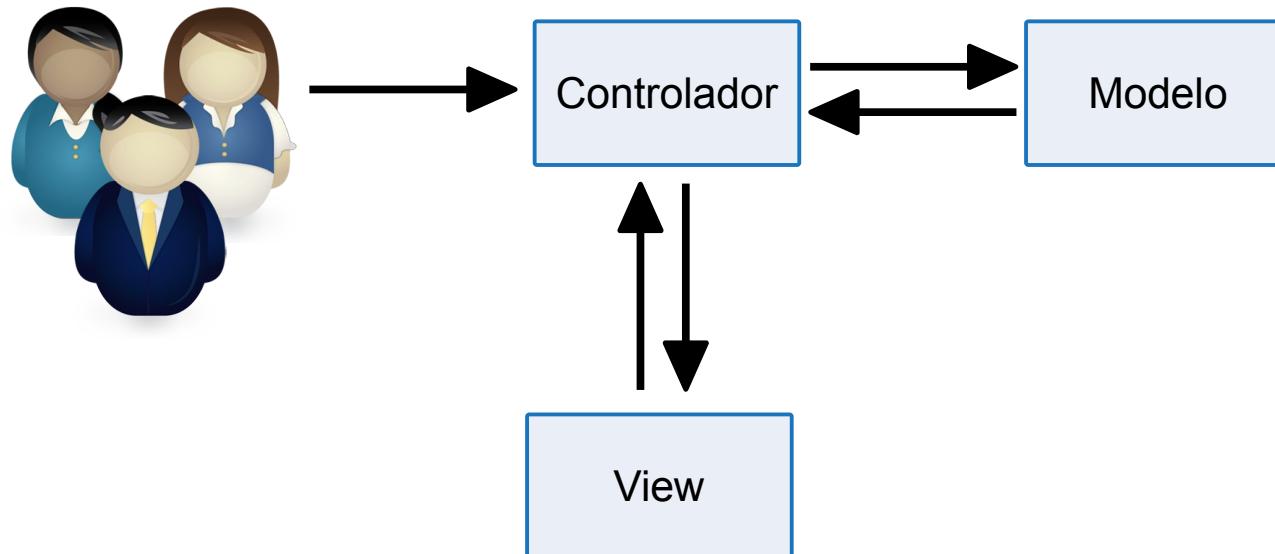




**Prof. David Fernandes de Oliveira  
Instituto de Computação  
UFAM**

# O MVC

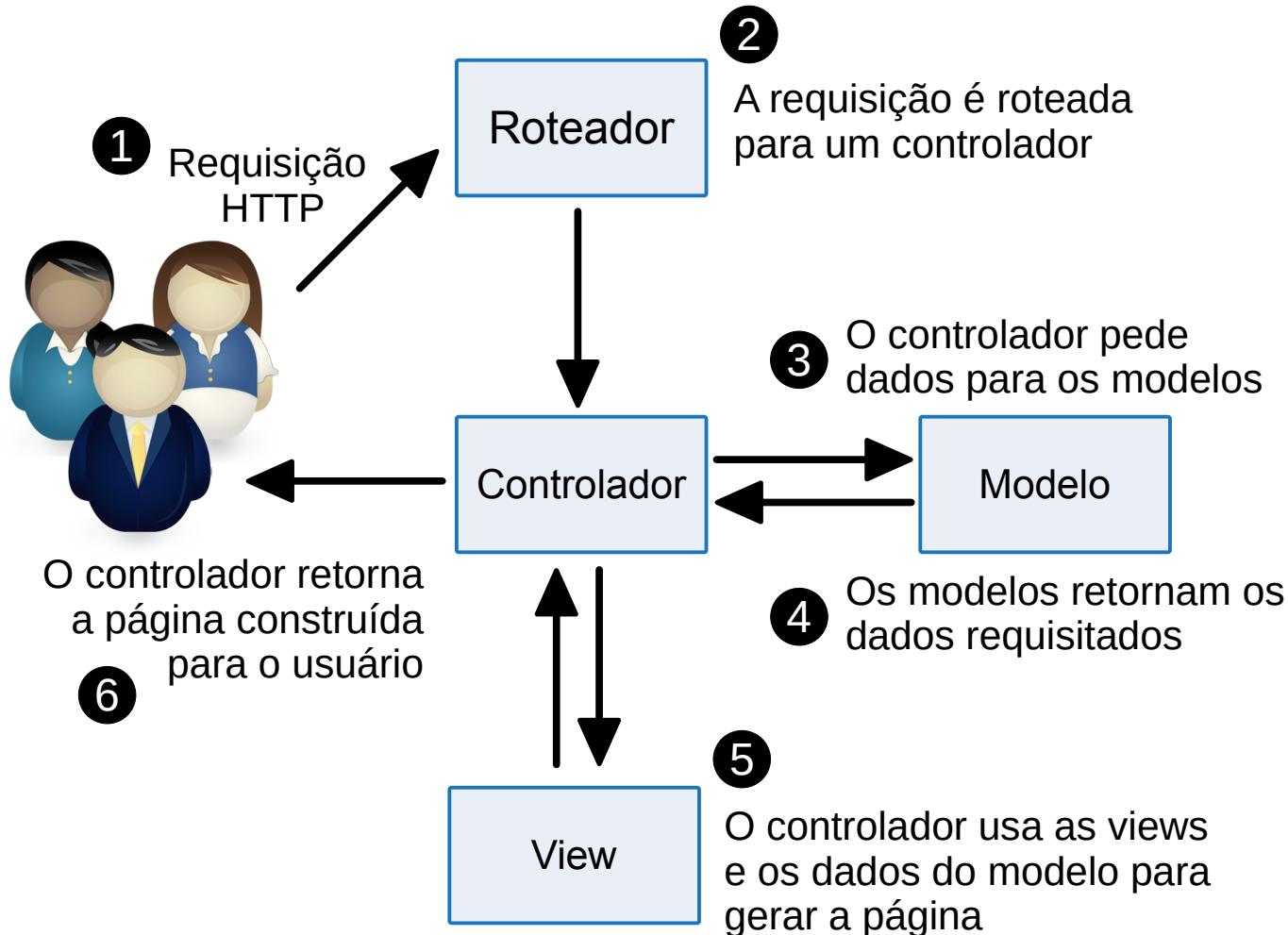
- O MVC é um padrão de arquitetura de software que separa as aplicações em 3 camadas: **Modelos, Views e Controladores**
  - O objetivo de separar a arquitetura nas três camadas é facilitar a organização, compreensão e a manutenção do código
- Frameworks Web MVC: Yii2, Laravel, Sails, Adonis, Django, etc



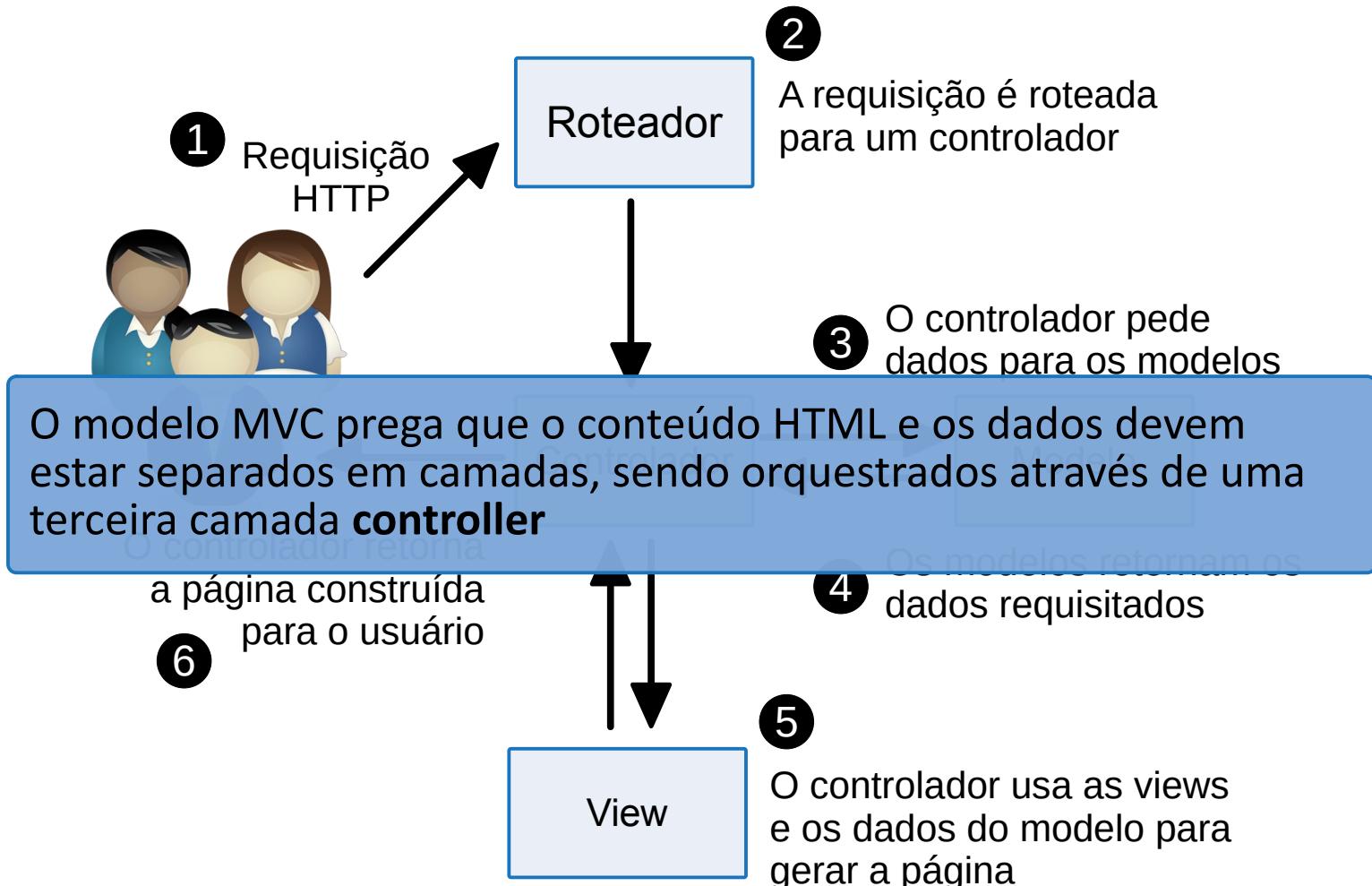
# O MVC

- No modelo MVC, as 3 camadas possuem funções específicas e estão conectadas entre si:
  - **Modelo**, responsável pela leitura e escrita dos dados provenientes do SGBD utilizado pela aplicação
  - **Visão**, responsável por gerar o conteúdo HTML que será enviado para o usuário para que esse possa interagir com a aplicação
  - **Controlador**, responsável por responder as requisições dos usuários, fazendo uso dos modelos e apresentando os resultados através das views

# O MVC

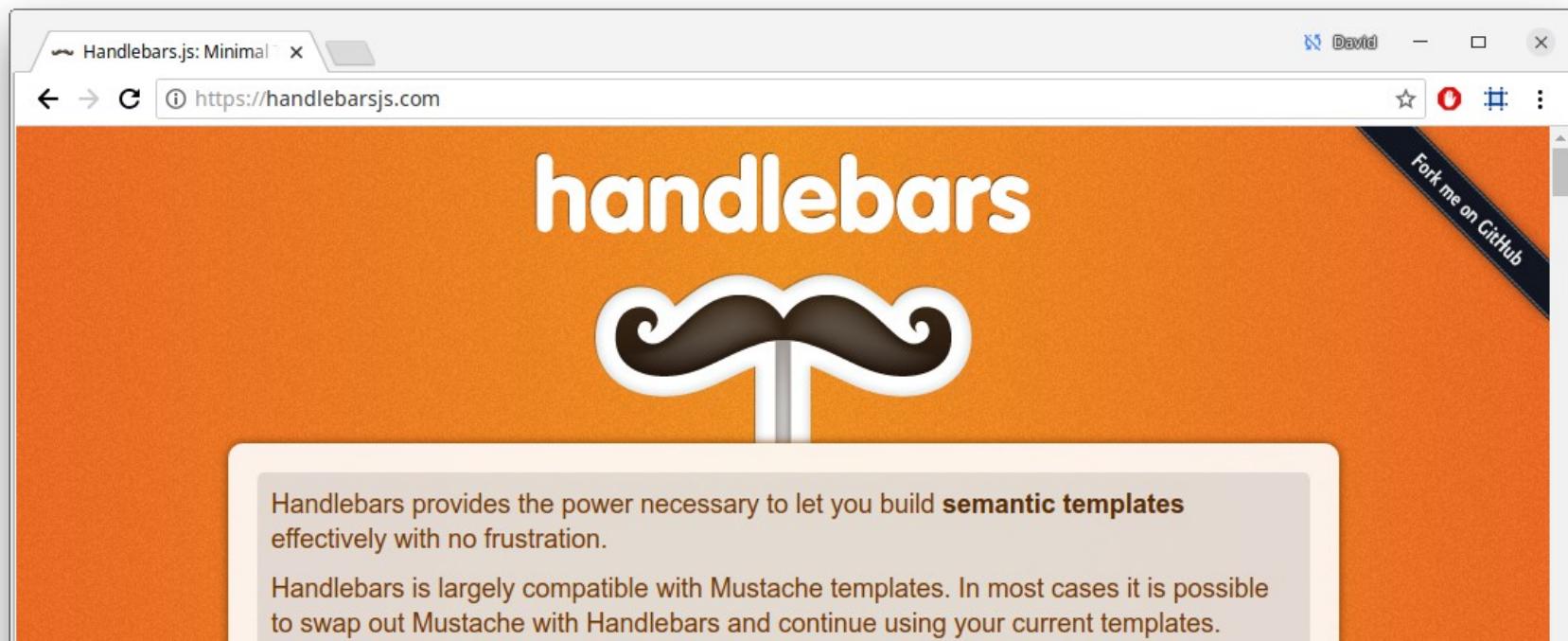


# O MVC



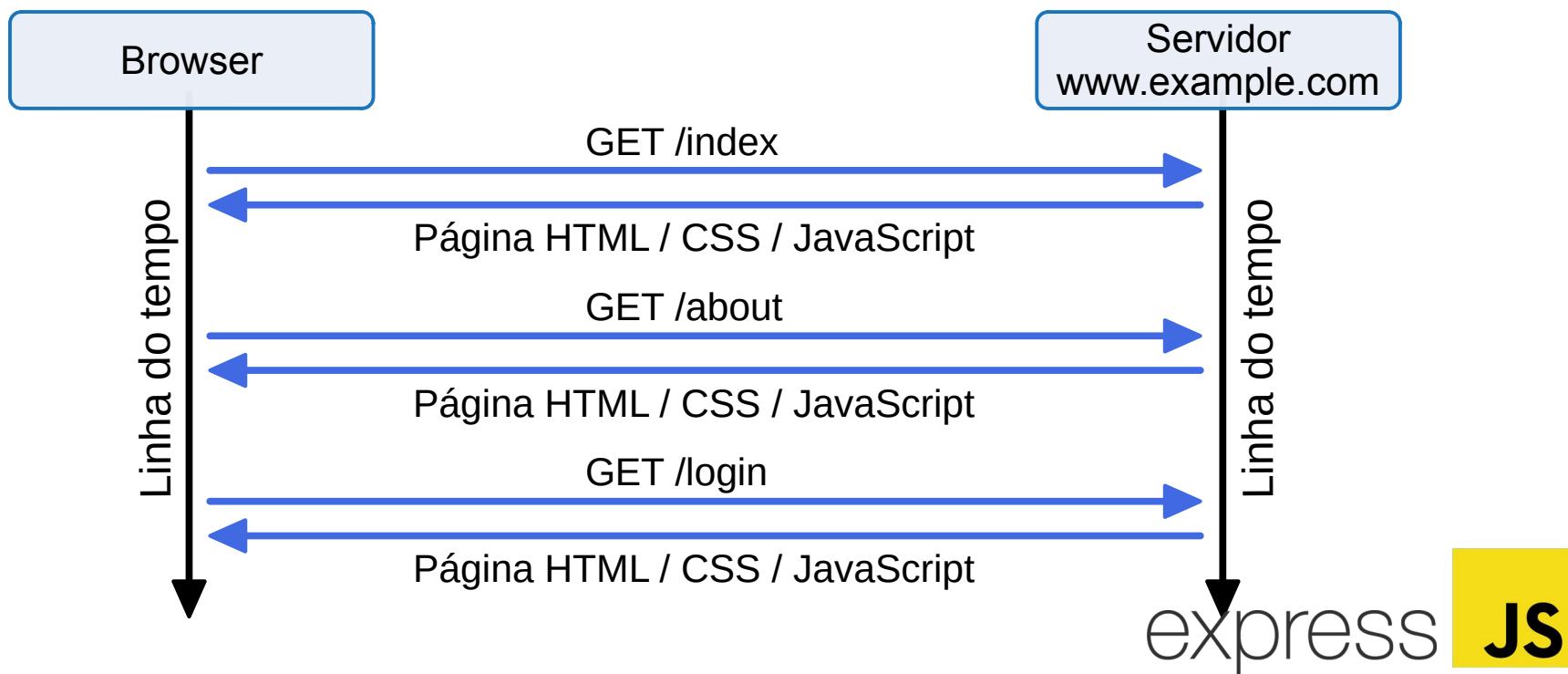
# Views

- 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



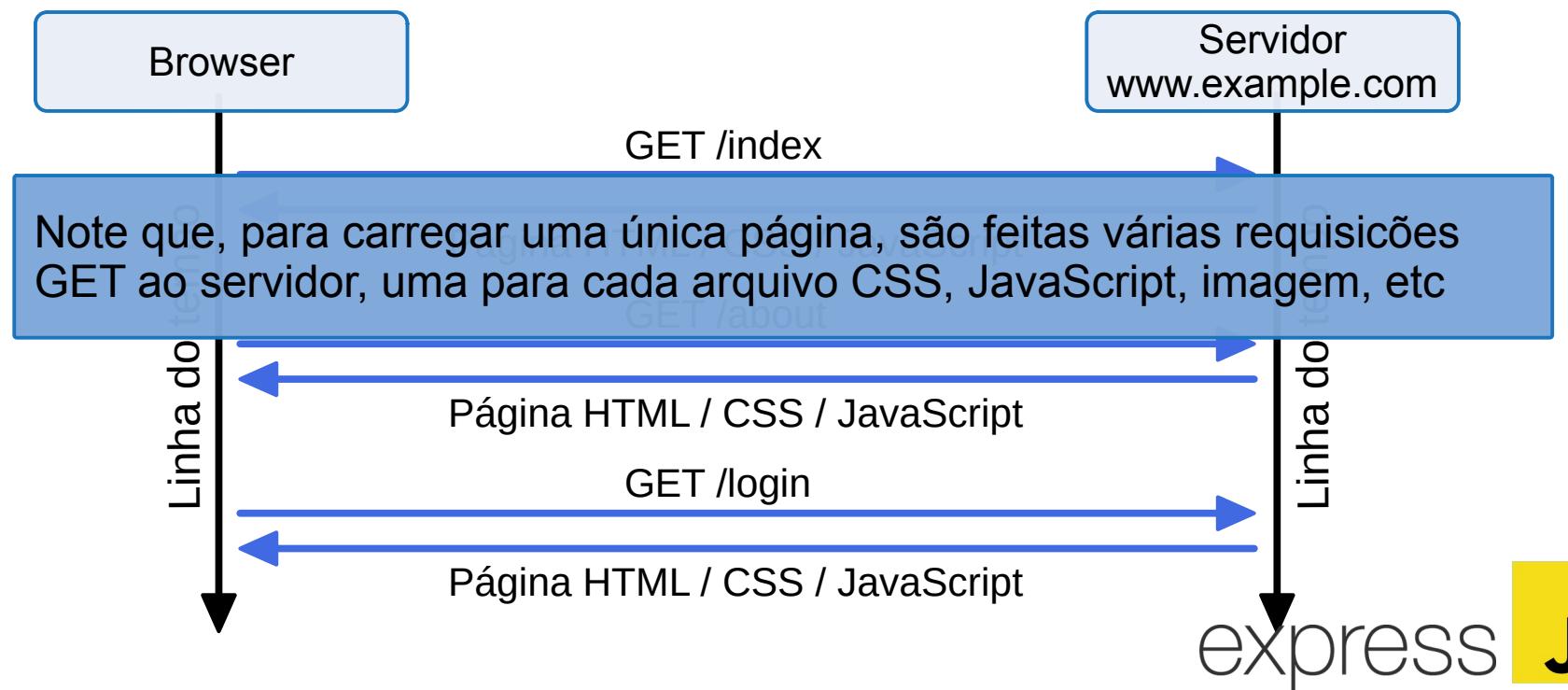
# Sistemas MVC

- Nos **sistemas MVC**, o servidor é responsável por executar a maior parte da lógica da aplicação
- A cada requisição ao sistema, o servidor precisa retornar o todo o conteúdo HTML, CSS e JavaScript do recurso desejado



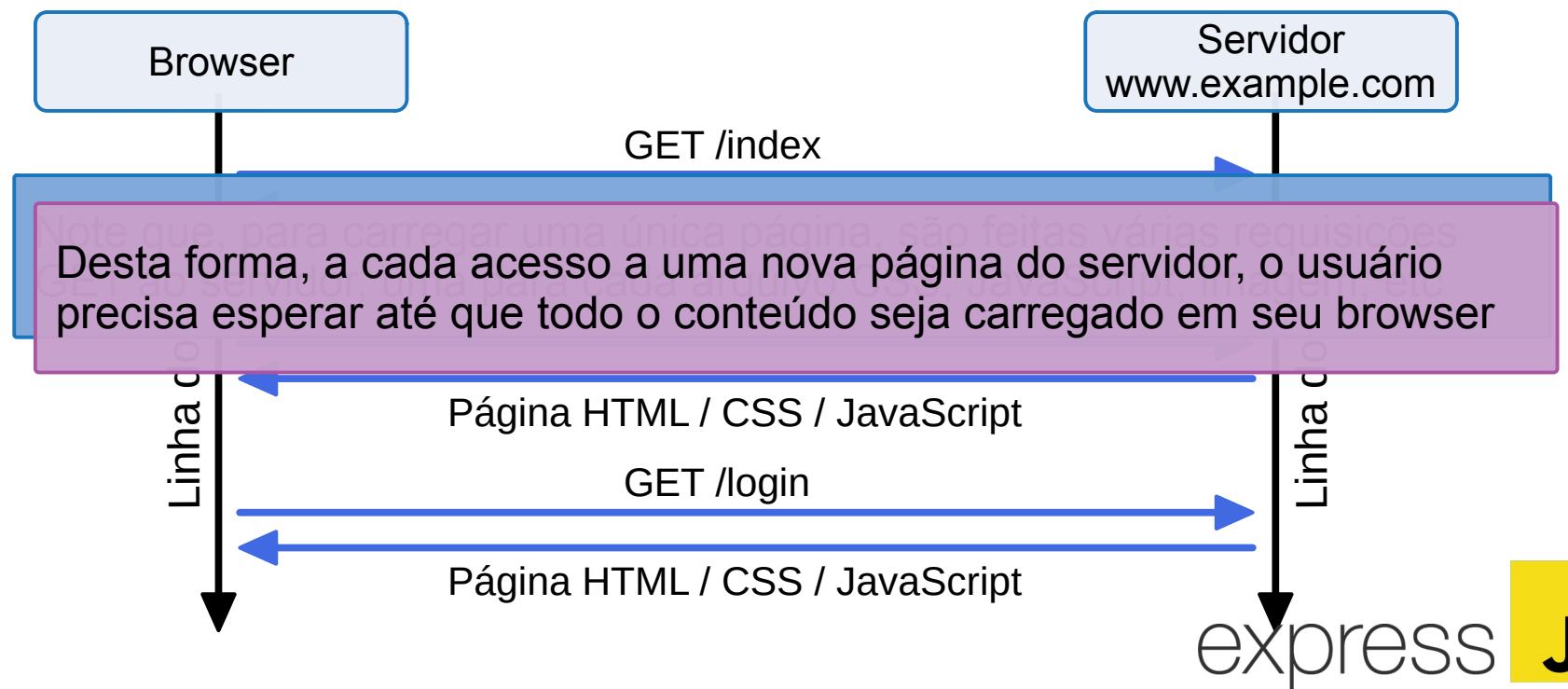
# Sistemas MVC

- Nos **sistemas MVC**, o servidor é responsável por executar a maior parte da lógica da aplicação
- A cada requisição ao sistema, o servidor precisa retornar o todo o conteúdo HTML, CSS e JavaScript do recurso desejado



# Sistemas MVC

- Nos **sistemas MVC**, o servidor é responsável por executar a maior parte da lógica da aplicação
- A cada requisição ao sistema, o servidor precisa retornar o todo o conteúdo HTML, CSS e JavaScript do recurso desejado



# Uma revolução chamada AJAX

- Em 2005, **Jesse Garrett** introduziu o conceito de AJAX, tornando possível a criação de páginas muito mais dinâmicas
- Um dos primeiros grandes sistemas a usar essa nova tecnologia de forma realmente produtiva foi o **Gmail**

The screenshot shows the Gmail inbox interface. At the top, there's a search bar with 'Cari di Surat' and 'Cari di Web' buttons, and links for 'Tampilan pilihan pencarian' and 'Buat filter'. Below the search bar, the message list starts with a header: '1 - 12 dari 12'. The first message is selected (indicated by a checked checkbox) and has a subject: 'Writeboard' followed by a detailed description of a new writeboard created for other purposes. The list continues with various messages from different senders, each with a checkbox and a subject line. At the bottom of the list, there's a note in Indonesian: 'Kini Anda dapat memakai Gmail dalam lebih banyak bahasa! [Ingin tahu?](#)' and a summary: 'Anda sekarang memakai 0 MB (0%) dari kuota Anda sebesar 2654 MB.' There are also links for 'Syarat-syarat Penggunaan', 'Kebijakan Privasi', 'Kebijakan Program', and 'Beranda Google'. The footer includes the text '©2005 Google' and the 'express JS' logo.

ran@gmail.com | Pengelolaan Email | Bantuan | Keluar

Tulis Surat

Surat Masuk

Surat Bintang ★

Surat Keluar

Draft

Semua Surat

Spam

Sampah

Daftar Teman

▼ Label

Perbaiki label

▼ Undang teman

Beri Gmail ke:

liah: Semua, Tak satupun, Baca, Belum Dibaca, Ditandai bintang, Redup

Arsipkan Ini Spam Mau Diapakan? Refresh

Pilih: Semua, Tak satupun, Baca, Belum Dibaca, Ditandai bintang, Redup

1 - 12 dari 12

Writeboard » Your new writeboard: "Hayabusa's board for other purposes™" - A new writeboard has been created for you. Oct 9  
no-reply » Welcome to elYsiun.com Forums - Welcome to elYsiun.com Forums Please keep this email for Oct 9  
Writeboard » Your new writeboard: "Hayabusa's board" - A new writeboard has been created for you: Hayabus Oct 8  
Signup processing for. » Application for Hattrick - Welcome to Hattrick! You have applied for a team called Medan Knight Oct 4  
Administrator » [Gatra.com] Registrasi Anggota - Anda telah mendaftar sebagai anggota di www.gatra.com. Sila Sep 26  
Robin Cherry » Graphis Annual Report & Posters CALL FOR ENTRY - Account.OrganizationName ANNUAL RE Sep 21  
registration » Activate Your FREE latimes.com Account - Do Not Delete - Thank you for registering at latimes Aug 27  
Godote's kickass foru. » Welcome to Godote's kickass forum! - Thank you for registering in the forums at http://www.god Aug 26  
livedoor Wiki (2) » livedoor Wiki - 新規作成完了のお知らせ - livedoor Wiki(ウィキ)からのお知らせ http://wiki.livedo Aug 23  
livedoor (2) » livedoor ID 登録完了のお知らせ - 洪 揭揚様 ... Aug 23  
info » Seesaa サービスへようこそ！ - このメールは、大切に保管してください。Seesaa サービスをご利用頂 Aug 16  
Gmail Team » Gmail is different. Here is what you need to know. Text in "Subject:" line. - First of all, welcome Jul 9

Kini Anda dapat memakai Gmail dalam lebih banyak bahasa! [Ingin tahu?](#)

Anda sekarang memakai 0 MB (0%) dari kuota Anda sebesar 2654 MB.

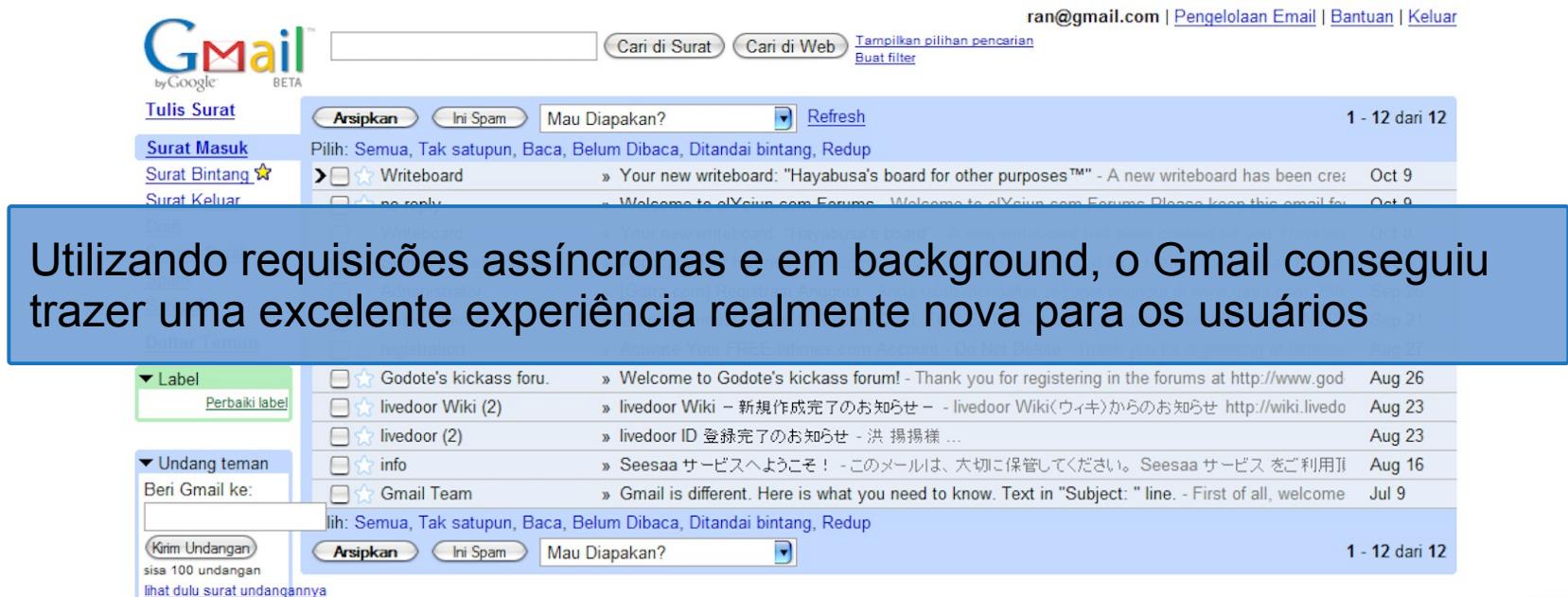
[Syarat-syarat Penggunaan](#) - [Kebijakan Privasi](#) - [Kebijakan Program](#) - [Beranda Google](#)

©2005 Google

express JS

# Uma revolução chamada AJAX

- Em 2005, **Jesse Garrett** introduziu o conceito de AJAX, tornando possível a criação de páginas muito mais dinâmicas
- Um dos primeiros grandes sistemas a usar essa nova tecnologia de forma realmente produtiva foi o **Gmail**



The screenshot shows the early version of the Gmail interface. At the top, there's a search bar with 'Cari di Surat' and 'Cari di Web' buttons, and links for 'Tampilan pilihan pencarian' and 'Buat filter'. Below the search bar, there are buttons for 'Arsipkan', 'Ini Spam', and 'Mau Diapakan?'. A message summary is displayed: 'Pilih: Semua, Tak satupun, Baca, Belum Dibaca, Ditandai bintang, Redup'. The main area shows 12 messages, with the first one being a writeboard message from 'Writeboard' with a subject like 'Your new writeboard: "Hayabusa's board for other purposes™" - A new writeboard has been created'. The interface is clearly based on the Google Mail design of the time.

Utilizando requisições assíncronas e em background, o Gmail conseguiu trazer uma excelente experiência realmente nova para os usuários

Kini Anda dapat memakai Gmail dalam lebih banyak bahasa! [Ingin tahu?](#)

Anda sekarang memakai 0 MB (0%) dari kuota Anda sebesar 2654 MB.

[Syarat-syarat Penggunaan](#) - [Kebijakan Privasi](#) - [Kebijakan Program](#) - [Beranda Google](#)

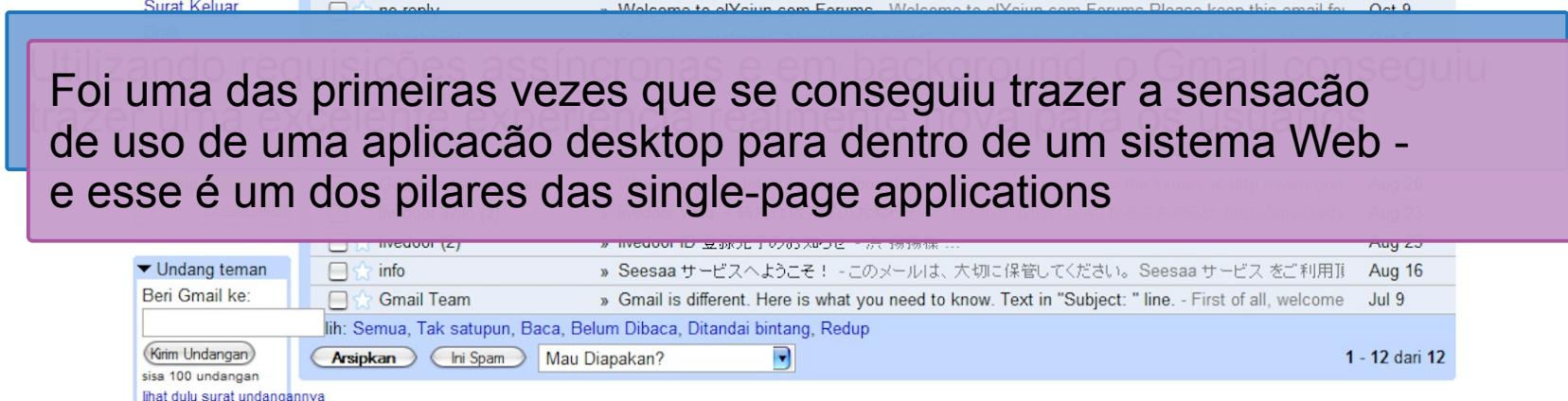
©2005 Google

# Uma revolução chamada AJAX

- Em 2005, **Jesse Garrett** introduziu o conceito de AJAX, tornando possível a criação de páginas muito mais dinâmicas
- Um dos primeiros grandes sistemas a usar essa nova tecnologia de forma realmente produtiva foi o **Gmail**



Foi uma das primeiras vezes que se conseguiu trazer a sensação de uso de uma aplicação desktop para dentro de um sistema Web - e esse é um dos pilares das single-page applications



Kini Anda dapat memakai Gmail dalam lebih banyak bahasa! [Ingin tahu?](#)

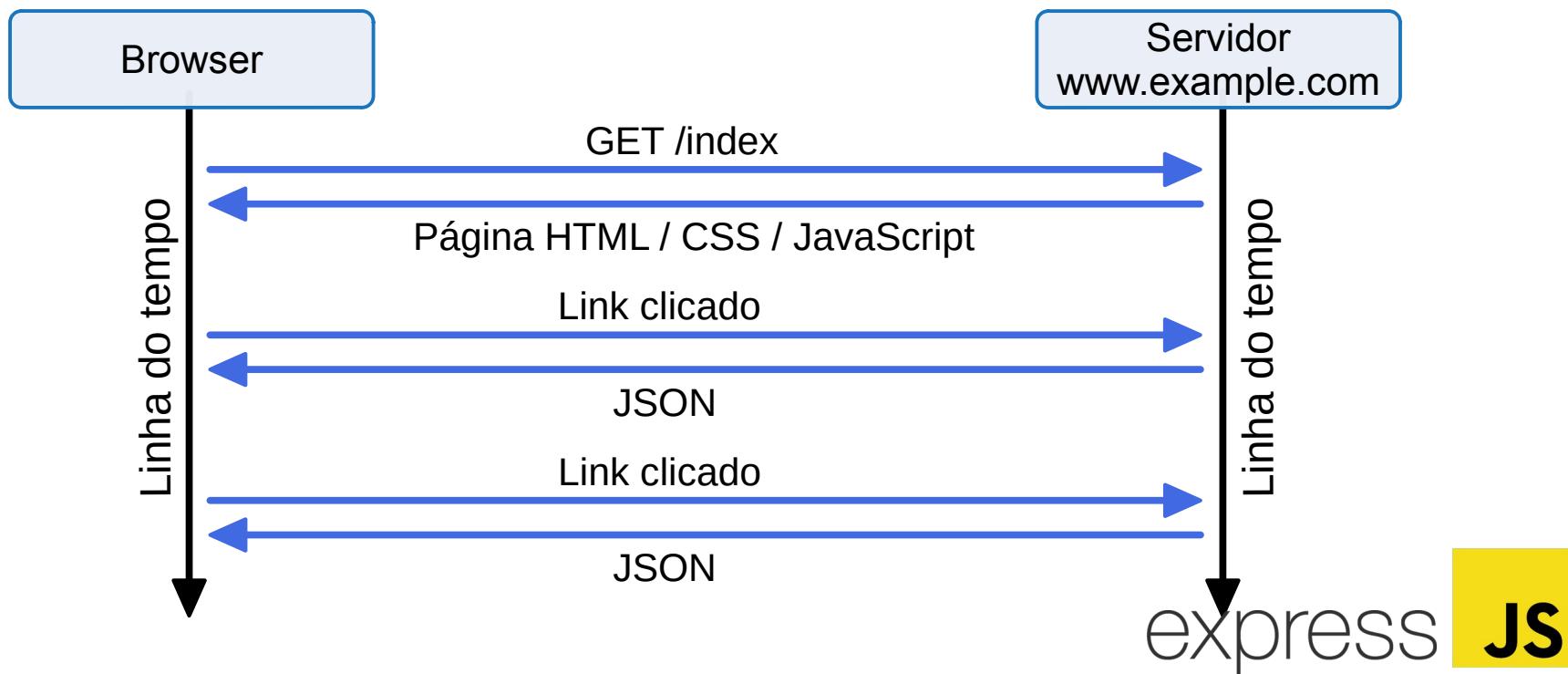
Anda sekarang memakai 0 MB (0%) dari kuota Anda sebesar 2654 MB.

[Syarat-syarat Penggunaan](#) - [Kebijakan Privasi](#) - [Kebijakan Program](#) - [Beranda Google](#)

©2005 Google

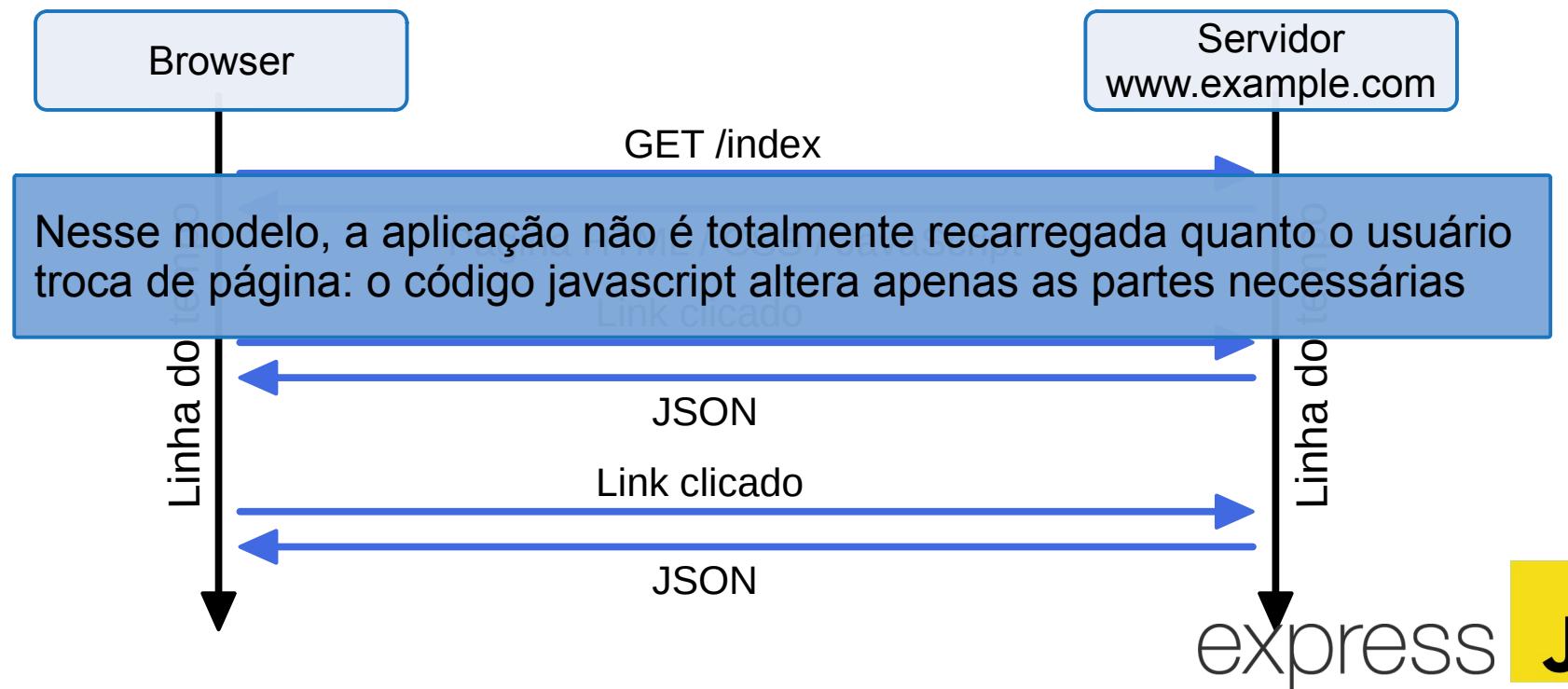
# Single-Page Applications

- Um SPA é uma aplicação web que roda em uma única página, de uma forma similar à uma aplicacao desktop ou mobile
- Executam a maior parte da lógica da aplicacão no browser, comunicando-se com o servidor através de APIs



# Single-Page Applications

- Um SPA é uma aplicação web que roda em uma única página, de uma forma similar à uma aplicacao desktop ou mobile
- Executam a maior parte da lógica da aplicacão no browser, comunicando-se com o servidor através de APIs



# Vantagens e Desvantagens

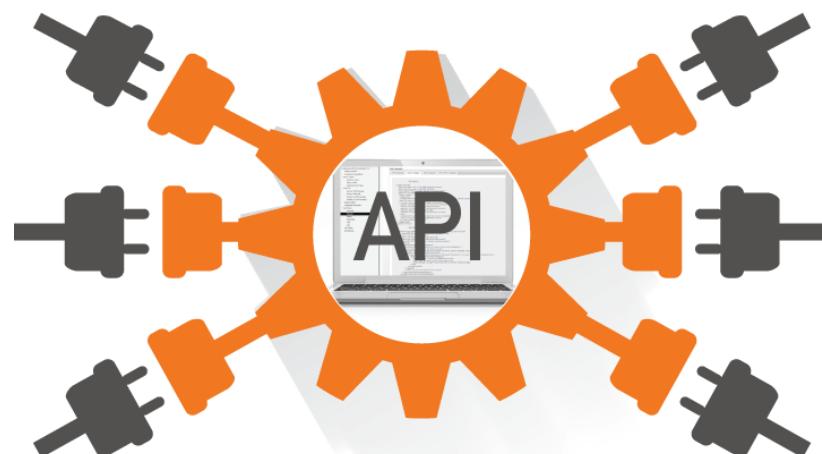
- Vantagens das **Single-Page Applications**
  - Páginas mais reativas, interação com o usuário mais fluida
  - Alto desacoplamento entre backend e frontend
  - Vários frameworks para o frontend
- Desvantagens
  - Requer uma política de **Search Engine Optimization** diferenciada
  - Carregamento inicial com muito mais código
  - Requer conhecimentos sólidos de programação JavaScript
  - Perigo de descontinuidade das bibliotecas usadas, ou geração de novas versões incompatíveis com as anteriores

# Vantagens e Desvantagens

- Vantagens dos **Sistemas Web Tradicionais**
  - Técnicas mais consolidadas
  - **Search Engine Optimization** mais simples
  - Mais fácil de implementar
  - Menor acoplamento com código javascript no lado cliente
- Desvantagens:
  - Experiência de usuário inferior, pois todo o conteúdo da página é recarregada a cada nova requisição
  - Forte acoplamento dentre frontend e backend
  - Arquitetura defasada

# REST APIs

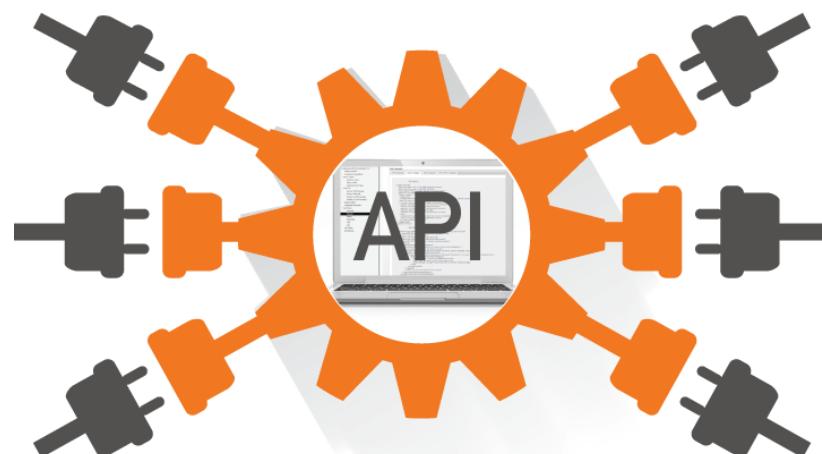
- O REST – Representational State Transfer – é caracterizado como um paradigma de desenvolvimento de software semelhante aos webservices
  - Nesse paradigma, um serviço (normalmente chamado de API) é fornecido para acesso e manipulação dos dados de uma aplicação
- API – Application Programming Interface – é um conjunto de rotinas usadas na comunicação entre duas partes de uma aplicação



# REST APIs

- O REST – Representational State Transfer – é caracterizado como um paradigma de desenvolvimento de software semelhante aos webservices
  - Nesse paradigma, um serviço (normalmente chamado de API) é fornecido para acesso e manipulação dos dados de uma aplicação
- O Front, que consome os dados da API, pode ser desenvolvido através de vários tipos de frameworks frontend, como o **React**, **Angular** e o **Vue.js**

aplicação



# Sistema de Loja Virtual

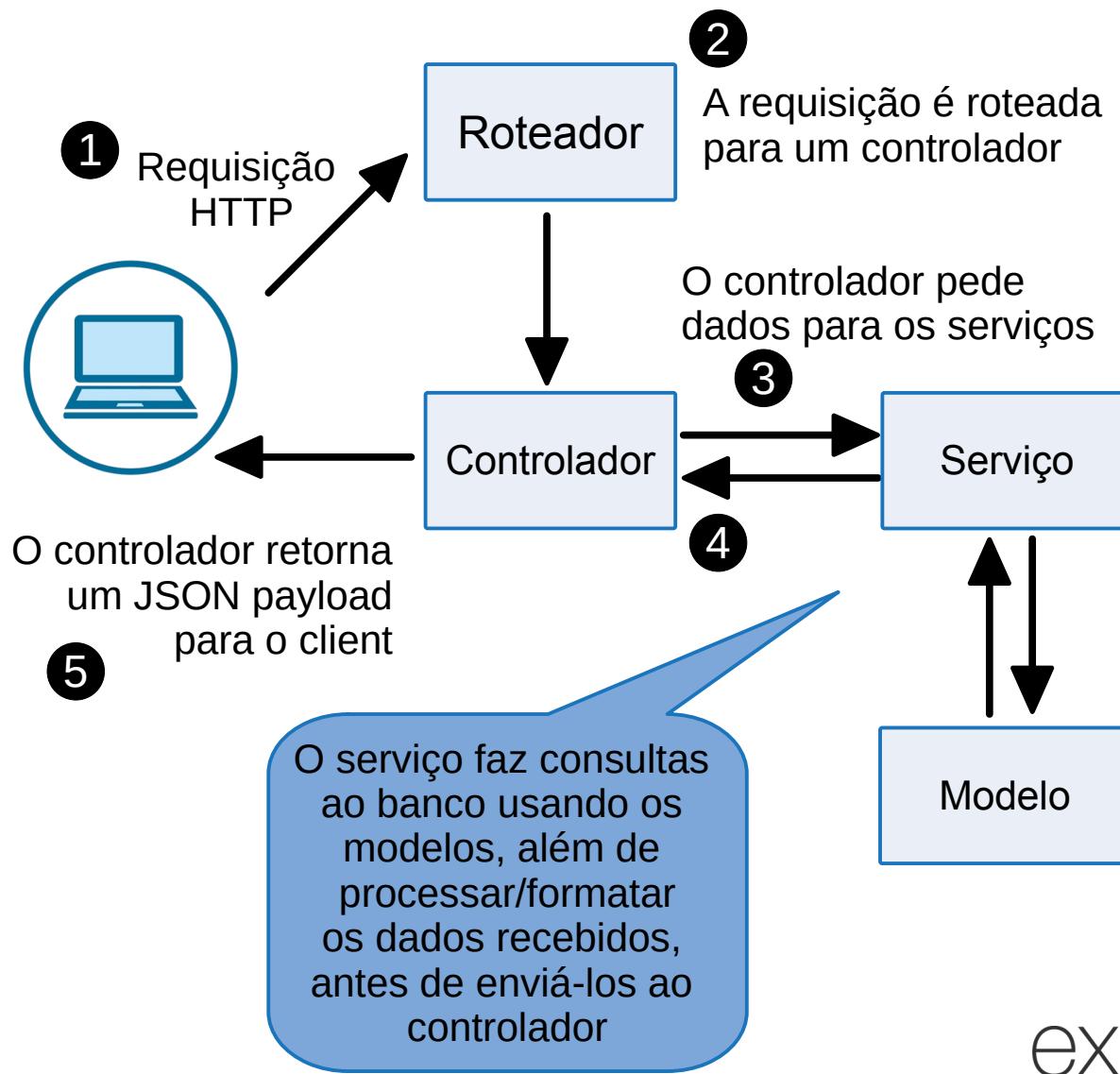
- Como prática de desenvolvimento nesta etapa do curso, cada aluno irá desenvolver um sistema SPA para uma loja virtual
- Além da API, que será desenvolvida no presente módulo, a loja virtual também terá um frontend, testes integrados, além de uma estratégia de CI/CD



# Movendo para o padrão REST

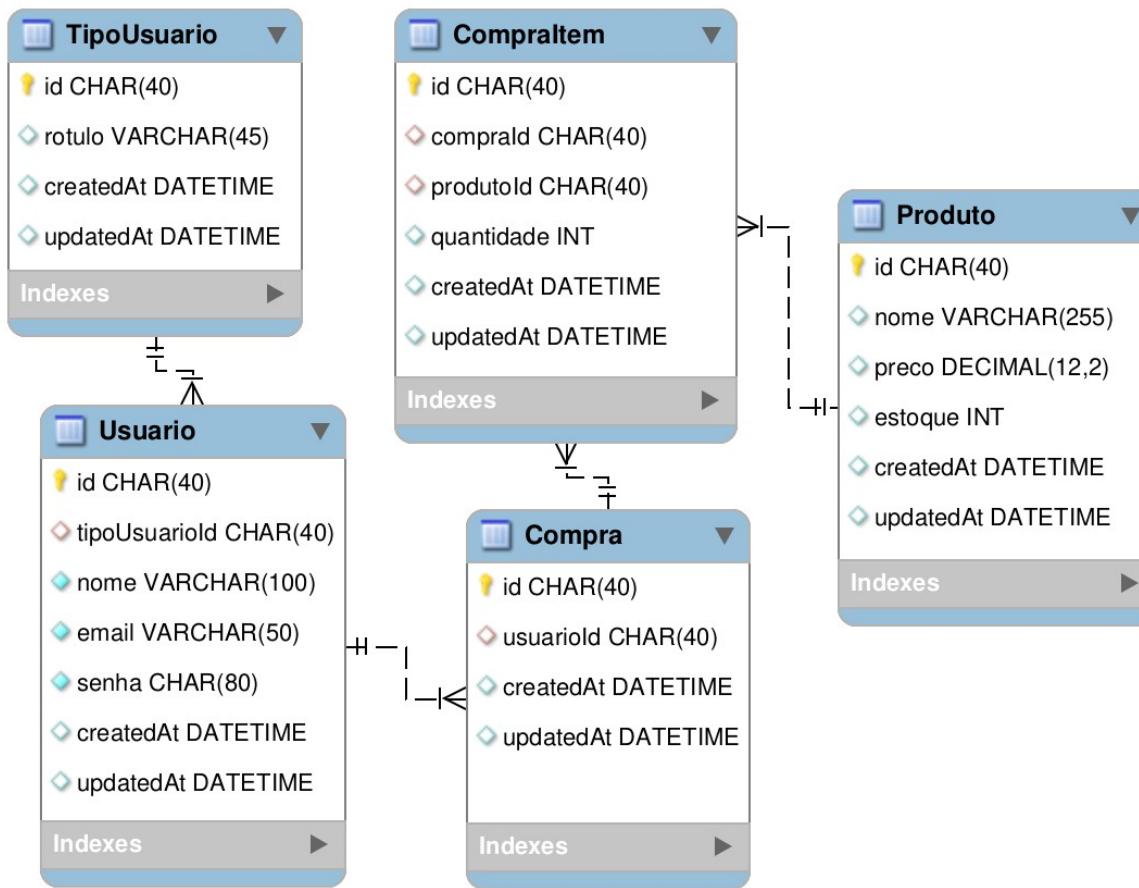
- Nas aplicações MVC, cada página é construída através de uma action de um dado controlador
  - Por exemplo a **função** `about` do controlador **main** tem por objetivo construir e retornar o conteúdo HTML da página `/about`
- Nas aplicações REST, por outro lado, as páginas de uma aplicação são definidas no lado Front e não no lado Back
  - O Back nesse caso é responsável por responder a chamadas HTTP do Front, realizando os processos de negócio e de persistência que sejam pertinentes a cada situação

# Movendo para o padrão REST



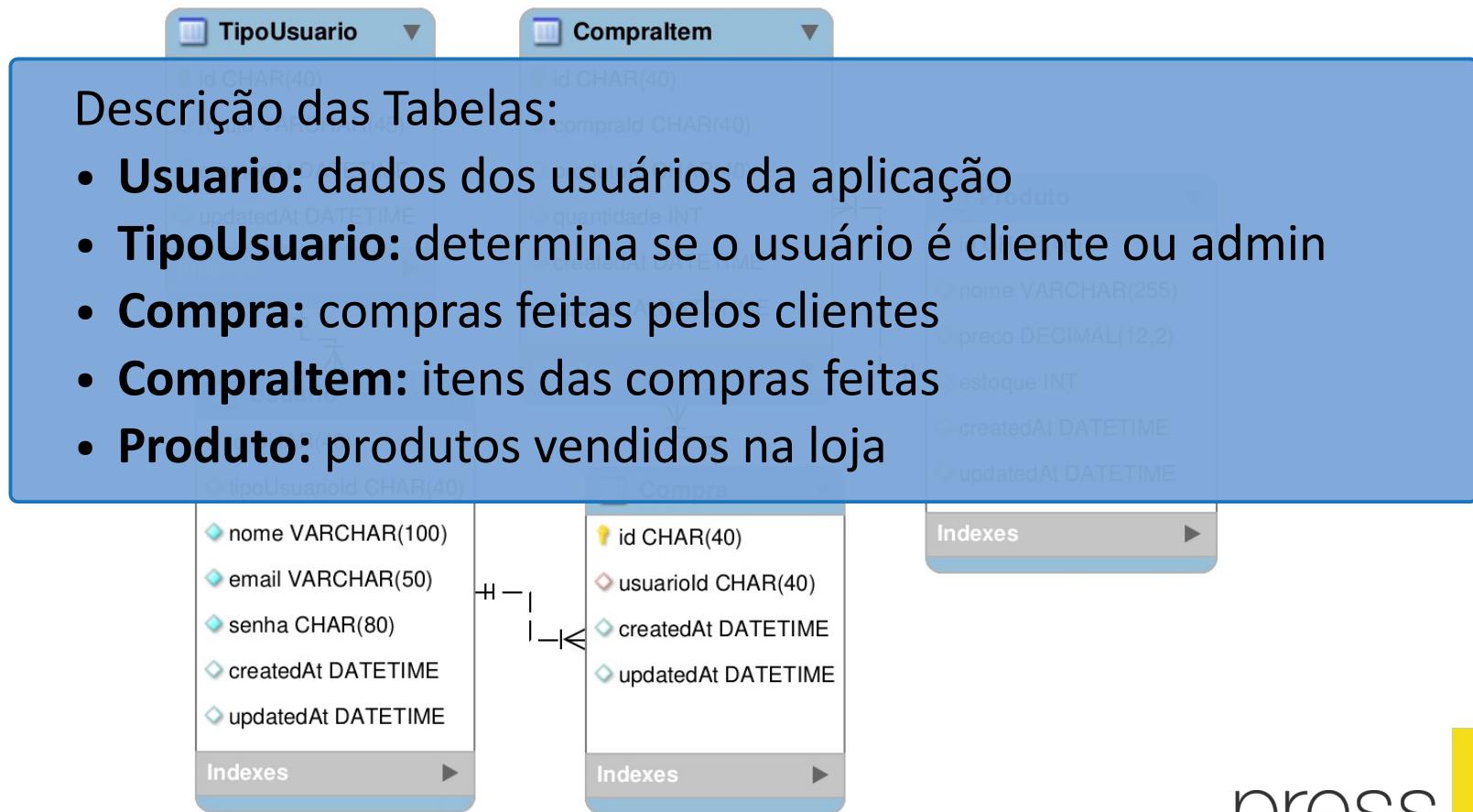
# Esquema de banco de dados e ORM

- Nossa aplicação usará o ORM Prisma, e os modelos serão definidos no diretório **prisma/schema.prisma**



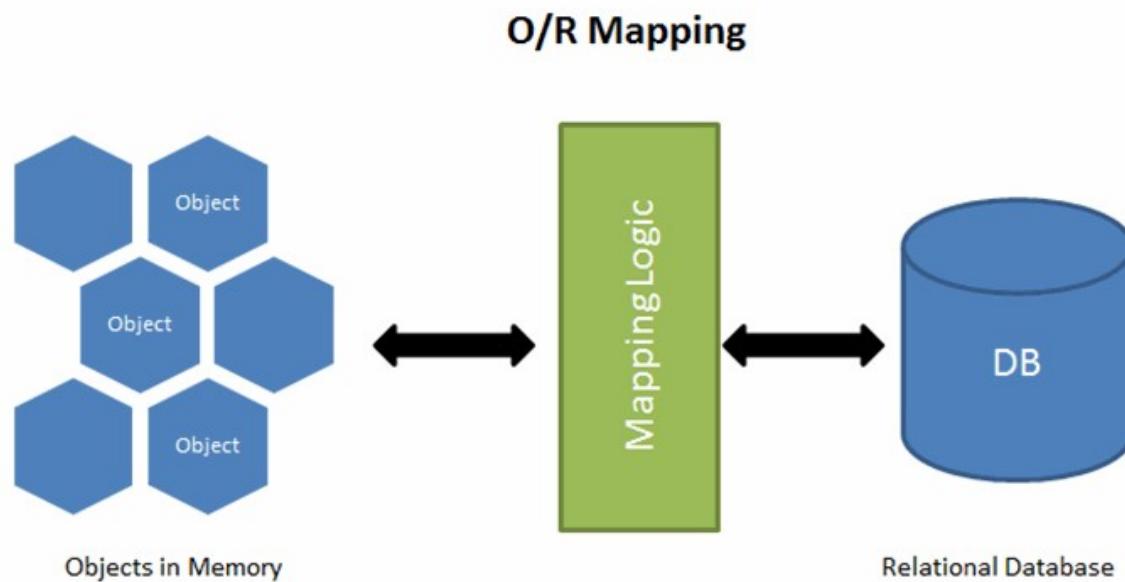
# Esquema de banco de dados e ORM

- Nossa aplicação usará o ORM Prisma, e os modelos serão definidos no diretório **prisma/schema.prisma**

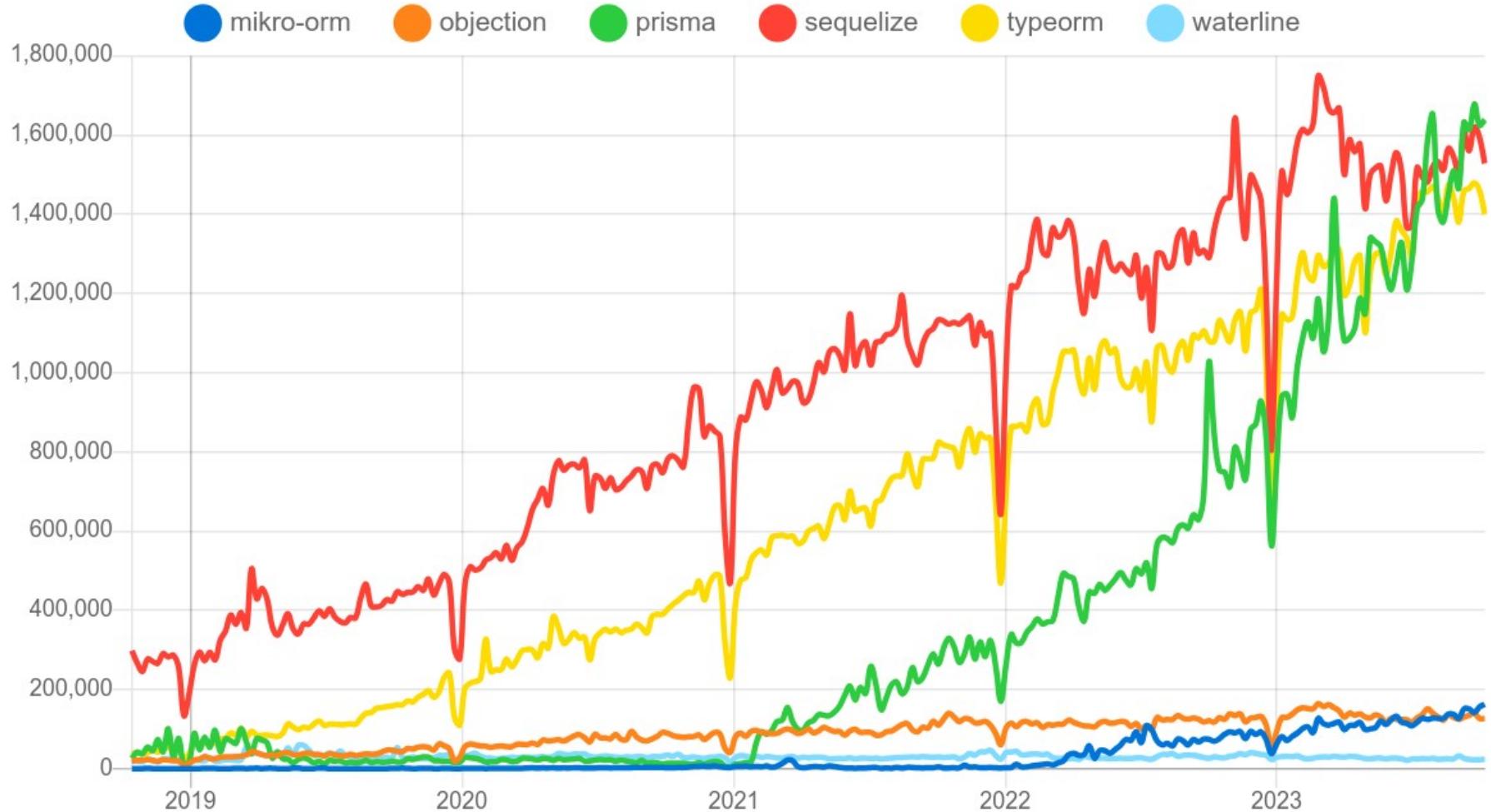


# Object Relational Mapper - ORM

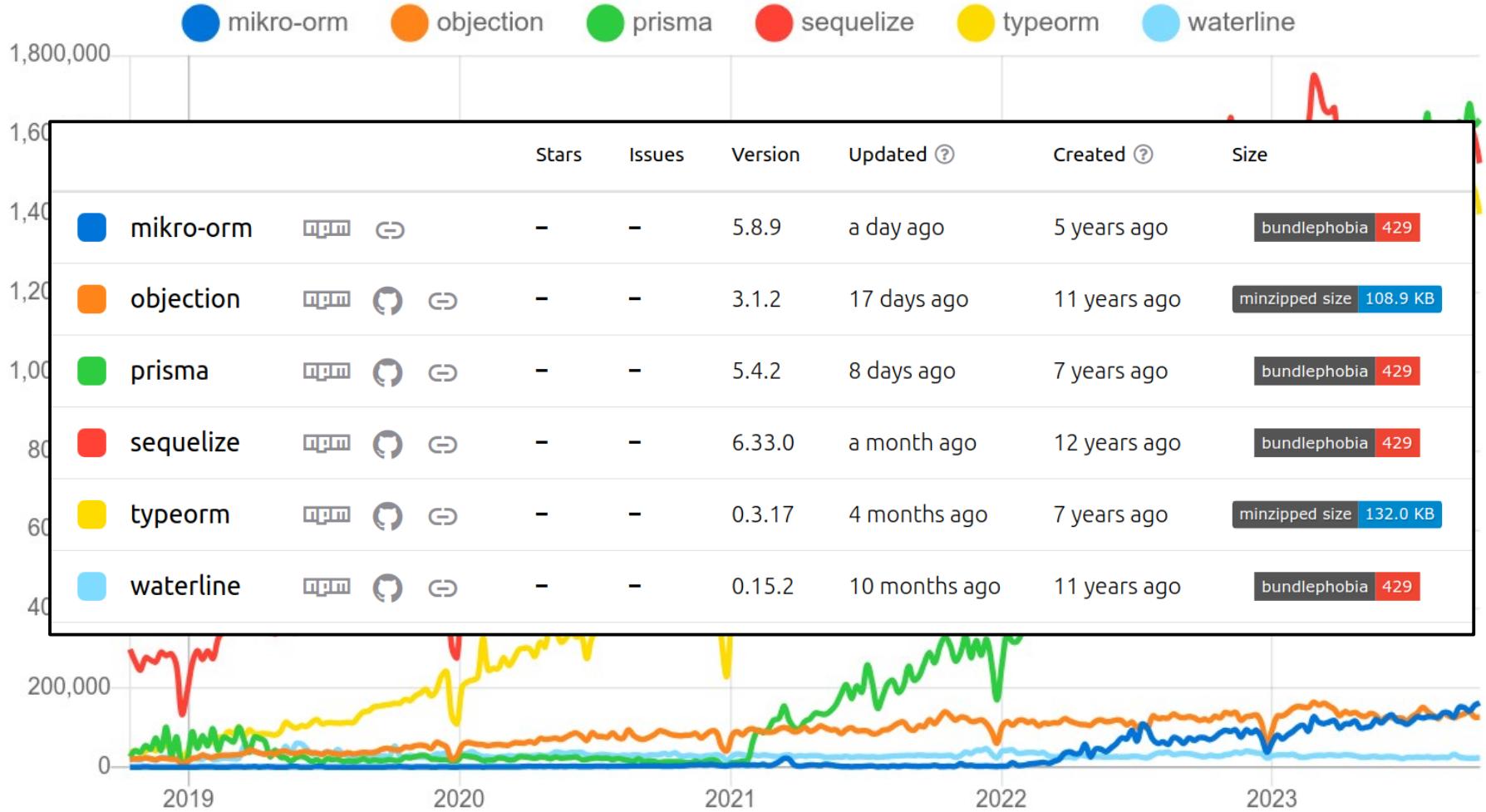
- ORM é uma técnica que permite **consultar e manipular dados** de um database usando o paradigma de orientação a objetos
- Desta forma, o acesso aos dados não é feito através da linguagem SQL, e sim através de objetos



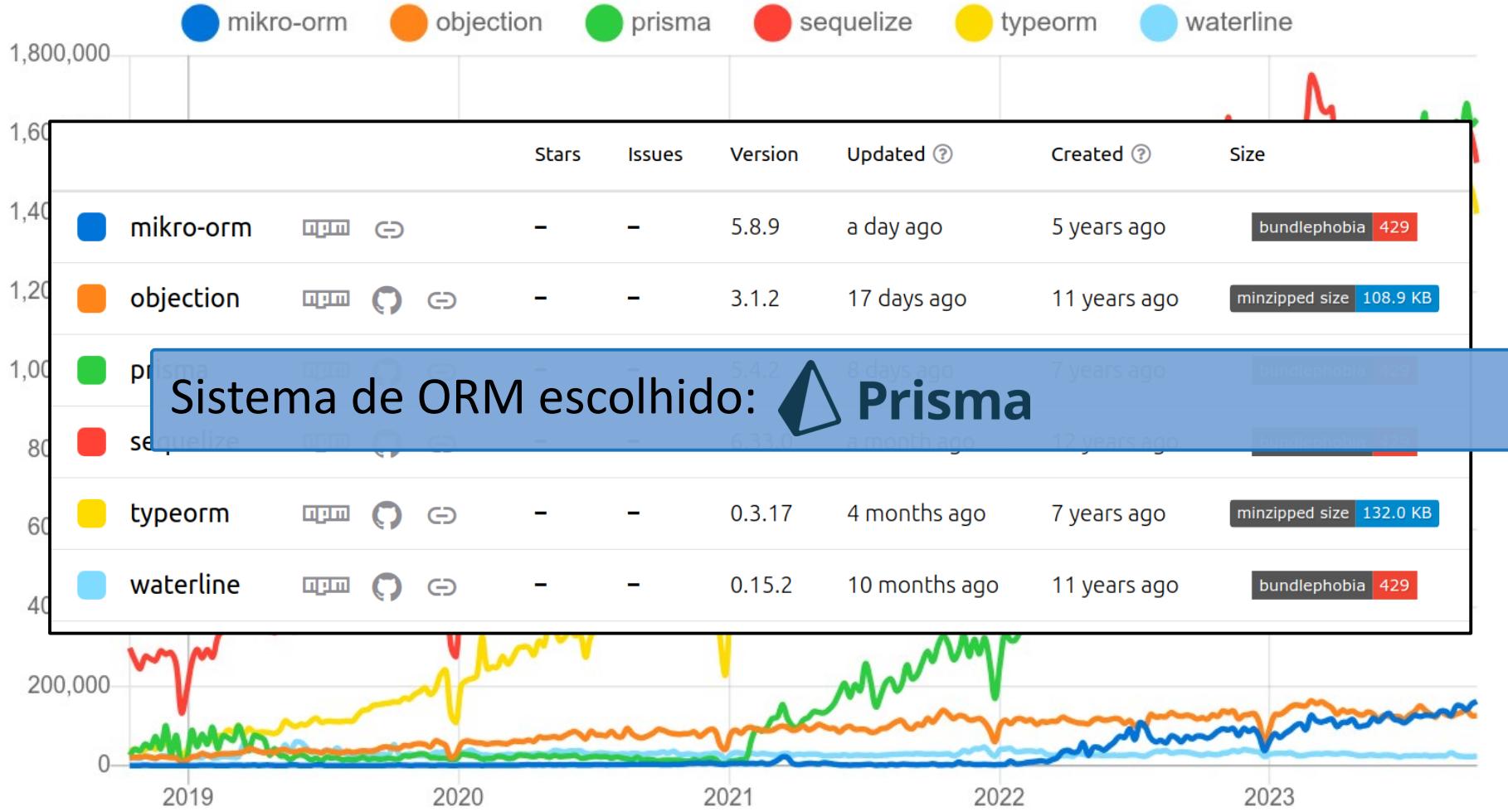
# Escolhendo o ORM



# Escolhendo o ORM



# Escolhendo o ORM





- Use os comandos abaixo para instalar o **Prisma** como dependência de desenvolvimento

```
$ npm install -D prisma
```

- Após isso, execute o comando abaixo para que o Prisma crie os arquivos iniciais de configuração

```
$ npx prisma init
```



~/d/e/backend



david@coyote ~/dev/expApi/backend [main]x \$ npx prisma init

✓ Your Prisma schema was created at `prisma/schema.prisma`

You can now open it in your favorite editor.

warn You already have a `.gitignore` file. Don't forget to add `'.env'` in it to not commit any private information.

\$ npm install -D prisma

Next steps:

1. Set the `DATABASE URL` in the `.env` file to point to your existing database. If your database has no tables yet, read <https://pris.ly/d/getting-started>
2. Set the `provider` of the `datasource` block in `schema.prisma` to match your database: `postgresql`, `mysql`, `sqlite`, `sqlserver`, `mongodb` or `cockroachdb`.
3. Run `prisma db pull` to turn your database schema into a Prisma schema.
4. Run `prisma generate` to generate the Prisma Client. You can then start querying your database.

More information in our documentation:

<https://pris.ly/d/getting-started>

david@coyote ~/dev/expApi/backend [main]x \$



- O comando **prisma init** adiciona uma variável DATABASE\_URL no arquivo **.env**, contendo uma string de conexão com o banco
- Será preciso editar o valor dessa variável conforme a realidade do banco de dados utilizado

```
DATABASE_URL="mysql://user:senha123@localhost:3306/loja"
```

SGBD

usuário  
do  
banco

senha  
do  
usuário

host

porta

nome  
do  
banco



- O comando **prisma init** também cria um arquivo **prisma/schema.prisma**, onde serão criados os modelos da aplicação
- Nesse arquivo, é importante alterar o **provider** para **mysql**, conforme mostrado abaixo

```
generator client {  
  provider = "prisma-client-js"  
}  
  
datasource db {  
  provider = "mysql"  
  url      = env("DATABASE_URL")  
}
```

Mudar o  
**provider**  
para mysql



# Prisma

Extension: Prisma - wa-api - Visual Studio Code

File Edit Selection View Go Run Terminal Help

EXTEN... 🔍 ⌂ ⌂ ...

schema.prisma U Extension: Prisma X

prisma

**Prisma** v5.4.2 46ms  
Adds syntax highlighting, formating,...  
Prisma 24

**Prisma** v5.4.2 2K  
Quernest Install

**Prisma - Insider** 89K  
This is the Insider Build...  
Prisma Install

**Prisma NextJS** 32K  
Automagically add types...  
William Luke Install

**Prisma Import** 3K  
Adds import statement...  
Prisma

DETAILS FEATURE CONTRIB

Prisma VS Code Extension

Adds syntax highlighting, linting, code completion, formatting, just-in-time-definition and more for Prisma

Languages

Extension

Go Live

...

main\* 🔍 ⌂ ⌂ 0 ⚠ 0 {..} : 0 ⌂

Para usar o Prisma,  
é fortemente  
recomendado instalar  
a extensão Prisma,  
do Visual Studio

express **JS**

# Adicionando um modelo

- Ainda no arquivo **prisma/schema.prisma**, vamos adicionar o primeiro modelo de nossa aplicação

```
model Produto {  
    id      String  @id @default(uuid()) @db.Char(40)  
    nome   String  @unique @db.VarChar(100)  
    preco   Decimal @db.Decimal  
    estoque Int     @db.Int  
    createdAt DateTime @default(now()) @map("created_at")  
    updatedAt DateTime @updatedAt @map("updated_at")  
  
    @@map("produtos")  
}
```

# Adicionando um modelo

- Após isso, usamos o comando **npx prisma migrate dev --name** para gerar a migração e criar a tabela produtos

```
david@coyote ~/dev/expApi/backend [main]x $ npx prisma migrate dev --name create-produto-table
Environment variables loaded from .env
Prisma schema loaded from prisma/schema.prisma
Datasource "db": MySQL database "loja" at "localhost:3306"

Applying migration `20231019144451_create_produto_table`

The following migration(s) have been created and applied from new schema changes:

migrations/
└─ 20231019144451_create_produto_table/
    └─ migration.sql

Your database is now in sync with your schema.

✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 63ms
```

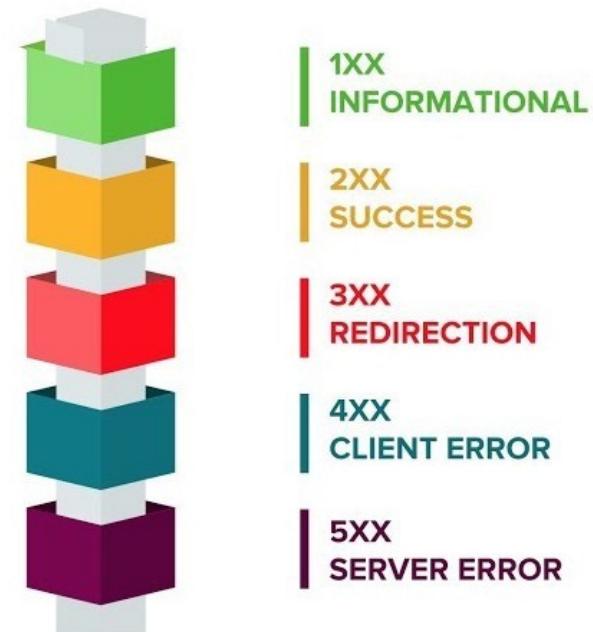
# Elementos de uma Requisição

- O **endpoint** é o caminho usado para fazer uma requisição, possuindo um **resource** e opcionalmente uma **query string**

- O **método HTTP** define o tipo de ação desejada pela requisição, sendo que os métodos mais usados são:
    - **Get**, usado para buscar dados do servidor
    - **Post**, usado para enviar dados para o servidor
    - **Put e Patch**, usado para atualizar dados
    - **Delete**, usado para apagar registros no servidor

# Elementos de uma Requisição

- O **body** é o corpo da mensagem enviada na requisição, e é usado apenas com os métodos POST, PUT e PATCH
- Os **HTTP status codes** servem para indicar se uma requisição HTTP foi corretamente concluída
- Os principais códigos utilizados para as respostas de um endpoint são o 200 (OK), o 201 (CREATED), o 204 (NO CONTENT), o 404 (NOT FOUND) e o 400 (BAD REQUEST).



HTTP Cats

http.cat

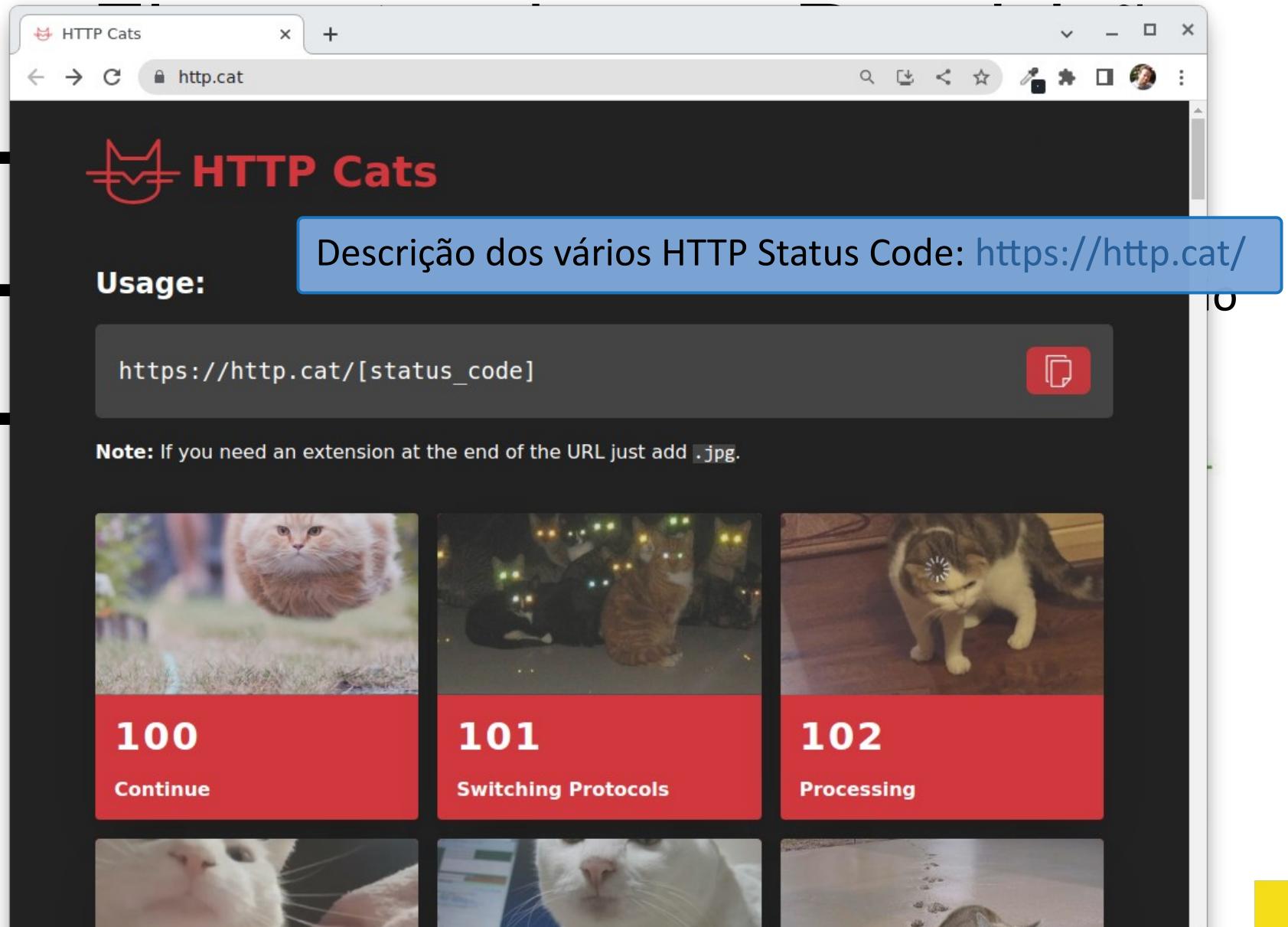
# HTTP Cats

Usage:

Descrição dos vários HTTP Status Code: <https://http.cat/>

https://http.cat/[status\_code]

Note: If you need an extension at the end of the URL just add .jpg.

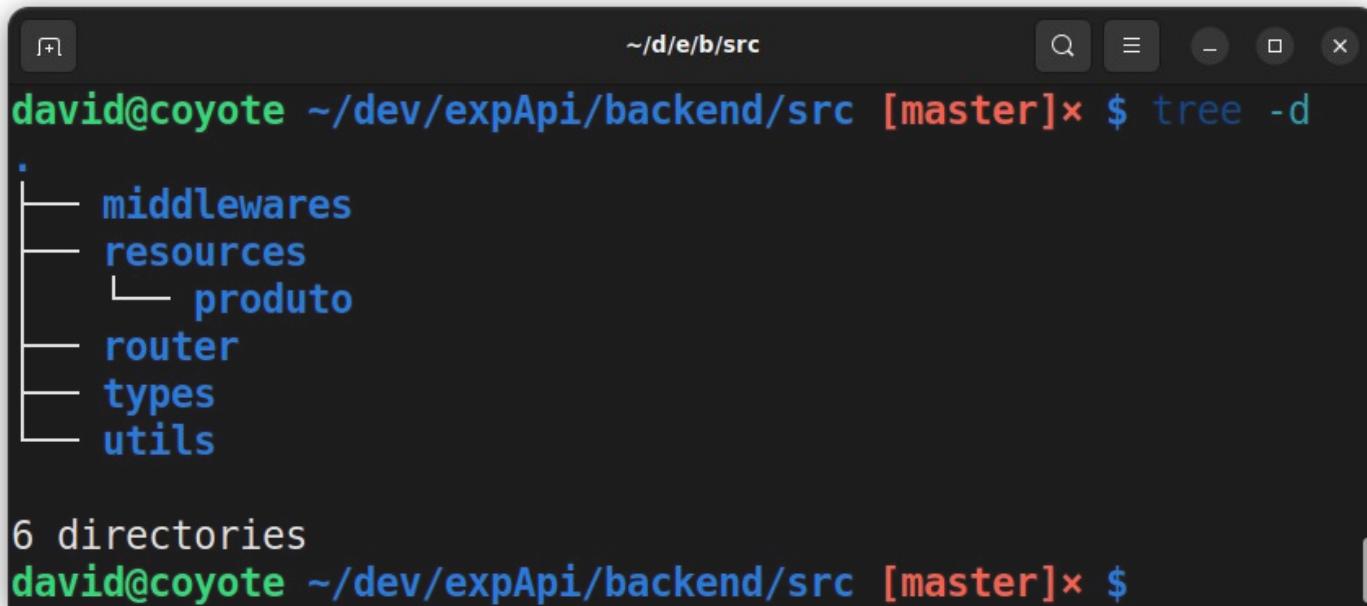


<b>100</b> Continue	<b>101</b> Switching Protocols	<b>102</b> Processing
		

express JS

# Movendo para o padrão REST

- Para o padrão REST, optamos por organizar os arquivos de nossa aplicação usando o esquema abaixo
- O diretório **src** terá todos os arquivos fontes da aplicação, exceto os **modelos**



A screenshot of a terminal window titled 'david@coyote ~/dev/expApi/backend/src [master]\$. tree -d'. The window shows the following directory structure:

```
.
├── middlewares
├── resources
│   └── produto
├── router
├── types
└── utils
```

The terminal also displays the command used to generate the tree and the number of directories found.

# Movendo para o padrão REST

- Para o padrão REST, optamos por estruturar nossa aplicação usando o esquema de diretórios RESTful.
- O diretório **resources** terá um subdiretório para cada entidade da aplicação, exceção feita para os módulos de middleware.

Dentro dos subdiretórios, haverá um roteador, um controlador, um serviço e um arquivo de tipos

```
david@coyote ~/dev/expApi/backend/src [master]x $ tree -d
.
├── middlewares
└── resources
    └── produto
```

```
david@coyote ~/dev/expApi/backend/src/resources/produto [master]x $ ls
produto.controller.ts  produto.service.ts
produto.router.ts      produto.types.ts
david@coyote ~/dev/expApi/backend/src/resources/produto [master]x $ ls
6 directories
david@coyote ~/dev/expApi/backend/src [master]x $
```

# O Roteador

- O **Roteador** de cada resource contém as rotas associadas ao resource, referenciando as actions do controlador

```
// Arquivo src/resources/produto/produto.router.ts

import { Router } from 'express';
import produtoController from './produto.controller';
const router = Router();

// Produto controller
router.get('/', produtoController.index);
router.post('/', produtoController.create);
router.get('/:id', produtoController.read);
router.put('/:id', produtoController.update);
router.delete('/:id', produtoController.remove);

export default router;
```

# O Roteador

- O **Roteador** de cada resource contém as rotas associadas ao resource, referenciando as actions do controlador

```
// Arquivo src/resources/produto/produto.router.ts
```

```
import { Router } from 'express';
import produtoController from './produto.controller';
const router = Router();

// Embora não faça parte do CRUD, o objetivo da rota /produto
// é listar os produtos existentes
router.post('/', produtoController.create);
router.get('/:id', produtoController.read);
router.put('/:id', produtoController.update);
router.delete('/:id', produtoController.remove);

export default router;
```

# O Roteador

- O **Roteador** de cada resource contém as rotas associadas ao resource, referenciando as actions do controlador

```
// Arquivo src/resources/produto/produto.router.ts

import { Router } from 'express';
import produtoController from './produto.controller';
const router = Router();

// Embora não faça parte do CRUD, o objetivo da rota /produto
// é Note que as rotas para read, update e remove terminam com
// uma string :id, que representa um parâmetro utilizado para
// informar que produto se deseja ler, atualizar ou apagar
router.put('/:id', produtoController.update);
router.delete('/:id', produtoController.remove);

export default router;
```

# O Roteador

- O **Roteador** de cada resource contém as rotas associadas ao resource, referenciando as actions do resource.

```
// Arquivo src/resources/produto/produto.router.ts  
import { Router } from 'express';
```

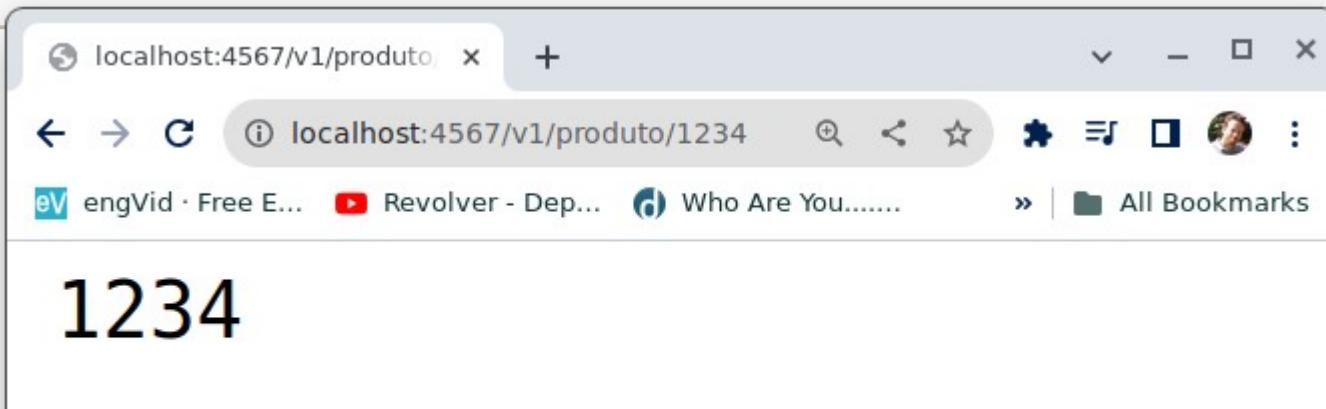
```
// Arquivo src/router/v1Router.ts  
import express from 'express';  
import produtoRouter from '../resources/produto/produto.router';  
  
const router = express.Router();  
  
router.use('/produto', produtoRouter);  
  
export default router;
```

O arquivo de rotas principal irá importar as rotas de cada resource

# O Roteador

- Os **parâmetros** permitem passar dados informações adicionais para o endpoint desejado
  - Por exemplo, na url **http://localhost:3000/produto/1234**, o valor do parâmetro **id** é 1234
- Para ler o valor de **id** dentro de uma função, podemos usar o atributo **param** de **req** (objeto da requisição do usuário):

```
async function read (req, res) {  
  const produtoId = req.params.id;  
  res.end(produtoId);  
},
```



# O Roteador

- O Express possui um middleware chamado **json()**, que é usado para extrair os dados do corpo da requisição (**req.body**)
- Para usá-lo, basta inserir a linha abaixo no arquivo **src/index.ts** antes da chamada ao middleware router:

```
// Arquivo src/index.ts
...
app.use(express.json());
app.use(router);
```

- Após isso, o **express.json()** irá extrair os dados do **request body** das requisições e copiá-los no objeto **req.body**

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/resources/produto/produto.service.ts

import { PrismaClient, Produto } from '@prisma/client';
import { CreateProdutoDto } from './produto.types';
const prisma = new PrismaClient();

export async function getAllProdutos(): Promise<Produto[]> {
  return await prisma.produto.findMany();
}

export async function createProduto(
  produto: CreateProdutoDto
): Promise<Produto> {
  return await prisma.produto.create({ data: produto });
}
```

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/resources/produto/produto.service.ts  
  
import { PrismaClient, Produto } from '@prisma/client';
```

Além das funções mostradas, o serviço de produtos precisa ter funções como:

```
const produtoJaExiste = async (nome: string): Promise<boolean>  
const getProduto = async (id: string): Promise<Produto>  
const updateProduto = async (id: string, produto: ProdutoCreateDto):  
  Promise<[affectedCount: number]>  
const removeProduto = async (id: string): Promise<number>  
  
  produto. CreateProduto  
(): Promise<Produto> {  
  return await prisma.produto.create({ data: produto });  
}
```

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/resources/produto/produto.service.ts  
import { PrismaClient, Produto } from '@prisma/client';
```

Além das funções mostradas, o serviço de produtos precisa ter funções como:

- Uma vantagem do uso de serviços é que suas funções podem ser utilizadas em outras partes da aplicação, diminuindo a réplica de códigos em vários arquivos.

```
Promise<[affectedCount: number]>  
  
const removeProduto = async (id: string): Promise<number>  
  produto. CreateProduct  
(): Promise<Produto> {  
  return await prisma.produto.create({ data: produto });  
}
```

# Camada de Serviços

- Os **serviços** têm como função orquestrar as regras de negócio e servir de intermediários entre controladores e modelos

```
// Arquivo src/resources/produto/produto.service.ts  
import { PrismaClient, Produto } from '@prisma/client';
```

Além das funções mostradas, o serviço de produtos precisa ter funções como:

- Uma vantagem do uso de serviços é que suas funções podem ser reutilizadas.
- Outra vantagem dos serviços é que, caso se queira mudar o ORM da aplicação, o esforço será muito menor. Isso porque eles serão os únicos arquivos que usam os recursos do ORM para recuperar, atualizar e criar dados.

```
const removerProduto = async (id: string): Promise<number> >  
  produto. createProduct  
( ): Promise<Produto> {  
  return await prisma.produto.create({ data: produto });  
}
```

# Data Transfer Objects (DTO)

- Os arquivos **resources/\*\*/\*.types.ts** possuem as **interfaces** e **types**, em especial os **DTOs**, usados dentro do resource
- DTO é uma interface ou type usado para representar os objetos de dados que são trocados entre a API e as aplicações client
  - Por exemplo, para criar um novo produto, a aplicação cliente precisa enviar para a API os dados desse novo produto, sendo o formato desses dados é definido através de um DTO

# Data Transfer Objects (DTO)

- Os DTOs geralmente contêm um subconjunto dos atributos de um dado modelo, e para gerá-los podemos usar o comando Pick

```
// Arquivo src/resources/produto/produto.types.ts  
  
import { Produto } from '../../models/Produto';  
  
type ProdCreateDto = Pick<Produto, 'nome' | 'preco' | 'estoque'>;  
type ProdUpdateDto = Pick<Produto, 'nome' | 'preco' | 'estoque'>;  
  
export default { ProdCreateDto, ProdUpdateDto}
```

Cria um novo type contendo apenas as propriedade nome, preço e estoque do modelo Produto

# Data Transfer Objects (DTO)

- Os DTOs são usados principalmente na camada de serviço, mas também podem ser utilizados nos controladores

```
// Arquivo src/resources/produto/produto.service.ts

import { PrismaClient, Produto } from '@prisma/client';
import { CreateProdutoDto } from './produto.types';
const prisma = new PrismaClient();

export async function createProduto(
  produto: CreateProdutoDto
): Promise<Produto> {
  return await prisma.produto.create({ data: produto });
}
```

# Camada do Controlador

- Os controladores são responsáveis por **recepçionar** e **responder** as respostas dos usuários

```
// Arquivo src/resources/produto/produto.controller.ts

import { Request, Response } from 'express';
import { createProduto, jaExiste } from './produto.service';

async function create(req: Request, res: Response) {
  const produto = req.body;
  try {
    if (await jaExiste(produto.nome)) {
      return res.status(400).json({ msg: 'Produto já existe' });
    }
    const newProduto = await createProduto(produto);
    res.status(201).json(newProduto);
  } catch (err) {
    res.status(500).json(err);
  }
}
```

# Camada do Controlador

- Os controladores são responsáveis por **recepçionar** e **responder** as respostas dos usuários

```
// Arquivo src/resources/produto/produto.controller.ts
```

É importante notar que erros na inserção de um registro disparam uma exceção no bloco try do Controlador, ocasionando um status 500 para a requisição.

```
const produto = req.body;
try {
  if (await jaExiste(produto.nome)) {
    return res.status(400).json({ msg: 'Produto já existe' });
  }
  const newProduto = await createProduto(produto);
  res.status(201).json(newProduto);
} catch (err) {
  res.status(500).json(err);
}
```

# Insomnia

- Existem várias ferramentas para testar os endpoints de uma API, como o **Insomnia**, o **Postman** e o **Thunder Client**

The screenshot shows the homepage of the Insomnia API development tool. At the top, there's a navigation bar with links for 'Products', 'Docs', 'Pricing', 'Plugins', and 'Login'. A prominent 'Get Started for Free' button is located on the right side of the header. Below the header, a large white text area displays the slogan: 'The easy way to design, debug, and test APIs'. Underneath the slogan, a smaller text block reads: 'Build better APIs faster and collaboratively with a dev-friendly interface, built-in automation, and an extensible plugin ecosystem.' At the bottom of the page, there's another 'Get Started for Free' button. The footer features a sidebar with 'Kong Org' dropdown, 'PROJECTS (2)', and items like 'Konnect' and 'On Premise'. It also includes a search bar with 'Filter...' and a '+ Create' button, along with several thumbnail previews of API documents and collections.

# Insomnia

- Existem várias ferramentas para testar os endpoints de uma API, como o **Insomnia**, o **Postman** e o **Thunder Client**

The screenshot shows the homepage of the Insomnia API development tool. At the top, there's a navigation bar with links for 'Products', 'Docs', 'Pricing', 'Plugins', and 'Login'. A prominent 'Get Started for Free' button is located on the right side of the header. Below the header, a large banner features the text 'The easy way to design, debug, and test APIs'. A blue callout box contains the message: 'Nesse curso optamos por usar o **Insomnia**, dado sua facilidade e recursos'. At the bottom, there's a preview of the Insomnia interface showing a sidebar with 'Kong Org' and 'PROJECTS (2)', and a main area displaying API collections and documents.

The easy way to design, debug, and test APIs

Nesse curso optamos por usar o **Insomnia**, dado sua facilidade e recursos

Get Started for Free

Kong Org

PROJECTS (2)

Document Collection Document

Cloud OS Login Google Play Mo... Partner Cloud OS Login

Upgrade

Podemos usar o plugin **insomnia-plugin-faker** para gerar requisições com dados fakes

Podemos usar o plugin **insomnia-plugin-faker** para gerar requisições com dados fakes

POST `_.base_url /v1/produto`

Auth ▾ Query Header 1 Docs

```
1 "nome": "Faker → Commerce",
2 "preco": Faker → Commerce,
3 "estoque": Faker → Random
4 }
```

Beautify JSON

201 Created 22.3 ms 179 B 2 Minutes Ago ▾

Preview ▾ Header 7 Cookie Timeline

```
1 {
2   "id": "a09edab0-1a5d-11ee-ac7e-a531058f3526",
3   "nome": "Gorgeous Granite Bike",
4   "preco": 490,
5   "estoque": 33,
6   "updatedAt": "2023-07-04T11:26:32.412Z",
7 }
```

\$.store.books[\*].author

Status  
500 causados  
por erros de  
validação no  
Sequelize

POST `_.base_url /v1/produto`

```
1▼ {  
2   "nome": "ABC",  
3   "preco": Faker → Commerce ,  
4   "estoque": Faker → Random  
}
```

500 Internal Server Error

7.87 ms

482 B

1 Minute Ago ▾

Preview ▾

Header 7

Cookie

Timeline

```
1▼ {  
2   "name": "SequelizeValidationError",  
3   "errors": [  
4     {  
5       "message": "O nome do produto deve ter entre 4 e 50 caracteres",  
6       "type": "Validation error",  
7     }  
]
```

`$.store.books[*].author`





Application Edit View Window Tools Help

Insomnia - Loja - Create Produto

Application Edit View Window Tools Help

Insomnia - Loja - Create Produto



Insomnia / Loja ▾



## Insomnia Preferences – v2022.3.0

General

Data

Themes

Keyboard

Account

Plugins

Import format will be automatically detected. Supported formats include: Insomnia v1, Insomnia v2, Insomnia v3, Insomnia v4, Postman, Postman Environment, HAR 1.2, cURL, Swagger 2.0, OpenAPI 3.0, WSDL

Your format isn't supported? [Add Your Own.](#)

Export Data ▾

Import Data ▾

Create Run Button

\* Tip: You can also paste Curl commands into the URL bar

É possível exportar os endpoints gerados no Insomnia e salvá-los no repositório da aplicação (normalmente em `.insomnia/insomnia.json`). Use esse artifício para compartilhar os endpoints com os demais devs.

DEL Delete Produto

`$.store.books[*].author`



On Premise



Document



Collection



Document

Cloud OS Login

Google Play Mo... Partner

Cloud OS Login



# Design, debug, and test APIs locally or in the cloud

Choose Local, Cloud, or Git storage to build better APIs collaboratively with a dev-friendly UI, built-in automation, and an extensible plugin ecosystem.

**Exercício:** Crie uma API REST usando o framework Express contendo os endpoints **index**, **create**, **read**, **update** e **delete** para o resource **produto**. Use o Insomnia para testar a API.

github  
**ExpAPI**

express JS

The screenshot shows the Insomnia application's interface. On the left, there's a sidebar with icons for Kong Org, Projects (2), Konnect, and On Premise. The main area displays a list of collections: 'Cloud OS Login v2.8.4', 'Collection v1.1' (with 'Google Play Mo... Partner'), and another 'Cloud OS Login v2.8.4'. A red oval on the right side of the screen contains the text 'github' and 'ExpAPI'. Below this, the words 'express' and 'JS' are stacked vertically. The overall theme of the slide is API development using Express and GitHub.

# Validação de Requests

- Em APIs, é importate implementar alguma forma de proteção contra dados inválidos
- O **Joi** é uma biblioteca para **validação de dados**, usada para garantir a integridade das solicitações em APIs
  - Com o Joi, podemos definir que campos são obrigatórios, que tipos de dados são aceitáveis e quais regras os valores devem cumprir
- Use o comando abaixo para adicionar o Joi na aplicação

```
$ npm install joi
```

# Validação de Requests

- Em APIs, é importate implementar alguma forma de proteção contra dados inválidos
- O **Joi** é uma biblioteca para **validacão de dados**. usada para  
ga Note que neste curso o Joi será usado para validação das  
– requests no **lado servidor**, mas na maioria dos casos é  
recomendado que a validação também seja feita do lado cliente
- Use o comando abaixo para adicionar o Joi na aplicação

```
$ npm install joi
```

# Validação de Requests

- Após isso, já podemos usar o joi para validar dados:

```
import Joi from 'joi';

const schemaProduto = Joi.object().keys({
  nome: Joi.string().min(3).max(50).required(),
  preco: Joi.number().required(),
  estoque: Joi.number().integer().required()
});

const celularMoto = {
  nome: 'Smartphone Motorola Edge 30',
  preco: 1499,
  estoque: 3
};

const result = schemaProduto.validate(celularMoto);
// result.error == null significa válido
```

Esquema  
de  
validação

Usamos  
o esquema  
para validar  
o dado

# Validação de Requests

- O Joi oferece uma variedade de validadores que podem ser usados para definir esquemas de validação
  - **string()**, **number()**, **boolean()**, **object()**, **array()**, **date()**, **required()**, **min()**, **max()**, **email()**, **uri()**, etc
- Também oferece regras de validação mais complexas
  - **pattern()**: verifica se o valor corresponde a um padrão
  - **allow()**: especifica os valores permitidos para um campo
  - **forbidden()**: especifica valores que não permitidos
  - **valid()**: define uma lista de valores válidos
  - **custom()**: permite a definição de funções de validação personalizadas

# Validação de Requests

- Para validar as requests do resource **produtos**, vamos criar um arquivo **resources/produto/produto.schemas.ts**

```
import Joi from 'joi';

const schema = Joi.object({
  nome: Joi.string().min(3).max(100).required(),
  preco: Joi.number().required(),
  estoque: Joi.number().integer().required()
});

export default schema;
```

# Validação de Requests

- Adicionalmente, vamos criar um middleware **validate.ts** para validar os dados das requests (diretório **src/middlewares**)

```
import { Request, Response, NextFunction } from 'express';
import { Schema } from 'joi';

const validate = (schema: Schema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const { error } = schema.validate(req.body, {
      abortEarly: false
    });
    if (error) res.status(422).json({ error: error.details });
    else next();
  };
};

export default validate;
```

# Validação de Requests

- Adicionalmente, vamos criar um middleware **validate.ts** para validar os dados das requests (diretório **src/middlewares**)

```
import { Request, Response, NextFunction } from 'express';
import { Schema } from 'joi';

const validate = (schema: Schema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    const { error } = schema.validate(req.body, { abortEarly: false });
    if (error) res.status(422).json({ error: error.details });
    else next();
  };
};

export default validate;
```

Retorna todos os erros encontrados, e não apenas o primeiro

# Validação de Requests

- A partir desse momento, podemos usar o middleware de validação no roteador do resource produtos

```
import { Router } from 'express';
import produtoController from './produto.controller';
import schema from './produto.schemas';
import validate from '../../middlewares/validate';

const router = Router();

router.get('/', produtoController.index);
router.get('/test', produtoController.test);
router.post('/', validate(schema), produtoController.create);
router.get('/:id', produtoController.read);
router.put('/:id', validate(schema), produtoController.update);
router.delete('/:id', produtoController.remove);

export default router;
```

Insomnia - Loja Virtual – Create Produto

Application Edit View Window Tools Help

Insomnia / Loja Virtual ▾

No Environment ▾ Cookies

POST `_.base_url/v1/produto` Send

Filter +

JSON Auth Query Header 1 Docs

```
1▼ {  
2  "nome": "Faker → Commerce",  
3  "preco": Faker → Commerce,  
4  "estoque": "a"  
5 }
```

Beautify JSON

422 Unprocessable Entity 1.92 ms 152 B 5 Minutes Ago ▾

PUT Update Produto

GET Get Produto

POST Create Produto

GET Get Produtos

Preview Header 7 Cookie Timeline

```
1▼ {  
2  "error": [  
3    {  
4      "message": "\"estoque\" must be a number",  
5      "path": [  
6        "estoque"  
7      ],  
8      "type": "number_base"  
9    }  
10  }  
11 }  
12  
13 $.store.books[*].author
```

Insomnia - Loja Virtual – Create Produto

Application Edit View Window Tools Help

Insomnia / Loja Virtual ▾

No Environment ▾ Cookies

POST `_.base_url/v1/produto` Send

Filter +

Produto

PUT Update Produto

GET Get Produto

POST Create Produto

GET Get Produtos

JSON Auth Query Header 1 Docs

```
1▼ {  
2   "nome": "Faker → Commerce",  
3   "preco": Faker → Commerce,  
4   "estoque": "a"  
5 }
```

Beautify JSON

422 Unprocessable Entity 1.92 ms 152 B 5 Minutes Ago ▾

```
4   "message": "\"estoque\" must be a number"  
5▼  
6   "path": [  
7     "estoque"  
8   ],  
9   "type": "number_base"
```

\$.store.books[\*].author

**Exercício:** Usando Joi, implemente um mecanismo de validação dos endpoints do CRUD de produto.

github  
**ExpAPI**

express JS

# Cookies e Sessões

- HTTP é um protocolo que não mantém estado, isto é **não mantém uma conexão**
- Cada requisição que um cliente (browser, insomnia, etc) faz à API é independente das requisições anteriores
- No entanto, muitas aplicações necessitam manter o estado do usuário durante as requests feitas à API
  - Ex: carrinho de compras em sites de comércio eletrônico
- As aplicações possuem duas opções para manter as informações de estado dos clientes:
  - **Cookies**: mantém informações de estado no cliente (browser)
  - **Sessões**: matém informações de estado no lado servidor

# Cookies



- **Cookies** são variáveis enviadas pelo servidor Web para o cliente através do **protocolo HTTP**
  - Ficam armazenados no lado cliente
  - São enviados para o servidor em futuros acessos do cliente

## Requests do Cliente

1 Get / HTTP1.1  
Host: http://api.mydomain.com

## Responses de http://api.mydomain.com

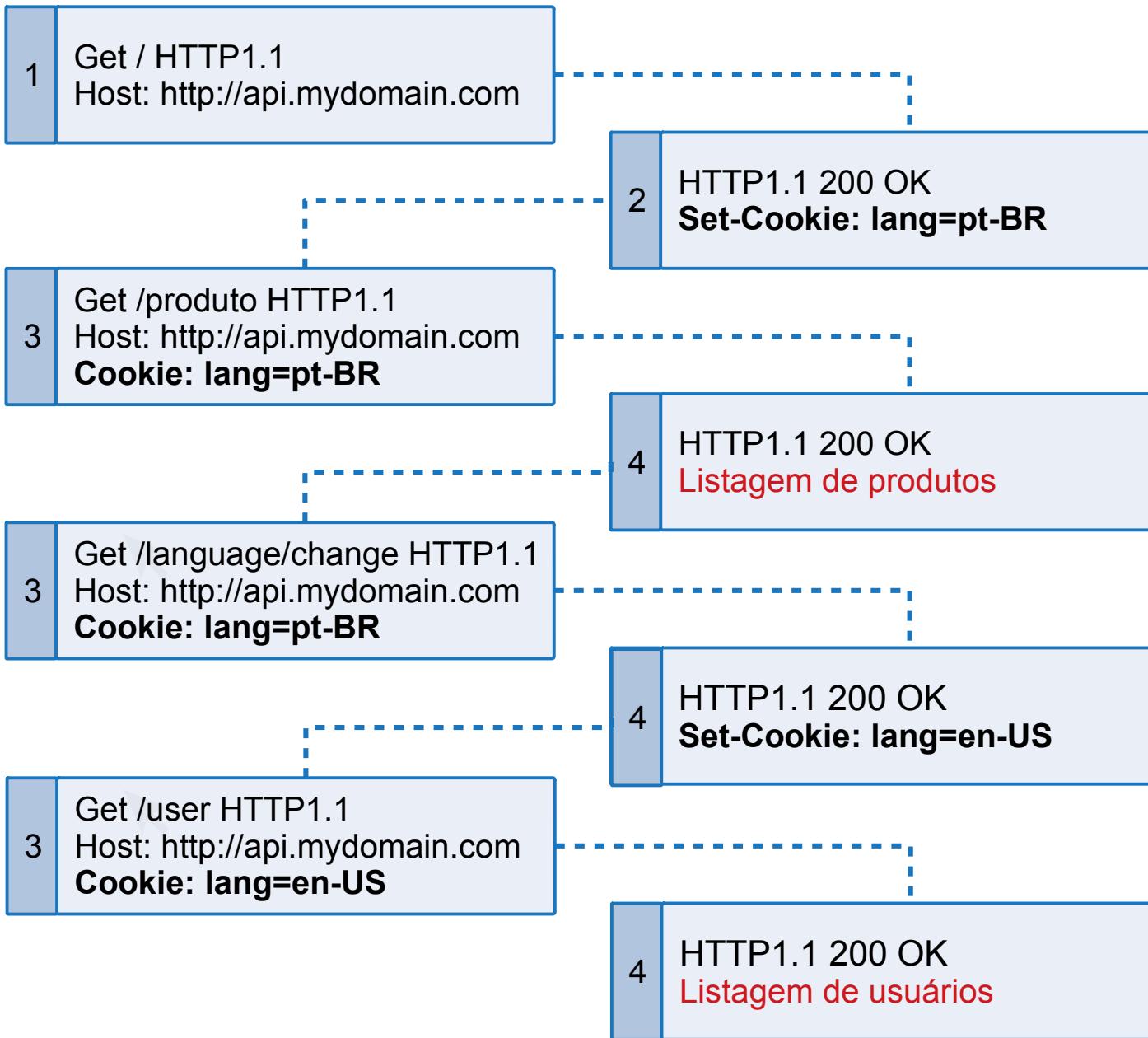
2 HTTP1.1 200 OK  
**Set-Cookie: lang=pt-BR**

3 Get /produto - HTTP1.1  
Host: http://api.mydomain.com  
**Cookie: lang=pt-BR**

4 HTTP1.1 200 OK  
**Listagem de produtos**

## Requests do Cliente

## Responses de http://api.mydomain.com



# Cookies



- Para habilitar o uso de cookies por sua aplicação, é necessário instalar o middleware **cookie-parser**

```
$ npm install cookie-parser  
$ npm install -D @types/cookie-parser
```

- Para usar o middleware, precisamos dar um **import** no módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo src/index.ts  
import cookieParser from 'cookie-parser';  
app.use(cookieParser());
```

# Cookies



- A partir deste momento, podemos i) criar novos cookies e ii) identificar os cookies enviados pelo browser para o servidor
  - O array **req.cookies** armazena os cookies enviados pelo cliente
  - O método **res.cookie** é usado para enviar um pedido de criação de um novo cookie no lado cliente (browser, insomnia, etc)

```
// Arquivo src/middlewares/setLangCookie

const setLangCookie = (req, res, next) => {
  if (!('lang' in req.cookies)) res.cookie('lang', 'pt-BR');
  next();
};

export default setLangCookie
```

# Cookies



- Após criar o middleware, podemos adicioná-lo em `src/index.ts`

```
// Arquivo src/index.ts
...
import cookieParser from 'cookie-parser';
import { setLangCookie } from './middlewares/setLangCookie';
...
app.use(cookieParser());
app.use(setLangCookie);
```

- Com isso, o cookie `lang` será criado no primeiro acesso do usuário, com o valor default `pt-BR`

Insomnia - Loja Virtual - Get Produtos

Application Edit View Window Tools Help

Insomnia / Loja Virtual ▾

No Environment ▾ Cookies GET ▾ base\_url /v1/produto/ Send

Filter + ▾

Bo Body Query Header Docs

Clique para ver os cookies

200 OK 11.4ms 1983 B 1 Minute Ago ▾

Language

GET Change Language

Produto

PUT Update Produto

GET Get Produto

POST Create Produto

GET Get Produtos

Preview ▾ Header 8 Cookie 1 Timeline

```
1▼ [  
2▼ {  
3 "id": "02578185-fb3e-4c12-81f4-8f84709f4c80",  
4 "nome": "Bacon",  
5 "preco": "269",  
6 "estoque": 6,  
7 "createdAt": "2023-10-21T14:09:36.456Z",  
8 "updatedAt": "2023-10-21T14:09:36.456Z"  
9 },  
10▼ {  
11 "id": "243b4ac5-4c2a-4375-8253-8b5ec789683a",  
12 "nome": "Chair",  
$.store.books[*].author  
?
```

express JS

Ap Insomnia - Loja Virtual - Get Produtos

Application Edit View Window Tools Help

Insomnia / Loja Virtual ▾

Manage Cookies X

Filter Cookies

twitter.com

DOMAIN	COOKIE	Actions
localhost	lang=pt-BR; Path=/; Domain=localhost	<span>Edit</span> <span>&gt;Delete</span>

\* cookies are automatically sent with relevant requests Done

\$.store.books[\*].author

# Cookies



- Podemos criar um **resource** para gerenciamento das linguagens, contendo um endpoint para mudar a linguagem

```
// Arquivo src/resources/languages/language.controller
import { Request, Response } from 'express';

function changeLanguage(req: Request, res: Response) {
  const { lang } = req.body;
  res.cookie('lang', lang);
  res.json({ lang });
}

export default { changeLanguage };
```

```
// Arquivo src/resources/languages/language.router
import { Router } from 'express';
import languageController from './language.controller';

const router = Router();
router.get('/change', languageController.changeLanguage);
export default router;
```

Insomnia - Loja Virtual - Get Produtos

Application Edit View Window Tools Help

Insomnia / Loja Virtual ▾

No Environment ▾ Cookies

GET `base_url/v1/language/change` Send

JSON Auth Query Header 1 Docs

```
1▼ {  
2   "lang": "en-US"  
3 }
```

Beautify JSON

200 OK 12.5 ms Just Now ▾

Preview Header 8 Cookie 1 Timeline

```
1▼ {  
2   "lang": "en-US"  
3 }
```

\$.store.books[\*].author

Requisição de mudança da linguagem

Filter +

Language

Change Language

Produto

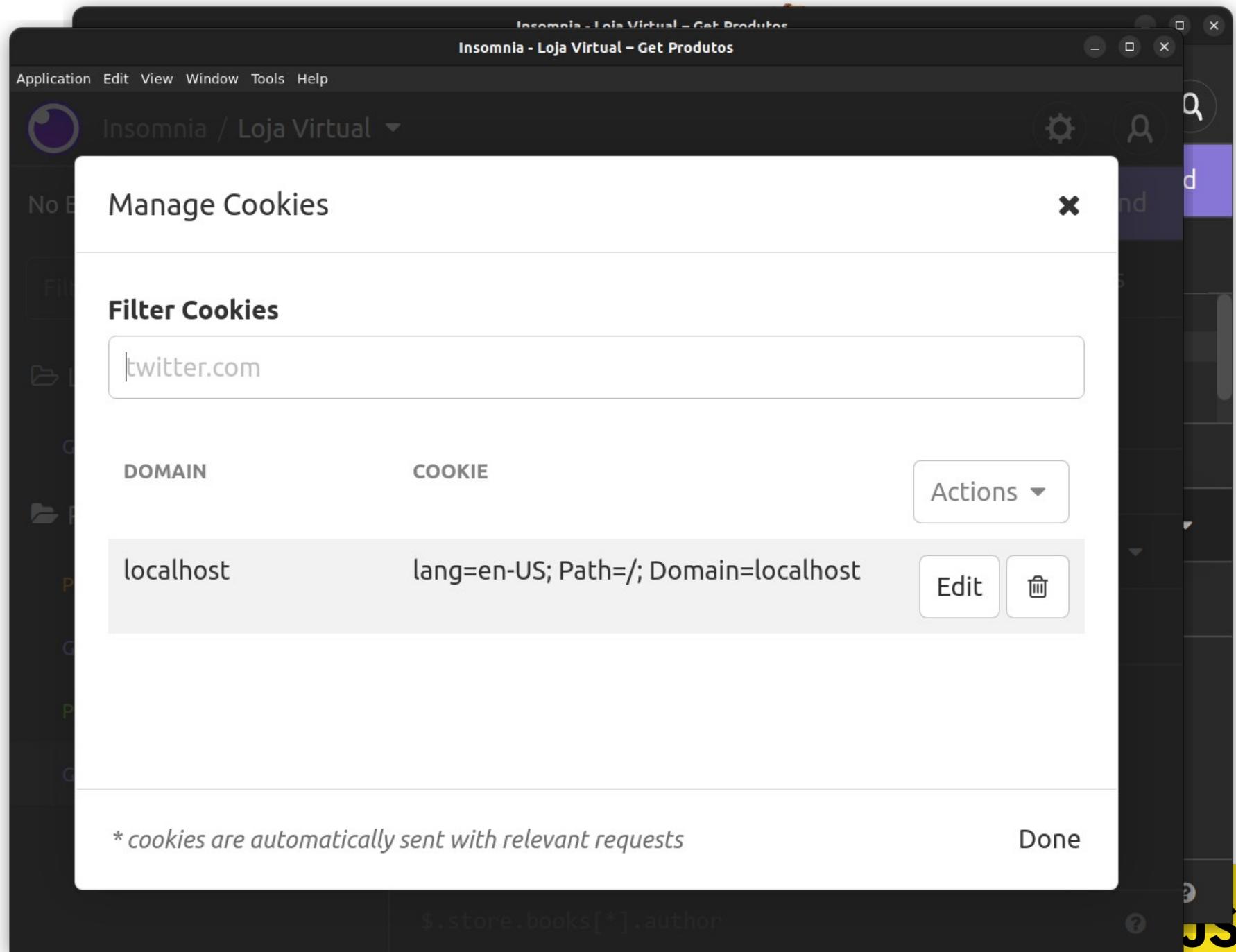
Update Produto

Get Produto

Create Produto

Get Produtos

Get default router



# Cookies



- Também podemos criar cookies com data de expiração

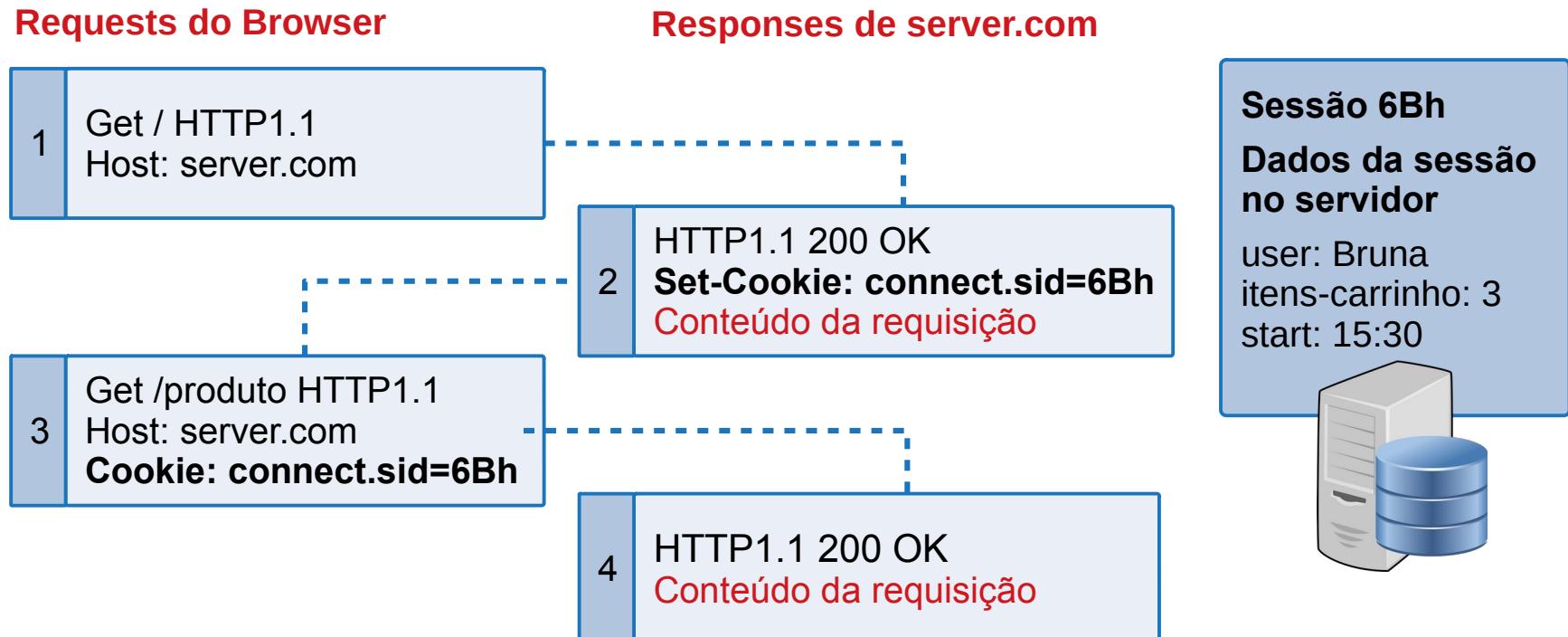
```
// Expira 360000 ms (6 minutos) após ser criado  
res.cookie(name, 'value', { maxAge: 360000 } );
```

- Se o cookie for criado sem data de expiração, ele será apagado após o fechamento da janela do browser
- Usamos a função **clearCookie** para apagar um cookie já criado

```
res.clearCookie('lang');
```

# Sessões

- Através de sessões, podemos armazenar informações de estado (variáveis) no lado servidor
- Em vez do browser guardar um cookie por dado, ele guarda apenas um cookie contendo um **id de sessão (connect.sid)**



# Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](#)

```
$ npm install uuid  
$ npm install -D @types/uuid
```

- Os UUIDs são valores de 128 bits que podem ser usados como ID únicos de qualquer coisa em sistemas computacionais
  - Ex: f0221c72-ac30-4796-83f5-fd7a8a4f6b15
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

# Sessões

- Para usarmos as sessões, precisamos instalar um módulo para geração de **valores únicos para os IDs** das sessões
- Uma opção é o módulo **uuid** – Universally Unique Identifier – que é uma implementação do UUID descrito na [RFC 4122](#)

```
$ npm install uuid  
$ npm install -D @types/uuid
```

- Conforme dito anteriormente, em nosso sistema usamos uuids como **chaves primárias** de tabelas, ao invés de IDs auto incrementados
- Embora a probabilidade de um UUID ser duplicado não seja nula, ela é próximo o suficiente de zero e pode ser ignorada

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

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

- Para usar o middleware, precisamos importar o módulo e adicioná-lo em nossa aplicação com o método **use**

```
// Arquivo index.ts
import session from 'express-session';
import { v4 as uuidv4 } from 'uuid';

...
app.use(session({
  genid: (req) => uuidv4(),
  secret: 'Hi9Cf#mK98',
  resave: true,
  saveUninitialized: true
}));
```

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/expr
```

- Para usar o middleware, precisaremos importá-lo em nossa aplicação e adicioná-lo em nosso **app.use**

Os UUIDs são usados para gerar o **id de sessão**

```
// Arquivo index.ts  
import session from 'express-session';  
import { v4 as uuidv4 } from 'uuid';  
  
...  
app.use(session({  
    genid: (req) => uuidv4(),  
    secret: 'Hi9Cf#mK98',  
    resave: true,  
    saveUninitialized: true  
}));
```

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/expr
```

- Para usar o middleware adicioná-lo em nossa aplicação

```
// Arquivo index.ts  
import session from 'express-session'  
import { v4 as uuidv4 } from 'uuid'  
...  
app.use(session({  
    genid: (req) => uuidv4(),  
    secret: 'Hi9Cf#mK98',  
    resave: true,  
    saveUninitialized: true  
}));
```

Usado para adicionar uma assinatura (similar ao checksum) ao **id de sessão** enviado para o usuário. Quando o usuário devolve o session.id, a assinatura é usada para checar se o session.id é válido. Usa uma técnica chamada HMAC.

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/expr
```

- Para usar o middleware, é necessário adicionar-o em nossa aplicação.

```
// Arquivo index.js  
import session from 'express-session'  
import { v4 as uuidv4 } from 'uuid'  
...  
app.use(session({  
    genid: (req) => uuidv4(),  
    secret: 'H19Cf#mK98',  
    resave: true,  
    saveUninitialized: true  
}));
```

Quando **true**, a sessão do usuário é salva a cada requisição, mesmo que os dados da sessão não tenham sido modificados durante a requisição. Isso mantém a sessão ativa, visto que ela pode ser deletada após algum tempo de desuso.

# Sessões

- Para habilitar o uso de sessões em sua aplicação, é necessário instalar o middleware **express-session**

```
$ npm install express-session  
$ npm install -D @types/expr
```

- Para usar o middleware é necessário importá-lo no seu código e adicioná-lo em nossa aplicação.

```
// Arquivo index.js  
import session from 'express-session'  
import { v4 as uuidv4 } from 'uuid'  
...  
app.use(session({  
    genid: (req) => uuidv4(),  
    secret: 'H190#mK98',  
    resave: true,  
    saveUninitialized: true  
}));
```

Quando **true**, a sessão do usuário é salva a cada modificação.

Quando **true**, força que as sessões não inicializadas sejam salvas no store. Uma sessão não inicializada ocorre quando a sessão é nova e ainda não foi modificada.



No E

## Manage Cookies



## Filter Cookies

twitter.com

## DOMAIN

## COOKIE

Actions ▾

localhost

lang=en-US; Path=/; Domain=localhost

Edit



localhost

connect.sid=s%3A6401e87c-2d1a-4f1e-  
-892d-39ed495183a7.%2BRalpppgJj0  
J%2BaFHI3cPBNojkH3VJlzKATWun5s  
4z%2FA; Path=/; HttpOnly; Domain=lo  
calhost

Edit

*\* cookies are automatically sent with relevant requests*

Done

\$.store.books[\*].author



S

# Cadastro de Usuários

- O cadastro de usuário envolve dois modelos Prisma, que precisam ser incluídos no arquivo **prisma/schema.prisma**

```
model Usuario {  
    id      String    @id @default(uuid()) @db.Char(40)  
    nome    String    @db.VarChar(100)  
    email   String    @unique @db.VarChar(100)  
    senha   String    @db.VarChar(100)  
    createdAt DateTime @default(now()) @map("created_at")  
    updatedAt DateTime @updatedAt @map("updated_at")  
  
    tipoUsuario  TipoUsuario @relation(fields: [  
        tipoUsuarioId  
    ], references: [id])  
    tipoUsuarioId String  
  
    @@map("usuarios")  
}
```

# Cadastro de Usuários

- O cadastro de usuário envolve dois modelos Prisma, que precisam ser incluídos no arquivo **prisma/schema.prisma**

```
model Usuario {  
    id      String    @id @default(uuid()) @db.Char(40)  
    rotulo  String    @unique @db.VarChar(10)  
    created_at DateTime @default(now()) @map("created_at")  
    updated_at DateTime @updatedAt @map("updated_at")  
    usuarios Usuario[]  
  
    @@map("tipos_usuarios")  
}  
  
    @@map("usuarios")  
}
```



~/d/e/backend



```
david@coyote ~/dev/expApi/backend [main]x $ npx prisma migrate dev
```

```
--name add-usuarios-tables
```

```
Environment variables loaded from .env
```

```
Prisma schema loaded from prisma/schema.prisma
```

```
Datasource "db": MySQL database "loja" at "localhost:3306"
```

```
model Usuario {
```

```
  Applying migration `20231028115757_add_usuarios_tables` (40)
```

```
The following migration(s) have been created and applied from new schema changes:
```

```
  @id @default(uuid()) @db.Char(40)  
  rotulo String @unique @db.VarChar(10)
```

```
  migrations/  
    20231028115757_add_usuarios_tables/  
      updatedAt DateTime @default(now()) @map("created_at")  
      @updatedAt @map("updated_at")
```

```
  Usuario Usuario[]
```

```
Your database is now in sync with your schema.
```

```
  @@map("tipos_usuarios")
```

```
}✓ Generated Prisma Client (v5.4.2) to ./node_modules/@prisma/client in 129ms
```

```
  @@map("usuarios")
```

```
david@coyote ~/dev/expApi/backend [main]x $
```

# Tipos de Usuários

- Em nossa app, teremos os tipos de usuários **client** e **admin**

```
// File src/resources/tipoUsuario/tipoUsuario.constants.ts

export enum TiposUsuarios {
  "ADMIN" = "7edd25c6-c89e-4c06-ae50-c3c32d71b8ad",
  "CLIENT" = "6a4cda94-fbb6-476b-be29-f4124cae9058",
}
```

```
// File prisma/seed.ts

async function main() {
  await prisma.tipoUsuario.createMany({
    data: [
      { id: TiposUsuarios.CLIENT, rotulo: "cliente" },
      { id: TiposUsuarios.ADMIN, rotulo: "admin" },
    ],
    skipDuplicates: true,
  });
}
```

# Tipos de Usuários

- Em nossa app, teremos os tipos de usuários **client** e **admin**

```
// File src/resources/tipoUsuario/tipoUsuario.constants.ts

export enum TiposUsuarios {
  "ADMIN" = "7edd25c6-c89e-4c06-ae50-c3c32d71b8ad",
```



The terminal window shows the command `npx prisma db seed` being run in the directory `~/dev/expApi/backend`. The output indicates that environment variables are loaded from `.env`, the seed command is running, and the seed has been successfully executed.

```
david@coyote ~/dev/expApi/backend [main] $ npx prisma db seed
Environment variables loaded from .env
Running seed command
File /Users/david/.ts-node-prisma/seeds.ts` ...
async function main() {
  The seed command has been executed.
}
david@coyote ~/dev/expApi/backend [main] $
```

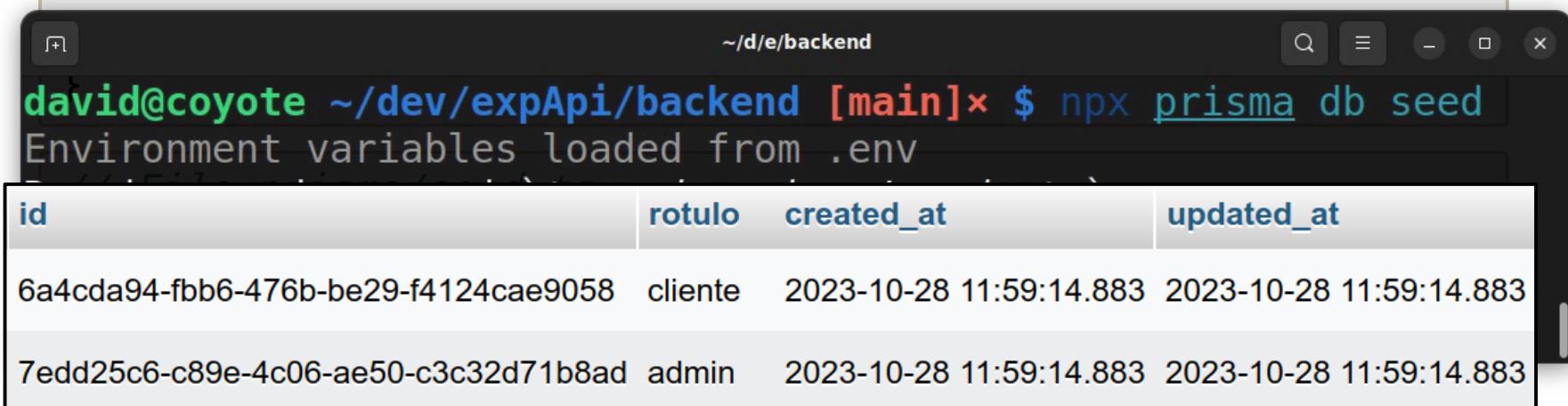
```
data: [
  { id: TiposUsuarios.CLIENT, rotulo: "cliente" },
  { id: TiposUsuarios.ADMIN, rotulo: "admin" },
],
skipDuplicates: true,
});
```

JS

# Tipos de Usuários

- Em nossa app, teremos os tipos de usuários **client** e **admin**

```
// File src/resources/tipoUsuario/tipoUsuario.constants.ts
export enum TiposUsuarios {
  "ADMIN" = "7edd25c6-c89e-4c06-ae50-c3c32d71b8ad",
```



```
david@coyote ~/dev/expApi/backend [main]x $ npx prisma db seed
Environment variables loaded from .env

```

id	rotulo	created_at	updated_at
6a4cda94-fbb6-476b-be29-f4124cae9058	cliente	2023-10-28 11:59:14.883	2023-10-28 11:59:14.883
7edd25c6-c89e-4c06-ae50-c3c32d71b8ad	admin	2023-10-28 11:59:14.883	2023-10-28 11:59:14.883

```
      { id: TiposUsuarios.CLIENT, rotulo: "cliente" },
      { id: TiposUsuarios.ADMIN, rotulo: "admin" },
    ],
    skipDuplicates: true,
  );
}
```

# Criptografando as senhas

- Para criar um novo usuário, é necessário criptografar a sua senha antes de salvá-la no banco de dados
- Mas por quê? Por que não guardar a senha crua?
  - Todas as pessoas com acesso ao banco poderiam ver a senha
  - Os usuários frequentemente **usam a mesma senha** em vários sites
  - A senha iria aparecer nos **backups** do banco
  - Se o banco estiver na **cloud**, as senhas ficariam expostas na web
  - As senhas ficariam expostas a **ataques de SQL-injection**

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela Usuario

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

Submit

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela Usuario

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

Submit

```
SELECT estado, capital FROM estado  
WHERE estado = 'Amazonas'
```

estado	capital
Amazonas	Manaus

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

estado	capital
Rio de Janeiro	Rio de Janeiro
Amazonas	Manaus
Minas Gerais	Belo Horizonte
Ceará	Fortaleza

Tabela Usuario

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM estado  
WHERE estado = " UNION SELECT login,  
senha FROM Usuario WHERE login != "

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

estado	capital
Rio de Janeiro	Belo Horizonte

Tabela Usuario

login	senha
alberto	flamengo

Os ataques de SQL-Injection só são possíveis se o invasor conseguir incluir caracteres como ' (aspas simples), " (aspas duplas) ou \b (caracter de backspace) nos inputs dos formulários.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM estado  
WHERE estado = " UNION SELECT login,  
senha FROM Usuario WHERE login != "

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

# Ataques de SQL-Injection

- Caso os desenvolvedores não tomem os devidos cuidados, os formulários podem ficar vulneráveis a ataques de SQL-injection

Tabela Estado

estado	capital
Rio de Janeiro	Belo Horizonte
São Paulo	Brasília

Tabela Usuario

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

O Prisma resolve este problema adicionando um caracter de **escape** a tais elementos. Por exemplo, o caracter ' vira \' , " vira \" e \b vira \\b. Vide [SQL injection no Prisma](#). Mesmo assim, é sempre saudável fazer validação dos inputs.

Busca da capital pelo estado

' UNION SELECT login, senha FROM Usuario WHERE login != '

Submit 

SELECT estado, capital FROM estado  
WHERE estado = " UNION SELECT login,  
senha FROM Usuario WHERE login != "

login	senha
alberto	flamengo
maria	teste123
fernanda	RmJ&AnhK@
matheus	pokemon

# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
  - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

```
david@coyote: ~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote: ~$ echo -n "ufam" | md5sum  
8996fe161805927c27a92fae2ca238d2 -  
david@coyote: ~$ echo -n "computacao" | md5sum  
d0cc5ed8aecb1898032c48af57057b9f -  
david@coyote: ~$ □
```

# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
  - Ex: md5, sha1, sha256, etc
- São funções de uma **mão-única**, isto é, não é possível recuperar o bloco de dados original a partir do hash gerado

The screenshot shows a terminal window with two tabs. The top tab is active and displays the command: `david@coyote:~$ echo -n "minhasenha" | md5sum`. The output is `7c6`. The bottom tab is inactive and displays the command: `david@coyote:~$ echo -n "minhasenha" | sha1sum`. The output is `93d51f52fbfe1e944f084727df24993e88caee7`. Below these, two more commands are shown: `david@coyote:~$ echo -n "ufam" | sha1sum` and `david@coyote:~$ echo -n "computacao" | sha1sum`, both resulting in long hash strings.

```
david@coyote:~$ echo -n "minhasenha" | md5sum
7c6
david@coyote:~$ echo -n "minhasenha" | sha1sum
93d51f52fbfe1e944f084727df24993e88caee7
david@coyote:~$ echo -n "ufam" | sha1sum
4ae250e25cb5e0bdb06cfcc2513cb451f4a610d0e
david@coyote:~$ echo -n "computacao" | sha1sum
915769113d302e7893cc3019c5fc5dcb36682bca
david@coyote:~$
```

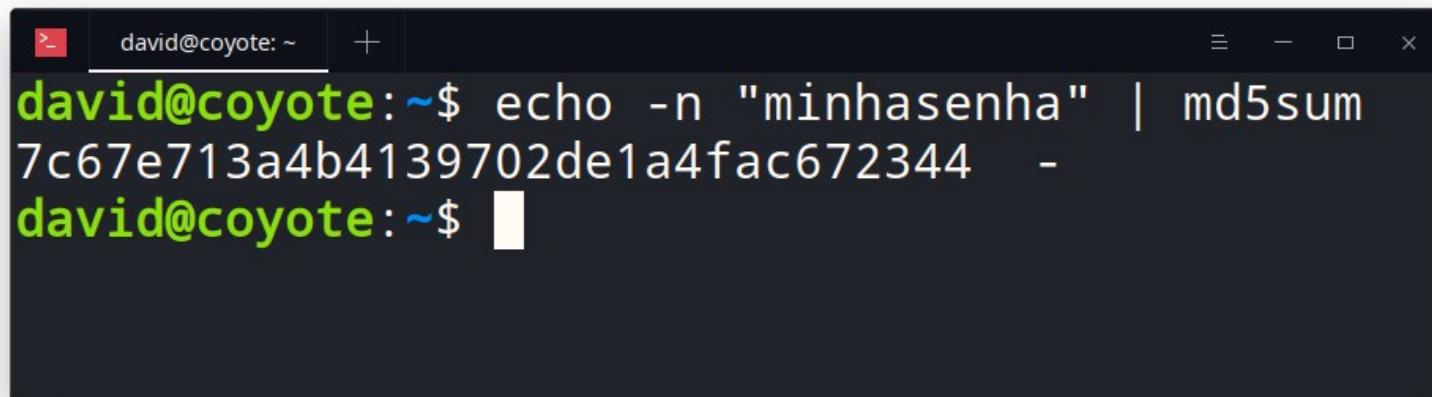
# Funções HASH

- Uma função Hash é um algoritmo que transforma um bloco de dados qualquer em série de caracteres de comprimento fixo
  - Os comandos **md5sum** e **sha1sum** são instalados por padrão em sistemas UNIX , GNU/Linux e BSD. Eles são usados para verificar a integridade de dados transmitidos através da Web. Para maiores informações, vide [Soma de Verificação \(Checksum\)](#)
- 

The screenshot shows a terminal window with two tabs. The top tab is active and displays the command: `david@coyote:~$ echo -n "minhasenha" | md5sum`. The output is `7c6`. The bottom tab is inactive and displays the command: `david@coyote:~$ echo -n "minhasenha" | sha1sum`. The output is `93d51f52fbfe1e944f084727df24993e88caee7`. Below that, another command is shown: `david@coyote:~$ echo -n "ufam" | sha1sum`, with the output `4ae250e25cb5e0bdb06cfcc2513cb451f4a610d0e`. Finally, the command `david@coyote:~$ echo -n "computacao" | sha1sum` is shown with the output `915769113d302e7893cc3019c5fc5dcb36682bca`.

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
  - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
  - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco



```
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
    - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
    - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que **ninguém** poderá descobrir a senha do usuário caso acesso a tabela Usuário do banco de dados

```
david@coyote:~$ echo -n "minhasenha" | md5sum  
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

# Funções HASH

- Considere um sistema que usa a função hash MD5 para criptografar as senhas de seus usuários
    - Ex: se a senha de um usuário é **minhasenha**, a string armazenada no banco será **7c67e713a4b4139702de1a4fac672344**
    - Quando o usuário tentar logar no sistema, o servidor irá comparar o md5 da senha informada com a string armazenada no banco
- Note que ninguém poderá descobrir a senha do usuário caso a Será mesmo? Então vamos no Google e inserir a string hash **7c67e713a4b4139702de1a4fac672344** no campo de busca

```
7c67e713a4b4139702de1a4fac672344 -  
david@coyote:~$ █
```

# Funções HASH

- Considerar criptografia
  - Ex: se a senha é armazenada no banco de dados
  - Quando o usuário digita a senha, o sistema md5 a e compara com a que está no banco de dados

7c67e713a4b4139702de1a4fac672344

Todas

Maps

Vídeos

Imagens

Shopping

Mais



Aproximadamente 43 resultados (0,27 segundos)

[7c67e713a4b4139702de1a4fac672344 - Hash Toolkit](#)

<https://hashtoolkit.com/reverse-md5-hash> › 7c67e71... - Traduzir esta página

Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344 ... md5,

7c67e713a4b4139702de1a4fac672344, minhasenha ...

[Hash Md5: 7c67e713a4b4139702de1a4fac672344 - MD5Hashing.net](#)

<https://md5hashing.net/hash> › hash › 7c67e713a4b4139702de1a4fac672344

Decoded hash Md5: 7c67e713a4b4139702de1a4fac672344: minhasenha.

[Usando o md5 - Recursos do PHP](#)

[recursosdophp.blogspot.com](http://recursosdophp.blogspot.com/2011/02/normal-0-21-false-false-false-pt-br-x.html) › normal-0-21-false-false-false-pt-br-x

md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você mudar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



# Funções HASH

Best MD5 Password Decoder

Secure | https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344

## Hash Toolkit

Search in 15,786,077,212 decrypted md5 / sha1 hashes.

Hash: 7c67e713a4b4139702de1a4fac672344

Decrypt md5 Hash Results for: 7c67e713a4b4139702de1a4fac672344

Algorithm	Hash	Decrypted
md5	7c67e713a4b4139702de1a4fac672344	minhasenha

recursosdophp.blogspot.com › normal-0-21-false-false-false-pt-br-x ▾  
md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você mudar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .

Google

# Funções HASH

Best MD5 Password Decryption

Secure | https://hashtoolkit.com/reverse-md5-hash/7c67e713a4b4139702de1a4fac672344

## Hash Toolkit

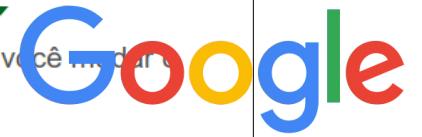
Search in 15,786,077,212 decrypted md5 / sha1 hashes.

Sistemas como o **Hash Toolkit** usam tabelas contendo bilhões de valores de hash pré-calculados. Essas tabelas normalmente são chamadas de **rainbow tables**.

Decrypt md5 Hash Results for: **7c67e713a4b4139702de1a4fac672344**

Algorithm	Hash	Decrypted
md5	7c67e713a4b4139702de1a4fac672344	Q minhasenha Q

recursosdophp.blogspot.com › normal-0-21-false-false-false-pt-br-x  
md5("minhasenha") = "7c67e713a4b4139702de1a4fac672344". Com isso se você mudar o valor de "\$senha" ele vai dar senha Invalida. Bom, isso até mais .



# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência adicional de caracteres aleatórios, chamada salt
- Para exemplificar, podemos aplicar o salt **Us8#upK12MjsM** à senha original do usuário, **minhasenha**

```
david@coyote: ~$ echo -n "Us8#upK12MjsMminhasenha" | md5sum  
d3eeb71a83744a4bbaa3ce41be87a292 -  
david@coyote: ~$ █
```

# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma sequência aleatória de caracteres.

A screenshot of a Google search results page. The search bar at the top contains the query "d3eeb71a83744a4bbbaa3ce41be87a292". Below the search bar, the navigation menu includes "Todas", "Maps", "Vídeos", "Imagens", "Shopping", "Mais", and "Configurações". The main search results area displays the message: "Sua pesquisa - d3eeb71a83744a4bbbaa3ce41be87a292 - não encontrou nenhum documento correspondente." Underneath this message, there is a section titled "Sugestões:" with three bullet points:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

The Google logo is visible at the bottom right of the search results page.

# Salt



- O **salt** é uma estratégia utilizada para evitar que duas senhas idênticas produzam **hashes** idênticos
- A ideia é concatenar a senha original do usuário com uma string aleatória.
- De uma forma geral, o salt é gerado aleatoriamente para cada usuário e armazenado no banco junto com o hash MD5 resultante:  
**Us8#upK12MjsMd3eeb71a83744a4bbaa3ce41be87a292**

Sua pesquisa - **d3eeb71a83744a4bbaa3ce41be87a292** - não encontrou nenhum documento correspondente.

Sugestões:

- Certifique-se de que todas as palavras estejam escritas corretamente.
- Tente palavras-chave diferentes.
- Tente palavras-chave mais genéricas.

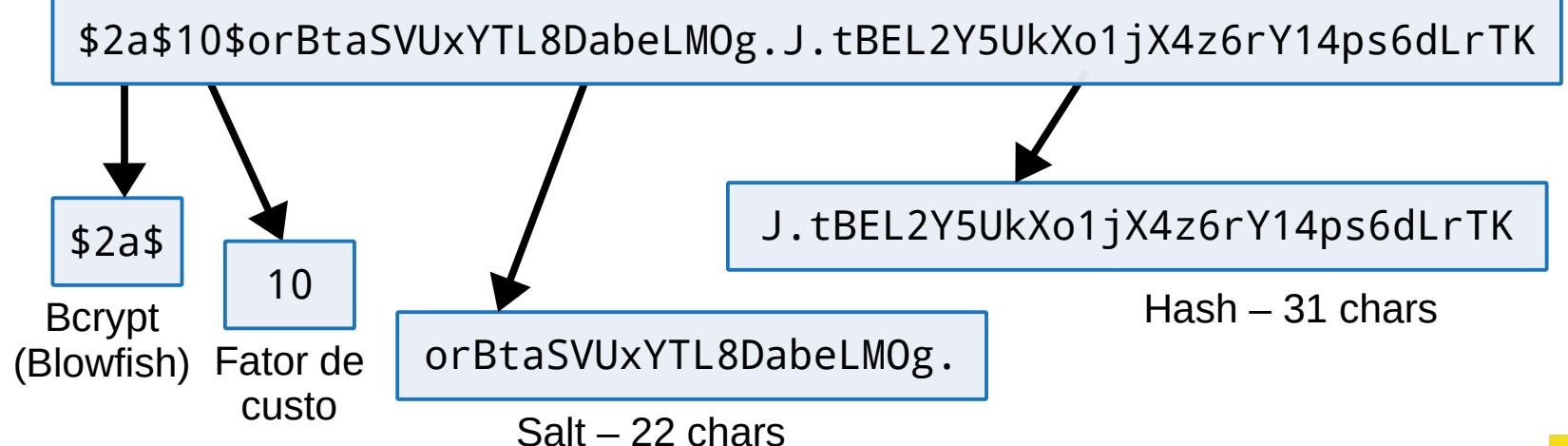


# O módulo bcrypt

- O módulo **bcrypt** é uma boa opção para geração de senhas, pois incorpora uma função hash à uma estratégia de salt

```
$ npm install bcryptjs  
$ npm install -D @types/bcryptjs
```

- O bcrypt é um **algoritmo de hash** usado para geração de senhas em sistemas como OpenBSD e algumas distribuições linux



# O módulo bcrypt

- Para gerar uma senha com salt podemos usar o código abaixo, onde rounds é o número de rounds, e **senha** é a senha informada pelo usuário

```
const salt = await bcrypt.genSalt(rounds);
const hash = await bcrypt.hash(senha, salt);
```

- Para verificar se uma senha está correta (no resource auth), podemos usar a função **compare** do bcrypt

```
const ok = await bcrypt.compare(senha, hash);
```

Senha  
criptografada  
(valor de hash  
do código  
anterior)

# CRUD de Usuários

- Os DTOs usados no CRUD de usuários serão **CreateUsuarioDto**, **UsuarioDto** e **UpdateUsuarioDto**
  - Definidos no arquivo `src/resources/usuario/usuario.types.ts`
- O código do roteador, mostrado abaixo, estabelece quais serão as rotas e funções do controlador

```
router.get('/', usuarioController.index);
router.post('/', usuarioController.create);
router.get('/:id', usuarioController.read);
router.put('/:id', usuarioController.update);
router.delete('/:id', usuarioController.remove);
```

- O serviço irá contar com as funções **getAllUsuarios**, **createUsuario**, **updateUsuario**, **buscaUsuarioPorEmail**, **buscaUsuarioPorId** e **deleteUsuario**

Insomnia - Loja - Create Usuario

Application Edit View Window Tools Help

Insomnia / Loja ▾

No Environment ▾ Cookies

POST `_.base_url/v1/usuario` Send

JSON Auth Query Header 1 Docs

```
1▼ {  
2   "tipoUsuarioId": "7edd25c6-c89e-4c06-ae50-c3c32d71b8ad",  
3   "nome": " Faker ⇒ Name ",  
4   "email": " Faker ⇒ Internet ",  
5   "senha": "12345678"  
6 }
```

Filter +

GET Get All Tipos

Usuário

GET Get All Usuarios

GET Read Usuario

POST Create Usuari

POST Create Usuári

POST Create Usuário Cliente

PUT Update Usuario

DEL Delete Usuario

Beautify JSON

Exercício: Desenvolva o CRUD de usuário na pasta `src/resources/usuario`

github  
**ExpAPI**

express JS

# Resource Auth

- Além do resource Usuário, será criado um resource Auth contendo as funções de **signup**, **login** e **logout**

```
// Arquivo src/resources/auth.router.ts
import { Router } from 'express';
import authController from './auth.controller';

const router = Router();

router.post('/signup', authController.signup);
router.post('/login', authController.login);
router.post('/logout', authController.logout);

export default router;
```

```
// Arquivo src/router/v1Router.ts
import authRouter from '../resources/auth/auth.router';
...
router.use('/', authRouter);
...
```

## Resources Auth

- Além do resource Usuário, o resource Auth contendo as funções de:

A função **signup** será usada para que novos clientes criem seu próprio cadastro na loja virtual

```
// Arquivo src/resources/auth/auth.router.ts
import { Router } from 'express';
import authController from './auth.controller';

const router = Router();

router.post('/signup', authController.signup);
router.post('/login', authController.login);
router.post('/logout', authController.logout);

export default router;
```

```
// Arquivo src/router/v1Router.ts
import authRouter from '../resources/auth/auth.router';
...
router.use('/', authRouter);
...
```

## Resources Auth

- Além do resource Usuário, o resource Auth contendo as funções de:

```
// Arquivo src/resources/auth/auth.router.ts
import { Router } from 'express';
import authController from '../controllers/authController';

const router = Router();

router.post('/signup', authController.signup);
router.post('/login', authController.login);
router.post('/logout', authController.logout);

export default router;
```

A função **signup** será usada para que novos usuários criem seu cadastro na loja virtual.

Ela difere da função **create** do resource Usuário, que será usada apenas por administradores para criar novos usuários.

```
// Arquivo src/router/v1Router.ts
import authRouter from '../resources/auth/auth.router';
...
router.use('/', authRouter);
...
```

# Resource Auth

- A função **signup** do controlador irá usar uma função **createUsuário**, da camada de serviço do **resource Usuario**

```
// Arquivo src/resources/auth.controller.ts

const signup = async (req: Request, res: Response) => {
  const usuario = req.body as SignUpDto;
  try {
    if (await buscaUsuarioPorEmail(usuario.email))
      return res
        .status(400)
        .json({ msg: 'Email informado já está sendo usado' });
    const newUsuario = await createUsuario({
      ...usuario,
      tipoUsuarioId: TiposUsuarios.CLIENT,
    });
    res.status(201).json(newUsuario);
  } catch (e: any) {
    res.status(500).json(e.errors);
  }
};
```

Signup só permite a criação de usuários **cliente**

Usa o **bcryptjs** para criptografar as senhas

JS

# Resource Auth

- A função **login** do controlador Auth irá usar uma função **checkAuth** da camada de serviço de Auth

```
// Arquivo src/resources/auth.service.ts

export const checkAuth = async (
  credenciais: LoginDto,
): Promise<Usuario | null> => {
  const { email, senha } = credenciais;
  const usuario = await Usuario.findOne({ where: { email } });
  if (!usuario) return null;
  const ok = await bcrypt.compare(senha, usuario.senha);
}
```



Usa o  
**bcryptjs** para  
verificar a senha  
digitada pelo  
usuário no  
login

# Resource Auth

- Ao efetuar o login, criamos as variável de sessão **uid** e **tipoUsuarioId** serão criadas

```
// Arquivo src/resources/auth.controller.ts

const login = async (req: Request, res: Response) => {
  const { email, senha } = req.body;
  try {
    const usuario = await checkAuth({ email, senha });
    if (!usuario)
      return res.status(401).json({
        msg: 'Email e/ou senha incorretos'
      });
    req.session.uid = usuario.id;
    req.session.tipoUsuarioId = usuario.tipoUsuarioId;
    res.status(200).json({ msg: 'Usuário autenticado' });
  } catch (e) {
    res.status(500).json(e);
  }
}
```

Variáveis  
de  
sessão

# Resource Auth

- Para que as variáveis de sessão funcionem, é preciso adicionar tais atributos à interface **SessionData** de **express-session**
- Para isso, adicionamos o seguinte código no começo (após a importação dos pacotes) do arquivo **src/index.ts**

```
declare module "express-session" {  
    interface SessionData {  
        uid: string;  
        tipoUsuario: string  
    }  
}
```

Insomnia - Loja - Login como Admin

Application Edit View Window Tools Help

Insomnia / Loja ▾

No Environment ▾ Cookies

POST `_.base_url` /v1/login Send

Filter +

Auth

POST Signup

POST Logout

POST Login como Admin

POST Login como Cliente

Tipo de Usuário

GET Get All Tipos

JSON Auth Query Header 1 Docs

```
1▼ {  
2   "email": "Rosina.Stark@gmail.com",  
3   "senha": "12345678"  
4 }
```

Beautify JSON

200 OK 61.5 ms 42 B 4 Hours Ago ▾

Preview Header 10 Cookie Timeline

```
1▼ {  
2   "msg": "Usuário autenticado com sucesso"  
3 }
```

`$.store.books[*].author` ?

express JS



## Manage Cookies



## Filter Cookies

twitter.com

Records

localhost

connect.sid=s%3A2eed0151-1312-44eb-8559-cbba  
8152c05b.J1zBsgOcoZLzgBRzLEYC4XuxtUmOzkbl  
NPiiJWdB23g; Path=/; HttpOnly; Domain=localhos  
t

Edit

*\* cookies are automatically sent with relevant requests*

Done

GET /api/tipos

\$.store.books[\*].author

# Mecanismo de Autorização

- Um middleware **isAdmin** pode ser criado para restringir usuários não autorizados de determinadas partes da aplicação

```
// Arquivo src/middlewares/isAdmin.ts

import { Request, Response, NextFunction } from 'express';
import { checkIsAdmin } from '../resources/auth/auth.service';

const isAdmin = async (
  req: Request,
  res: Response,
  next: NextFunction
) => {
  const uid = req.session.uid;
  if (uid && (await checkIsAdmin(uid))) next();
  else res.status(403).json({ msg: 'Não autorizado' });
};

export default isAdmin;
```

# Mecanismo de Autorização

- Uma vez que o middleware **isAdmin** seja criado, podemos impedir que usuários não logados ou clientes accessem rotas que devem ser usadas apenas por usuários admin

```
// Arquivo src/resources/usuario/usuario.router.ts

import { Router } from 'express';
import produtoController from './produto.controller';
import isAdmin from '../../middlewares/isAdmin';
const router = Router();

router.get('/', produtoController.index);
router.post('/', isAdmin, produtoController.create);
router.get('/:id', produtoController.read);
router.put('/:id', isAdmin, produtoController.update);
router.delete('/:id', isAdmin, produtoController.remove);

export default router;
```

# Mecanismo de Autorização

- Uma vez que o middleware **isAdmin** seja criado, podemos impedir que usuários não logados ou clientes acessem rotas que devem ser usadas apenas por usuários admin

```
// Arquivo src/resources/usuario/usuario.router.ts
```

```
import { Router } from 'express';
import produtoController from './produto.controller';
import isAdmin from '../../middlewares/isAdmin';
const router = Router();
```

**Exercício:** Desenvolva o middleware **isAdmin**, seguindo os passos mostrados anteriormente. Além disso, desenvolva um middleware **isAuth**, similar ao **isAdmin**, mas que apenas verifica se o usuário está logado ou não

```
export default router;
```

github

ExpAPI

express JS



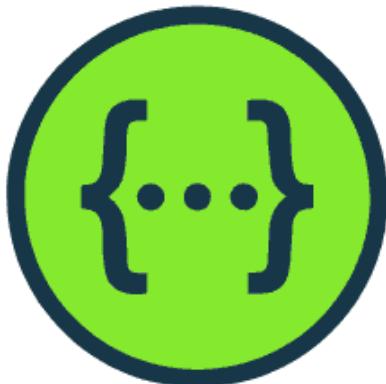
**Exercício:** Implemente um **resource compra** que permita o usuário i) adicionar produtos no carrinho de compra, e ii) concluir a compra salvando os itens do carrinho no banco de dados. Dica: o carrinho de compra pode ser representado por uma variável de sessão contendo um array dos produtos com suas respectivas quantidades.

[github](#)  
**ExpAPI**

express **JS**

# Swagger

- O **Swagger** é uma ferramenta para documentação, consumo e visualização dos endpoints de uma API Rest
- Ele é comumente usado para que devs backend gerem a documentação de APIs para devs frontend



Swagger<sup>TM</sup>  
Supported by SMARTBEAR

express 

Swagger UI

localhost:3333/api/#/

 **Swagger**  
Supported by SMARTBEAR

# API da Loja virtual

1.0.0 OAS 2.0

[ Base URL: localhost:3333/ ]

Documentação da API da Loja virtual implementada durante o Web Academy.

Schemes

HTTP

## Auth

POST /v1/auth/ Registro de novos clientes.

PUT /v1/auth/ Login de usuários.

DELETE /v1/auth/ Logout de usuário logado.

## Produto

GET /v1/produto/ Listagem de produtos.

POST /v1/produto/ Adiciona um novo produto na base.

JS

# Swagger

- Para usar o Swagger, vamos precisar dos pacotes **swagger-ui-express**, **swagger-jsdoc** e **swagger-autogen**

```
$ npm i swagger-ui-express
$ npm i -D swagger-autogen @types/swagger-ui-express
```

- A documentação oficial do Swagger é bastante completa e está disponível no endereço <https://swagger.io/>

The screenshot shows a web browser window with the following details:

- Address Bar:** Displays the URL [swagger.io/docs/specification/basic-structure/](https://swagger.io/docs/specification/basic-structure/).
- Header:** Features the Swagger logo (a green circle with white brackets) and the text "Supported by SMARTBEAR". It also includes links for "Why Swagger?", "Tools", "Resources", a search bar, a "Sign In" button, and a "Try Free" button.
- Left Sidebar:** A vertical sidebar with the "SwaggerHub" logo and a list of links: "Swagger Inspector", "Open Source Tools", "OpenAPI Guide" (which is currently selected), and "What Is OpenAPI?".
- Main Content Area:** The title "Basic Structure" is displayed prominently. Below it, a paragraph explains that you can write OpenAPI definitions in YAML or JSON, noting that YAML examples are used here. It provides a sample OpenAPI 3.0 definition in YAML.
- Sample YAML Definition:**

```
1. openapi: 3.0.0
2. info:
3.   title: Sample API
4.   description: optional multiline or single line description in CommonMark (https://spec.commonmark.org/)
```

# Swagger

- Gerar a documentação do Swagger é uma tarefa difícil, e por isso utilizamos ferramentas para automatizar esse processo
- Em nossa aplicação, vamos utilizar um pacote do framework Express chamad **Swagger Autogen** para gerar a documentação



- **Swagger Autogen**

# Swagger

- Para usar o **swagger-autogen**, o primeiro passo é criar um arquivo **src/swagger.ts**, com o conteúdo abaixo

```
// Arquivo src/swagger.ts

import swaggerAutogen from "swagger-autogen";
import dotenv from "dotenv";
dotenv.config();

const doc = {
  info: {
    title: "API da Loja virtual",
    description: "Documentação da API",
  },
  host: `${process.env.HOST}:${process.env.PORT}`,
};

const outputFile = "./swagger-output.json";
const routes = ["./src/router/index.ts"];

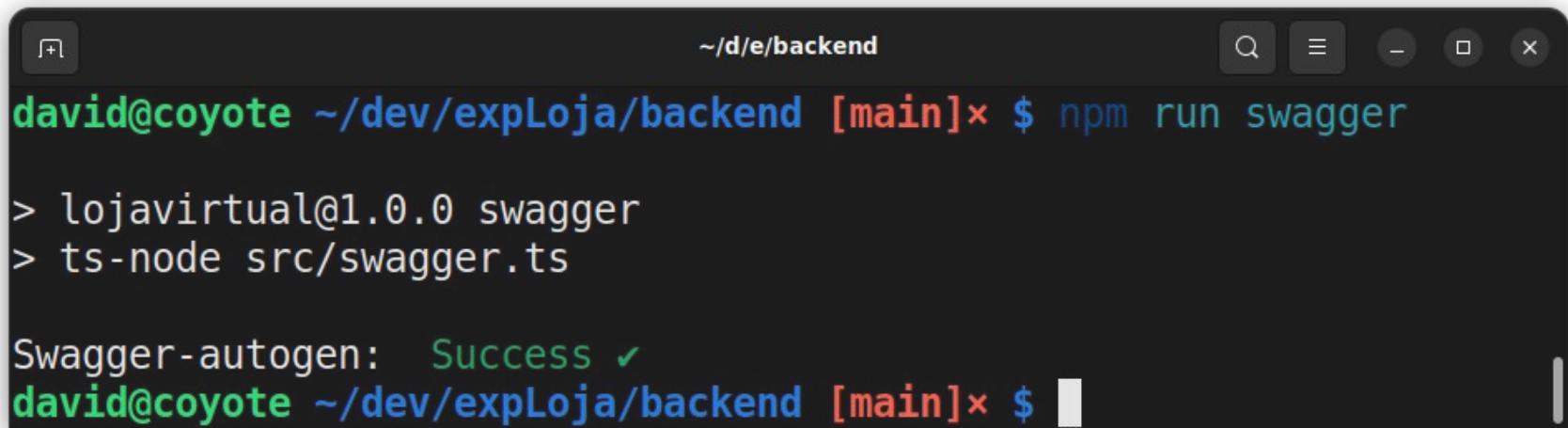
swaggerAutogen()(outputFile, routes, doc);
```

O swagger autogen usa os arquivos de rotas para documentar a API

# Swagger

- Após isso, criamos um novo script no arquivo **package.json** que será usado para gerar a documentação do Swagger

```
"scripts": {  
  "start": "nodemon -e js,json,ts,yaml src/index.ts",  
  "swagger": "ts-node src/swagger.ts"  
},
```

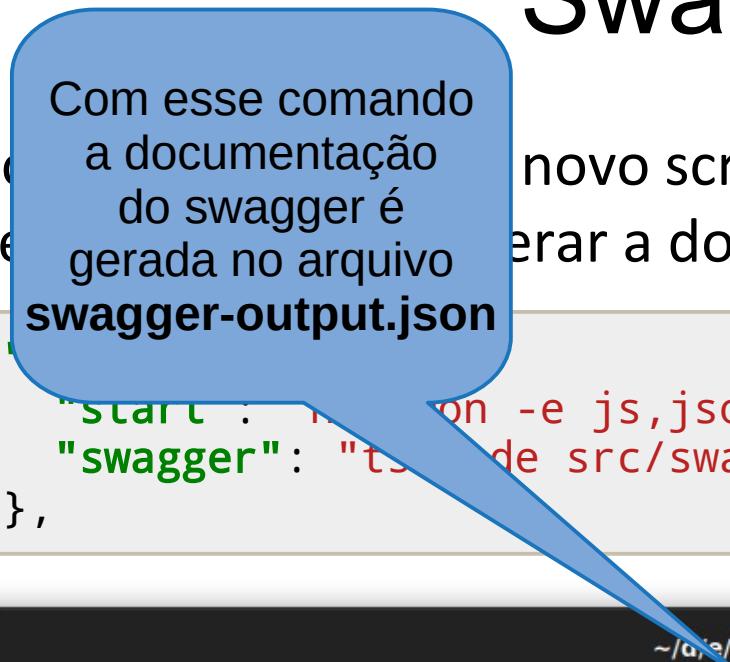


A screenshot of a terminal window titled '~ /d/e/backend'. The window shows the command `david@coyote ~/dev/expLoja/backend [main] $ npm run swagger` being run. The output shows the command `> lojavirtual@1.0.0 swagger` and `> ts-node src/swagger.ts`. At the bottom, it says `Swagger-autogen: Success ✓`.

# Swagger

- Após executar o comando, a documentação do swagger é gerada no arquivo **swagger-output.json**

```
"start": "ts-node --es module -e js,json,ts,yaml src/index.ts",
"swagger": "ts-node --es module src/swagger.ts"
},
```



```
david@coyote ~/dev/expLoja/backend [main]x $ npm run swagger

> lojavirtual@1.0.0 swagger
> ts-node src/swagger.ts

Swagger-autogen: Success ✓
david@coyote ~/dev/expLoja/backend [main]x $
```

# Swagger

- Para disponibilizar a documentação, é necessário configurar o pacote `swagger-ui-express` no `src/index.ts`

```
// Arquivo src/index.ts
...
import swaggerUi from "swagger-ui-express";
import swaggerFile from "./swagger-output.json";
...
app.use("/api", swaggerUi.serve, swaggerUi.setup(swaggerFile));
```

Swagger UI

localhost:3333/api/#/

## default

A documentação já pode ser acessada, mas precisa de ajustes

PUT /v1/usuario/{id}

DELETE /v1/usuario/{id}

POST /v1/linguagem/change

GET /v1/usuario/

POST /v1/usuario/

GET /v1/usuario/{id}

PUT /v1/usuario/{id}

DELETE /v1/usuario/{id}

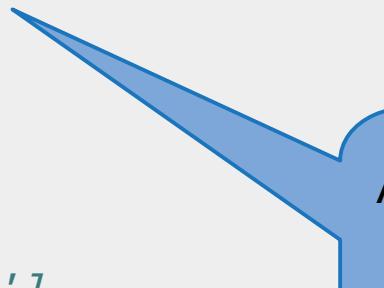
POST /v1/auth/

express JS

# Swagger

- O primeiro passo para melhorar a documentação é definir as tags de cada resource no arquivo **router/v1Router.ts**

```
router.use(  
  "/auth",  
  // #swagger.tags = ['Auth']  
  authRouter  
);  
  
router.use(  
  "/produto",  
  // #swagger.tags = ['Produto']  
  produtoRouter  
);  
  
router.use(  
  "/usuario",  
  // #swagger.tags = ['Usuario']  
  usuarioRouter  
);
```



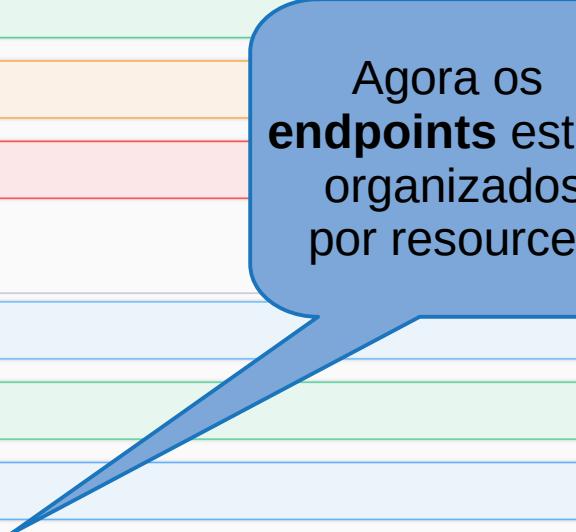
Atribuindo a tag  
Auth para os  
**endpoints** do  
resource Auth

▪ O

ta

■

Agora os  
**endpoints** estão  
organizados  
por resources



Swagger UI

localhost:3333/api/#/

**Auth**

- POST /v1/auth/
- PUT /v1/auth/
- DELETE /v1/auth/

**Produto**

- GET /v1/produto/
- POST /v1/produto/
- GET /v1/produto/{id}
- PUT /v1/produto/{id}
- DELETE /v1/produto/{id}

**Linguagem**

- POST /v1/linguagem/change

**Usuário**

- GET /v1/usuario/
- POST /v1/usuario/

JS

Swagger UI

localhost:3333/api/#/Produto/post\_v1\_produto\_

## Produto

GET /v1/produto/

POST /v1/produto/

Parameters

Name Description

body object (body)

Example Value | Model

```
{ "nome": "any" }
```

Parameter content type

application/json

Responses

Code Description

201 Created

400 Bad Request

500 Internal Server Error

GET /v1/produto/{id}

PUT /v1/produto/{id}

Try it out

No entanto, dentro de cada endpoint não existem muitas informações

JS

# Swagger

- Para a documentação dos endpoints, precisamos informar exemplos dos tipos de dados usados em `src/swagger.ts`

```
const doc = {  
  ...  
  definitions: {  
    CreateProdutoDto: {  
      nome: "Martelo",  
      preco: 29.0,  
      estoque: 10,  
    },  
    Produto: {  
      id: "8a2053de-5d92-4c43-97c0-c9b2b0d56703",  
      nome: "Bacon",  
      preco: 261,  
      estoque: 1,  
      createdAt: "2023-11-07T19:27:15.645Z",  
      updatedAt: "2023-11-07T19:27:15.645Z",  
    },  
  },  
};
```

# Swagger

- A partir disso, podemos adicionar informações de cada endpoint no controlador do resource

```
const create = async (req: Request, res: Response) => {
  /*
  #swagger.summary = 'Adiciona um novo produto na base.'
  #swagger.parameters['body'] = {
    in: 'body',
    schema: { $ref: '#/definitions/CreateProduto' }
  }
  #swagger.responses[200] = {
    schema: { $ref: '#/definitions/Produto' }
  }
  */
  const produto = req.body as CreateProdutoDto;
  ...
};
```

# Swagger

- A partir disso, podemos adicionar informações de cada endpoint no controlador do resource

```
const read = async (req: Request, res: Response) => {
  /*
    #swagger.summary = 'Recupera dados de um produto específico.'
    #swagger.parameters['id'] = { description: 'ID do produto' }
    #swagger.responses[200] = {
      schema: { $ref: '#/definitions/Produto' }
    }
  */
  const { id } = req.params;
  ...
};

const produto: TProduct = await createProduto(
  ...
);

};
```

Swagger UI

localhost:3333/api/#/

# Swagger

Supported by SMARTBEAR

## API da Loja virtual

1.0.0 OAS 2.0

[ Base URL: localhost:3333/ ]

Documentação da API da Loja virtual implementada durante o Web Academy.

Schemes

HTTP

### Auth

POST /v1/auth/ Registro de novos clientes.

PUT /v1/auth/ Login de usuários.

DELETE /v1/auth/ Logout de usuário logado.

### Produto

GET /v1/produto/ Listagem de produtos.

POST /v1/produto/ Adiciona um novo produto na base.

JS

Swagger UI    +

localhost:3333/api/#/

# Swagger

Supported by SMARTBEAR

## API da Loja virtual 1.0.0 OAS 2.0

[ Base URL: localhost:3333/ ]

Documentação da API da Loja virtual implementada durante o Web Academy.

Schemes

HTTP

### Auth

POST /v1/auth/ Registro de novos clientes.

D

**Exercício:** Crie a página com a documentação do swagger usando as instruções dos slides

github  
**ExpAPI**

### Produto

GET /v1/produto/ Listagem de produtos.

POST /v1/produto/ Adiciona um novo produto na base.

JS