

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROGRAMAÇÃO WEB II

Prof. Marcos Caldas

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROJETO BACKEND

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

PROF. MARCOS CALDAS
DESENVOLVIMENTO
DE SISTEMAS

Criar uma pasta para o projeto depois abrir o **cmd** e instalar **npm init -y**

O comando **npm init -y** apresenta perguntas sobre o seu projeto.

Com a opção **-y** os valores padrões são atribuídos.

Foi criado um arquivo chamado **package.json**

mudar index para app

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npm init -y
Wrote to C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND\package.json:
```

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

Express

O Express é um framework para Node.js que fornece um conjunto de recursos para o desenvolvimento de aplicações web ou móveis.

A ideia central de um framework é dispor de recursos que visam otimizar o processo de construção de aplicações. A documentação sobre o Express pode ser obtida no endereço: <https://expressjs.com/>.




Instalar o express

Express.js é um framework para **Node.js** que fornece recursos mínimos para construção de servidores web.

Ele cria a pasta **node-modules** e **package-lock.json**

npm i express

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npm i express  
  
added 58 packages, and audited 59 packages in 9s  
  
8 packages are looking for funding  
  run `npm fund` for details  
  
found 0 vulnerabilities  
  
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

Nome	Data de modificação	Tipo	Tamanho
 node_modules	20/07/2023 12:23	Pasta de arquivos	
 package.json	20/07/2023 12:23	JSON File	1 KB
 package-lock.json	20/07/2023 12:23	JSON File	40 KB

Com o VS Code aberto vamos criar um arquivo **app.js** na pasta raiz e inserir os códigos:

```
JS app.js × {} package.json
JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4
5  app.get('/', (req, res)=>{
6    res.send("Olá...Bem vindo!");
7  });
8
9  app.listen(port, ()=>{
10    console.log(`Servidor rodando em http://localhost:${port}`);
11  });
```

Para rodar o servidor abrir o CMD na pasta do projeto e digitar:

node app

Abrir o navegador e acessar o endereço: <http://localhost:3001>

O node app não atualiza o servidor, tem que fechar e abrir de novo.


```
C:\Users\Aluno\Desktop\MARCOS\BACKEND>node app  
Servidor rodando em http://localhost:3001
```

← → ↻ ⓘ localhost:3001

Olá...Bem vindo!

No arquivo **app.js**

Adicionar o comando antes do **app.listen** para adicionar uma nova rota.

```
JS app.js x
JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4
5  app.get('/', (req, res) => {
6    res.send("Olá... Bem vindo!");
7  });
8
9
10 app.get('/introducao', (req, res) => {
11   res.send('<h2>Introdução ao Express </h2>');
12 });
13
14
15 app.listen(port, () => {
16   console.log(`Servidor rodando em http://localhost:${port}`);
17 });
```

colocar :
antes
do \$

Nodemon

Quando o **Node.js** compila o código ele não escuta as alterações, dessa forma tem que finalizar o processo e abrir de novo.

Podemos adicionar o pacote **nodemon** para que tenha essa atualização automática.

Voltar no terminal e parar a execução e executar o comando

npm i --save-dev nodemon

caso não funcione, instalar o
e depois

npm i --save-dev nodemon

npm i nodemon -g

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npm i --save-dev nodemon
added 33 packages, and audited 92 packages in 7s

11 packages are looking for funding
  run `npm fund` for details


found 0 vulnerabilities

C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

Após esse comando verificar o **package.json**

Aproveitar para mudar o nome do
arquivo principal

"main": "app.js"



```
JS app.js x {} package.json x
() package.json > ...
1 {
2   "name": "projeto-backend",
3   "version": "1.0.0",
4   "description": "",
5   "main": "app.js",
6   "scripts": {
7     "test": "echo \"Error: no test specified\" && exit 1"
8   },
9   "keywords": [],
10  "author": "",
11  "license": "ISC",
12  "dependencies": {
13    "express": "^4.18.2"
14  },
15  "devDependencies": {
16    "nodemon": "^3.0.1"
17  }
18 }
19
```

Para testar a nova rota **introducao** e fazer com que o Node.js fique na “escuta” de alterações no código, rode o nodemon app

Rodar no terminal **nodemon app**

Vai aparecer essa tela

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>nodemon app
[nodemon] 3.0.1
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node app.js`
Servidor rodando em http://localhost3001
```

Testar no navegador a rota
localhost:3001/introducao



No programa app.js é possível observar que cada arrow function associada a uma rota recebe dois parâmetros: **req** (request) e **res** (response).

Eles podem receber qualquer nome, mas **req** e **res** são os geralmente utilizados.

O parâmetro **req** contém as informações que são passadas para a rota – por quem a acionou.

Já **res** refere-se à resposta da rota – como as mensagens exibidas nas duas rotas anteriormente criadas.

Antes do **listen()** acrescentar o seguinte código:

app.js

```
JS app.js • {} package.json
JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4
5  app.get('/', (req, res)=>{
6    res.send("Olá...Bem vindo!");
7  });
8
9
10 app.get('/introducao', (req, res)=>{
11   res.send('<h2>Introdução ao Express </h2>');
12 });
13
14 // para reconhecer os dados recebidos como sendo um objeto no formato JSON
15 app.use(express.json());
16 app.post('/filmes', (req, res)=>{
17   //const titulo = req.body.titulo;
18   //const genero = req.body.genero;
19   const {titulo, genero} = req.body;
20   res.send(`Filme:${titulo} - Gênero: ${genero}, recebido...`);
21 });
22
23 app.listen(port, ()=>{
24   console.log(`Servidor rodando em http://localhost:${port}`);
25 });
```


A linha **app.use(express.json());** adiciona um middleware para reconhecer que o formato 'combinado' para os dados a serem enviados pelo cliente e recebidos pelo programa no servidor é o JSON. Como se fosse um parse.

Na rota utilizamos o método **POST** ou **VERBO POST**.

Rotas com **GET** podem ser acessadas diretamente no navegador.

As demais podem ser testadas a partir de nossos programas, ou através de softwares como o INSOMNIA ou POSTMAN ou pelo site **reqbin.com** porém tem que instalar uma extensão no Chrome.

Middlewares

Um middleware é uma espécie de mediador entre duas partes, algo que fica no meio (middle).

Vamos construir um exemplo, para ilustrar uma possível funcionalidade para um middleware. Digamos que os administradores de um sistema gostariam de analisar quantas vezes ou em quais horários determinadas rotas estão sendo acionadas.

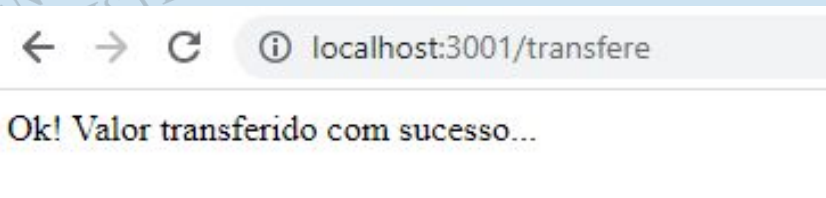
Algo como um controle de logs, que poderia registrar ações como transferências financeiras, exclusões ou alterações de dados, por exemplo.

Nesse caso, antes de acionar essas rotas, podemos fazer o sistema passar por um middleware que irá registrar essas ações. Acrescente o seguinte trecho ao **app.js**.

```

JS app.js x {} package.json
JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4
5  app.get('/', (req, res) => {
6    res.send("Olá... Bem vindo!");
7  });
8
9
10 app.get('/introducao', (req, res) => {
11   res.send('<h2>Introdução ao Express </h2>');
12 });
13
14 // para reconhecer os dados recebidos como sendo um objeto no formato JSON
15 app.use(express.json());
16 app.post('/filmes', (req, res) => {
17   //const titulo = req.body.titulo;
18   //const genero = req.body.genero;
19   const {titulo, genero} = req.body;
20   res.send(`Filme: ${titulo} - Gênero: ${genero}, recebido...`);
21 });
22
23 const log = (req, res, next) => {
24   console.log(`..... Acessado em ${new Date()}`);
25   next();
26 }
27 app.get('/transfere', log, (req, res) => {
28   res.send("Ok! Valor transferido com sucesso...");
29 });
30
31 app.listen(port, () => {
32   console.log(`Servidor rodando em http://localhost:${port}`);
33 });

```



```
Servidor rodando em http://localhost3001
[nodemon] restarting due to changes...
[nodemon] starting `node app.js`
Servidor rodando em http://localhost3001
..... Acessado em Thu Jul 20 2023 13:33:12 GMT-0300 (Horário Padrão de Brasília)
```

Instalar o pacote Knex para fazer a conexão com o banco de dados.

Instalar os pacotes:

npm i knex

npm i sqlite3

```
^C
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npm i knex
added 20 packages, and audited 112 packages in 5s
14 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npm i sqlite3
npm WARN deprecated @npmcli/move-file@1.1.2: This functionality has been moved to @npmcli/fs
added 101 packages, and audited 213 packages in 17s
18 packages are looking for funding
  run `npm fund` for details
found 0 vulnerabilities

C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

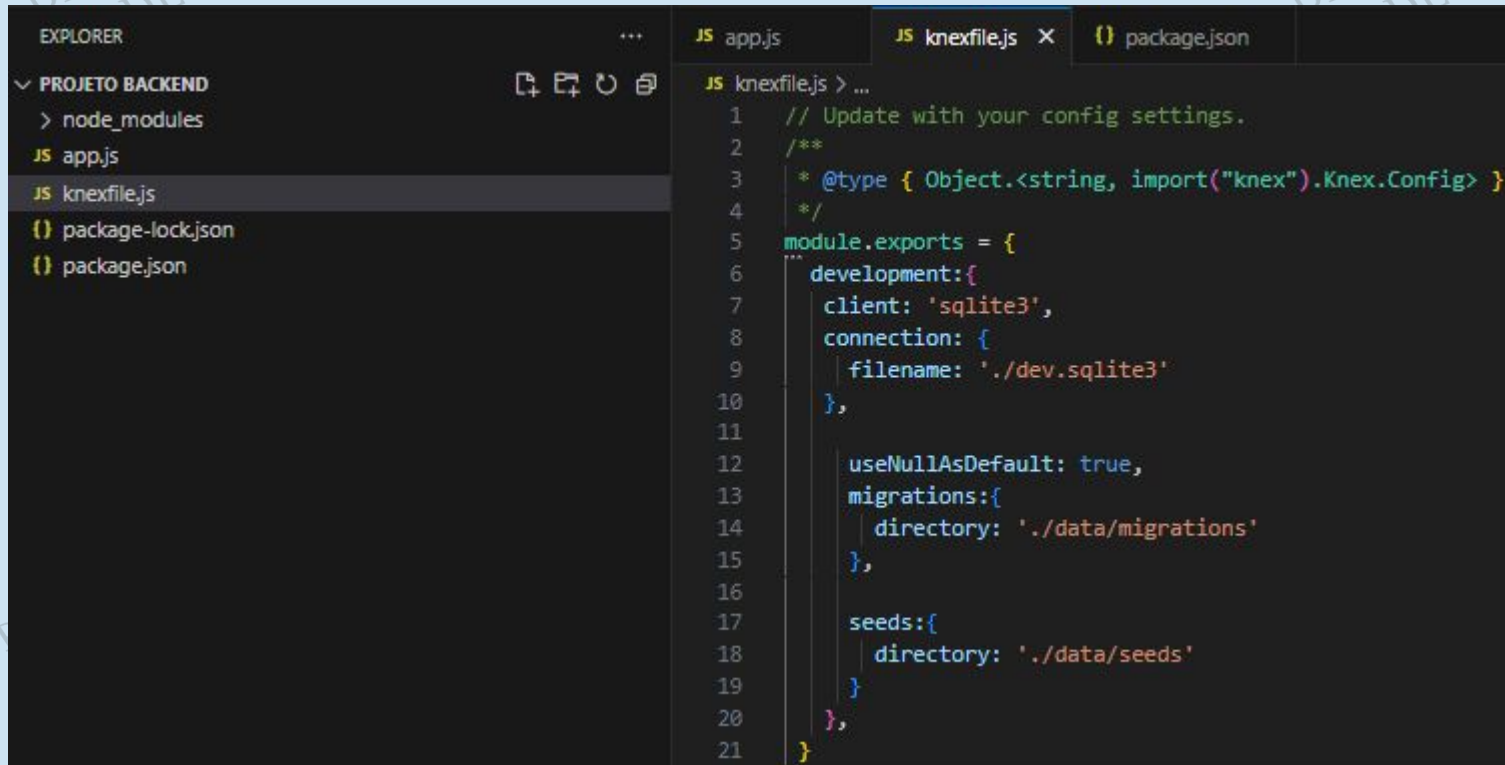
Para criar a tabela com Migrations instalar o pacote:

npx knex init

Vai ter como resposta Created ./knexfile.js para configurar a conexao

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npx knex init  
Created ./knexfile.js  
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

Editar o arquivo **knexfile.js** para o banco **sql3**



The image shows a screenshot of the Visual Studio Code editor. On the left, the Explorer sidebar displays the project structure under 'PROJETO BACKEND', including 'node_modules', 'app.js', 'knexfile.js' (selected), 'package-lock.json', and 'package.json'. The main editor area shows the 'knexfile.js' file with the following content:

```
JS knexfile.js > ...
1  // Update with your config settings.
2  /**
3   * @type { Object.<string, import("knex").Knex.Config> }
4   */
5  module.exports = {
6    development: {
7      client: 'sqlite3',
8      connection: {
9        filename: './dev.sqlite3'
10      },
11
12      useNullAsDefault: true,
13      migrations: {
14        directory: './data/migrations'
15      },
16
17      seeds: {
18        directory: './data/seeds'
19      }
20    },
21  }
```


PRESTAR A ATENÇÃO SE USAR O MYSQL

Editar o arquivo
knexfile.js

Para o banco mysql

**Instalar o pacote
npm i mysql2**

**Criar um banco no mysql
primeiro**

```
JS knexfile.js X
JS knexfile.js > ...
1 // Update with your config settings.
2 /**
3  * @type { Object.<string, import("knex").Knex.Config> }
4  */
5 module.exports = {
6   development: {
7     client: 'mysql2',
8     connection: {
9       host : '127.0.0.1',
10      port : 3306,
11      user : 'root',
12      password : 'suasenha',
13      database : 'teste'
14    },
15
16     useNullAsDefault: true,
17     migrations: {
18       directory: './data/migrations'
19     },
20
21     seeds: {
22       directory: './data/seeds'
23     }
24   },
25 }
```


Para criar o arquivo que irá conter a definição da estrutura da tabela de livros, rode o comando:

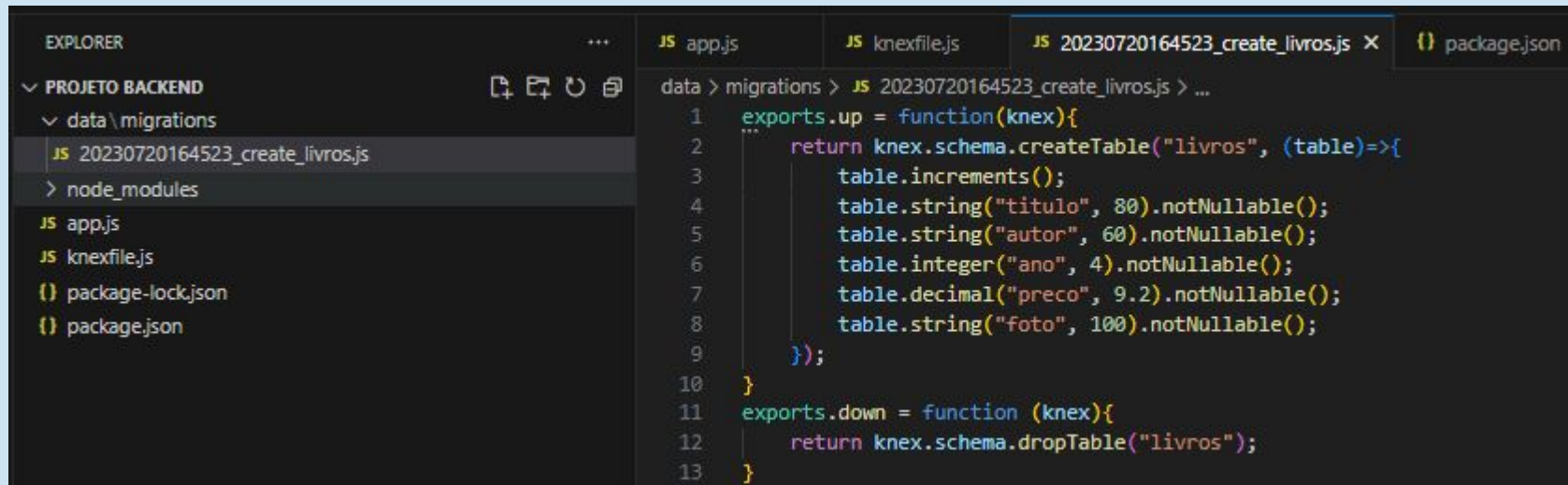
npx knex migrate:make create_livros

como resposta teremos:

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npx knex migrate:make create_livros
Using environment: development
Using environment: development
Using environment: development
Created Migration: C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND\data\migrations\20230720164523_create_livros.js
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

Editar o arquivo criado pelo migration

Pasta data\migration



The screenshot shows the Visual Studio Code interface. On the left, the Explorer sidebar displays the project structure: 'PROJETO BACKEND' with a subfolder 'data' containing a 'migrations' folder. The file '20230720164523_create_livros.js' is selected. The main editor area shows the content of this file, which defines an 'up' migration to create a 'livros' table with columns: 'titulo' (string, 80, not nullable), 'autor' (string, 60, not nullable), 'ano' (integer, 4, not nullable), 'preco' (decimal, 9.2, not nullable), and 'foto' (string, 100, not nullable). It also includes a 'down' migration to drop the table.

```
data > migrations > JS 20230720164523_create_livros.js > ...
1  exports.up = function(knex){
2      return knex.schema.createTable("livros", (table)=>{
3          table.increments();
4          table.string("titulo", 80).notNullable();
5          table.string("autor", 60).notNullable();
6          table.integer("ano", 4).notNullable();
7          table.decimal("preco", 9.2).notNullable();
8          table.string("foto", 100).notNullable();
9      });
10 }
11 exports.down = function (knex){
12     return knex.schema.dropTable("livros");
13 }
```

Retornar ao cmd e digitar **npx knex migrate:latest**

Seeds: “Semeando ” dados iniciais

digitar o comando

`npx knex seed:make 001_add_livros`

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>npx knex seed:make 001_add_livros
Using environment: development
Using environment: development
Using environment: development
Created seed file: C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND\data\seeds\001_add_livros.js
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

Editar esse arquivo dentro da pasta **`./data/seeds`**

▼ PROJETO BACKEND

▼ data

> migrations

▼ seeds

JS 001_add_livros.js

> node_modules

JS app.js

dev.sqlite3

JS knexfile.js

() package-lock.json

() package.json

data > seeds > JS 001_add_livros.js > ...

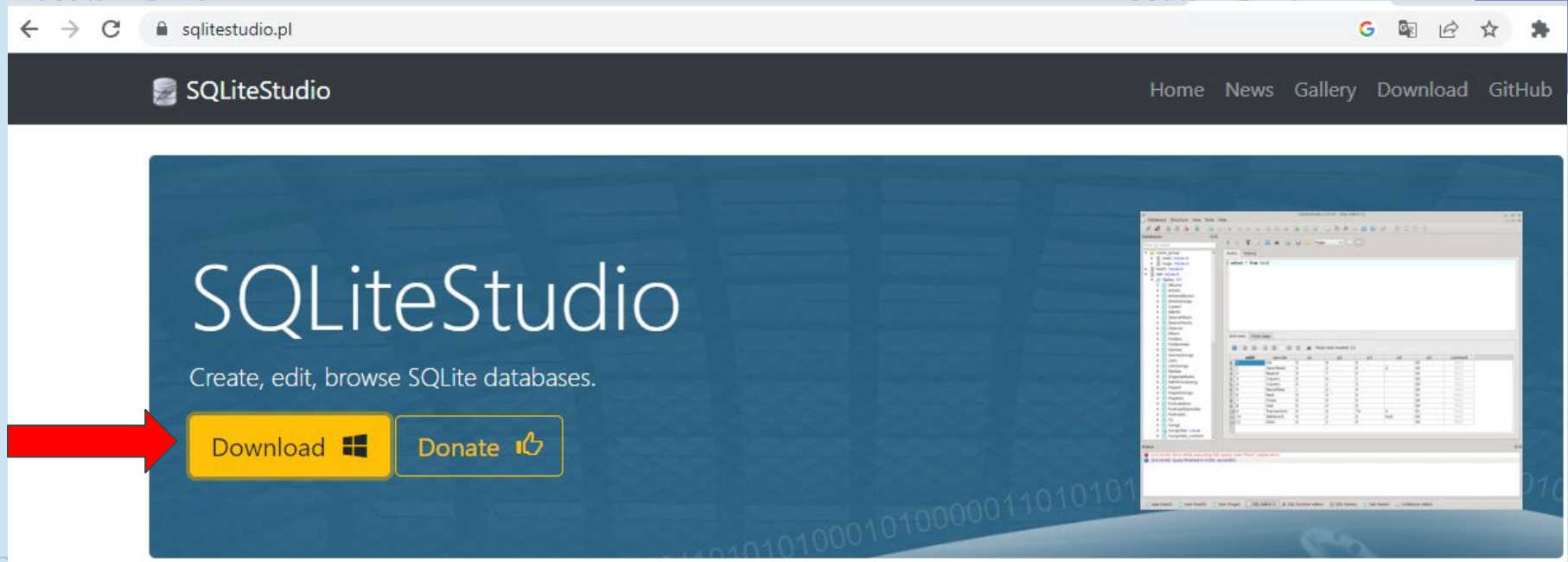
```
1  exports.seed = function (knex) {
2    return knex("livros").del()
3    .then(function () {
4      return knex("livros").insert([
5        {
6          titulo: "Web Design Responsivo", autor: "Maurício Samy Silva", ano: 2014,
7          preco: 73.0, foto: "https://s3.novatec.com.br/capas/9788575223925.jpg",
8        },
9        {
10         titulo: "Proteção Moderna de Dados", autor: "W. Curtis Preston", ano: 2021,
11         preco: 97.0, foto: "https://s3.novatec.com.br/capas/9786586057843.jpg",
12       },
13       {
14         titulo: "SQL em 10 Minutos por Dia", autor: "Ben Forta", ano: 2021,
15         preco: 79.0, foto: "https://s3.novatec.com.br/capas/9786586057447.jpg",
16       },
17       {
18         titulo: "CSS Grid Layout", autor: "Maurício Samy Silva", ano: 2017,
19         preco: 45.0, foto: "https://s3.novatec.com.br/capas/9788575226322.jpg",
20       },
21       {
22         titulo: "Python para análise de dados", autor: "Wes McKinney", ano: 2018,
23         preco: 132.0, foto: "https://s3.novatec.com.br/capas/9788575226476.jpg",
24       },
25     ])
26   })
27 }
```

Para inserir os dados digite o comando:

npx knex seed:run

```
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND> npx knex seed:run
Using environment: development
Ran 1 seed files
C:\Users\G-Fire\Documents\PW2 2023\PROJETO BACKEND>
```

<https://sqlitestudio.pl/>



SQLiteStudio (3.4.4) - [livros (dev)]

Database Structure View Tools Help

Databases filter by name

- dev (SQLite 3)
 - Tables (3)
 - knex_...
 - knex_...
 - livros
 - Views

Structure Data Constraints Indexes Triggers DDL

dev Table name: livros ☐ WITHOUT ROWID ☐ STRICT

	Name	Data type	Primary Key	Foreign Key	Unique	Check	Not NULL	Collate	Generated	Default value
1	id	integer								NULL
2	titulo	varchar (80)								NULL
3	autor	varchar (60)								NULL
4	ano	integer								NULL
5	preco	float								NULL

Type Name Details

Status

[16:06:59] Query finished in 0.001 second(s).

livros (dev) SQL editor 1

Habilitar editor de SQL ou Alt + E

The screenshot shows the SQLiteStudio 3.4.4 interface. A red arrow points to the 'SQL editor' tab in the top menu bar. The main window displays a SQL query: `SELECT * FROM LIVROS`. Below the query editor, the 'Form view' tab is active, showing a table with 5 rows and 7 columns: id, titulo, autor, ano, preco, and foto. The status bar at the bottom indicates that the query finished in 0.001 second(s).

id	titulo	autor	ano	preco	foto
1	Web Design Responsivo	Maurício Samy Silva	2014	73	https://s3.novatec.com.br/capas/9788575223925.jpg
2	Proteção Moderna de Dados	W. Curtis Preston	2021	97	https://s3.novatec.com.br/capas/9786586057843.jpg
3	SQL em 10 Minutos por Dia	Ben Forta	2021	79	https://s3.novatec.com.br/capas/9786586057447.jpg
4	CSS Grid Layout	Maurício Samy Silva	2017	45	https://s3.novatec.com.br/capas/9788575226322.jpg
5	Python para análise de dados	Wes McKinney	2018	132	https://s3.novatec.com.br/capas/9788575226476.jpg

Durante o desenvolvimento do nosso projeto já trabalhamos com os conceitos:

- Database-first;
- Model-first, Code-first e Migrations;
- Mapeamento objeto-relacional (ORM).

Para fixar nossos aprendizados faremos uma pesquisa para expandir mais os conceitos apresentados.

Atividade em Grupo.

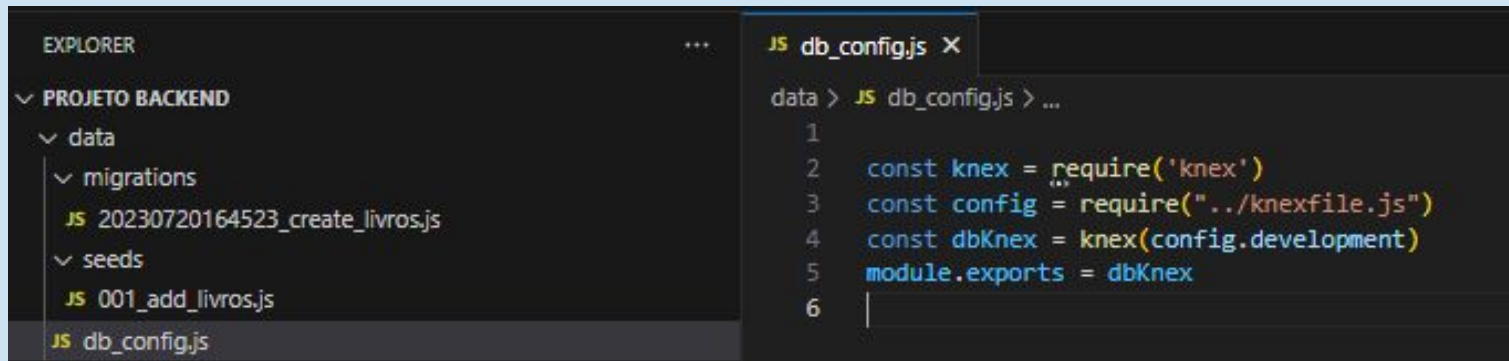
Estrutura da pesquisa:

- INTRODUÇÃO
- DESENVOLVIMENTO
- CONCLUSÃO COM OPINIÃO PESSOAL DA EQUIPE;
- FONTE DE PESQUISA.

O arquivo de configuração criado anteriormente (`knexfile.js`) serve para definir o driver do banco de dados e parâmetros da conexão utilizados no processo de criação das migrations e seeds da aplicação.

Agora, precisamos criar um arquivo que vai conter esses detalhes de conexão com o banco de dados a serem utilizados pelo programa.

Para isso, crie o arquivo `db_config.js`, dentro da pasta `data`, e insira as seguintes linhas:

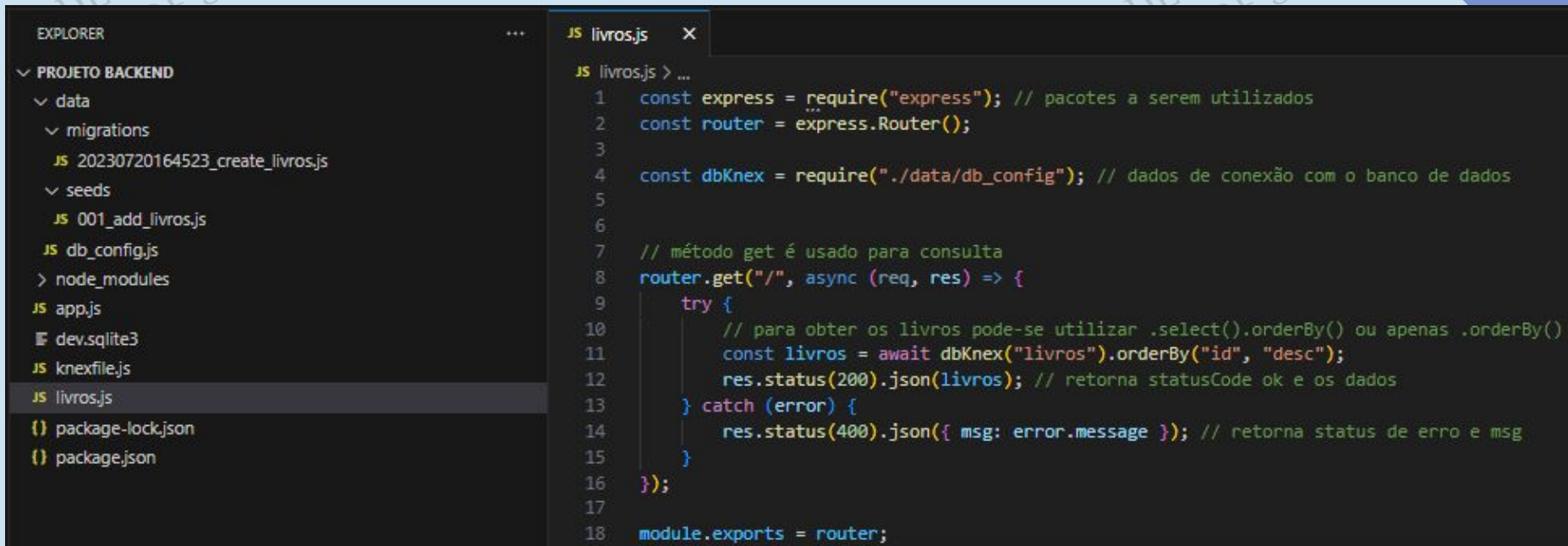


```
EXPLORER

▼ PROJETO BACKEND
  ▼ data
    ▼ migrations
      JS 20230720164523_create_livros.js
    ▼ seeds
      JS 001_add_livros.js
  JS db_config.js

JS db_config.js X
data > JS db_config.js > ...
1
2   const knex = require('knex')
3   const config = require("../knexfile.js")
4   const dbKnex = knex(config.development)
5   module.exports = dbKnex
6   |
```

Na pasta raiz do projeto criar uma arquivo **livros.js** para organizar a rotas.



```
JS livros.js
JS livros.js > ...
1  const express = require("express"); // pacotes a serem utilizados
2  const router = express.Router();
3
4  const dbKnex = require("../data/db_config"); // dados de conexão com o banco de dados
5
6
7  // método get é usado para consulta
8  router.get("/", async (req, res) => {
9      try {
10         // para obter os livros pode-se utilizar .select().orderBy() ou apenas .orderBy()
11         const livros = await dbKnex("livros").orderBy("id", "desc");
12         res.status(200).json(livros); // retorna statusCode ok e os dados
13     } catch (error) {
14         res.status(400).json({ msg: error.message }); // retorna status de erro e msg
15     }
16 });
17
18 module.exports = router;
```

Abrir o arquivo
app.js e adicionar
as linhas 33 até 36:

```

EXPLORER
  PROJETO BACKEND
    data
    migrations
      JS 20230720164523_create_livros.js
    seeds
      JS 001_add_livros.js
      JS db_config.js
    > node_modules
  JS app.js
  dev.sqlite3
  JS knexfile.js
  JS livros.js
  {} package-lock.json
  {} package.json

JS app.js > ...
1  const express = require('express');
2  const app = express();
3  const port = 3001;
4
5  app.get('/', (req, res) => {
6    res.send("Olá... Bem vindo!");
7  });
8
9
10 app.get('/introducao', (req, res) => {
11   res.send('<h2>Introdução ao Express </h2>');
12 });
13
14 // para reconhecer os dados recebidos como sendo um objeto no formato JSON
15 app.use(express.json());
16 app.post('/filmes', (req, res) => {
17   //const titulo = req.body.titulo;
18   //const genero = req.body.genero;
19   const {titulo, genero} = req.body;
20   res.send(`Filme: ${titulo} - Gênero: ${genero}, recebido...`);
21 });
22
23 const log = (req, res, next) => {
24   console.log('..... Acessado em ${new Date()}');
25   next();
26 }
27
28 app.get('/transfere', log, (req, res) => {
29   res.send("Ok! Valor transferido com sucesso...");
30 });
31
32
33 //arquivo com rotas para o cadastro de livros
34 const livros = require('./livros');
35
36 app.use('/livros', livros) //identificação da rota e da const (require) associada
37
38 app.listen(port, () => {
39   console.log(`Servidor rodando em http://localhost:${port}`);
40 });
  
```

Rodar **nodemon app**

Consultar <http://localhost:3001/livros>

Para uma melhor visualização adicionar extensões no Chrome

JSON Formatter

```
[
  {
    "id": 5,
    "titulo": "Python para análise de dados",
    "autor": "Wes McKinney",
    "ano": 2018,
    "preco": 132,
    "foto": "https://s3.novatec.com.br/capas/9788575226476.jpg"
  },
  {
    "id": 4,
    "titulo": "CSS Grid Layout",
    "autor": "Maurício Samy Silva",
    "ano": 2017,
    "preco": 45,
    "foto": "https://s3.novatec.com.br/capas/9788575226322.jpg"
  },
  {
    "id": 3,
    "titulo": "SQL em 10 Minutos por Dia",
    "autor": "Ben Forta",
    "ano": 2021,
    "preco": 79,
    "foto": "https://s3.novatec.com.br/capas/9786586057447.jpg"
  },
  {
    "id": 2,
    "titulo": "Proteção Moderna de Dados",
    "autor": "W. Curtis Preston",
    "ano": 2021,
    "preco": 97,
    "foto": "https://s3.novatec.com.br/capas/9786586057843.jpg"
  },
  {
    "id": 1,
```

Rotas para a realização do **CRUD**

Create, Read, Update, Delete

acrescentar ao arquivo livros.js as seguintes linhas:

Linhas 17
até 36

```
JS livros.js JS app.js

JS livros.js > ...
1  const express = require("express"); // pacotes a serem utilizados
2  const router = express.Router();
3
4  const dbKnex = require("../data/db_config"); // dados de conexão com o banco de dados
5
6
7  // método get é usado para consulta
8  router.get("/", async (req, res) => {
9      try {
10         // para obter os livros pode-se utilizar .select().orderBy() ou apenas .orderBy()
11         const livros = await dbKnex("livros").orderBy("id", "desc");
12         res.status(200).json(livros); // retorna statusCode ok e os dados
13     } catch (error) {
14         res.status(400).json({ msg: error.message }); // retorna status de erro e msg
15     }
16 });
17 // Método post é usado para inclusão
18 router.post("/", async (req, res) => {
19     // faz a desestruturação dos dados recebidos no corpo da requisição
20     const { titulo, autor, ano, preco, foto } = req.body;
21
22     // se algum dos campos não foi passado, irá enviar uma mensagem de erro e retornar
23     if (!titulo || !autor || !ano || !preco || !foto) {
24         res.status(400).json({ msg: "Enviar titulo, autor, ano, preco e foto do livro" });
25         return;
26     }
27
28     // caso ocorra algum erro na inclusão, o programa irá capturar (catch) o erro
29     try {
30         // insert, faz a inserção na tabela livros (e retorna o id do registro inserido)
31         const novo = await dbKnex("livros").insert({ titulo, autor, ano, preco, foto });
32         res.status(201).json({ id: novo[0] }); // statusCode indica Create
33     } catch (error) {
34         res.status(400).json({ msg: error.message }); // retorna status de erro e msg
35     }
36 });
37
38
39 module.exports = router;
```

Linhas 37
até 48

```
37 // Método put é usado para alteração. id indica o registro a ser alterado
38 router.put("/:id", async (req, res) => {
39     const id = req.params.id; // ou const { id } = req.params
40     const { preco } = req.body; // campo a ser alterado
41     try {
42         // altera o campo preco, no registro cujo id coincidir com o parâmetro passado
43         await dbKnex("livros").update({ preco }).where("id", id); // ou .where({ id })
44         res.status(200).json(); // statusCode indica Ok
45     } catch (error) {
46         res.status(400).json({ msg: error.message }); // retorna status de erro e msg
47     }
48 });
49
50
51
52 module.exports = router;
```

Linhas 48
até 58

```
49 // Método delete é usado para exclusão
50 router.delete("/:id", async (req, res) => {
51   const { id } = req.params; // id do registro a ser excluído
52   try {
53     await dbKnex("livros").del().where({ id });
54     res.status(200).json(); // statusCode indica Ok
55   } catch (error) {
56     res.status(400).json({ msg: error.message }); // retorna status de erro e msg
57   }
58 });
59
60
61
62 module.exports = router;
```

Linhas 59
até 73

```
59 ✓ //ADICIONANDO FILTROS
60
61 // ----- Filtro por título ou autor
62 ✓ router.get("/filtro/:palavra", async (req, res) => {
63     const palavra = req.params.palavra; // palavra do título ou autor a pesquisar
64     ✓ try {
65         // para filtrar registros, utiliza-se .where(), com suas variantes
66         const livros = await dbKnex("livros")
67             .where("titulo", "like", `${palavra}%`)
68             .orWhere("autor", "like", `${palavra}%`);
69         res.status(200).json(livros); // retorna statusCode ok e os dados
70     ✓ } catch (error) {
71         res.status(400).json({ msg: error.message }); // retorna status de erro e msg
72     }
73     });
74
75 module.exports = router;
```

Linhas 74
até 88

```
74 // ----- Resumo do cadastro de livros
75 router.get("/dados/resumo", async (req, res) => {
76   try {
77     // métodos que podem ser utilizados para obter dados estatísticos da tabela
78     const consulta = await dbKnex("livros")
79       .count({ num: "*" })
80       .sum({ soma: "preco" })
81       .max({ maior: "preco" })
82       .avg({ media: "preco" });
83     const { num, soma, maior, media } = consulta[0];
84     res.status(200).json({ num, soma, maior, media: Number(media.toFixed(2)) });
85   } catch (error) {
86     res.status(400).json({ msg: error.message }); // retorna status de erro e msg
87   }
88 });
89
90
91 module.exports = router;
```

Linhas 89
até 99

```
89 // ----- Soma dos preços, agrupados por ano
90 router.get("/dados/grafico", async (req, res) => {
91   try {
92     // obtém ano e soma do preço dos livros (com o nome total), agrupados por ano
93     const totalPorAno = await dbKnex("livros").select("ano")
94       .sum({ total: "preco" }).groupBy("ano");
95     res.status(200).json(totalPorAno);
96   } catch (error) {
97     res.status(400).json({ msg: error.message }); // retorna status de erro e msg
98   }
99   });
100
101 module.exports = router;
```


INSTALAR O CORS

npm i cors

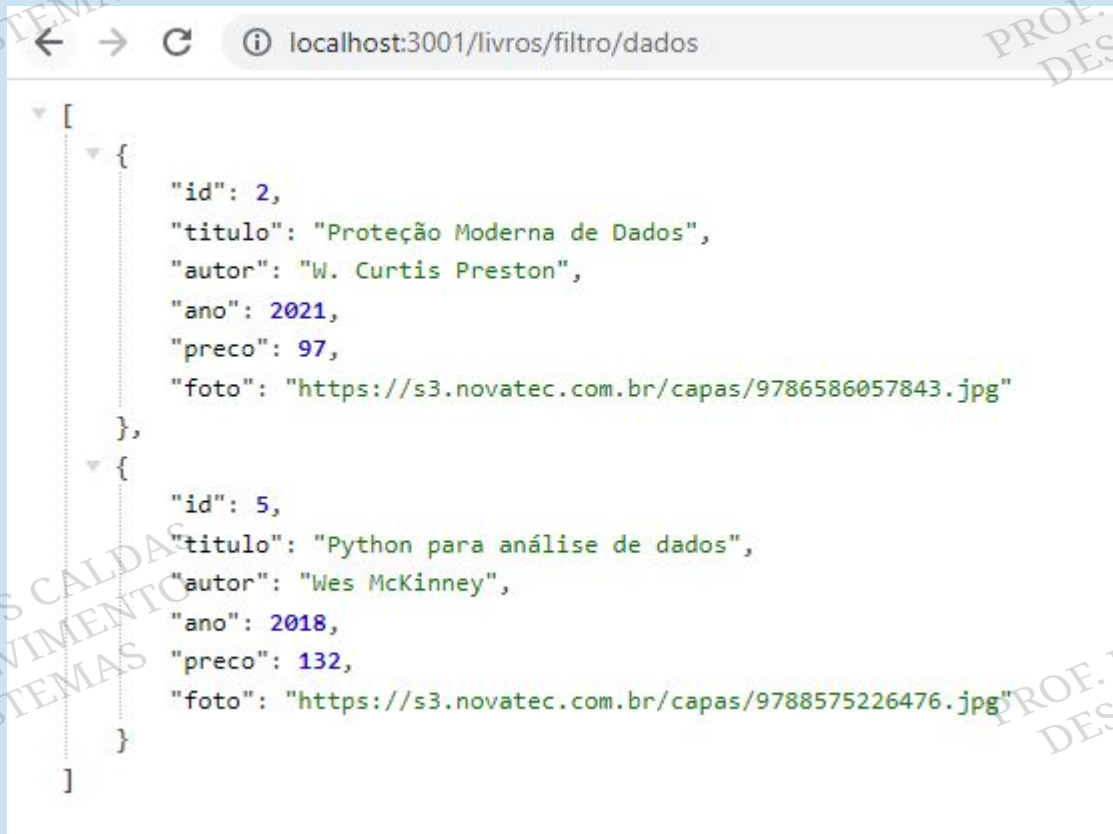
O que é uma requisição CORS?

CORS — Cross Origin Resource Sharing (*Compartilhamento de recursos com origens diferentes*) é um sistema que utiliza headers adicionais ao tradicionais para determinar as permissões de uma aplicação ao acessar dados de um servidor de origem distinta. Um exemplo de requisição cross-site é quando seu frontend está hospedado em “http://a.com” e a partir dele você precisa pedir alguns dados para sua API que está hospedada em “http://b.com”. Por segurança, os navegadores por padrão bloqueiam esse tipo de requisição iniciado por um script (fetch, axios.get ...), isso significa que sua aplicação só poderá fazer solicitações para a API se incluir os cabeçalhos CORS corretos e se o servidor estiver configurado corretamente.

Acrescente depois das duas primeiras linhas do arquivo **livros.js** o código a seguir:

```
JS livros.js > ...
1  const express = require("express"); // pacotes a serem utilizados
2  const router = express.Router();
3  const cors = require("cors");
4  router.use(cors());
5
6  const dbKnex = require("../data/db_config"); // dados de conexão com o banco de dados
```


TESTANDO NOSSA APLICAÇÃO



```
[
  {
    "id": 2,
    "titulo": "Proteção Moderna de Dados",
    "autor": "W. Curtis Preston",
    "ano": 2021,
    "preco": 97,
    "foto": "https://s3.novatec.com.br/capas/9786586057843.jpg"
  },
  {
    "id": 5,
    "titulo": "Python para análise de dados",
    "autor": "Wes McKinney",
    "ano": 2018,
    "preco": 132,
    "foto": "https://s3.novatec.com.br/capas/9788575226476.jpg"
  }
]
```



A screenshot of a web browser window displaying a JSON array. The address bar shows the URL `localhost:3001/livros/dados/grafico`. The JSON data is as follows:

```
[
  {
    "ano": 2014,
    "total": 73
  },
  {
    "ano": 2017,
    "total": 45
  },
  {
    "ano": 2018,
    "total": 132
  },
  {
    "ano": 2021,
    "total": 176
  }
]
```