



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

METEOR

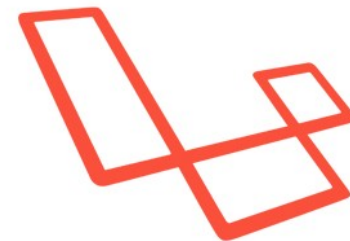
express

sails



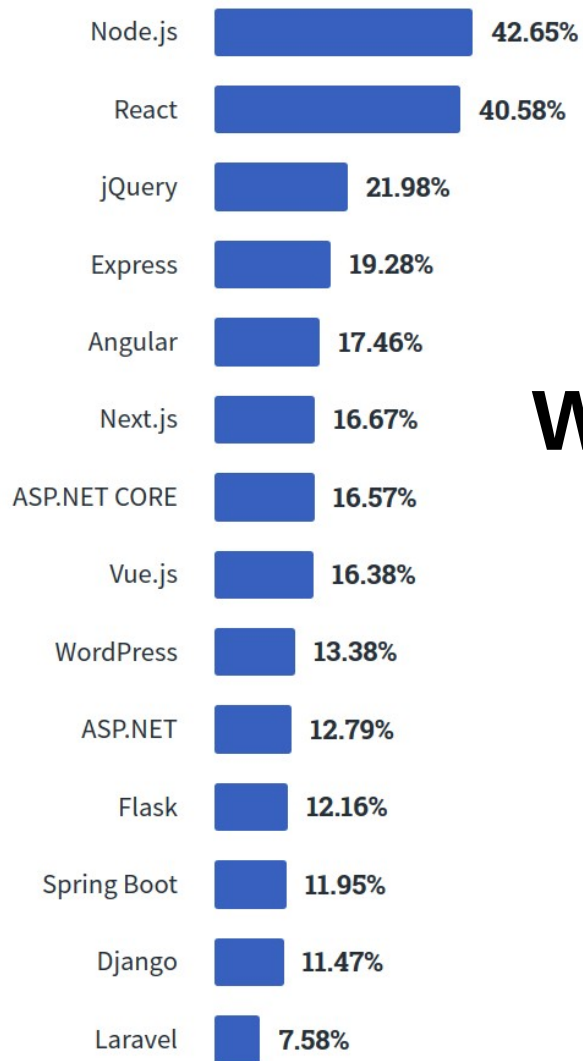
yii

django



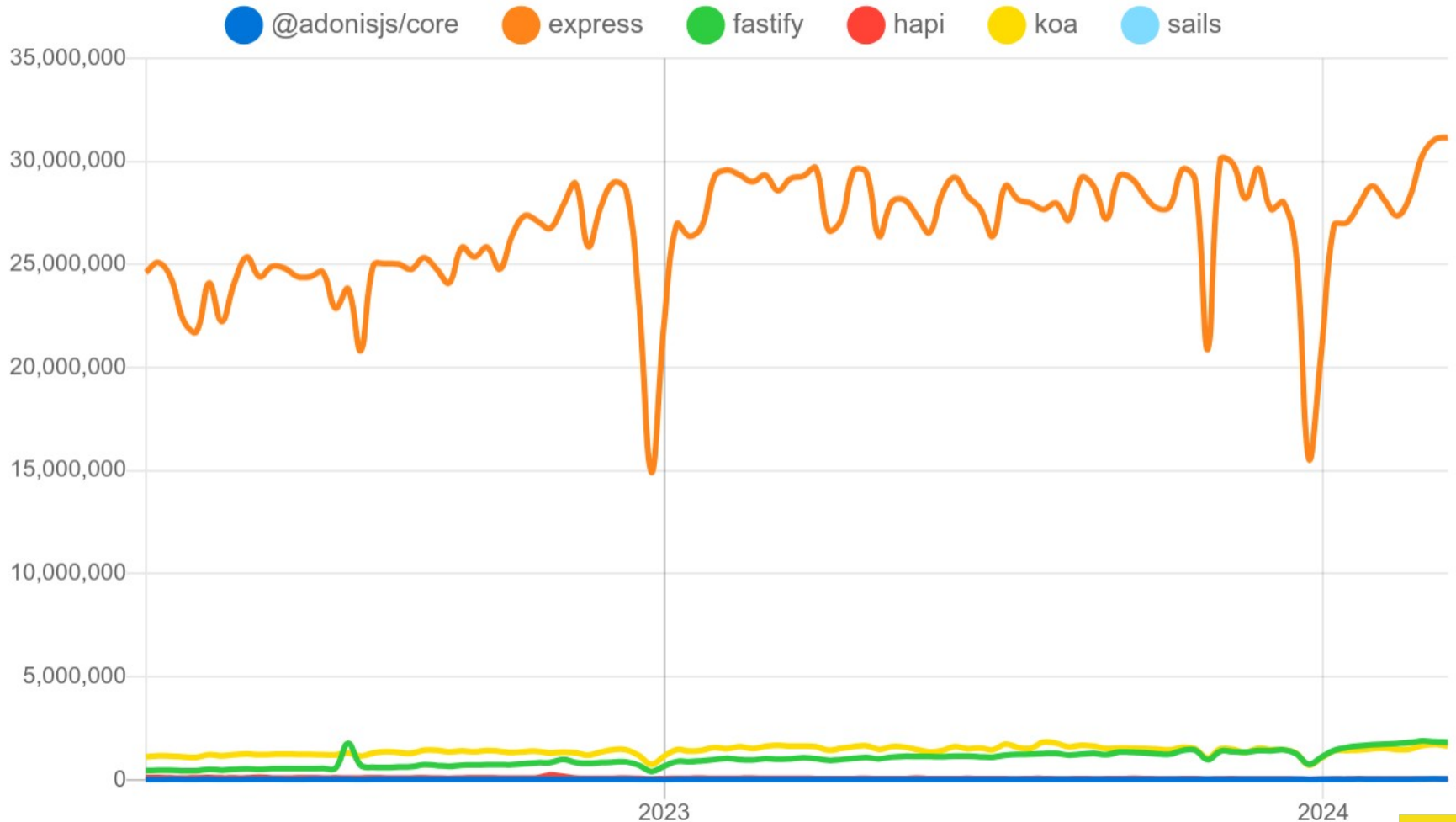
laravel

StackOverflow Surveys



Web Frameworks 2023

NPM Trends



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 ap x +


← → ↻ expressjs.com 🔍 ☆ ⛶ ⌵ 📄 👤 ⋮

Black Lives Matter.
Support the Equal Justice Initiative.

Express 🔍 search Home Getting started Guide API reference Advanced topics Resources

Express 4.18.3
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

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web and mobile applications.

APIs

With a myriad of HTTP utility methods and middleware at your disposal, creating a robust API is quick and easy.

Performance

Express provides a thin layer of fundamental web application features, without obscuring Node.js features that you know and love.

Frameworks

Many [popular frameworks](#) are based on Express.

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

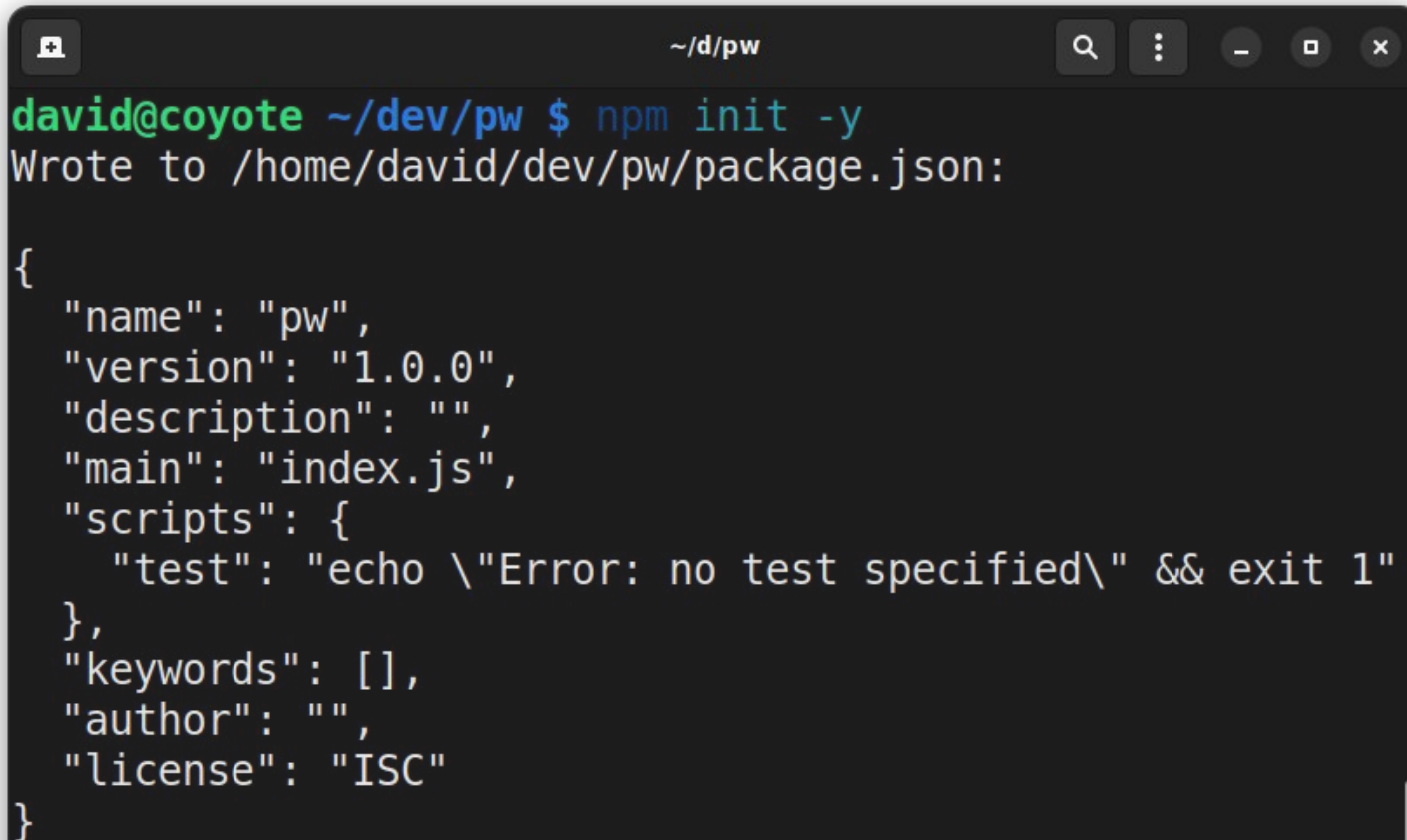
- O primeiro passo no desenvolvimento de uma aplicação node.js é a criação de um arquivo chamado **package.json**
 - A função desse arquivo é armazenar vários metadados sobre a aplicação, incluindo suas dependências

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

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

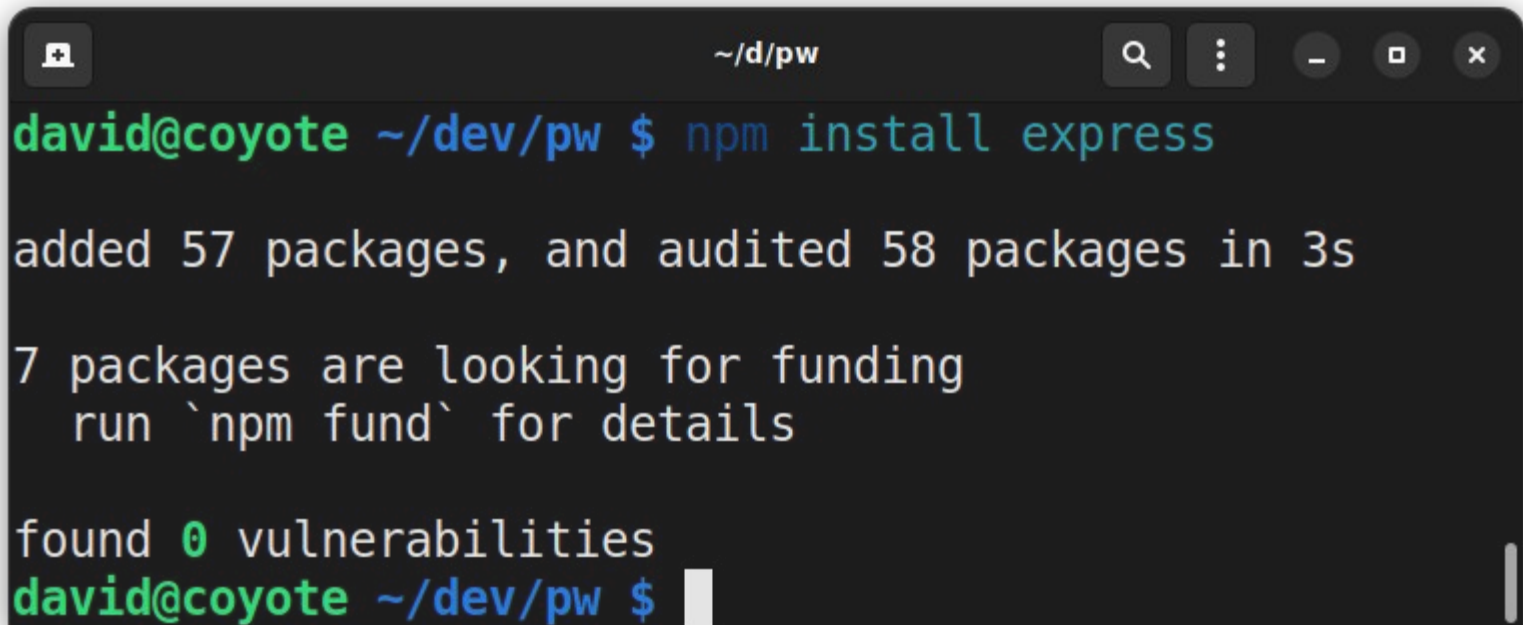


```
david@coyote ~/dev/pw $ npm init -y
Wrote to /home/david/dev/pw/package.json:

{
  "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

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

```
~ /d/pw $ cat package.json
```

```
{
  "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"
  }
}
```

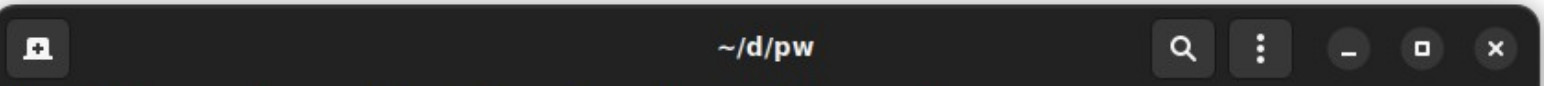
```
~ /d/pw $
```

```
~ /d/pw $
```

Adicionando o Express no Projeto

- Para incluirmos o framework express no projeto, basta

executar



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 aplicação é dependente do Express versão 4.*.*

Alguns módulos podem ser prefixados com um ~, indicando que eles devem ser atualizados apenas 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",
  "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 || 3333;
```

```
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 indicada
em **.env**

Hello World em Express

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

```
node src/index.js ~/d/express
```

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

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

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

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

app.listen(PORT, () => {
  console.log(`Server is running on port ${PORT}`);
});
```

localhost:3333

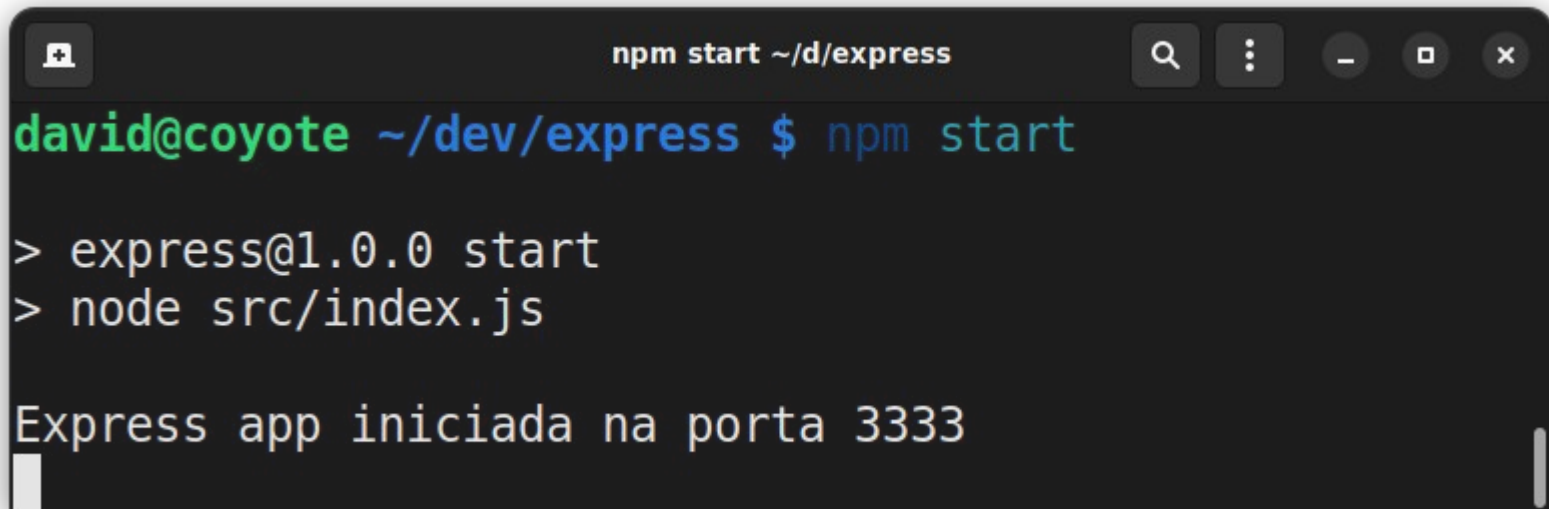
Hello World!

Realiza o servidor HTTP na porta indicada no .env

Hello World em Express

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

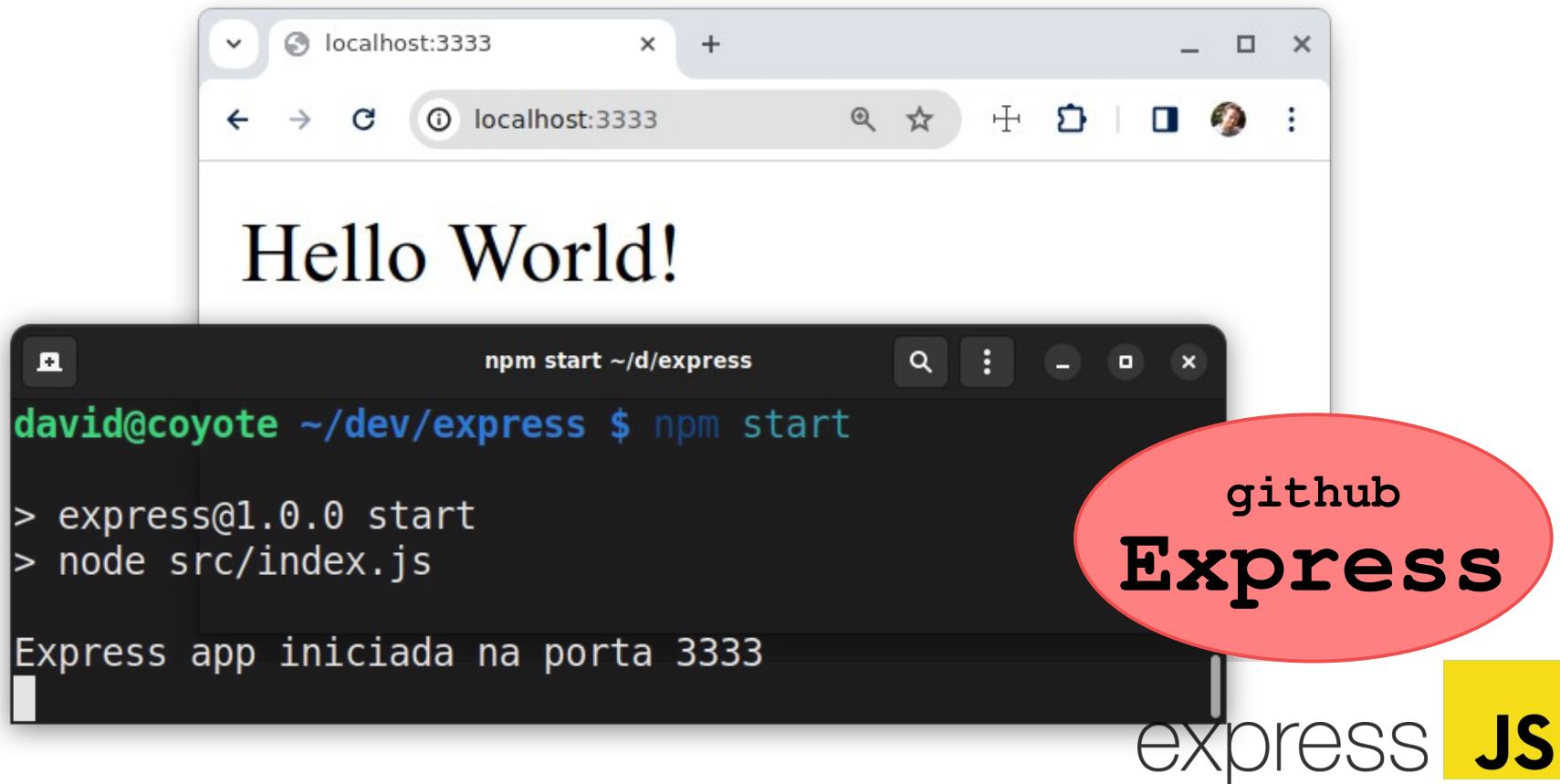
```
"scripts": {  
  "start": "node src/index.js"  
},
```



```
npm start ~/d/express  
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



Instalação do TypeScript

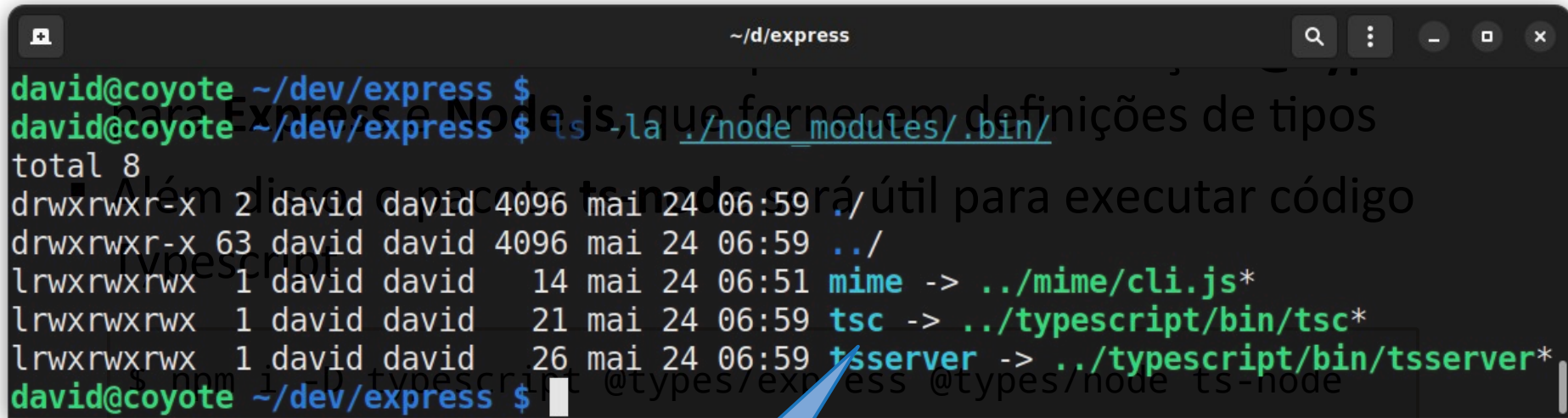
- Após a instalação do Express, vamos instalar o **TypeScript** como dependência de desenvolvimento da aplicação
- Também serão instalados os pacotes de declaração **@types** para **Express** e **Node.js**, que fornecem definições de tipos
- Além disso, o pacote **ts-node** será útil para executar código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

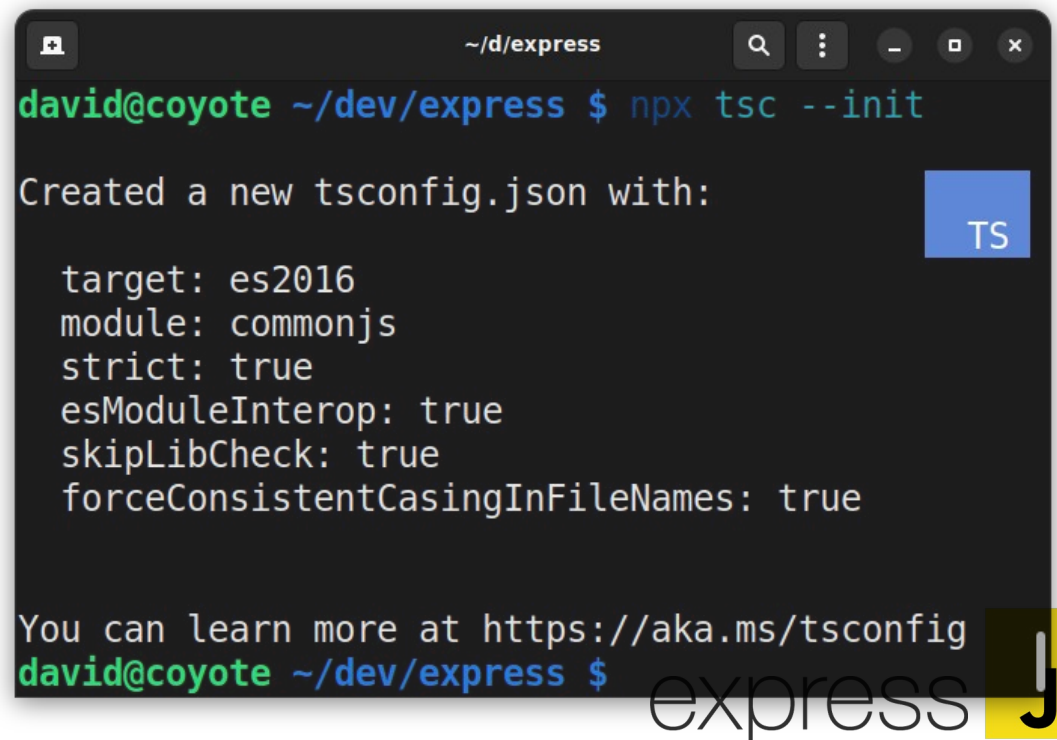
Compilador
TypeScript

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



```
~/d/express
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 $
```

The screenshot shows a terminal window with the command `npx tsc --init` executed. The output indicates that a new `tsconfig.json` file has been created with the following configuration: `target: es2016`, `module: commonjs`, `strict: true`, `esModuleInterop: true`, `skipLibCheck: true`, and `forceConsistentCasingInFileNames: true`. A link to learn more about `tsconfig` is provided at the bottom. The terminal window has a dark theme and a blue tab labeled 'TS'.

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 o

Hello World!

ts e

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

Express app iniciada na porta `${PORT}`.

Hello World em Express/TypeScript

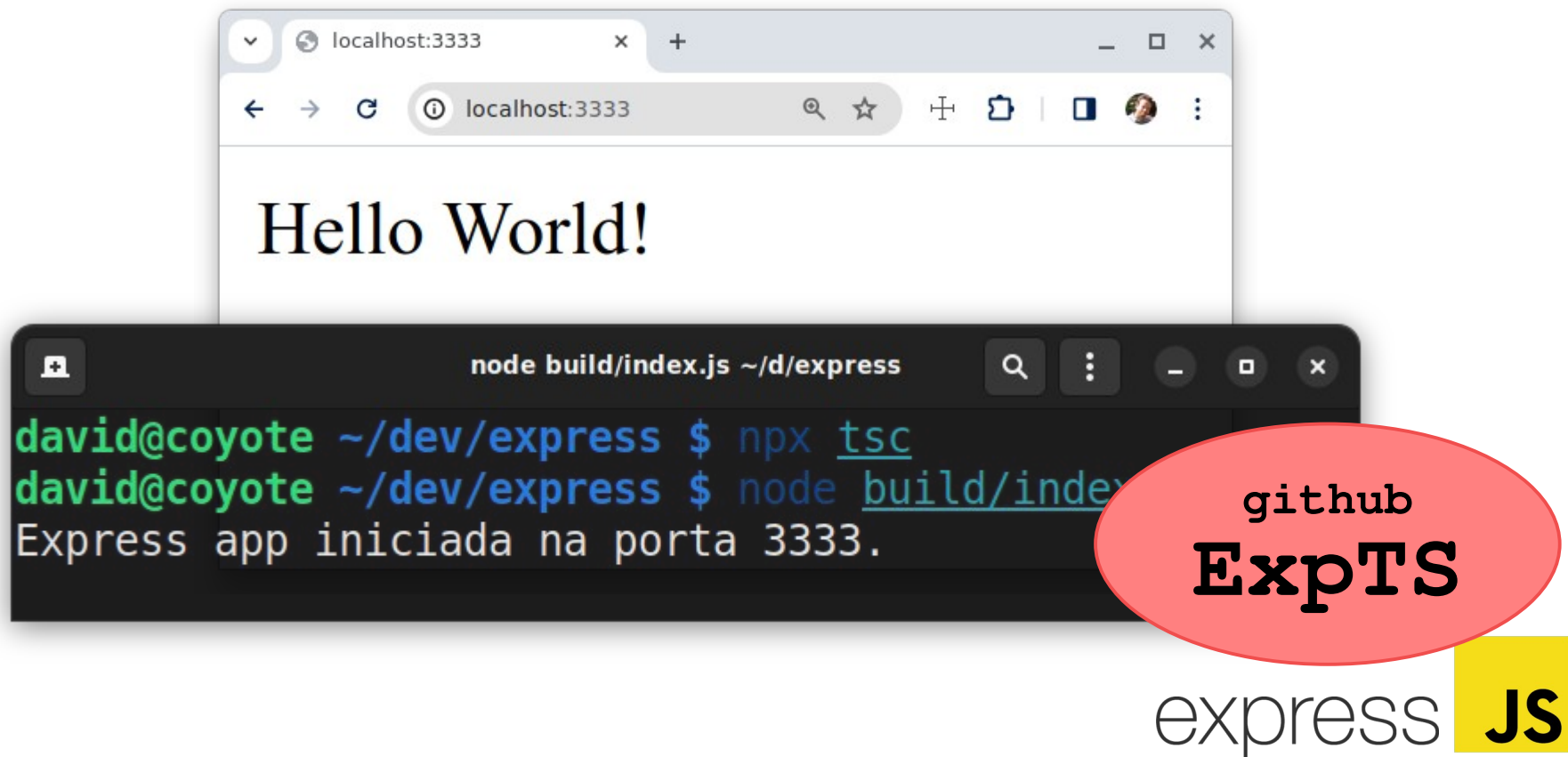
- Podemos inicializar a aplicação em ambiente de desenvolvimento usando o **nodemon**, de forma que os scripts em **package.json** ficam assim:

```
"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.



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");  
})
```

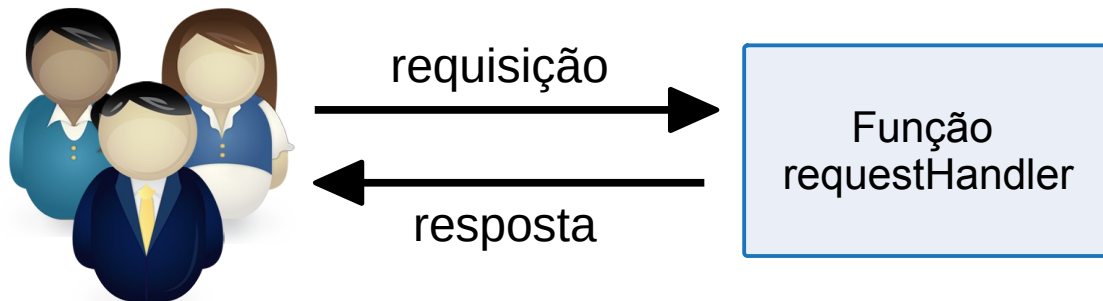
Os Fundamentos do Express

- 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

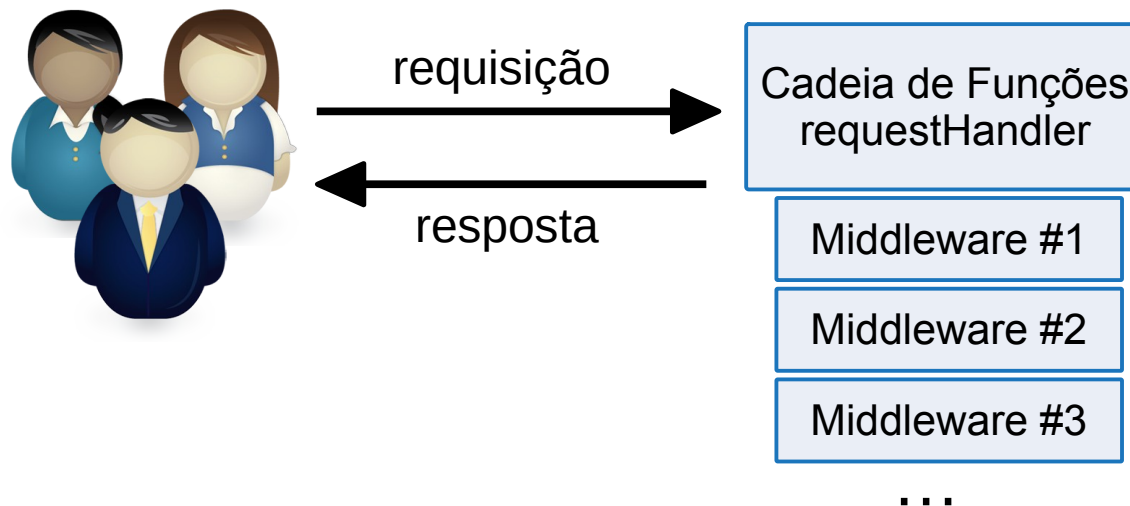
- Usando Node.js puro, usamos o callback de **http.createServer** para responder as requisições dos usuários

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



Middleware

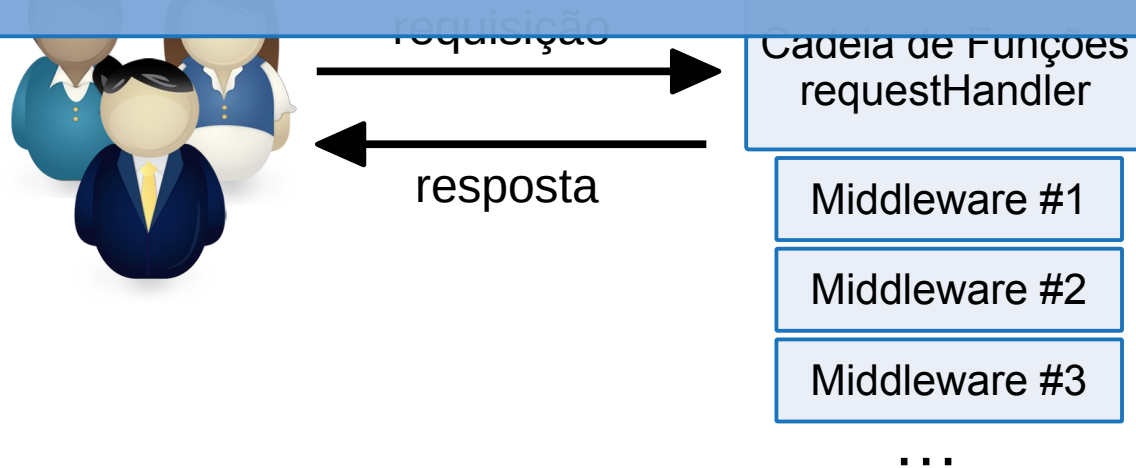
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental: ao invés de um callback, podemos criar vários em sequência



Middleware

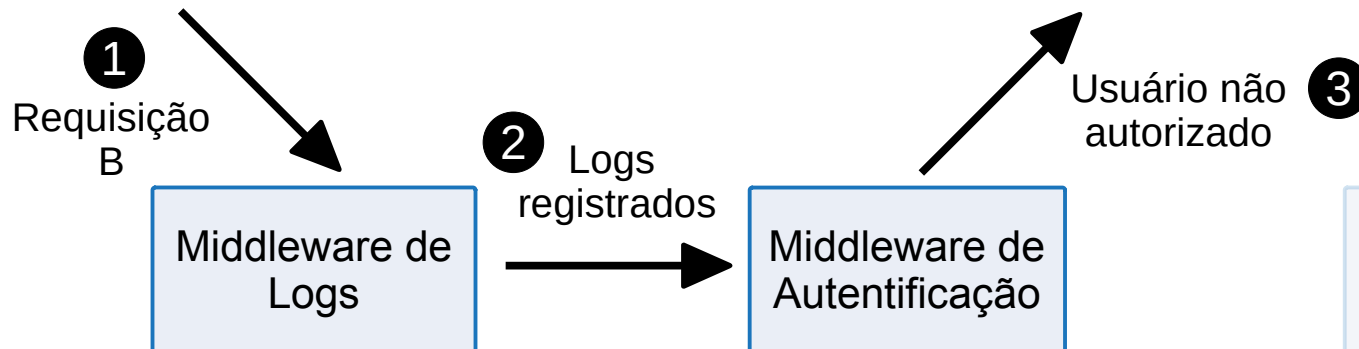
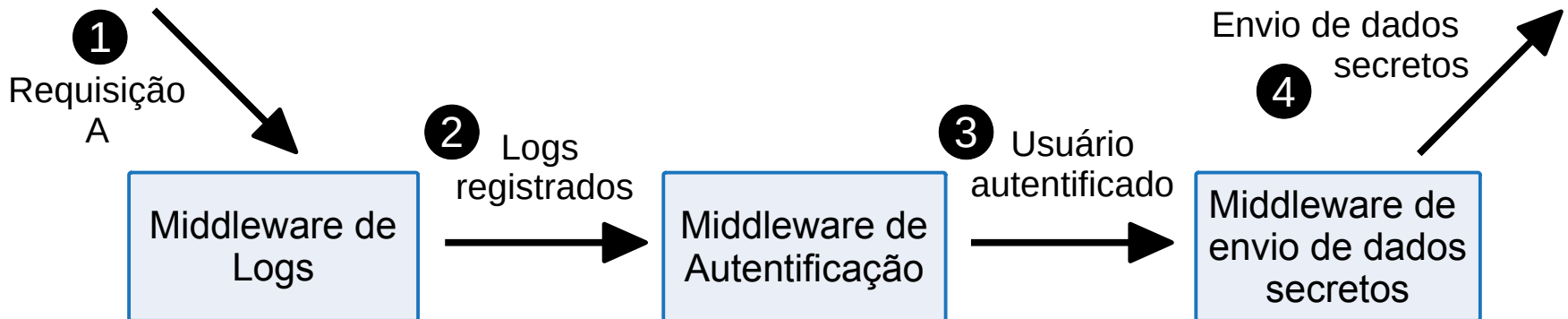
- Os **middlewares** são funções similares ao callback de **http.createServer**, mas possuem uma diferença fundamental: ao invés de um callback, podemos criar vários em sequência

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



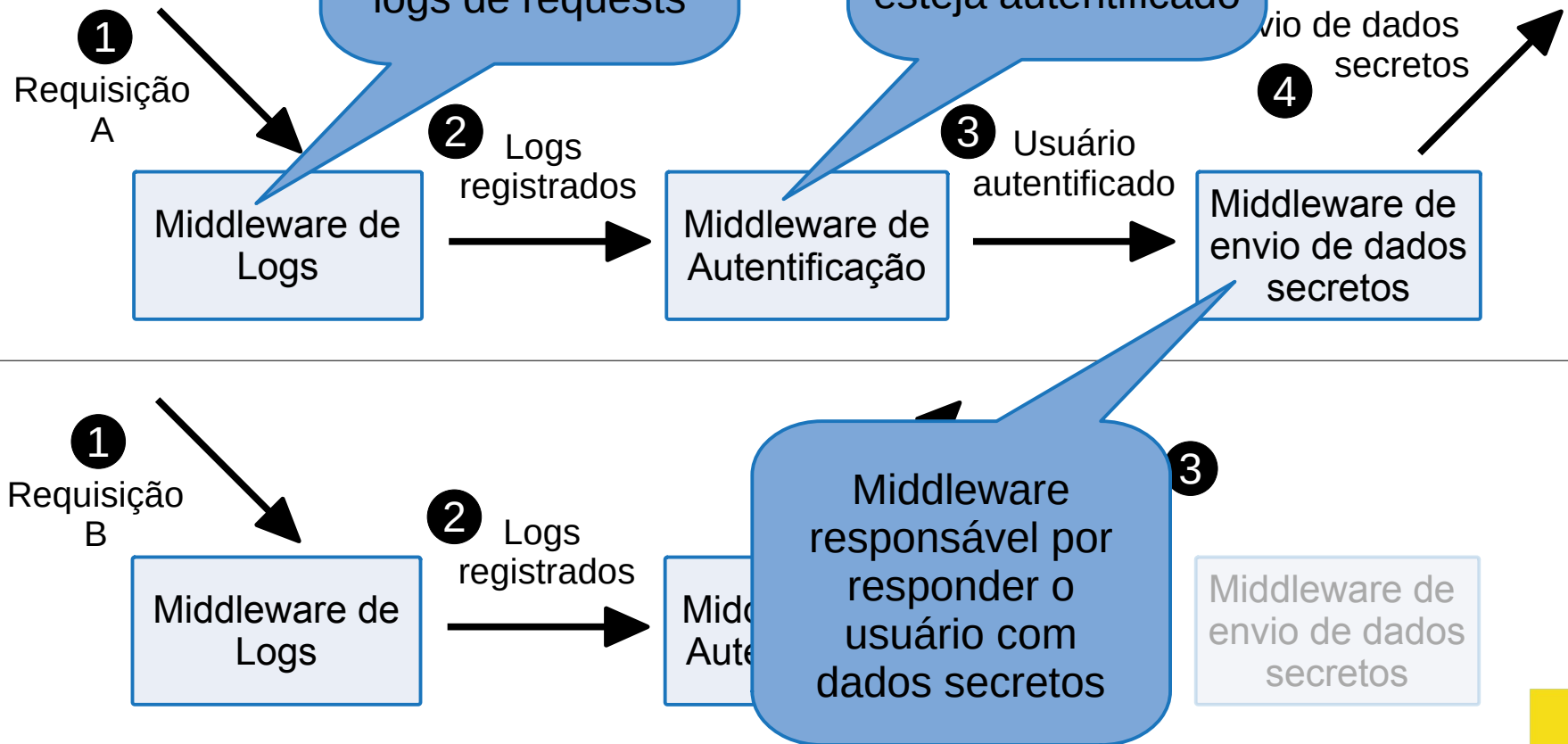
Middleware

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



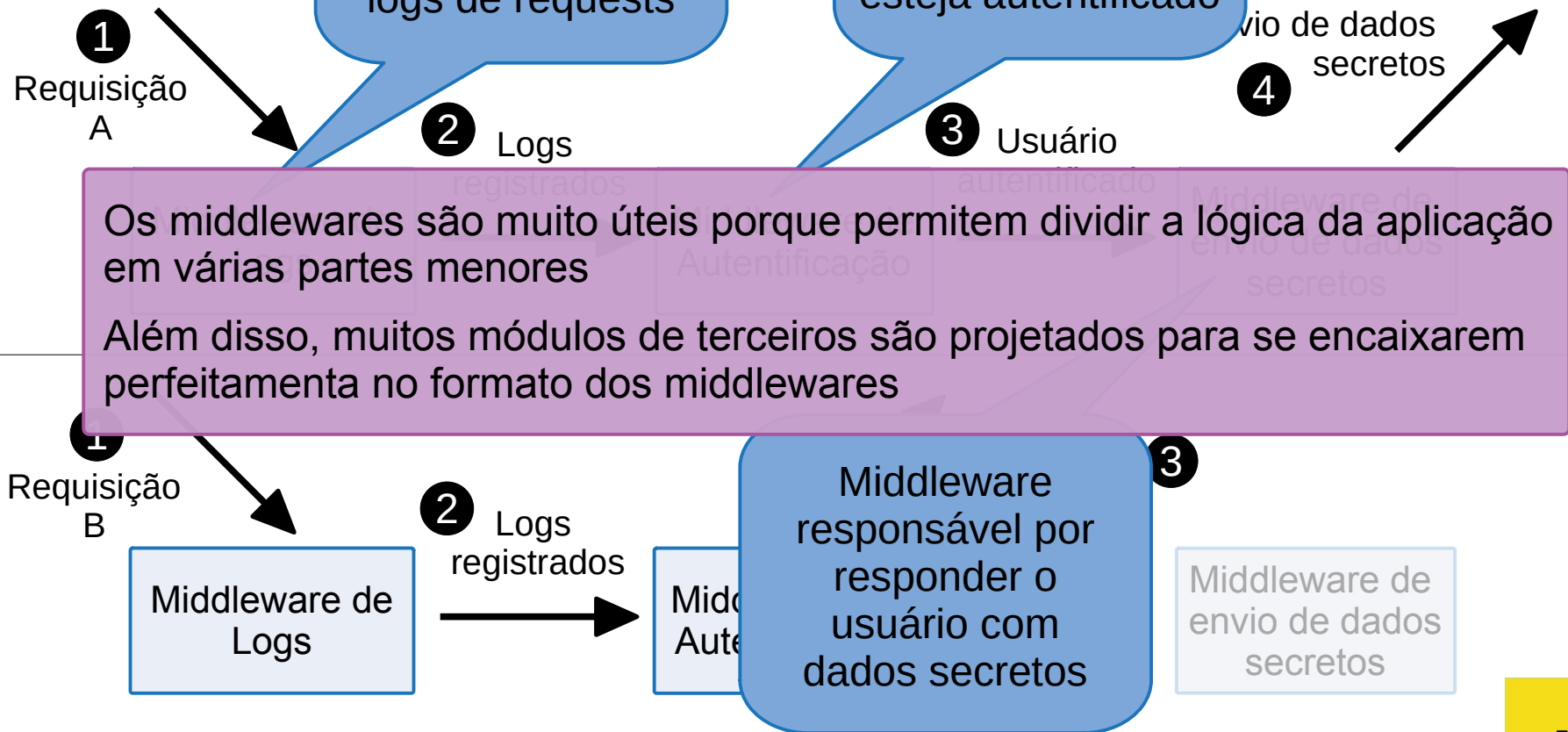
Middleware

- A Figura 2 mostra um exemplo de middleware de Logs e um exemplo de middleware de autenticação. O middleware de Logs é uma função responsável por criar logs de requests. O middleware de autenticação irá deixar o usuário seguir caso ele esteja autenticado.



Middleware

- A Figura 2 mostra um exemplo de middleware de Logs e um exemplo de middleware de autenticação. O middleware de Logs é uma função responsável por criar logs de requests e o middleware de autenticação irá deixar o usuário seguir caso ele esteja autenticado.



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 middlewares também podem receber o **next** como parâmetro.

ser usado

```
[nodemon] starting ~/d/express node src/index.ts  
Express app iniciada na porta 3333
```

```
app.use((req, res, next) => {  
  console.log('Requisição GET /');  
  next();  
});  
app.get('/about', (req, res) => {  
  res.send('Hello world!');  
});
```

localhost:3000

localhost:3000

Hello world!

localhost:3000/about

localhost:3000/about

Hello world!

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.json({ msg: "Usuário não autenticado" })  
  }  
});  
  
app.use((req, res) => {  
  res.json({ dados_secretos: { codigo: 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 primeiro

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

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

```
dotenv.config();  
const PORT = process.env.PORT ?? 3000;  
  
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 milhares de middlewares disponíveis para uso, e um dos mais conhecidos é o **Morgan**, usado para registro de logs

- O primeiro

```
[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
```

morgan("short")
retorna uma
função
middleware

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

Exercício 3

- 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 os arquivos da 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 terminal window with a dark background and light-colored text. The title bar shows the path ~/d/e/public. The prompt is david@coyote. The first command is ls, and the output is css/, html/, img/, and js/. The second command is ls img/, and the output is codebench.png*.

```
~ /d/e/public
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 rodando na porta ${PORT}`);
});
```

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

Executado em toda requisição que comece 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';
```



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

Executado em
toda requisição
que comece
com **/img**

express

JS

next
quivo
ja
ado

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
 - **Express.json** – 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.send("Bem-vindo ao meu site!");  
});
```

Executado só
na requisição
GET /

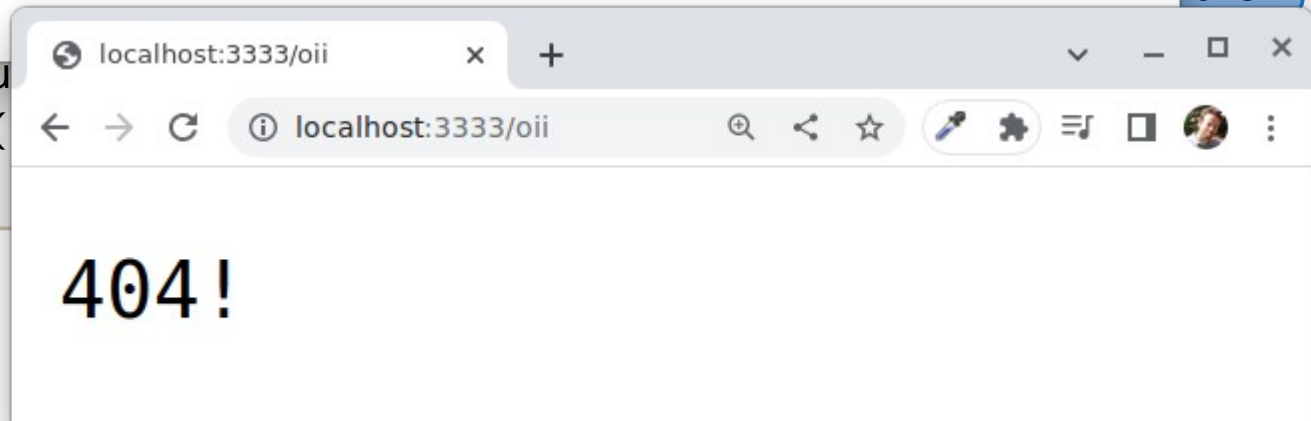
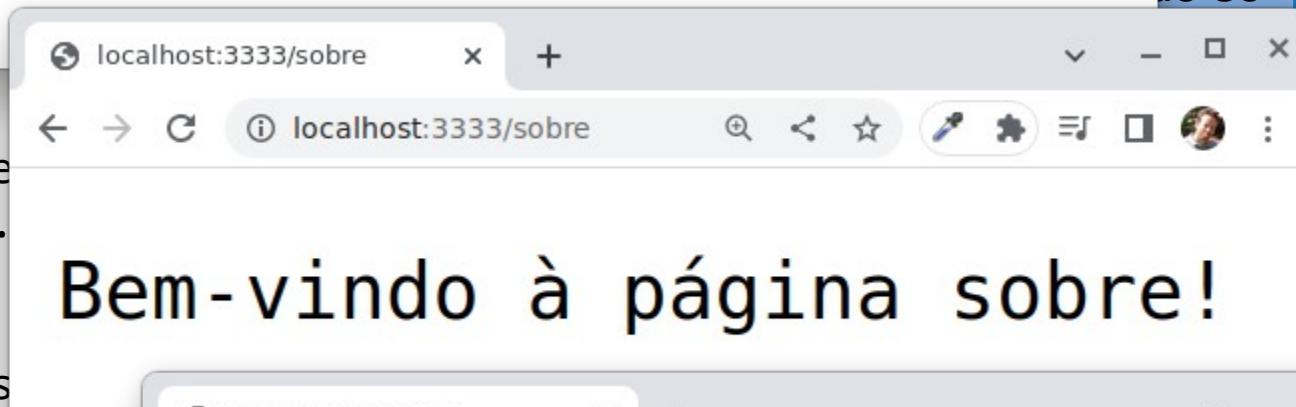
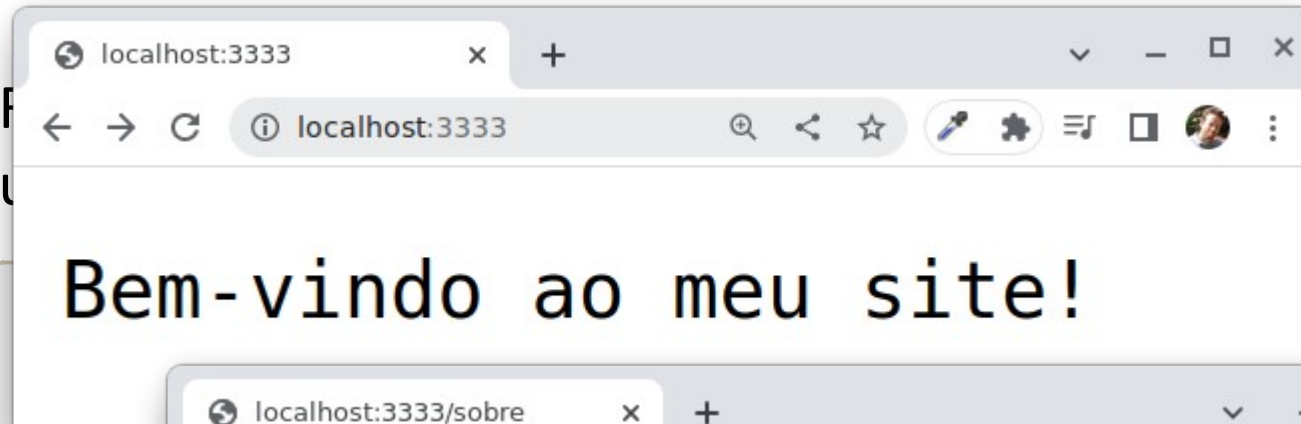
```
app.get("/sobre", (req, res) => {  
  res.send("Bem-vindo à página sobre!");  
});
```

Executado só
na requisição
GET /sobre

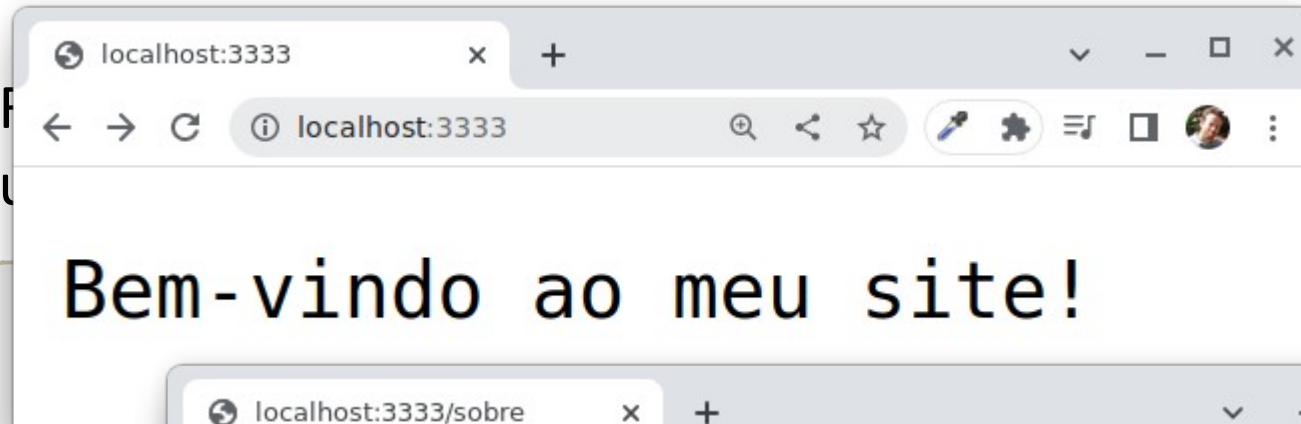
```
app.use((req, res) => {  
  res.statusCode = 404;  
  res.send("404!");  
});
```

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

Roteamento



Roteamento



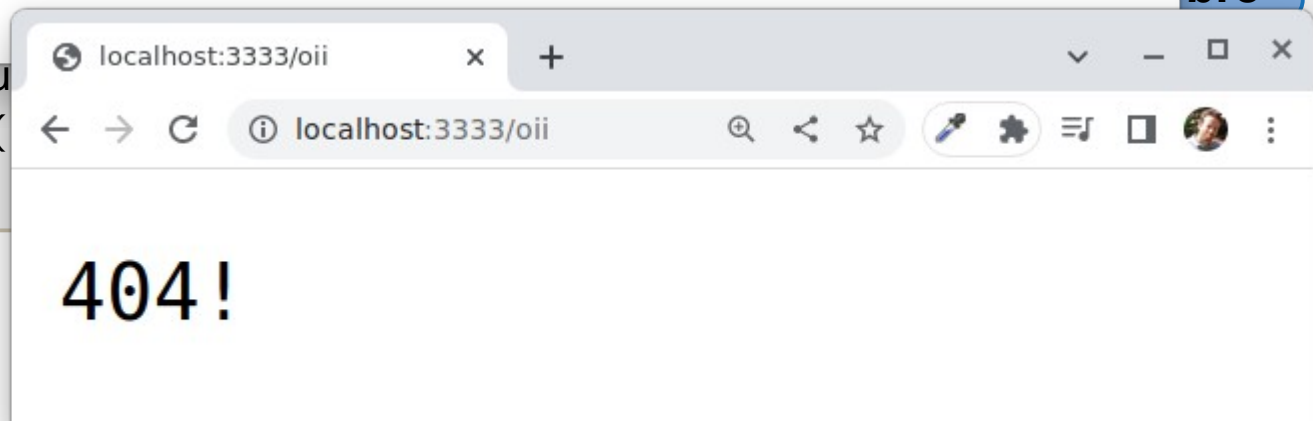
es serão

do só

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

bre

```
app.use  
res.statu  
res.send(  
});
```



Roteamento

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

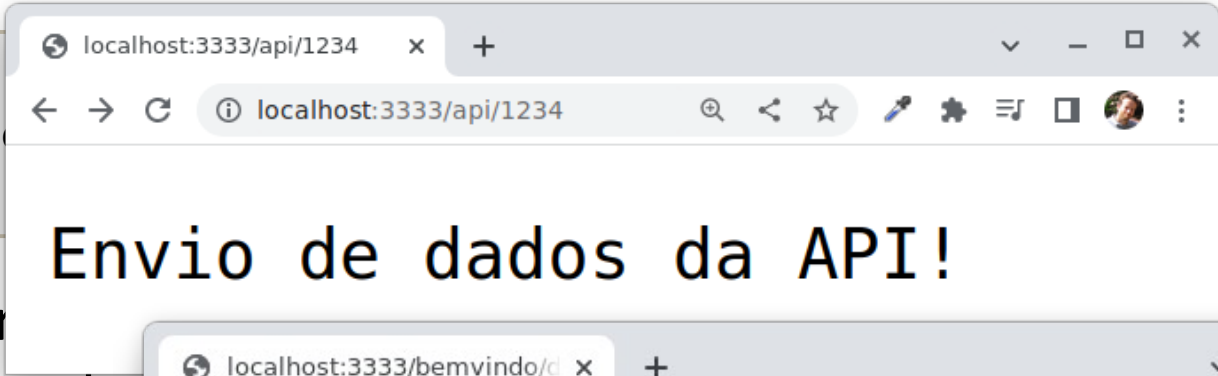
```
app.get( /^\/(api|rest)\/.+$/, (req, res) => {  
  res.send("Envio de dados da API!");  
});
```

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

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


Roteamento

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




localhost:3333/api/1234

Envio de dados da API!

This screenshot shows a web browser window with the address bar set to 'localhost:3333/api/1234'. The page content displays 'Envio de dados da API!' in a monospace font. The browser's developer tools are open, showing the console.

- Outra forma de usar rotas é através da URL, tal como



localhost:3333/bemvindo/david

Seja bem vindo david

This screenshot shows a web browser window with the address bar set to 'localhost:3333/bemvindo/david'. The page content displays 'Seja bem vindo david' in a monospace font. The browser's developer tools are open, showing the console.

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.");  
});
```

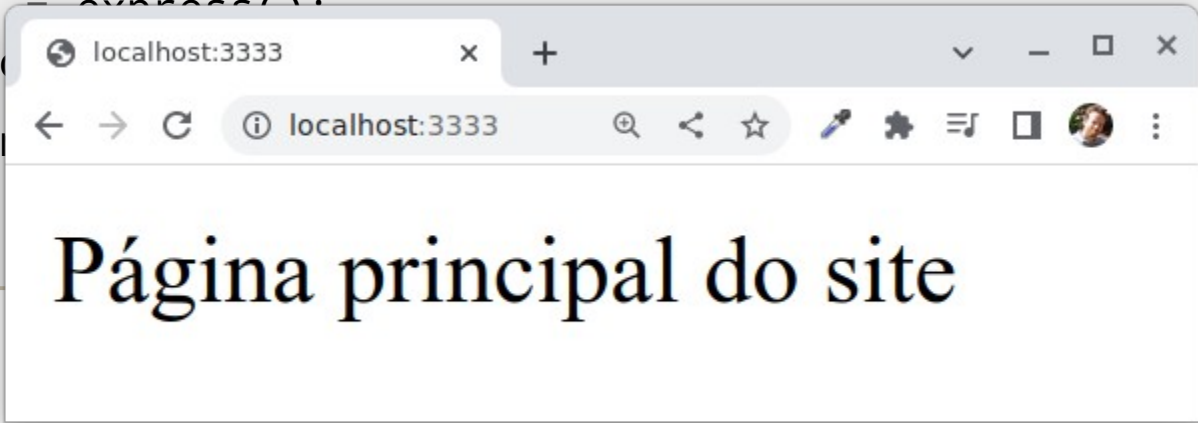
Arquivo Separado de Rotas

- 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');  
});
```



Página principal do site

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.

```
console.log('...');  
});
```

Página principal do site

Extensões de Req/Res

- **Request** (req) e **response** (res) são objetos passados para todas as funções que tratam as requisições de usuários
- Eles já estavam 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

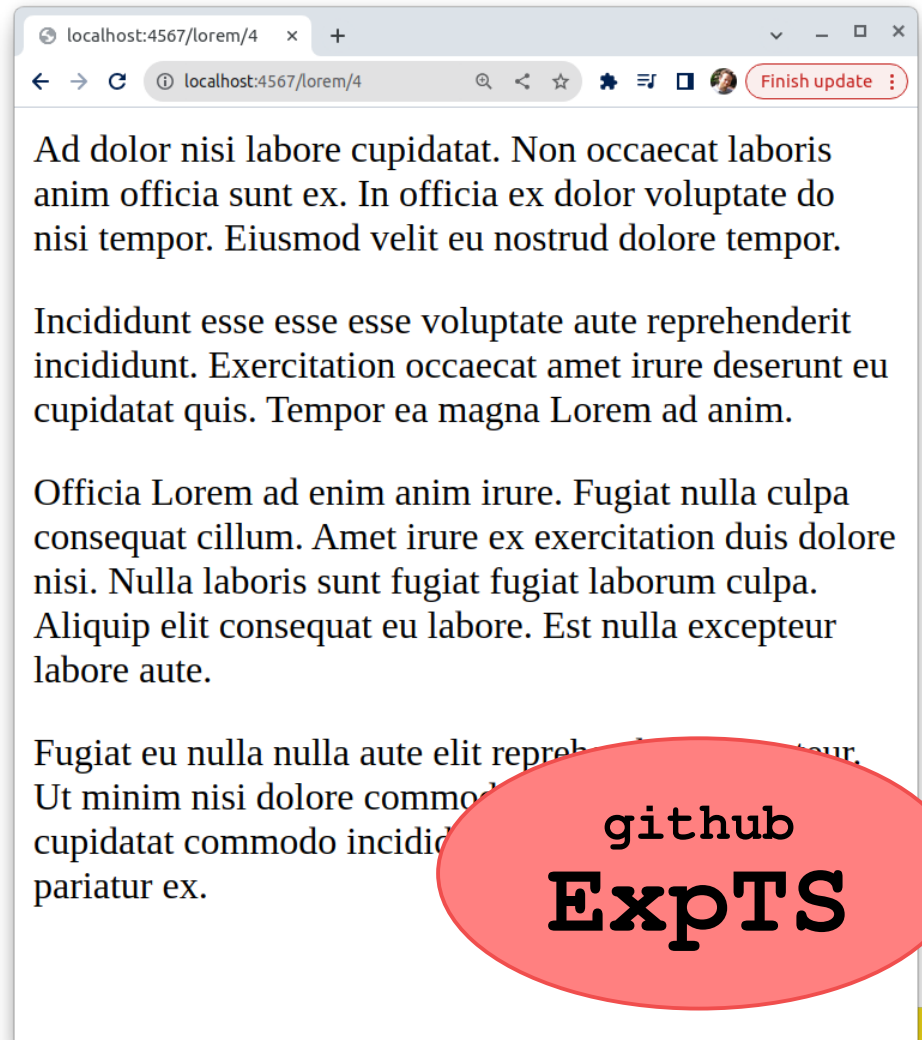
- Outro exemplo é o método **sendFile()**

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

Envia o
arquivo para
o browser

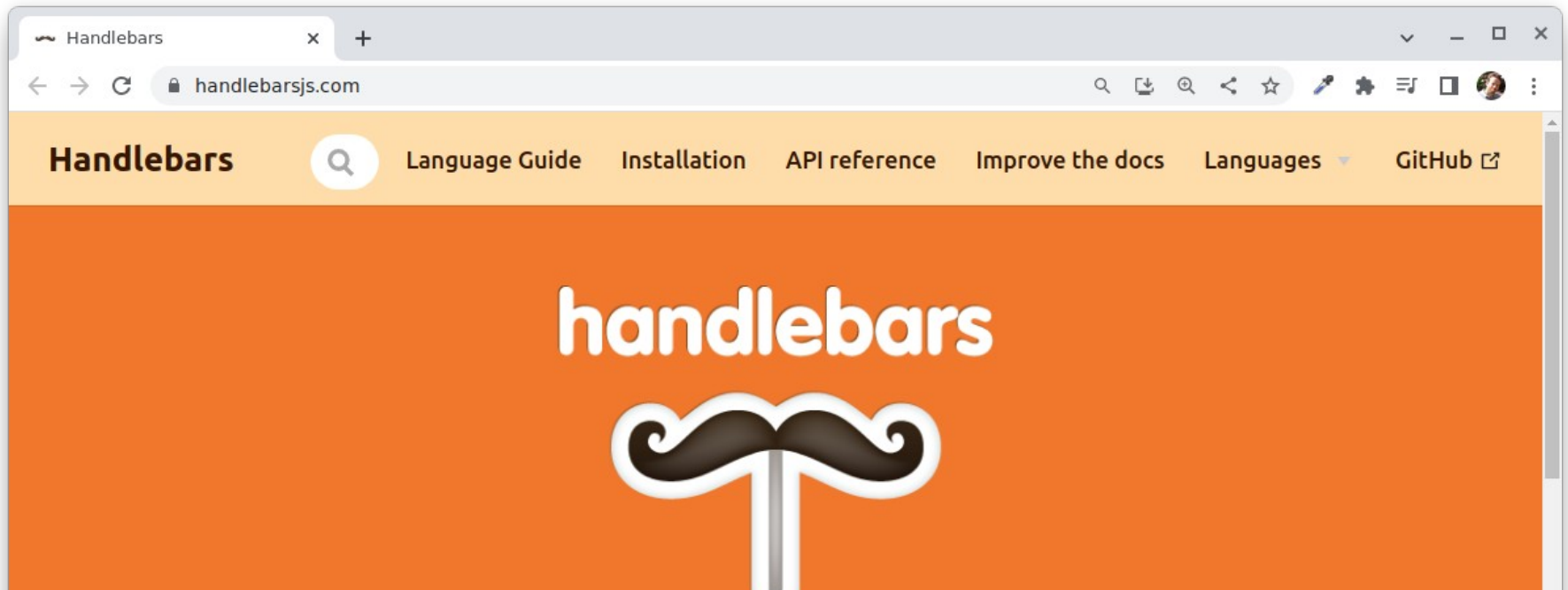
Exercício 4

Faça uma aplicação que possua uma rota **/lorem** e que receba como parâmetro o número de parágrafos **Lorem Ipsum** desejados. O número de parágrafos impressos deve ser igual ao parâmetro informado. As rotas da aplicação devem estar em um **arquivo separado**.



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,

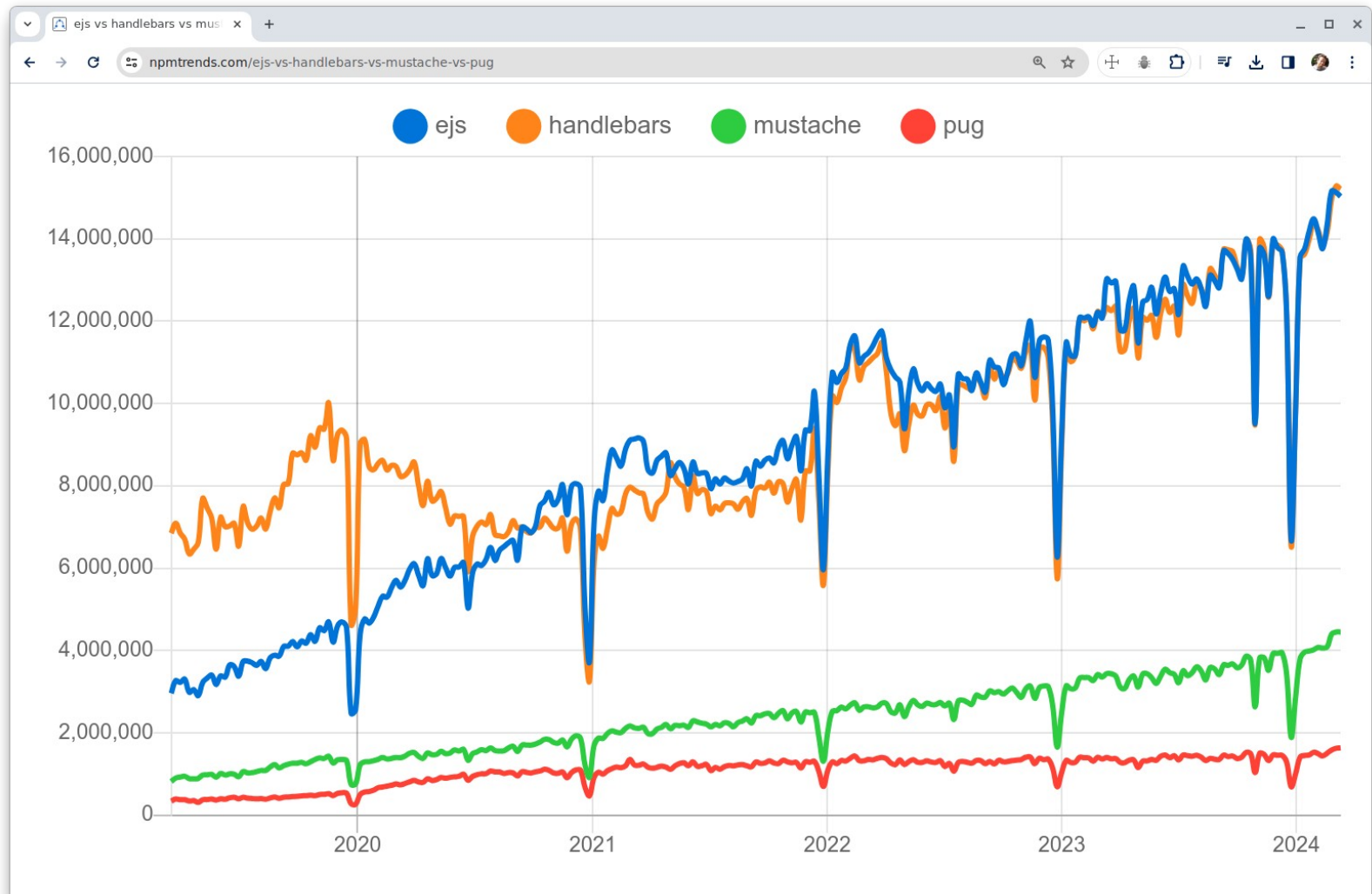
Conforme já dito, o Express é um **framework não opinativo** e requer uma série de acessórios de terceiros para construir uma aplicação completa

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

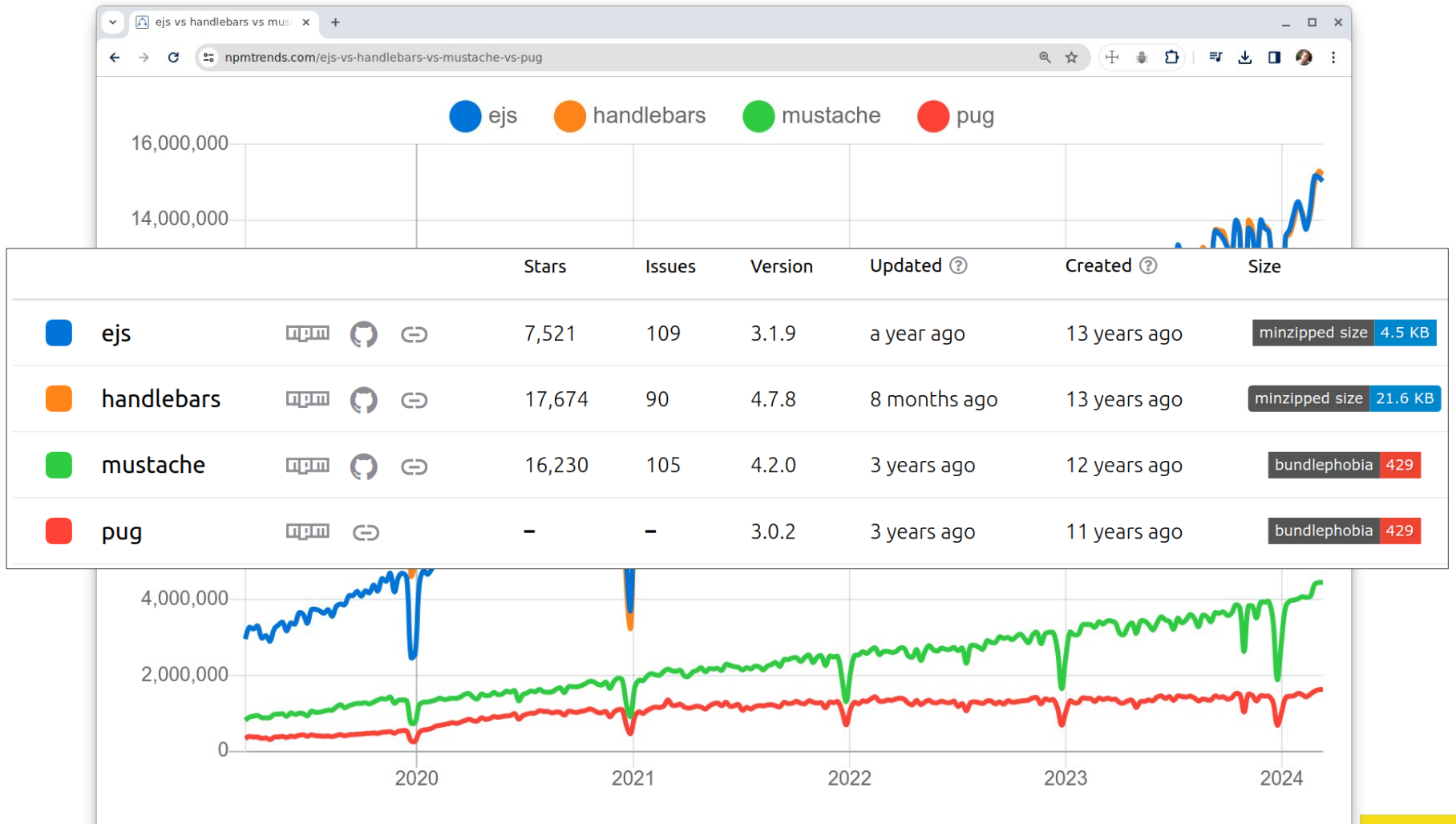
handlebars



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**

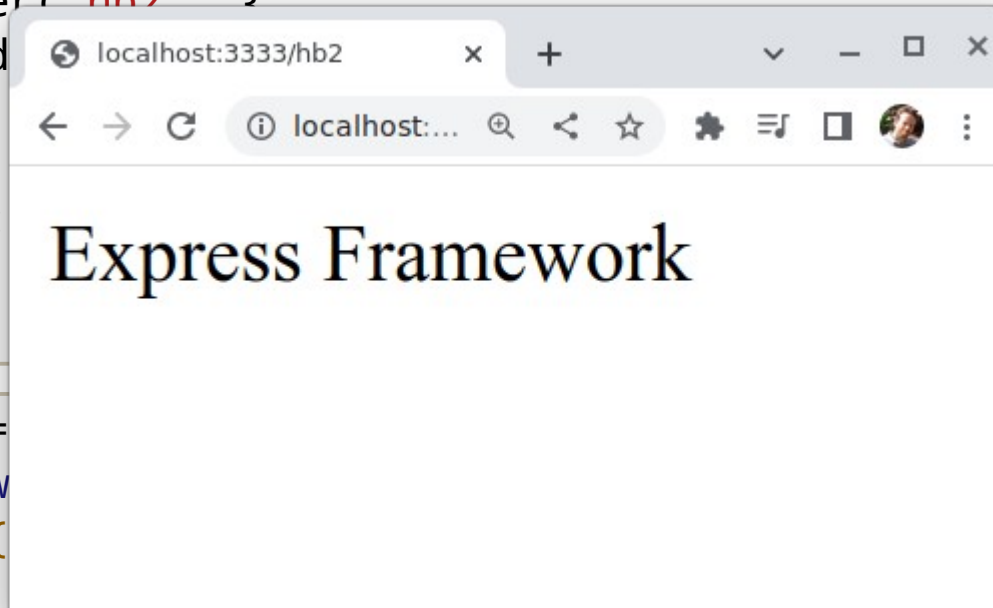
```
app.get('/hb2', (req, res) => {  
  res.render('hb2', {  
    poweredByNodejs: true,  
    name: 'Express',  
    type: 'Framework',  
    layout: false,  
  });  
});
```

```
<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:  
    name:  
    type:  
    layout:  
  });  
});
```



```
<div class=  
  {{#if pow  
    <span>{{  
  {{/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('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 });  
});
```

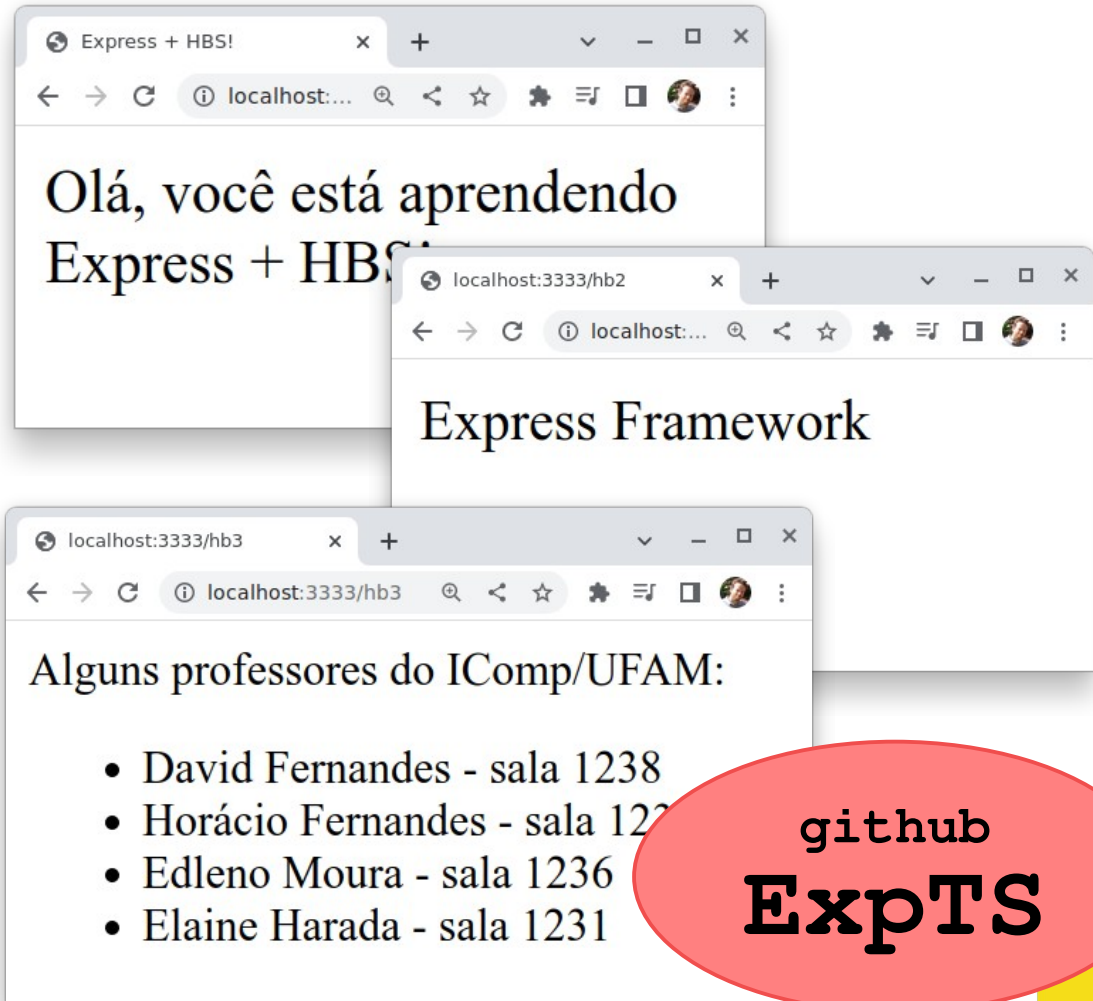
Alguns professores do IComp/UFAM:

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

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

Exercício 5

- Implemente as páginas das rotas **/hb1**, **/hb2** e **/hb3** 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", 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) => `- ${p.nome}-${p.sala}</li>`);
  return `
${list.join('')}</ul>`;
}

```

```
// Arquivo /views/helpers/helpersTypes.ts
export interface Prof {
  nome: string;
  sala: string;
}
```

Helpers Handlebars

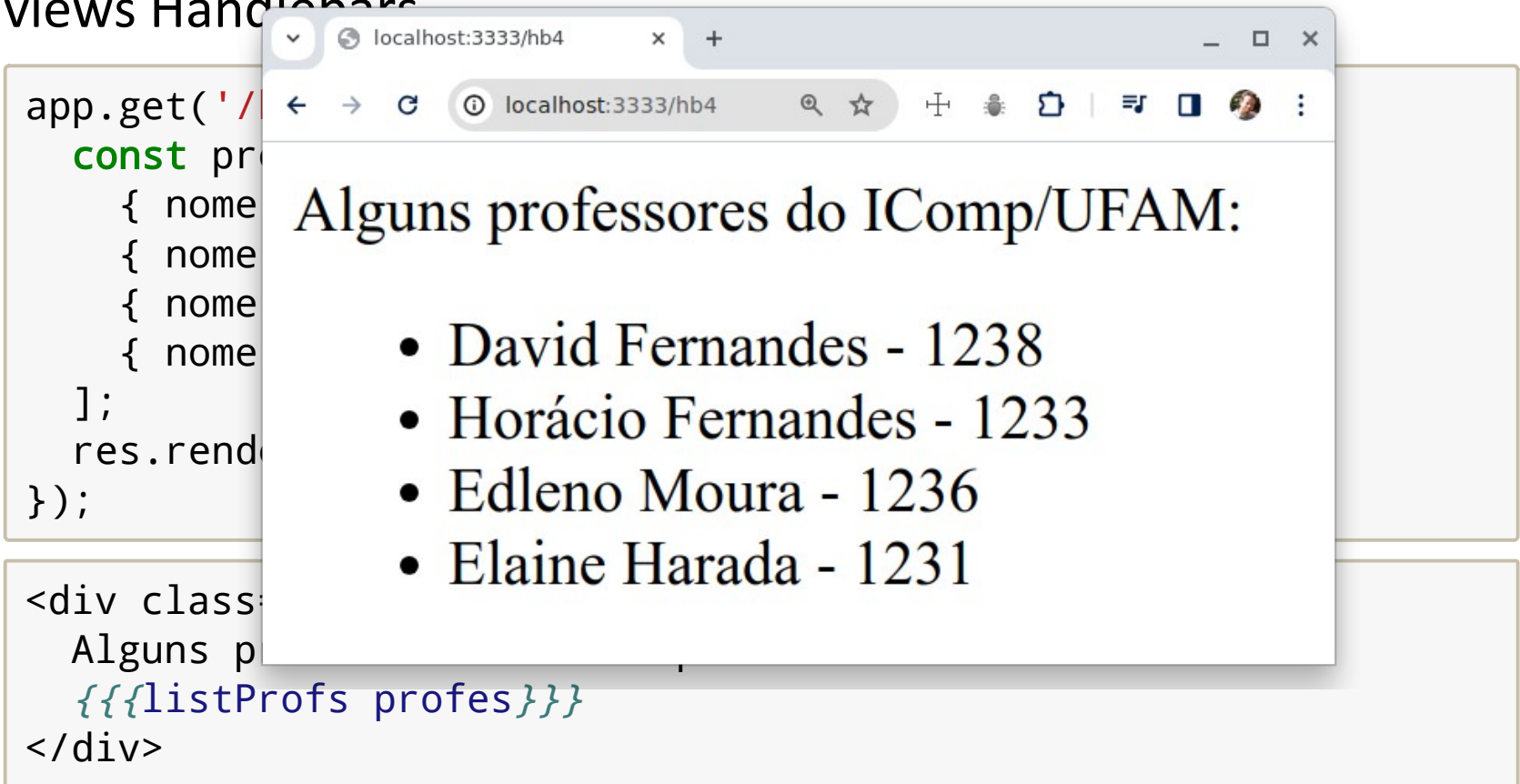
- A partir deste momento, a função **listProfs** pode ser usada nas views Handlebars

```
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



The image shows a web browser window at localhost:3333/hb4 displaying a list of professors. To the left, there are two code snippets: one for the Express.js backend route and one for the Handlebars frontend view.

Backend Code (app.get):

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

Frontend Code (Handlebars View):

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

Browser Display:

Alguns professores do IComp/UFAM:

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

Helpers Handlebars

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

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

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

```
app.get('/',
```

```
  { nome  
    { nome  
  };  
  res.render  
});
```

```
<div class=  
  Alguns p  
  {{{listProfs profes}}}  
</div>
```


Exercício 6

github
ExptS

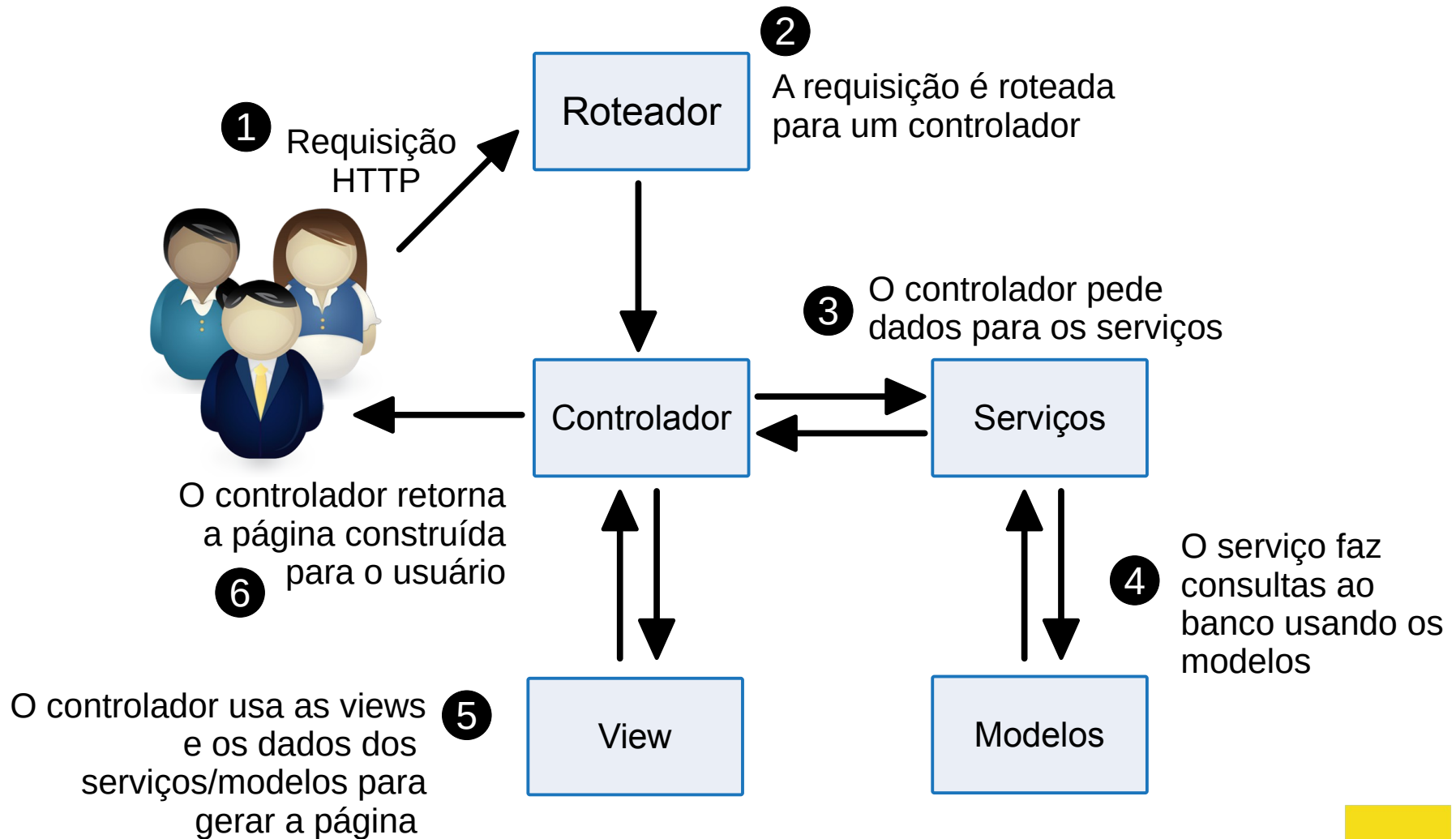
Crie uma rota **/hb4** que usa um **helper handlebars** capaz de receber um vetor de tecnologias, com os campos **name** (string), **type** (string) e **poweredByNodejs** (boolean) e imprima a lista de tecnologias desenvolvidas com Node.JS

Technologias baseadas no NodeJS:

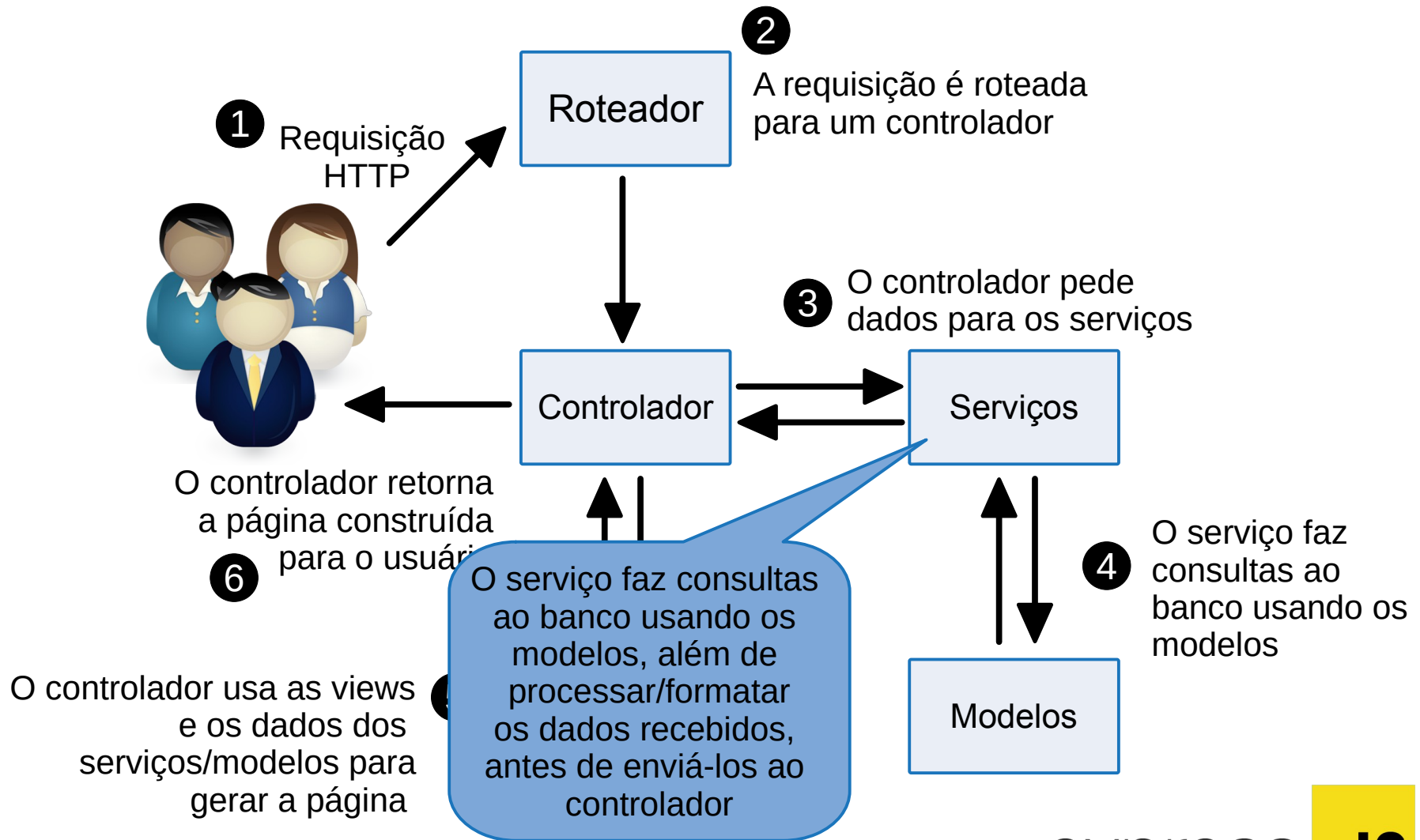
- Express - Framework
- React - Library
- Handlebars - Engine View
- Sequelize - ORM tool

```
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



Movendo o Express para o MVC

- Até este momento, nossa aplicação ainda não possui nenhum diretório específico para **controladores** e **serviços**

```
~/d/express
david@coyote ~/dev/express $ tree -d
.
├── logs
├── public
│   ├── css
│   ├── html
│   ├── img
│   └── js
└── src
    ├── middlewares
    ├── router
    ├── utils
    ├── views
    └── helpers

12 directories
david@coyote ~/dev/express $
```

Movendo o Express para o MVC

- Criando os diretórios **controllers** e **services**, o diretório **src** passa a ter a seguinte estrutura de subdiretórios

```
~ /d/e/src
david@coyote ~/dev/express/src $ mkdir controllers services
david@coyote ~/dev/express/src $ tree -d
.
├── controllers
├── middlewares
├── router
├── services
├── utils
├── views
└── helpers

7 directories
david@coyote ~/dev/express/src $
```

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 criadas até este momento
- Primando pela organização do código, criamos um diretório **src/views/main** que irá conter as views do controlador main

```
fish /home/david/dev/express/src/controllers
david@coiote ~/dev/express/src/controllers $ touch main.ts
david@coiote ~/dev/express/src/controllers $ ls
main.ts
david@coiote ~/dev/express/src/controllers $
```

```
fish /home/david/dev/express/src/views/main
david@coiote ~/dev/express/src/views/main $ ls
hb1.handlebars  hb3.handlebars
hb2.handlebars  hb4.handlebars
david@coiote ~/dev/express/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 criadas até este momento
- Primeira view possui o conteúdo html completo de suas páginas

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

```
david@coiote:~/dev/express/src/views/main$
```

```
hb1.handlebars
```

```
hb2.handlebars
```

```
hb4.handlebars
```

```
david@coiote ~/dev/express/src/views/main $
```

Movendo o Express para o MVC

- O **controlador main** irá conter os middlewares (actions) associados a cada uma das rotas desenvolvidas

```
// Arquivo src/controllers/main.ts  
import { Request, Response } from 'express';  
const index = (req: Request, res: Response) => {  
  res.end('Welcome to Web academy!');  
};  
  
const hb2 = (req: Request, res: Response) => {  
  res.render('main/hb2', {  
    nome: 'React',  
    tipo: 'library',  
    poweredByNode: true,  
    layout: false,  
  });  
};  
  
...  
export default { index, hb2, ... };
```


Movendo o Express para o MVC

- O **controlador main** irá conter os middlewares (actions) associados a cada uma das rotas desenvolvidas

```
// Arquivo src/controllers/main.ts  
  
import { Request, Response } from 'express';  
  
const index = (req: Request, res: Response) => {
```

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**, **lorem** e **hb1** são **actions** do **controlador main**.

```
    nome: 'React',  
    tipo: 'library',  
    poweredByNode: true,  
    layout: false,  
  });  
};  
  
...  
  
export default { index, hb2, ... };
```

Movendo o Express para o MVC

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

```
import { Router } from 'express';
import mainController from '../controllers/main';

const router = Router();

// Main Controller
router.get('/', mainController.index);
router.get('/lorem', mainController.lorem);
router.get('/bemvindo/:nome', mainController.bemvindo);
router.get('/hb1', mainController.hb1);
router.get('/hb2', mainController.hb2);
router.get('/hb3', mainController.hb3);
router.get('/hb4', mainController.hb4);

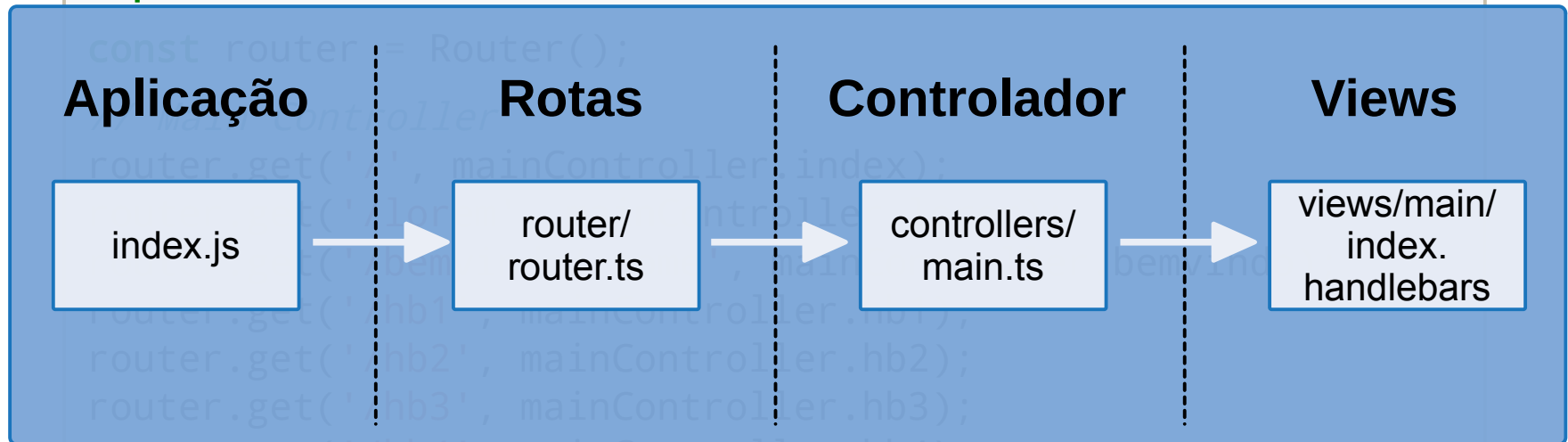
// User Controller

export default router;
```

Movendo o Express para o MVC

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

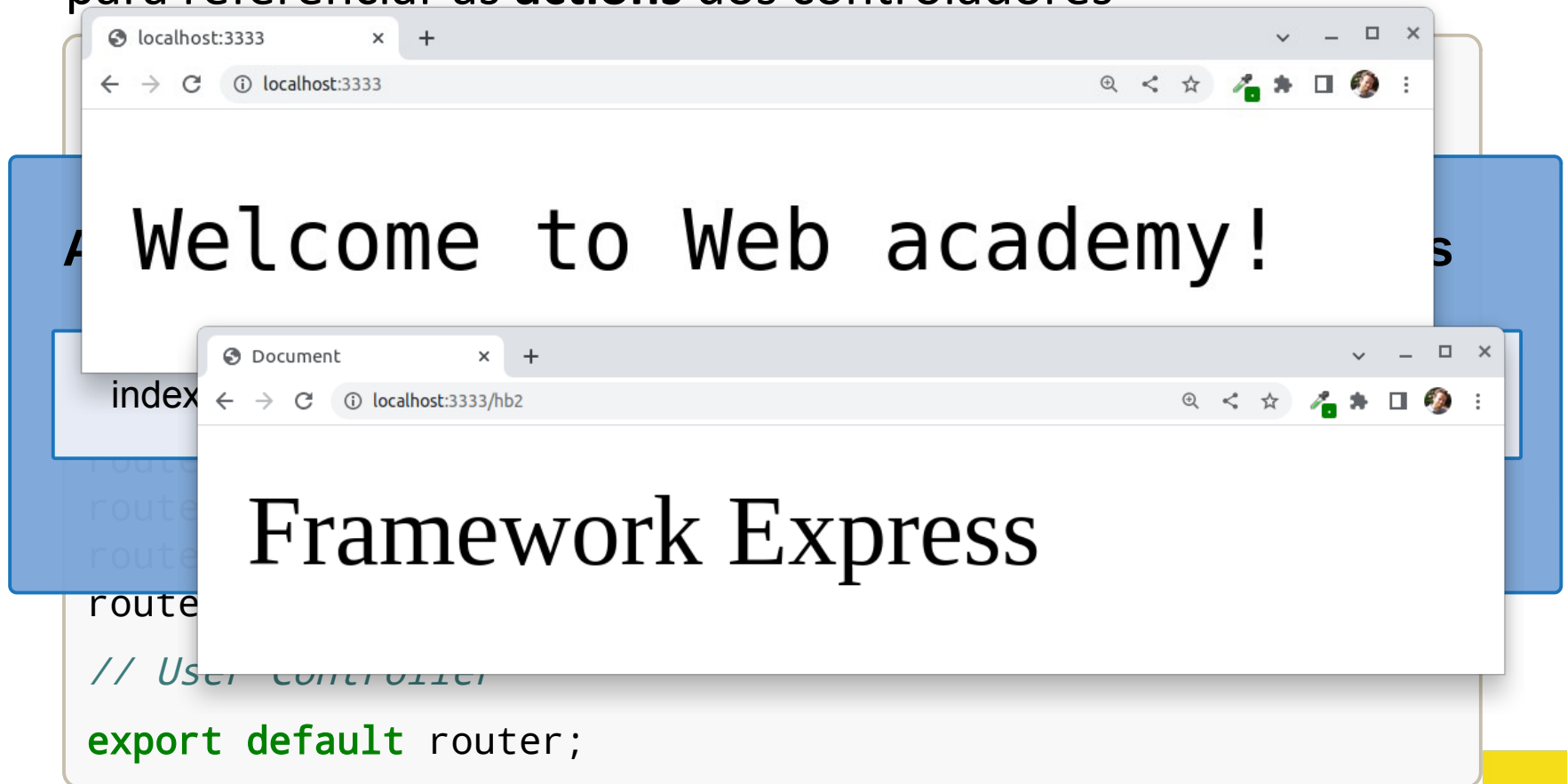
```
import { Router } from 'express';  
import mainController from '../controllers/main';
```



```
router.get('/hb4', mainController.hb4);  
  
// User Controller  
export default router;
```

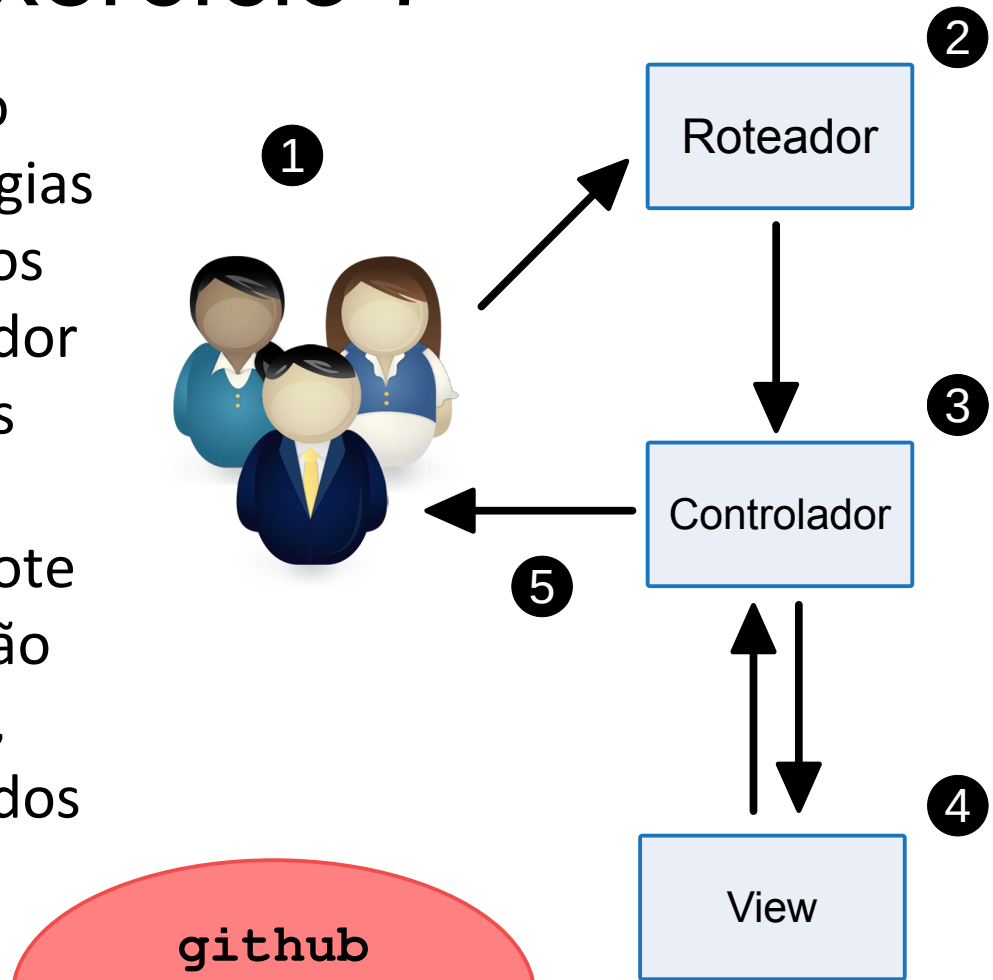
Movendo o Express para o MVC

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



Exercício 7

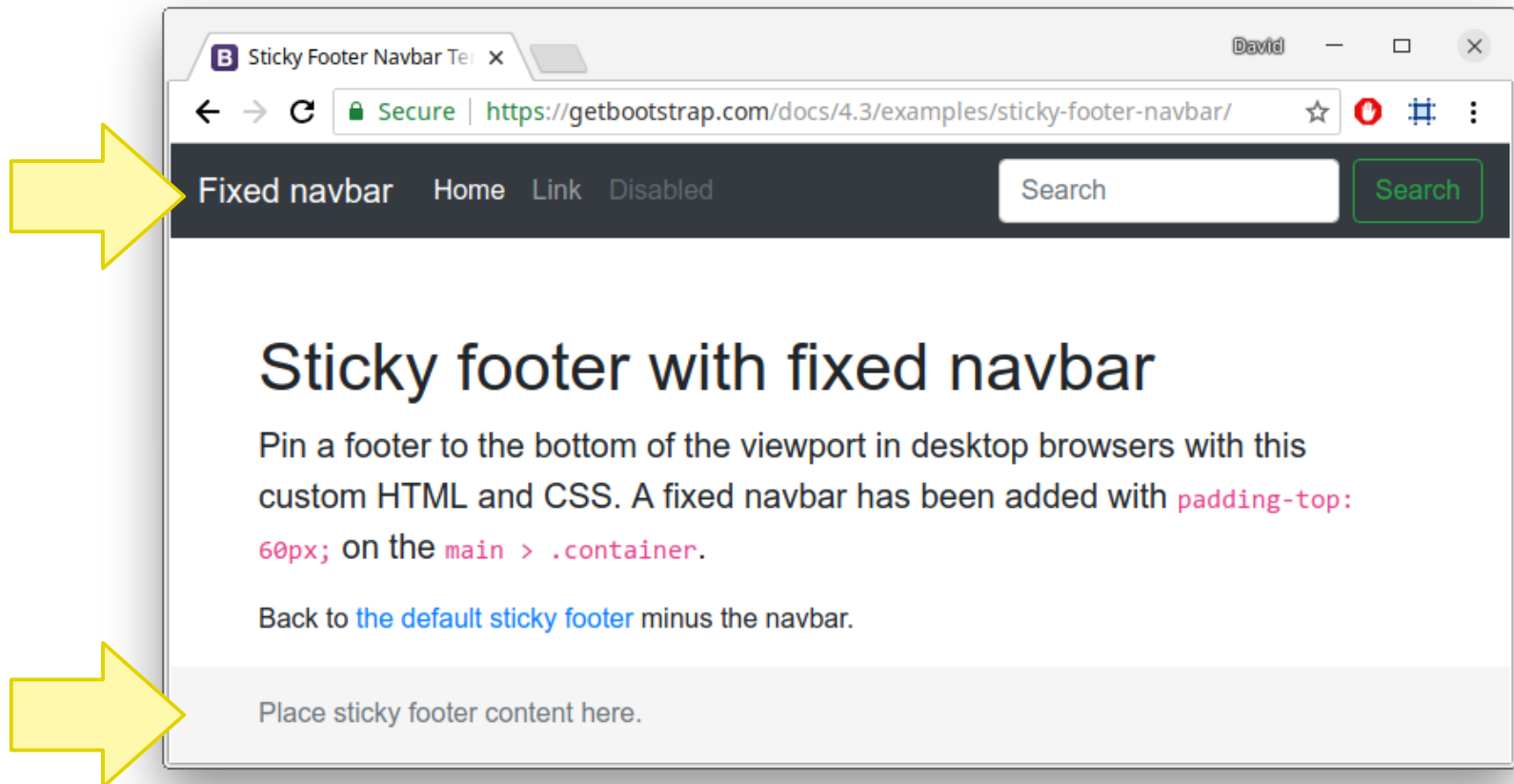
Mova a aplicação para o MVC, usando as estratégias apresentadas nos últimos slides. Crie um controlador **main** com todas as rotas desenvolvidas nos exercícios anteriores. Note que a aplicação ainda não tem modelos e serviços, que somente serão criados nas próximas etapas do curso.



github
Expts

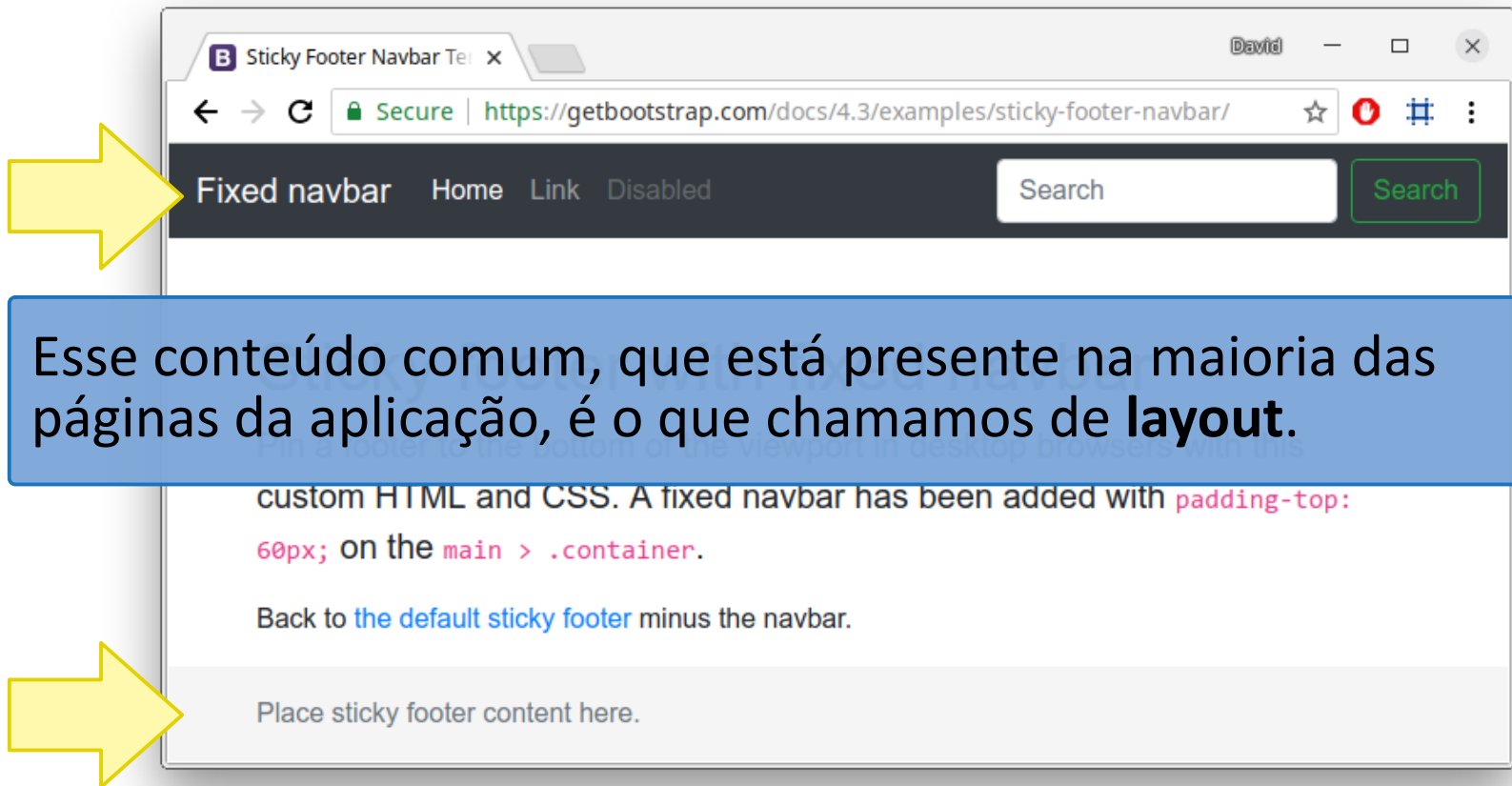
Layout X Views

- 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**



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', engine({  
  layoutsDir: `_${__dirname}/views/layoutsdir`,  
  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 contêm as tags `<html>`, `<head>`, `<body>`, `<style>` e `<meta>` da aplicação
- 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 geração da página

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

Local onde as views serão carregadas

Layout X Views

- Os layouts geralmente contêm 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 código das views passa a ser somente

```
<div>{{conteudo}}</div>
```

código `{{{body}}}`

- 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 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

- Na rota `/hb1` por exemplo, o código da view passa a ser somente `<div>{{tipo}} {{nome}}</div>`

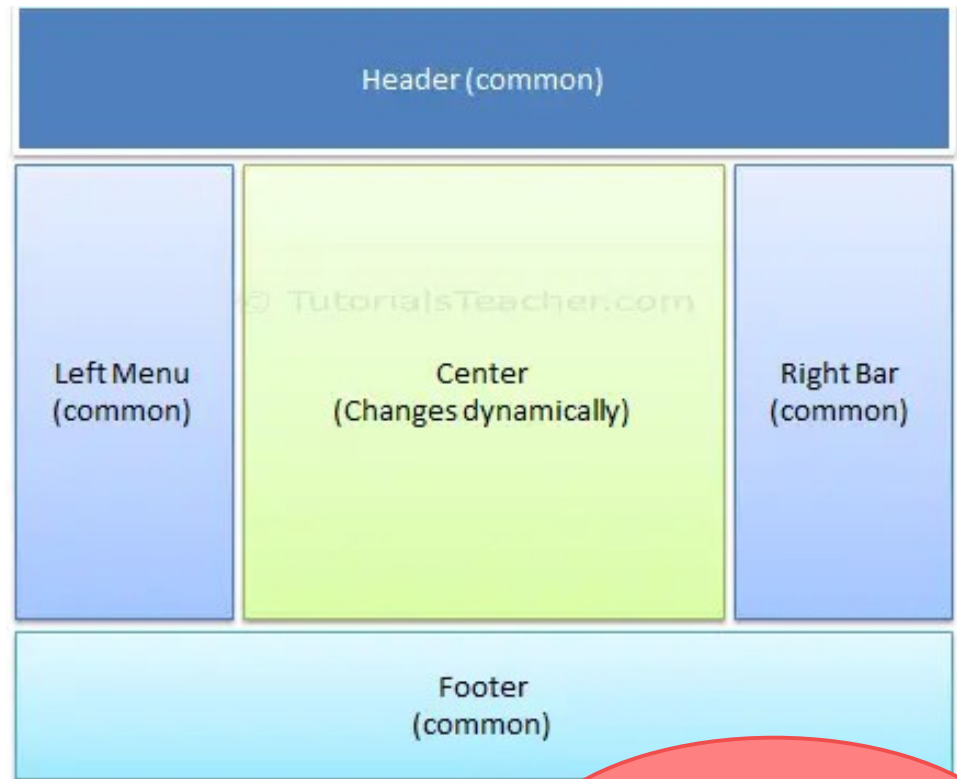
código

- Em ou
o código
momento



Exercício 8

Até então cada view possui seu próprio template (ex, tags `<html>`, `<header>`, `<body>`, etc). 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.

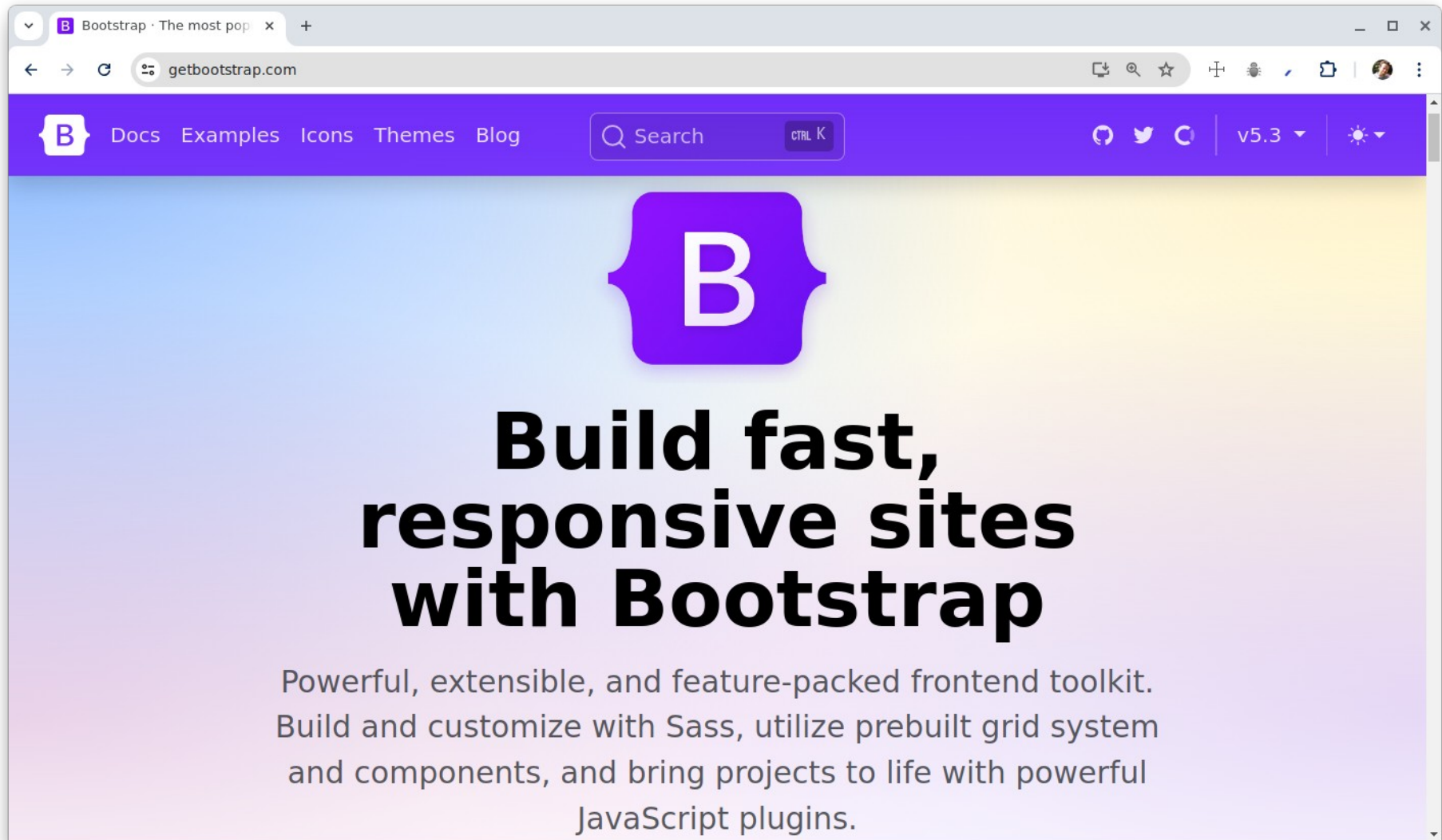


github

ExpTS

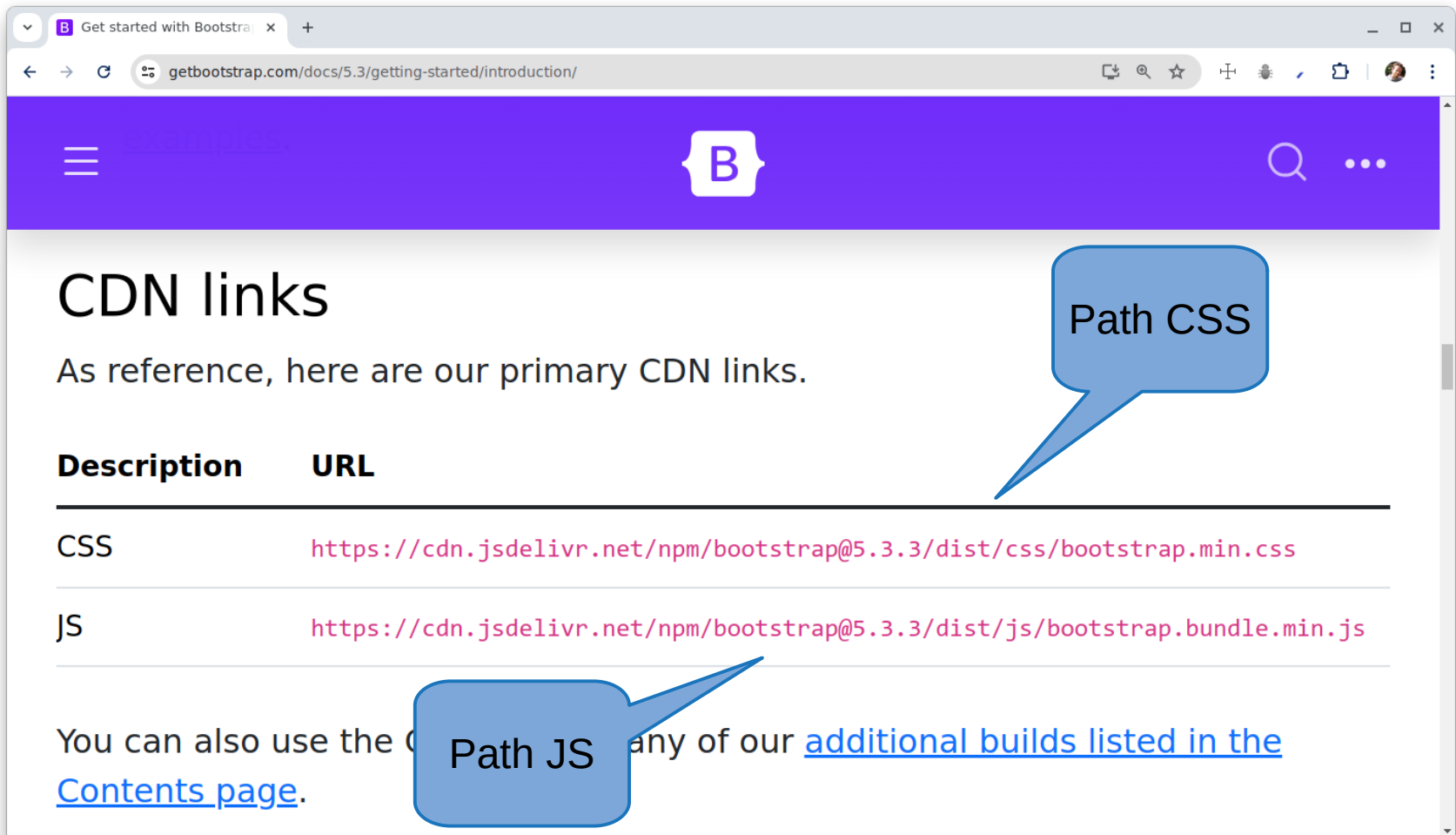
express JS

Adicionando o Bootstrap



Adicionando o Bootstrap

- Para instalar o Bootstrap, podemos usar links CDN



The screenshot shows the Bootstrap documentation page for getting started with Bootstrap 5.3. The page has a purple header with the Bootstrap logo and a search icon. The main content area is titled "CDN links" and includes a paragraph: "As reference, here are our primary CDN links." Below this is a table with two columns: "Description" and "URL".

Description	URL
CSS	https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css
JS	https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js

Below the table, there is a paragraph: "You can also use the C... any of our [additional builds listed in the Contents page](#)." Two blue callout boxes are present: one pointing to the "CSS" row labeled "Path CSS" and another pointing to the "JS" row labeled "Path JS".

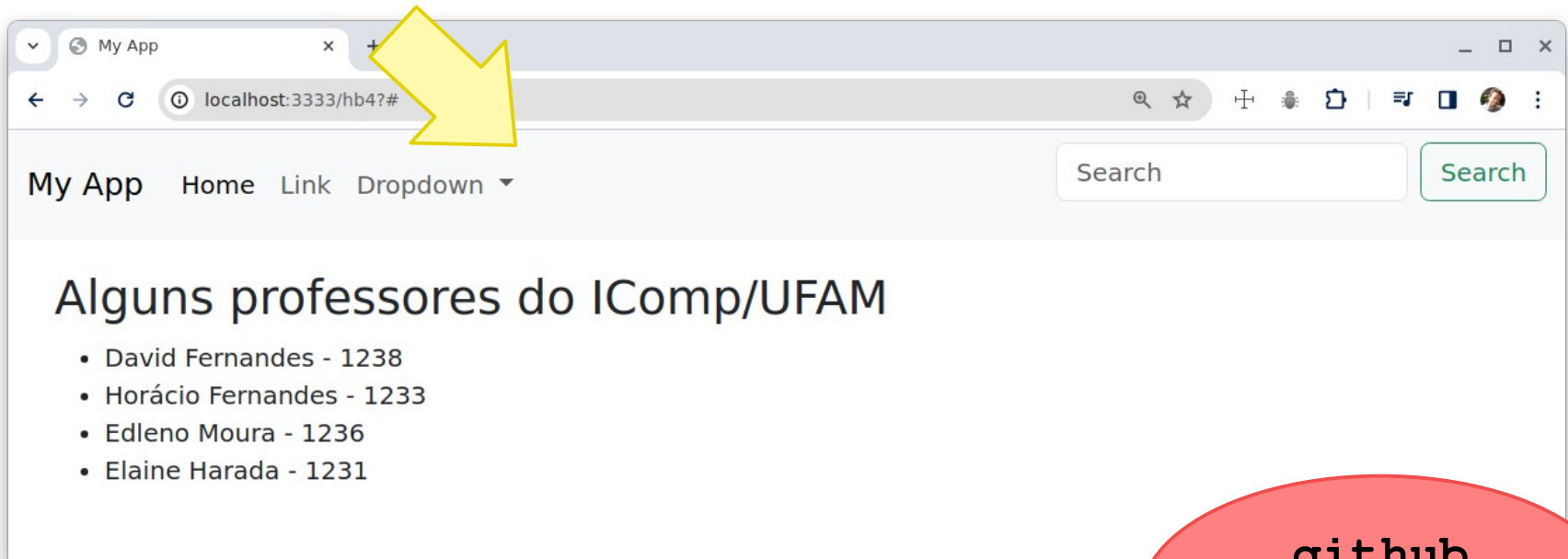
Adicionando o Bootstrap

- Edite o arquivo de layout de sua aplicação e adicione os paths CSS e Javascript da CDN

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Minha Loja</title>
  <link href="Path CSS" rel='stylesheet' />
</head>
<body>
  {{{body}}}
  <script src="Path JS"></script>
</body>
</html>
```

Exercício 9

- Coloque um **Navbar Bootstrap** no layout do handlebars. Use os exemplos de Navbar disponíveis em <https://getbootstrap.com/docs/5.3/components/navbar/>



github
ExpTS