

Introdução ao desenvolvimento de APIs

Node + Express

Prof. Luís Eduardo Tenório Silva
luís.silva@garanhuns.ifpe.edu.br

- Me. Luís Eduardo Tenório Silva

- **Experiência acadêmica:**

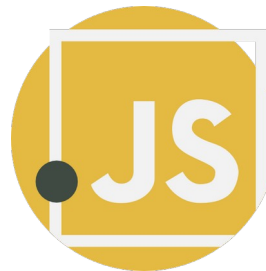
- » Técnico em Redes de computadores – IFAL (Palmeira dos Índios);
- » Graduado em Ciência da Computação – UFAL (Arapiraca);
- » Especialista em Testes e Desenvolvimento de sistemas – Motorola (Recife);
- » Mestrado em Ciência da Computação – UFPE (Recife);
- » **Doutorando em Ciência da Computação – UFCG.**

- **Experiência profissional:**

- » Engenheiro de Software – Motorola;
- » Engenheiro DevOps – LSD (UFCG);
- » Engenheiro de Software – Inatel (Santa Rita do Sapucaí – MG);
- » Engenheiro de Software – Zup (Banco Itaú, SP);
- » Professor no IFPE *campus* Vitória de Santo Antão – PE;
- » **Professor no IFPE *campus* Garanhuns – PE.**

- Como um **sistema web moderno** é projetado?
 - » Divisão de responsabilidades;
 - » **Frontend**: responsável pela parte visual (o que é renderizado pelo navegador)
 - Tecnologias bases utilizadas: HTML, CSS, Javascript;
 - Outras Tecnologias: React, Vue, Svelte, Angular, Next.JS.

- Qualquer elemento no frontend pode ser manipulado pelo usuário (parte visual):
 - » Backend abstrai partes que podem ser acessados pelo cliente de partes que somente a aplicação conhece.
- Processa solicitações vindas do cliente (um serviço, frontend, etc)
- Funções do backend:
 - » Segurança
 - » Banco de dados
 - » Serviços externos
 - » Regras de negócio
 - » Validações de dados
 - » *Logs* e monitoramento

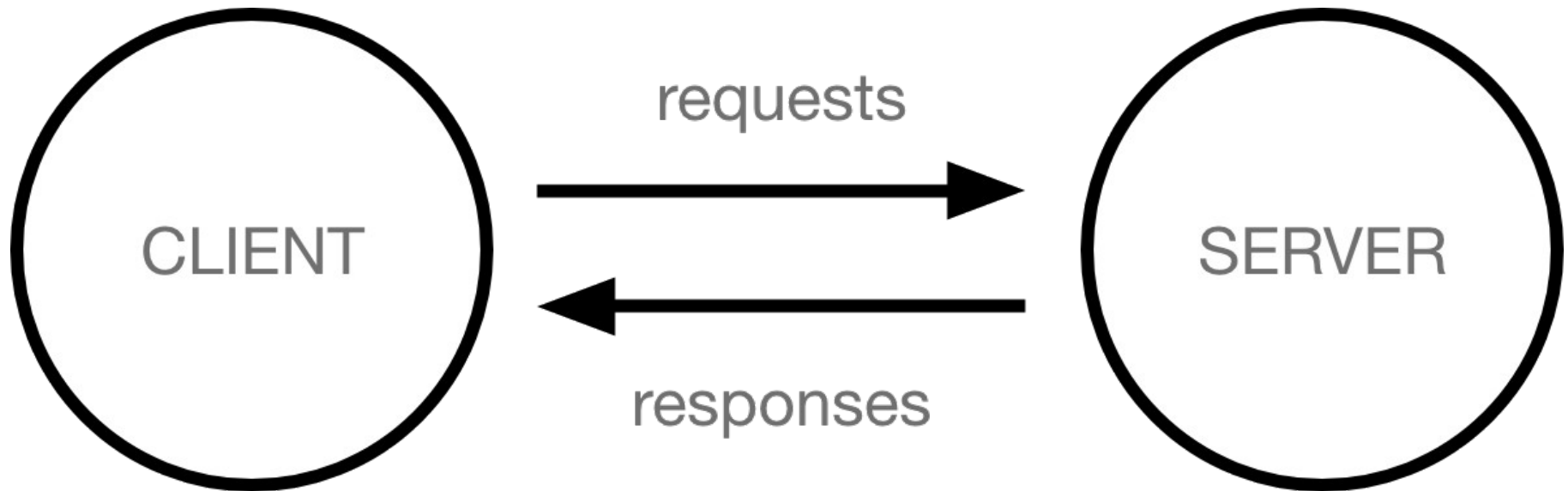


- API (Application Programming Interface)
 - » Comunicação entre sistemas diferente;
 - Sem necessidade de conhecimento de detalhes internos de implementação;
 - » Pode seguir um ou mais estilo arquitetural (modo como expõe os serviços para serem consumidos pelos sistemas):
 - REST (*Representational State Transfer*)
 - SOAP
 - RPC (*Remote Procedure Call*)
 - » Expõe **pontos de acesso (rotas)** que são acessados através de um protocolos de comunicação (**por ex: HTTP**);

Como a web funciona?

6

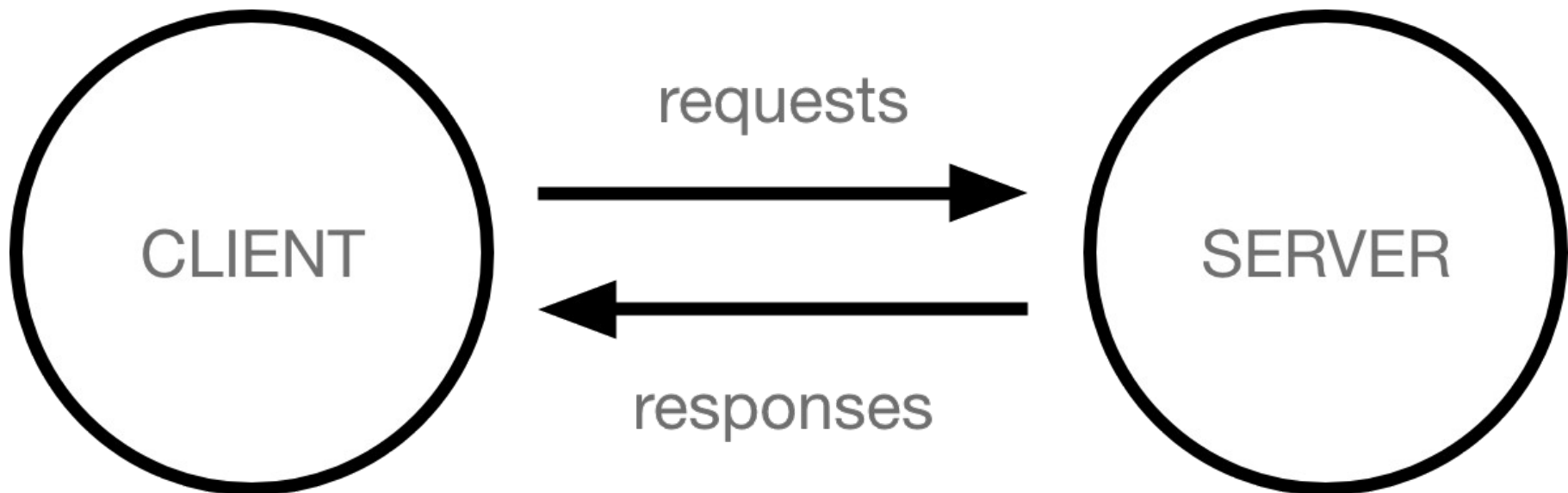
- **Arquitetura cliente x servidor**



Arquitetura cliente x servidor

- Clientes

- » Realizam requisições de recursos Web (páginas, imagens, áudio, vídeo) ao servidor;
- » Clientes não se conectam a outros clientes diretamente;
- » Diversos dispositivos podem ser clientes



Arquitetura cliente x servidor

- As requisições web são realizadas através do protocolo **HTTP (HyperText Transfer Protocol)**
 - » Protocolo de transferência de hipertexto;
 - » Permite o cliente requisitar um recurso web:
 - Páginas HTML
 - Arquivos CSS, Javascript, Imagens, Vídeos
- O servidor recebe uma requisição web, processa-a, e encaminha ao cliente uma resposta web usando o protocolo **HTTP**.

- Ambiente de execução JS no lado servidor;
- Construído sobre o motor v8:
 - » Interpretador Javascript;
 - » Utilizado no Google Chrome.
- Características:
 - » Assíncrono;
 - » Utilização de módulos e pacotes;
 - » Escalável;



Implementando uma API REST

- Criar um novo diretório
 - » *mkdir eic-pin-2024*
 - » *cd eic-pin-2024*
- Criar um projeto node
 - » *npm init -y*
- Instalar dependências de desenvolvimento
 - » *npm i -D nodemon*
- Instalar dependências da aplicação
 - » *npm i express*

- *Node Package Manager;*
- Gerenciador de dependências do node;
 - » Permite adicionar e remover dependências para um projeto;
 - » Gerencia dependências de desenvolvimento e produção;
- Gerencia ***scripts*** para atividades frequentes:
 - » Iniciar a aplicação;
 - » Iniciar em modo de desenvolvimento;
 - » Preparar as tabelas de um BD;
 - » Executar testes;
 - » Verificação de linters

package.json

12

```
{
  "name": "eic-pin-2024",
  "version": "1.0.0",
  "description": "Uma API de filmes apresentada no EIC2024",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "EIC2024",
    "express",
    "api-filmes"
  ],
  "author": "Luís Eduardo",
  "license": "MIT",
  "dependencies": {
    "express": "^4.21.1"
  }
}
```

Adicionar

- Serve os arquivos produzidos pela equipe de frontend;
- Criar uma pasta na raiz denominada public
 - » *mkdir public*
- Copiar todos os arquivos da página web para dentro da pasta public

eic-pin-2024/

├─ public/

| └─ css/

| | └─ style.css

| └─ js/

| | └─ script.js

| └─ index.html

└─ index.js

Prover arquivos estáticos

14

- Na raiz do projeto, crie um arquivo denominado index.js
 - » touch index.js
- Adicione ao arquivo as linhas abaixo:

```
import express from "express";

const app = express();

app.use(express.static("public/"));

const porta = 3000;
app.listen(porta, () => {
  console.log(`App executando na porta ${porta}...`);
});
```

- Abra o navegador e digite:
 - » `http://localhost:3000/`
- O navegador realiza uma requisição HTTP com o método GET e obtém como resposta o status 200 OK;
 - » Está presente no corpo da resposta HTTP o arquivo HTML *index.html*.

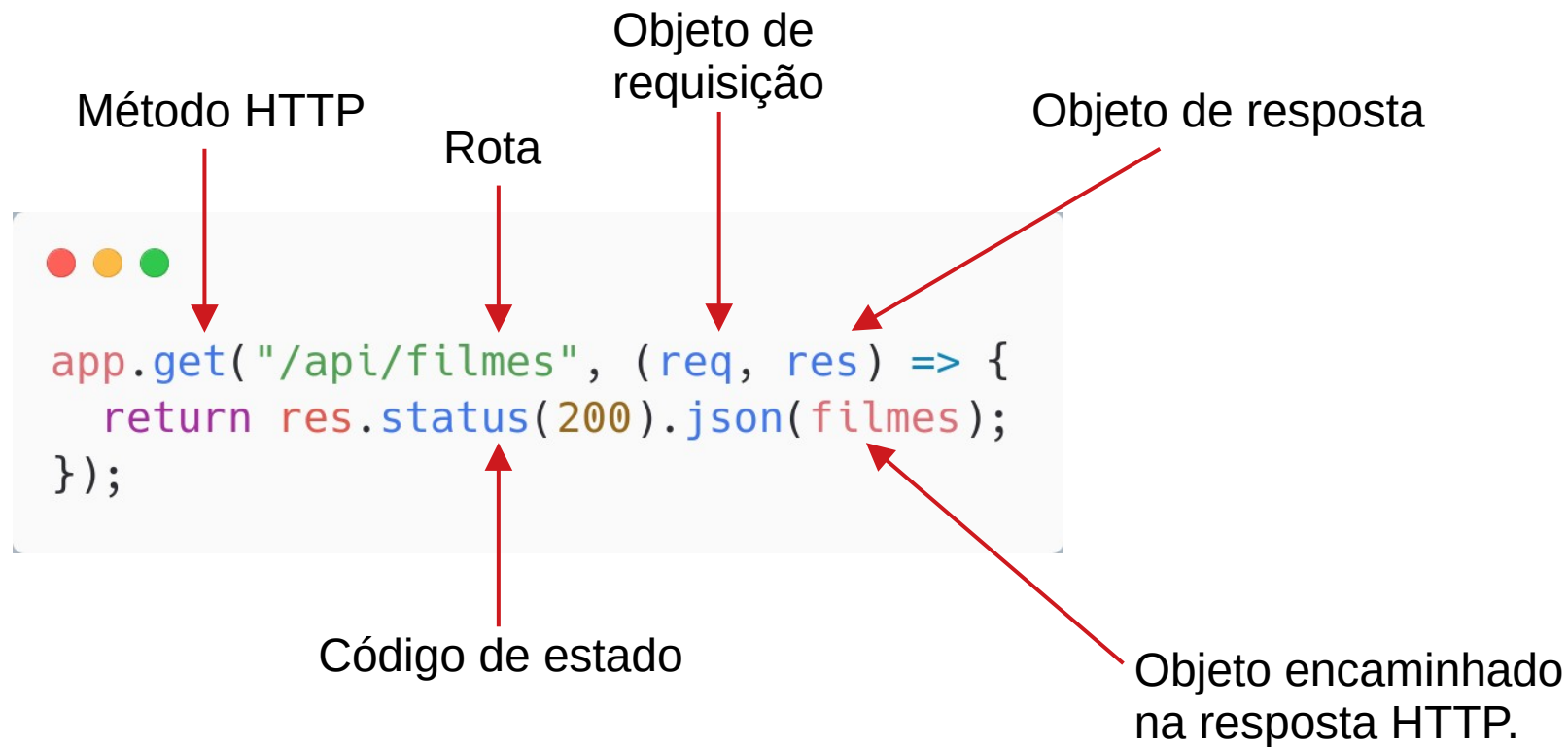
Lista de Filmes

A API não está online. Tente novamente mais tarde.

- Rota para **obter** todos os filmes:
 - » GET /api/filmes
- Rota para **obter** um filme:
 - » GET /api/filmes/<id>
- Rota para **adicionar** um filme:
 - » POST /api/filmes
- Rota para **atualizar** um filmes:
 - » PUT /api/filmes/<id>
- Rota para **remover** um filme:
 - » DELETE /api/filmes/<id>

- Requisição HTTP utilizada para **obter dados**;
- Não passa dados no corpo da requisição;
 - » Apenas dados na url.
 - » Ex: `http://localhost:3000/api/filmes&filter=a`
 - `filter=a` → parâmetro passado na URL que será tratado no backend.

- /api/filmes → obtém todos os filmes



Rota GET

19

```
let filmes = [  
  {  
    id: 0,  
    nome: "Tropa de Elite 2: O Inimigo Agora é Outro",  
    valor_ingresso: 25.0,  
    ano: 2010,  
  },  
  {  
    id: 1,  
    nome: "Dona Flor e Seus Dois Maridos",  
    valor_ingresso: 22.0,  
    ano: 1976,  
  },  
  {  
    id: 2,  
    nome: "Minha Mãe é uma Peça 3",  
    valor_ingresso: 20.0,  
    ano: 2019,  
  },  
  {  
    id: 3,  
    nome: "Se Eu Fosse Você 2",  
    valor_ingresso: 18.0,  
    ano: 2009,  
  },  
  {  
    id: 4,  
    nome: "De Pernas pro Ar 2",  
    valor_ingresso: 17.0,  
    ano: 2012,  
  },  
];
```

```
app.get("/api/filmes", (req, res) => {  
  return res.status(200).json(filmes);  
});
```

```
const filmes = [
  {
    id: 0,
    nome: "Tropa de Elite 2: O Inimigo Agora é Outro",
    valor_ingresso: 25.0,
    ano: 2010,
  },
  {
    id: 1,
    nome: "Dona Flor e Seus Dois Maridos",
    valor_ingresso: 22.0,
    ano: 1976,
  },
  {
    id: 2,
    nome: "Minha Mãe é uma Peça 3",
    valor_ingresso: 20.0,
    ano: 2019,
  },
  {
    id: 3,
    nome: "Se Eu Fosse Você 2",
    valor_ingresso: 18.0,
    ano: 2009,
  },
  {
    id: 4,
    nome: "De Pernas pro Ar 2",
    valor_ingresso: 17.0,
    ano: 2012,
  },
];

app.get("/api/filmes", (req, res) => {
  return res.status(200).json(filmes);
});
```

- No navegador, digite:

» <http://localhost:3000/api/filmes>

- Resultado:

```
[{"id":0,"nome":"Tropa de Elite 2: O Inimigo Agora é Outro","valor_ingresso":25,"ano":2010}, {"id":1,"nome":"Dona Flor e Seus Dois Maridos","valor_ingresso":22,"ano":1976}, {"id":2,"nome":"Minha Mãe é uma Peça 3","valor_ingresso":20,"ano":2019}, {"id":3,"nome":"Se Eu Fosse Você 2","valor_ingresso":18,"ano":2009}, {"id":4,"nome":"De Pernas pro Ar 2","valor_ingresso":17,"ano":2012}]
```

Respostas de sucesso (200-299)

22

- **200 OK:** Requisição bem sucedida
- **201 Created:** Requisição bem sucedida e novo recurso criado.
- **202 Accepted:** Nenhuma ação foi tomada sobre a requisição.
- **204 No Content:** Não existe um conteúdo para ser encaminhado ao cliente.
- **206 Partial Content:** Separa o conteúdo em vários fluxos.

- **301 Moved Permanently**: Objeto foi movido para outra URL (Definido no cabeçalho *Location* da mensagem de resposta).

Resposta de erro do cliente (400-499)

24

- **400 Bad Request**: Erro genérico. Requisição do cliente não está como esperado pelo servidor.
- **401 Unauthorized**: O cliente não está autenticado.
- **403 Forbidden**: O cliente não tem direitos de acesso ao conteúdo (autenticado).
- **404 Not Found**: Recurso requisitado não existe no servidor.
- **405 Method Not Allowed**: Método utilizado pelo cliente não é permitido (PUT, POST, DELETE...).

Resposta de erro do servidor (500-599)

25

- **500 Internal Server Error**: Erro genérico.
- **502 Bad Gateway**: Erro de comunicação com *gateway* do servidor.
- **503 Service Unavailable**: O servidor está sobrecarregado e não pode atender a requisição. Cabeçalho *Retry-After* deverá dizer o tempo para usuário retentar.
- **505 HTTP Version Not Supported**: Versão do protocolo HTTP não suportada pelo servidor.

- /api/filmes/<id> → obtém o filme com um id específico

```
● ● ●  
app.get("/api/filmes/:id", (req, res) => {  
  const id = req.params.id;  
  
  let resultado;  
  for (let x = 0; x < filmes.length; x++) {  
    if (filmes[x]["id"] == id) {  
      resultado = filmes[x];  
    }  
  }  
  
  if (resultado) {  
    return res.status(200).json(resultado);  
  } else {  
    return res.status(404).json(resultado);  
  }  
});
```

Parâmetro esperado

Obtém parâmetro id do objeto de requisição

Código de estado NOT FOUND

- Usando o **paradigma funcional**, pode-se simplificar a escrita da função anterior:

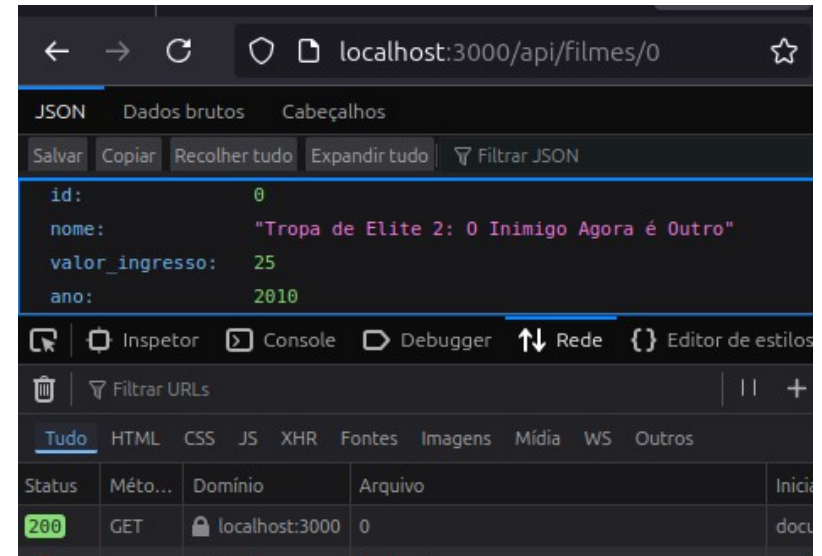
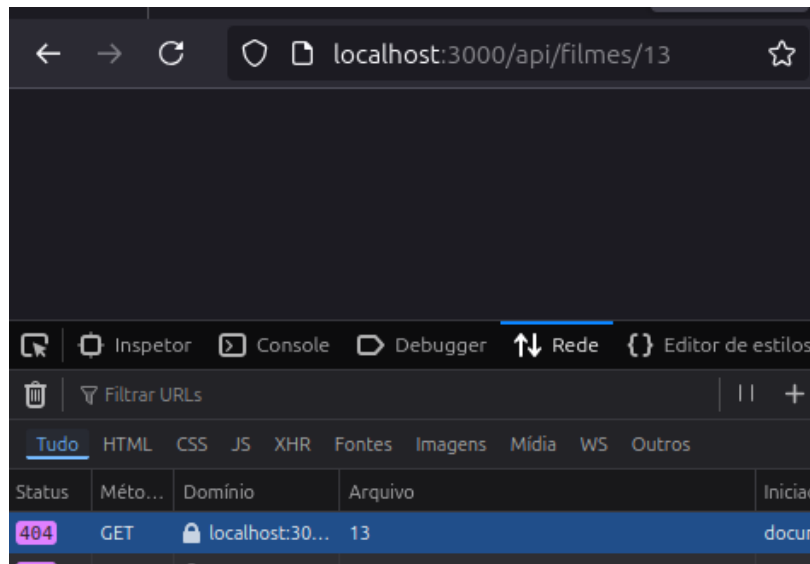


```
app.get("/api/filmes/:id", (req, res) => {  
  const { id } = req.params;  
  
  const resultado = filmes.find(f => f.id === id);  
  return res.status(resultado ? 200 : 404).json(resultado);  
});
```

Rota GET

28

- No navegador, pressione com o botão direito do mouse → inspecionar → rede



- Requisição HTTP utilizada para **encaminhar dados** ao lado do servidor;
 - » Os dados são postos no corpo (*body*) de uma requisição HTTP.
- Utilizado para:
 - » Criação de novos objetos;
 - » Encaminhar dados encriptados (desde que utilizando https);
 - » Requisições que precisam de autenticação;
 - » ...



```
app.use(express.static("public/"));  
app.use(express.json());
```

← Transforma o corpo da requisição em um objeto JSON

- /api/filmes: Adiciona um filme, encaminhado no corpo da requisição a lista de filmes:

```
app.post("/api/filmes", (req, res) => {  
  const filme = req.body; ← Corpo da requisição  
  
  for (let x = 0; x < filmes.length; x++) {  
    if (filmes[x].id == filme.id) {  
      return res.status(403).json( ← FORBIDDEN  
        {  
          mensagem: "Já existe um filme cadastrado com esse ID"  
        }  
      );  
    }  
  }  
  filmes.push(filme);  
  return res.status(201).json({mensagem: "Filme adicionado com sucesso"});  
})
```

← CREATED

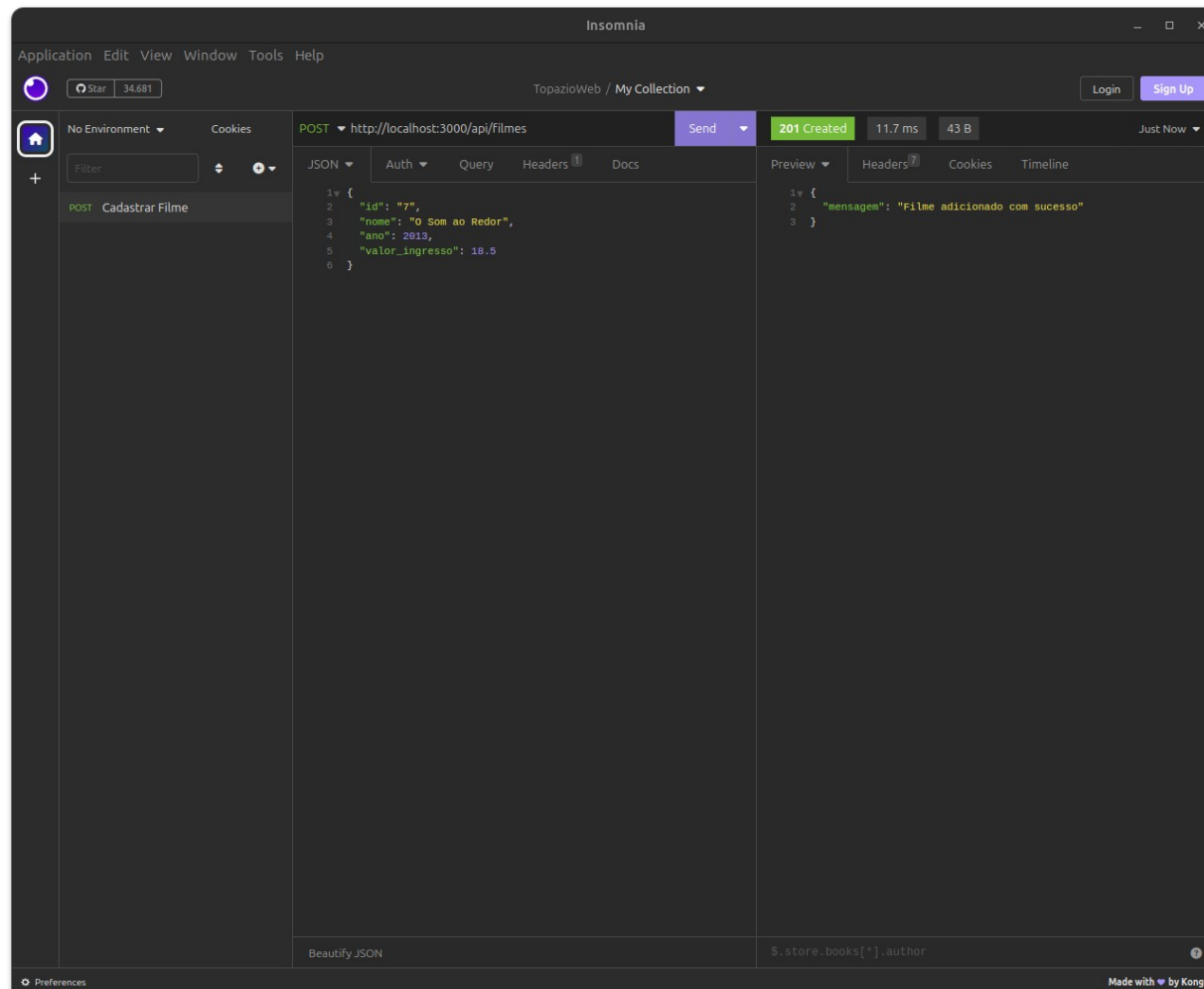
- /api/filmes: Adiciona um filme, encaminhado no corpo da requisição a lista de filmes:



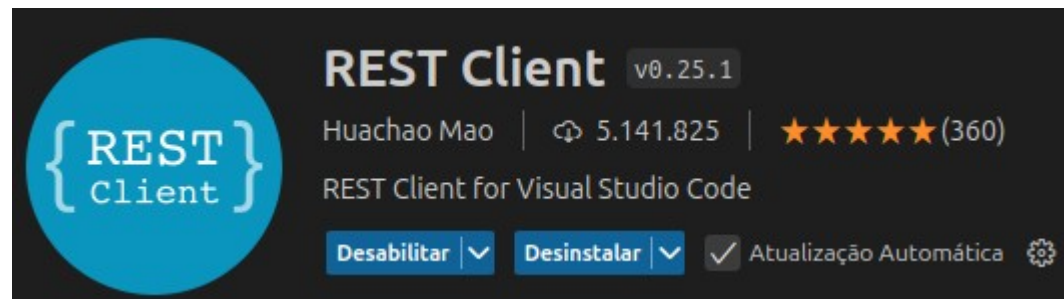
```
app.post("/api/filmes", (req, res) => {
  const filme = req.body;

  for (let x = 0; x < filmes.length; x++) {
    if (filmes[x].id == filme.id) {
      return res.status(403).json({
        mensagem: "Já existe um filme cadastrado com esse ID"
      });
    }
  }
  filmes.push(filme);
  return res.status(201).json({mensagem: "Filme adicionado com sucesso"});
})
```



- Como encaminhar uma requisição POST?
 - » Ferramentas como Insomnia, Postman



- Como encaminhar uma requisição POST?
 - » Através do VSCode: Extensão REST Client



- /api/filmes/<id>: Atualiza um objeto antigo com dados do novo objeto



```
app.put("/api/filmes/:id", (req, res) => {  
  const id = req.params.id; ← Parâmetro da requisição  
  const filme = req.body; ← Corpo da requisição  
  
  for (let x = 0; x < filmes.length; x++) {  
    if (filmes[x].id == id) {  
      filmes[x].ano = filme.ano;  
      filmes[x].nome = filme.nome;  
      filmes[x].valor_ingresso = filme.valor_ingresso;  
  
      return res.status(204).json({}); ← NO CONTENT  
    }  
  }  
  
  return res.status(404).json({}); ← NOT FOUND  
});
```

- Usando o **paradigma funcional**, pode-se simplificar a escrita da função anterior:



```
app.delete("/api/filmes/:id", (req, res) => {  
  const { id } = req.params;  
  
  const novoFilmes = filmes.filter(f => f.id !== id);  
  
  if (novoFilmes.length === filmes.length) {  
    return res.status(404).json({});  
  }  
  
  filmes = novoFilmes;  
  return res.status(200).json({ mensagem: "Filmes removidos com sucesso" });  
});
```

- Criar uma pasta *src* na raiz do projeto:
 - » `mkdir src`
- Dentro de *src*, criar os diretórios *controllers* e *routes*:
 - » `mkdir src/controllers`
 - » `mkdir src/routes`
- Dentro de *src/controllers*, crie um arquivo chamado *filmesControllers.js*:
 - » `touch src/controllers/filmesControllers.js`
- Dentro de *src/routes*, crie um arquivo chamado *filmesRoutes.js*:
 - » `touch src/routes/filmesRoutes.js`

- Sistema de gerenciamento de banco de dados (SGBD);
- Relacional;
- SQL (*Structure Query Language*);
- Leve e portátil;
 - » Dados salvos em um único arquivo.
- Zero configuração:
 - » Dispensa configurações de administração:
 - Instalação, configuração de usuários e permissões, etc.
- Multiplataforma.




- Criar um diretório na raiz do projeto chamado *database*;
 - » *mkdir database*
- Criar um arquivo *db.js* e um arquivo *seed.js* dentro da pasta *database*;
 - » *touch database/db.js* → Descrição das tabelas do BD;
 - » *touch database/seed.js* → Entradas iniciais do BD.
- Instalar os pacotes *sqlite3*:
 - » *npm i sqlite3 sqlite*
- Instalar plugin VSCode:
 - » SQLite

- Definindo conexão



```
1 import { open } from "sqlite";
2 import sqlite3 from "sqlite3";
3
4 const dbPromise = open({
5   filename: "./database/filmes.db",
6   driver: sqlite3.Database,
7 });
```

- Realizando conexão



```
1 const db = await dbPromise;
```


- Executando uma query:

```
1 const filmes = await db.all('SELECT * FROM filmes');
```

- Inserindo elemento no BD:

```
1 await db.run(  
2     "INSERT INTO filmes (id, titulo, img, descricao, genero, diretor, valor, ano) VALUES (?, ?, ?, ?, ?, ?, ?, ?)",  
3     [  
4         id,  
5         filme.titulo,  
6         filme.img,  
7         filme.descricao,  
8         filme.genero,  
9         filme.direcao,  
10        filme.valor,  
11        filme.ano,  
12    ]  
13 );
```

- **db.run(sql, [params], [callback]):** Executa uma instrução SQL (INSERT, UPDATE, DELETE)
 - » params: parâmetros utilizados na inserção (opcional). Array;
 - » Callback: função executada após a execução do comando (opcional).

```
1  await db.run(  
2    "INSERT INTO filmes (id, titulo, img, descricao, genero, diretor, valor, ano) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?)",  
3    [  
4      id,  
5      filme.titulo,  
6      filme.img,  
7      filme.descricao,  
8      filme.genero,  
9      filme.direcao,  
10     filme.valor,  
11     filme.ano,  
12   ]  
13  );
```

- **db.get(sql, [params], [callback]):** Obtém apenas a primeira linha do resultado de uma consulta;
 - » params: parâmetros utilizados na inserção (opcional). Array;
 - » Callback: função executada após a execução do comando (opcional).



```
1 const filmes = await db.all('SELECT * FROM filmes');
```

- **db.all(sql, [params], [callback]):** Obtém todas as linhas da execução de uma consulta.
 - » params: parâmetros utilizados na inserção (opcional). Array;
 - » Callback: função executada após a execução do comando (opcional).



```
1 const filmes = await db.all('SELECT * FROM filmes');
```

```
CREATE TABLE IF NOT EXISTS filmes (  
    id INTEGER PRIMARY KEY,  
    nome TEXT,  
    ano INTEGER,  
    valor_ingresso REAL  
)
```

id	nome	ano	valor_ingress so
Inteiro (chave primária)	Texto	Inteiro	Float (real)

- Arquivo *database/db.js*



```
import { open } from "sqlite";  
import sqlite3 from "sqlite3";  
  
export const dbPromise = open({  
  filename: "./database/filmes.db",  
  driver: sqlite3.Database,  
});
```

- Arquivo *database/seed.js* (parte 1)



```
import { dbPromise } from "../db.js";

const db = await dbPromise;

async function criarTabelaFilmes() {
  await db.exec(`CREATE TABLE IF NOT EXISTS filmes (
    id INTEGER PRIMARY KEY,
    nome TEXT,
    ano INTEGER,
    valor_ingresso REAL
  )`);
}
```

- Arquivo *database/seed.js* (parte 2)

```
async function adicionarFilmes() {
  const filmes = [
    {
      id: 0,
      nome: "Tropa de Elite 2: O Inimigo Agora é Outro",
      valor_ingresso: 25.0,
      ano: 2010,
    },
    {
      id: 1,
      nome: "Dona Flor e Seus Dois Maridos",
      valor_ingresso: 22.0,
      ano: 1976,
    },
    {
      id: 2,
      nome: "Minha Mãe é uma Peça 3",
      valor_ingresso: 20.0,
      ano: 2019,
    },
    {
      id: 3,
      nome: "Se Eu Fosse Você 2",
      valor_ingresso: 18.0,
      ano: 2009,
    },
    {
      id: 4,
      nome: "De Pernas pro Ar 2",
      valor_ingresso: 17.0,
      ano: 2012,
    },
  ];

  for (const f of filmes) {
    try {
      await db.run(
        "INSERT INTO filmes (id, nome, ano, valor_ingresso) VALUES (?, ?, ?, ?)",
        [f.id, f.nome, f.ano, f.valor_ingresso]
      );
    } catch (e) {
      console.log(e);
    }
  }
}
```


- Arquivo *database/seed.js* (parte 3)




```
await criarTabelaFilmes( );  
await adicionarFilmes( );
```


- Definir no *package.json* um script de inicialização das configurações do BD:

```
{
  "name": "eic-pin-2024",
  "version": "1.0.0",
  "description": "Uma API de filmes apresentada no EIC2024",
  "main": "index.js",
  "type": "module",
  "scripts": {
    "start": "node index.js",
    "dev": "nodemon index.js",
    "db:seed": "node database/seed.js"
  },
  "keywords": [
    "EIC2024",
    "express",
    "api-filmes"
  ],
  "author": "Luís Eduardo",
  "license": "MIT",
  "devDependencies": {
    "nodemon": "^3.1.7"
  },
  "dependencies": {
    "express": "^4.21.1",
    "sqlite": "^5.1.1",
    "sqlite3": "^5.1.7"
  }
}
```

*script de preparação
do BD*



- Dentro de src, criar o diretório *models*:
 - » `mkdir src/models`
- Criar o arquivo *filmesModel.js* (concentrar toda lógica de comunicação com o BD):
 - » `touch src/models/filmesModel.js`



```
import { dbPromise } from "../../database/db.js";

const db = await dbPromise;

export async function obterTodosOsFilmesModel() {
  return await db.all("SELECT * FROM filmes");
}

export async function obterFilmePorIdModel(id) {
  return await db.get("SELECT * FROM filmes WHERE id=?", [id]);
}

export async function adicionarFilmeModel(filme) {
  await db.run(
    "INSERT INTO filmes (id, nome, ano, valor_ingresso) VALUES (?, ?, ?, ?)",
    [filme.id, filme.nome, filme.ano, filme.valor_ingresso]
  );
}

export async function atualizarFilmeModel(filme) {
  await db.run(
    "UPDATE filmes SET nome = ?, ano = ?, valor_ingresso = ? WHERE id = ? ",
    [filme.nome, filme.ano, filme.valor_ingresso, filme.id]
  );
}

export async function removerFilmeModel(id) {
  await db.run("DELETE FROM filmes WHERE id = ?", [id]);
}
```

- Refatorar os *controllers* para utilizar o *model* (parte 1)

```
import {
  adicionarFilmeModel,
  atualizarFilmeModel,
  obterFilmePorIdModel,
  obterTodosOsFilmesModel,
  removerFilmeModel,
} from "../models/filmesModel.js";

export async function obterFilmes(_, res) {
  const filmes = await obterTodosOsFilmesModel();
  return res.status(200).json(filmes);
}

export async function obterFilmeEspecifico(req, res) {
  const { id } = req.params;

  const resultado = await obterFilmePorIdModel(id);
  return res.status(resultado ? 200 : 404).json(resultado);
}
```

- Refatorar os *controllers* para utilizar o *model* (parte 2)

```
export async function atualizarFilme(req, res) {
  const id = req.params.id;
  const filme = req.body;

  const filmeBD = await obterFilmePorIdModel(id);
  if (filmeBD) {
    await atualizarFilmeModel(filme);
    return res.status(204).json({});
  }
  return res.status(404).json({});
}

export async function adicionarFilme(req, res) {
  const filme = req.body;

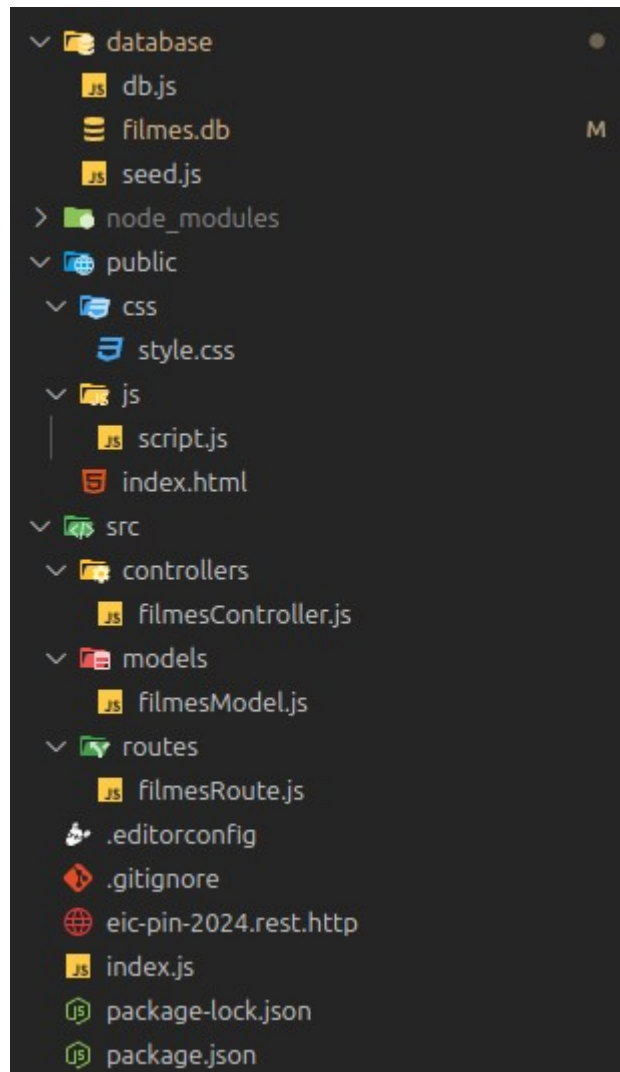
  const filmeBD = await obterFilmePorIdModel(filme.id);

  if (filmeBD) {
    return res.status(403).json({ mensagem: "Já existe um filme cadastrado com esse ID" });
  }
  await adicionarFilmeModel(filme);
  return res.status(201).json({ mensagem: "Filme adicionado com sucesso" });
}
```

- Refatorar os *controllers* para utilizar o *model* (parte 3)



```
export async function removerFilme(req, res) {  
  const { id } = req.params;  
  
  const filmeBD = await obterFilmePorIdModel(id);  
  if (filmeBD) {  
    await removerFilmeModel(id);  
    return res.status(200).json({ mensagem: "Filmes removidos com sucesso" });  
  }  
  return res.status(404).json({});  
}
```

- *Middlewares*;
- Validação e manipulação de requisição;
 - » joi, validator.js.
- Conexão com BD;
 - » Bancos não relacionais (MongoDB, Redis, etc).
- Autenticação e autorização;
 - » JWT, Sessions, cookies, bcrypt.
- Manipulação e tratativa de erros;
- Segurança
 - » Rate-limit, XSS, CSFR.
- Documentação
 - » swagger, postman.
- Testes
 - » jest, mocha, chai.
- Deploy
 - » AWS, DigitalOcean.

Dúvidas?