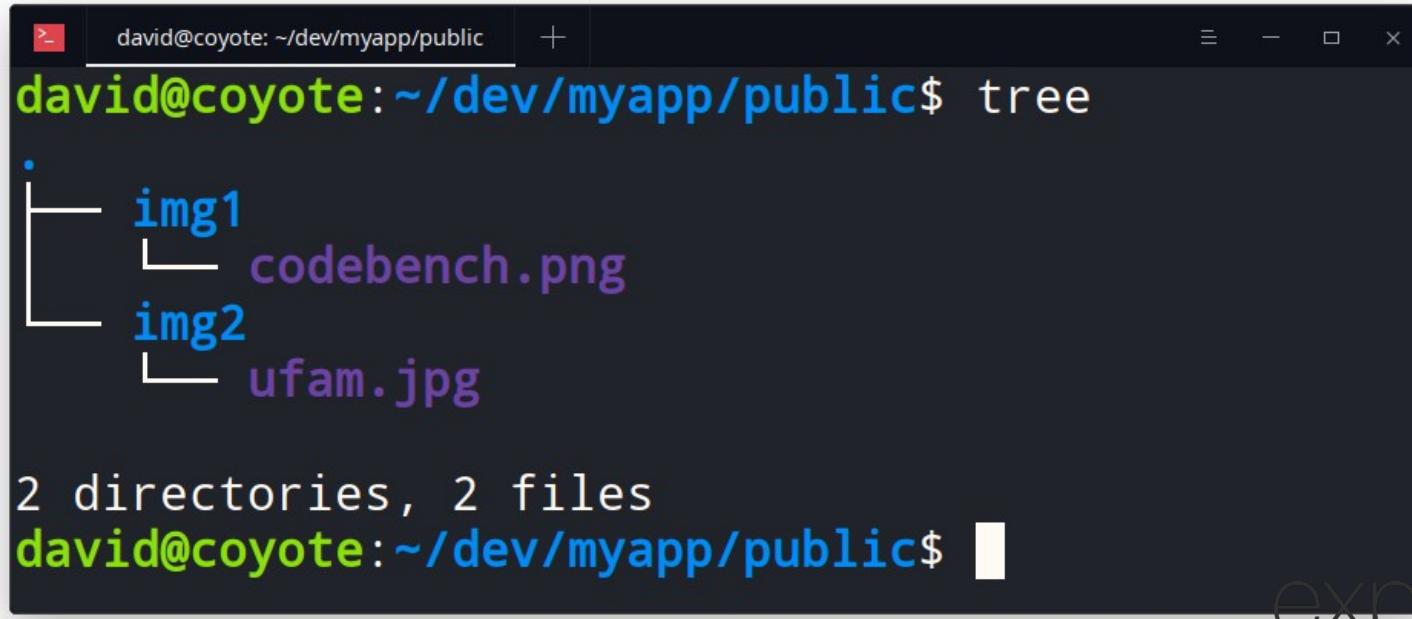
- Conforme já vimos, os assets são **arquivos estáticos** (js, css, img, etc) que são acessíveis pelos usuários da aplicação
- Para disponibilizar esses arquivos na aplicação, usamos o **middleware de arquivos estáticos** do Express
- Por exemplo, para disponibilizar todas as imagens do diretório **/public/img**, usamos o código abaixo:

```
app.use('/img', [
  express.static(`__dirname/public/img`)
]);
```

# Assets

- Também é possível disponibilizar imagens de 2 ou mais diretórios através de um único path na URL

```
app.use('/img', [
  express.static(`__dirname/public/img1`)
  express.static(`__dirname/public/img2`)
]);
```



A screenshot of a terminal window titled "david@coyote: ~/dev/myapp/public". The window shows the command "tree" being run, which displays the following directory structure:

```
david@coyote:~/dev/myapp/public$ tree
.
├── img1
│   └── codebench.png
└── img2
    └── ufam.jpg

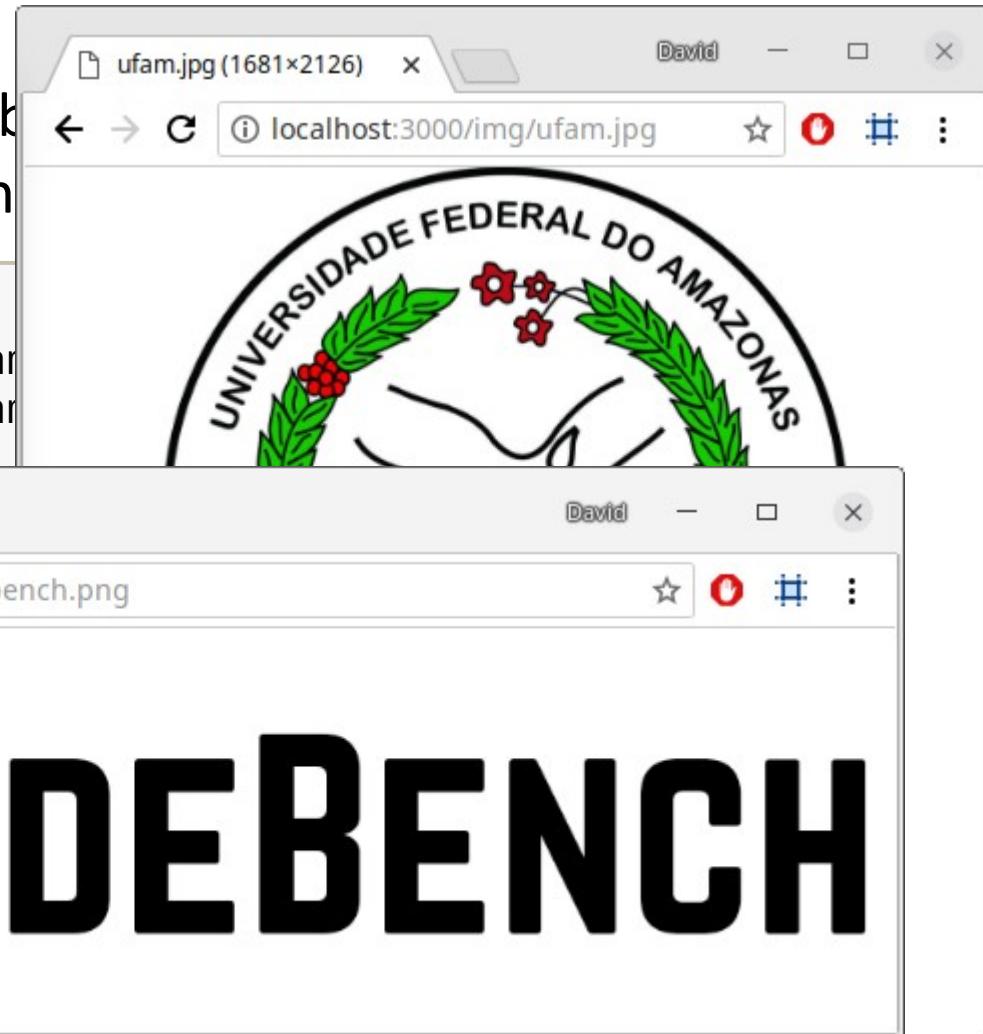
2 directories, 2 files
```

The terminal also shows the command "tree" being run again at the bottom.

# Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [
  express.static(`${__dirname}/img/ufam.jpg'),
  express.static(`${__dirname}/img/codebench.png`)
]);
```



2 directories, 2 files

david@coyote:~/dev/myapp/public\$

express JS

# Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [  
  express.static(`${__dirname}/img`),  
  express.static('public')  
]);
```



Se houver duas imagens com o mesmo nome nos dois diretórios informados, o Express irá retornar a imagem que encontrar primeiro, seguindo a sequência de diretórios informada

dav]\$



# CODEBENCH

2 directories, 2 files

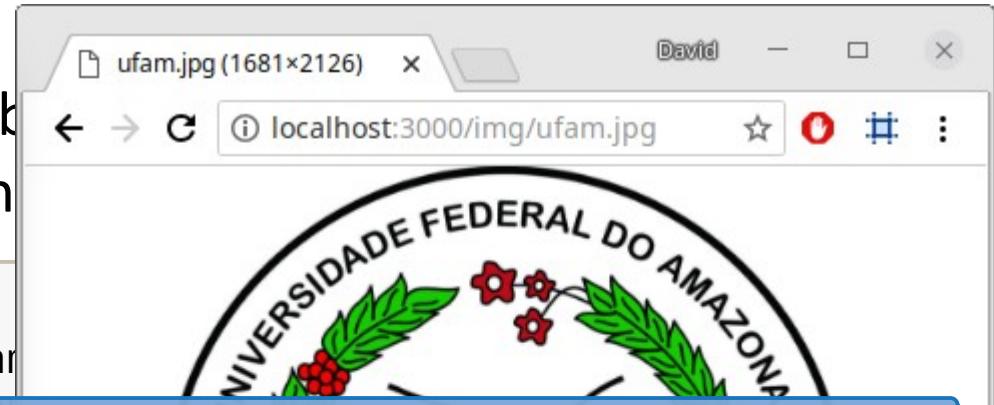
david@coyote:~/dev/myapp/public\$

express JS

# Assets

- Também é possível disponibilizar arquivos de diretórios através de um único endpoint

```
app.use('/img', [  
  express.static(`${__dirname}/img`),  
  express.static('public')])
```



Se houver duas imagens com o mesmo nome nos dois diretórios

Conforme veremos mais tarde, a inclusão de **arquivos JS** segue a mesma sequência de passos

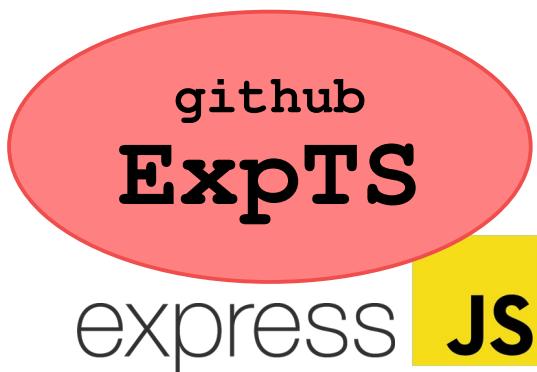


```
david@coyote:~/dev/myapp/public$
```

express **JS**

# Exercício 12

- Na página sobre, inclua uma imagem (além do texto)



express  JS

# Assets

- A inclusão de **arquivos CSS** pode ser feita através do middleware de arquivos estáticos do Express
- No entanto, em se tratando de arquivos de estilos, também é importante o uso de compiladores de estilos, tal como o **SASS**





- O SASS – Syntactically Awesome Stylesheets – é uma linguagem de **folha de estilos dinâmica** usada para simplificar a criação de código CSS
- Ela permite a criação de código CSS a partir de variáveis, operações, funções e mixins
- Instalação do SASS no sistema Linux:

```
$ sudo npm install sass -g
```

- Para compilar códigos SASS em CSS, usamos:

```
$ sass estilos.scss estilos.css
```

# Variáveis SASS

- SASS permite a definição de **variáveis** que podem ser utilizadas por toda a folha de estilos

SCSS

```
$color: #4D926F;  
  
#header {  
  color: $color;  
}  
  
h2 {  
  color: $color;  
}
```

CSS Compilado

```
#header {  
  color: #4D926F;  
}  
  
h2 {  
  color: #4D926F;  
}
```



~/d/m/p/scss



david@coyote ~/dev/myapp/public/scss \$ cat estilos.scss

```
$color: #4D926F;
```

- SASS permite a definição de **variáveis** que podem ser utilizadas por toda a folha de estilos
- ```
#header {  
    color: $color;  
}
```

SCSS

CSS Compilado

```
$color: #4D926F;  
    color: $color;  
#header {  
    color: $color;  
}  
#header {  
    color: #4D926F;  
}  
h2 {  
    color: $color;  
}  
h2 {  
    color: #4D926F;  
}
```

```
#header {  
    color: #4D926F;  
}  
h2 {  
    color: #4D926F;  
}
```

david@coyote ~/dev/myapp/public/scss \$

# Mixins *Sass*

- Os Mixins permitem a definição de estilos que podem ser reutilizados em toda a folha de estilo

## SCSS

```
@mixin bordered ($wth) {  
    border-top: dotted $wth black;  
    border-bottom: solid $wth black;  
}  
  
#menu a {  
    color: #111;  
    @include bordered(2px);  
}  
  
.post a {  
    @include bordered(1px);  
}
```

## CSS Compilado

```
#menu a {  
    color: #111;  
    border-top: dotted 2px black;  
    border-bottom: solid 2px black;  
}  
  
.post a {  
    border-top: dotted 1px black;  
    border-bottom: solid 1px black;  
}
```

# Regras Aninhadas



- Com o SCSS é possível aninhar seletores dentro de outros seletores para especificar a ancestralidade dos seletores

SCSS

```
#header {  
  color: black;  
  .navigation {  
    font-size: 12px;  
  }  
  .logo {  
    width: 300px;  
  }  
}
```

CSS Compilado

```
#header {  
  color: black;  
}  
#header .navigation {  
  font-size: 12px;  
}  
#header .logo {  
  width: 300px;  
}
```

# Operações



- O SCSS também conta com operações para somar, subtrair, dividir e multiplicar valores de propriedades

## SCSS

```
$the-border: 1px;  
  
#header {  
    border-left: $the-border;  
    border-right: $the-border * 2;  
}
```

## CSS Compilado

```
#header {  
    border-left: 1px;  
    border-right: 2px;  
}
```

# Outras Funções



- Além das funcionalidades mostradas, o SASS possui muitas outras vantagens sobre o CSS

## Comentários

```
/* Isto é um bloco  
de comentários! */  
$var: red;  
  
// Comentário em uma linha  
$var: white;
```

## Escopo

```
$var: red;  
#page {  
    $var: white;  
    #header {  
        color: $var; // white  
    }  
}
```

## Importação de estilos

```
@import "library"; // library.scss  
@import "typo.css";
```

## Manipulação de cores

```
lighten($color, 10%);  
darken($color, 10%);  
  
saturate($color, 10%);  
desaturate($color, 10%);  
  
opacity($color, 0.5);  
transparentize($color, 0.5);
```

# Adicionando SASS no Express

- Para usarmos o compilador SASS no Express, podemos recorrer ao pacote **node-sass-middleware**

```
$ npm install node-sass-middleware
```

- A configuração desse middleware é simples

```
const sass = require('node-sass-middleware');

app.use(sass({
  src: `${__dirname}/../public/scss`,
  dest: `${__dirname}/../public/css`,
  outputStyle: "compressed",
  prefix: "/css",
}));

app.use("/css", express.static(`${__dirname}/../public/css`));
```

# Adicionando SASS no Express

- Para usarmos o compilador SASS no Express, podemos recorrer ao pacote **node-sass-middleware**

```
~/.nvm/versions/node/v10.15.3/bin/npx -v  
david@coyote ~/dev/myapp $ npx -v  
1.1.0  
~/.nvm/versions/node/v10.15.3/bin/npx -v  
david@coyote ~/dev/myapp $ cat public/scss/estilos.scss  
$color: #4d926f;  
  
const sass = require('node-sass-middleware');  
h2 {  
  color: $color;  
}  
src: `__dirname/./public/scss`,  
dest: `__dirname/./public/css`  
outputStyle: "Compressed",  
prefix: "/css"  
});  
  
app.use("/css", express.static(`__dirname/./public/css`));  
  
david@coyote ~/dev/myapp $ cat public/scss/main.scss  
@import "estilos.scss";  
david@coyote ~/dev/myapp $ 
```

```
~/d/myapp
david@coyote ~/dev/myapp $ cat src/views/layouts/main.handlebars
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>My App</title>
    <link rel="stylesheet" href="/css/main.css" />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
david@coyote ~/dev/myapp $ cat public/scss/estilos.scss
<h2>{{conteudo}}</h2>
david@coyote ~/dev/myapp $
david@coyote ~/dev/myapp $ cat public/scss/main.scss
@import "estilos.scss";
david@coyote ~/dev/myapp $ 
app.use("/css", express.static(`$__dirname__`/..`/public/css`));
```

# express

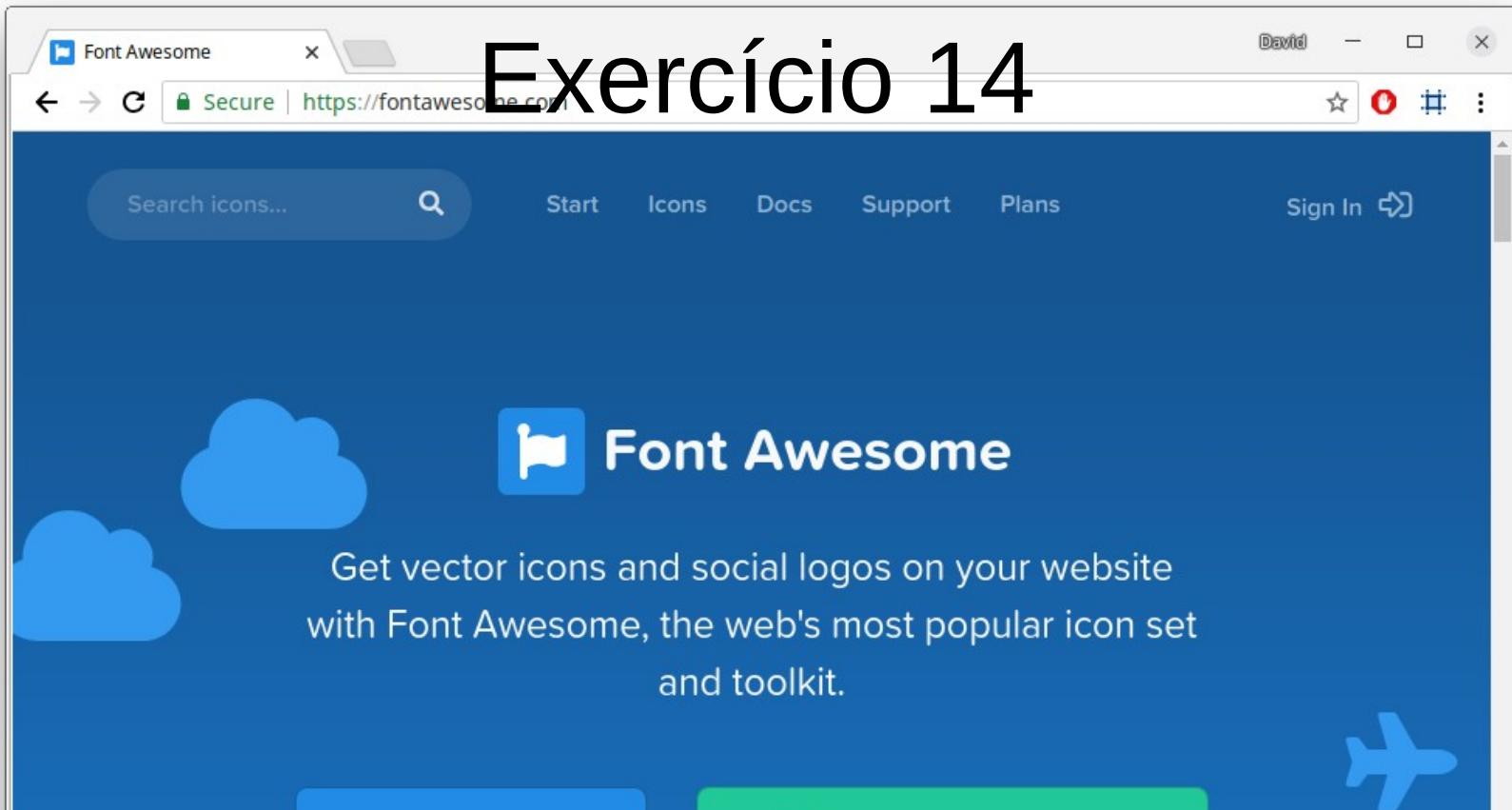
```
david@coyote ~/dev/myapp $ cat src/views/layouts/main.handlebars
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <title>My App</title>
    <link rel="stylesheet" href="/css/main.css" />
  </head>
  <body>
    {{{body}}}
  </body>
</html>
david@coyote ~/dev/myapp $ cat src/css/main.css
h2{color:#4d926f}

david@coyote ~/dev/myapp $ node ./bin/www
david@coyote ~/dev/myapp $ curl localhost:3000
david@coyote ~/dev/myapp $ curl localhost:3000/css/main.css
david@coyote ~/dev/myapp $ curl localhost:3000
```

# Exercício 13

- Adicione um SCSS para formatar alguma coisa

githubExptSexpress JS



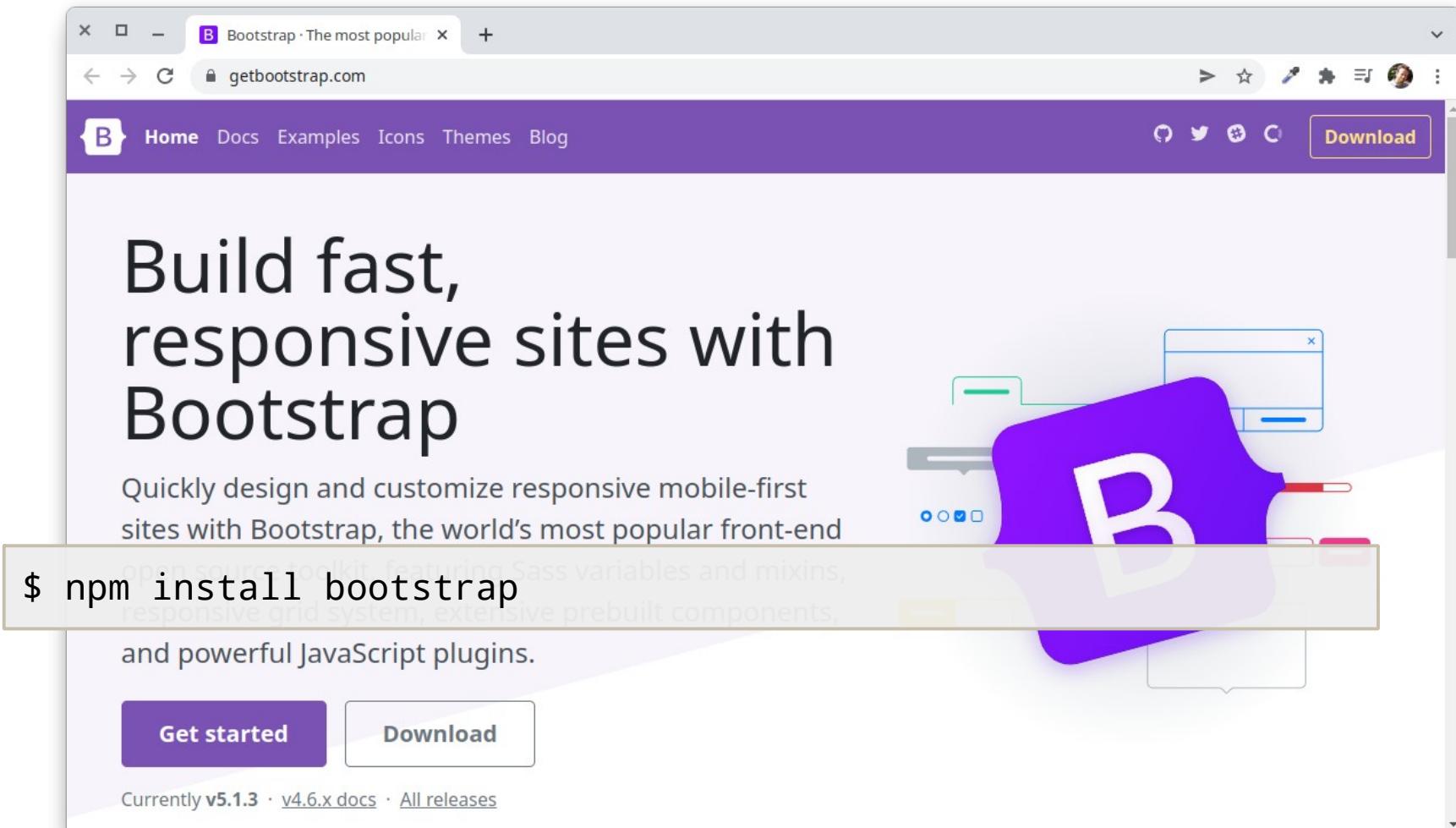
**Exercício 2:** Instalar o pacote do projeto **Font Awesome**, e disponibilizar os ícones para uso em sua aplicação. O nome do pacote NPM é **@fortawesome/fontawesome-free**. Siga as orientações disponíveis na [página oficial](#).

github  
**Game**

# Adicionando o Bootstrap



- O próximo passo é adicionar o **Bootstrap** na sua aplicação



The screenshot shows the official Bootstrap website at [getbootstrap.com](https://getbootstrap.com). The page features a purple header with the Bootstrap logo and navigation links for Home, Docs, Examples, Icons, Themes, and Blog. A large, bold headline reads "Build fast, responsive sites with Bootstrap". Below it, a sub-headline says "Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end". A command-line interface (CLI) command, "\$ npm install bootstrap", is highlighted in a purple box. At the bottom, there are two buttons: "Get started" (purple) and "Download" (white). The footer indicates the current version is v5.1.3.

Bootstrap · The most popular front-end framework for building responsive, mobile-first projects.

Build fast, responsive sites with Bootstrap

Quickly design and customize responsive mobile-first sites with Bootstrap, the world's most popular front-end.

```
$ npm install bootstrap
```

and powerful JavaScript plugins.

Get started   Download

Currently v5.1.3 · [v4.6.x docs](#) · [All releases](#)

# Adicionando o Bootstrap



- Após a instalação, é necessário incluir os paths dos arquivos JS dos três pacotes no **middleware de arquivos estáticos**

```
app.use('/js', [
  express.static(`__dirname`/../public/js`),
  express.static(`__dirname`../node_modules/bootstrap/dist/js/`)
]);
```

- Além disso, é preciso incluir o arquivo **bootstrap.scss** no arquivo **/public/scss/main.scss**

```
@import "../node_modules/bootstrap/scss/bootstrap.scss";
```

# Adicionando o Bootstrap



- Após isso, basta incluir o arquivo **bootstrap.bundle.js** no layout principal da aplicação

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>My Chess App</title>
  <link rel="stylesheet" href="/css/main.css">
</head>
<body>
  {{>body}}
  <script src="/js/bootstrap.bundle.js"></script>
</body>
</html>
```

# Exercício 15

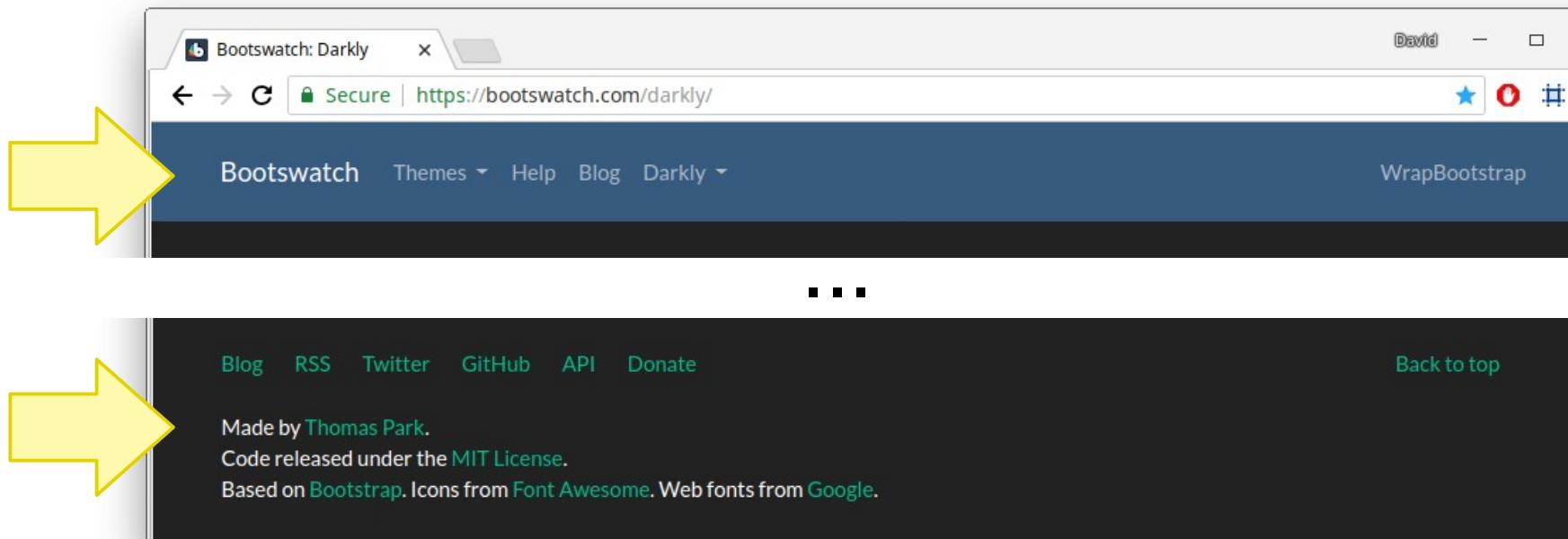
The screenshot shows a web browser window titled "My App" at "localhost:3334/ui". The page features a dark blue header with the "Bootswatch" logo and links for "Themes", "Help", "Blog", and "Darkly". On the right side of the header are "GitHub" and "Twitter" links. Below the header, the main content area has a dark background. The title "Darkly" is displayed in large white font, followed by the subtitle "Flatly in night mode". A section titled "Navbars" is shown with three examples of dark-themed navigation bars. Each navbar includes links for "Home", "Features", "Pricing", "About", and "Dropdown", along with a search bar. The entire content area is highlighted with a light blue rounded rectangle.

Exercício 3: O site **Bootswatch** – <https://bootswatch.com/> – possui um conjunto de temas baseados no **Bootstrap**. Vá até esse site, escolha um dos temas disponíveis e instale ele em sua aplicação.

github  
Game

# Dicas para o Exercício 3

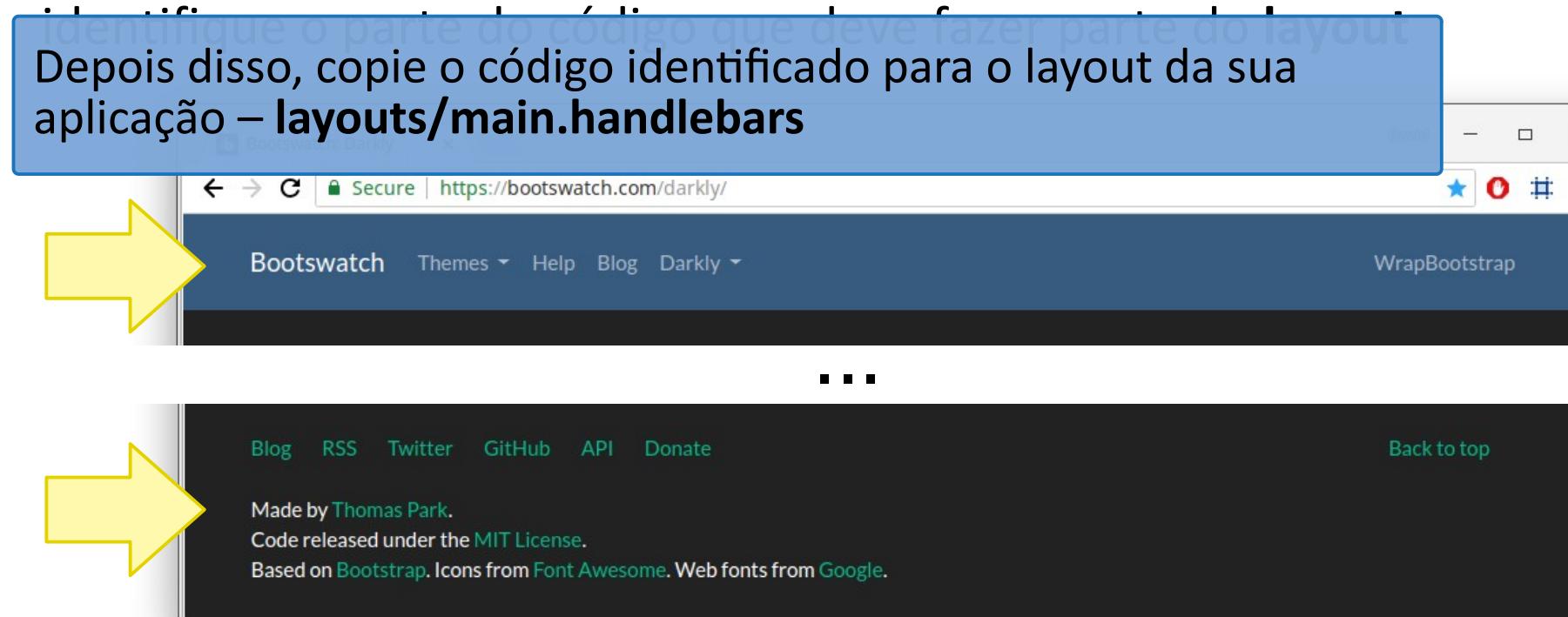
- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e identifique o parte do código que deve fazer parte do **layout**



# Dicas para o Exercício 3

- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e

Depois disso, copie o código identificado para o layout da sua aplicação – `layouts/main.handlebars`



# Dicas para o Exercício 3

- Copie os arquivos `_bootswatch.scss` e `_variables.scss` para `/public/scss` e faça seus **imports** no arquivo `main.scss`
- Acesse o código-fonte do preview do tema escolhido, e

Identifique a parte do código que deve fazer parte do layout.

Depois disso, copie o código identificado para o layout da sua aplicação.

a) Adicionalmente, crie uma **action** chamada **ui** e copie em sua **view** todo o código fonte do preview que não faça parte do layout.

O objetivo dessa view é conter um conjunto de **elementos de interface** que você poderá usar na sua aplicação (Ctrl+c Ctrl+v)

...

