

Avaliação Final Web-II – Acesso a Dados – CRUD - Mysql – ORM

Data: 28/11/2021

Aluno: Matheus de Oliveira

Rodvalho

Projeto: 30 –Filme x Elenco

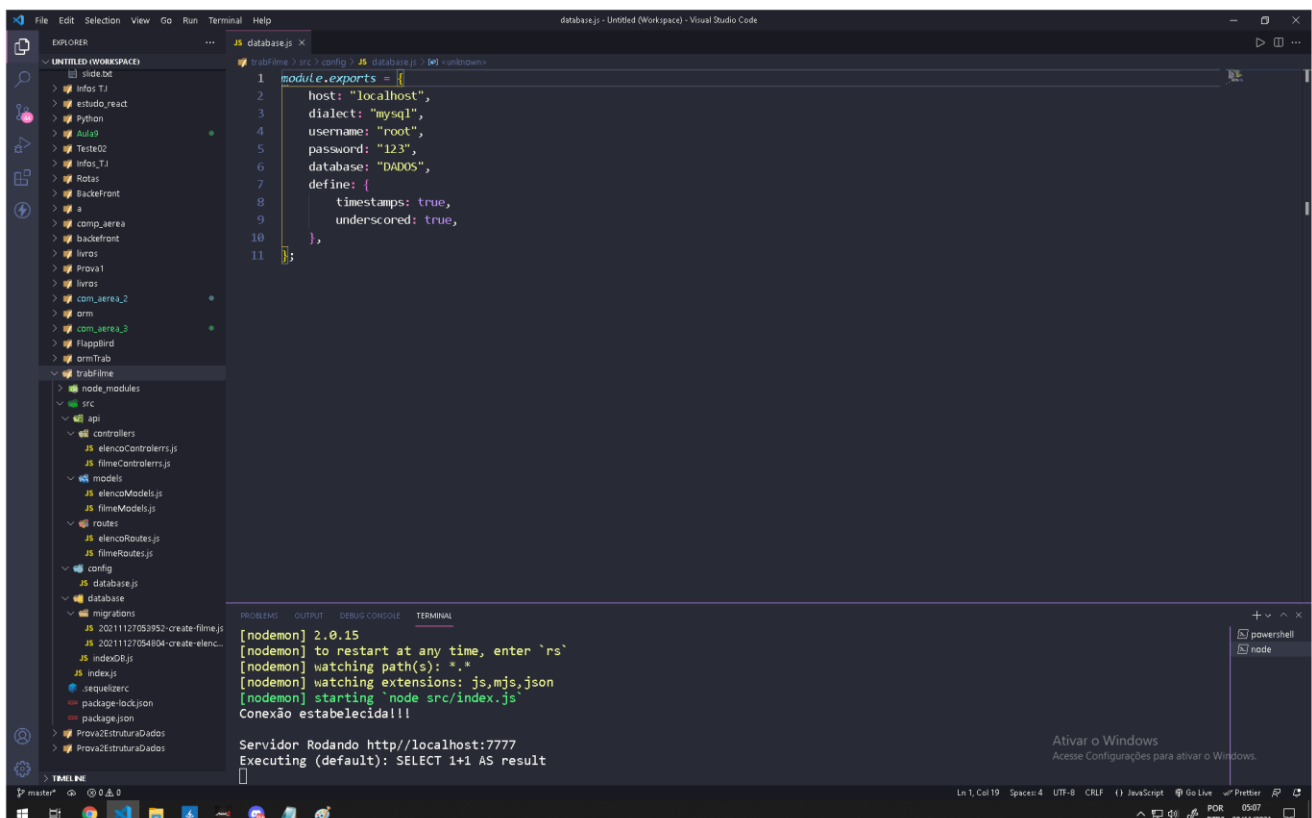
Especificação da Entidade – Tabela: FILME - FIL				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	fil_codigo		Chave primária da tabela
	varchar	fil_nomefilme	20	Nome do filme
	varchar	fil_produtores	15	Nome da produtora do filme
	varchar	fil_diretor	15	Nome do diretor do filme
	inteiro	fil_anoimagem		Ano da filmagem
	varchar	fil_pais	15	Nome do país onde o filme foi produzido

Especificação da Entidade – Tabela: ELENCO - ELE				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	ele_codigo		Chave primária da tabela
	varchar	ele_nome	20	Nome do artista
	char	ele_sexo		Sexo do artista
	varchar	ele_nacionalidade	15	Nacionalidade do artista
	varchar	ele_dt_nascimento		Data de nascimento do artista
FK	inteiro	fil_codigo		Código do Filme – chave estrangeira

- Para a comprovação do desenvolvimento do projeto o aluno apresentará alguns prints obrigatórios do projeto iniciando da estrutura das tabelas

1) Figura 1: Estrutura das tabelas – ao lado

2) Figura 2: imagem da área de desenvolvimento do projeto (**Visual Studio Code**) com a estrutura de pastas a esquerda todas abertas com o arquivo **database.js** em destaque, este arquivo fica localizado na pasta **config**. É importante que o rodapé da tela esteja visível na imagem.



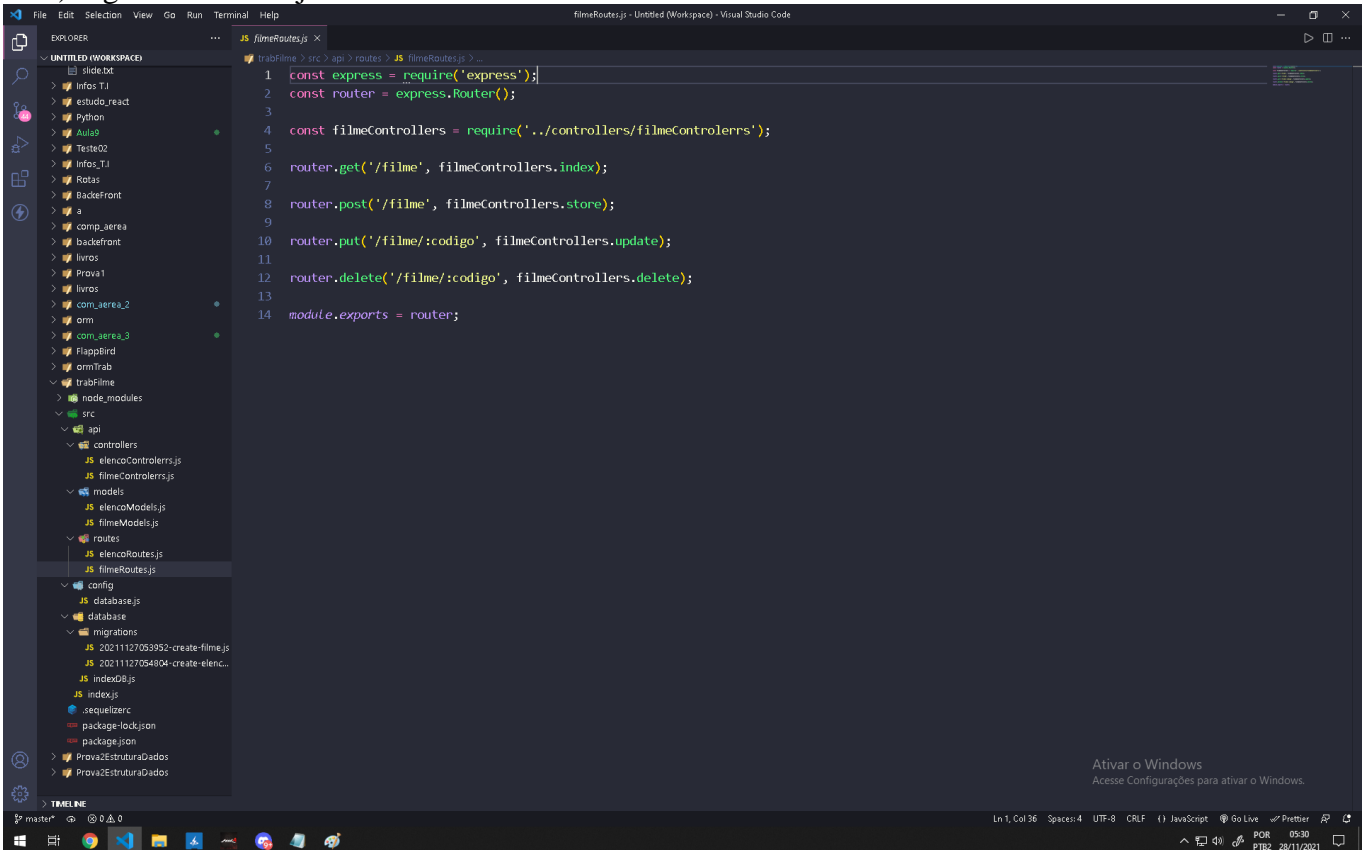
3) A partir de agora serão apresentados os prints sequenciais da primeira tabela informada no

documento da tarefa. Neste exemplo apresentaremos a sequência dos códigos na seguinte ordem:

- routes
- controllers
- models
- migrations
- workbench

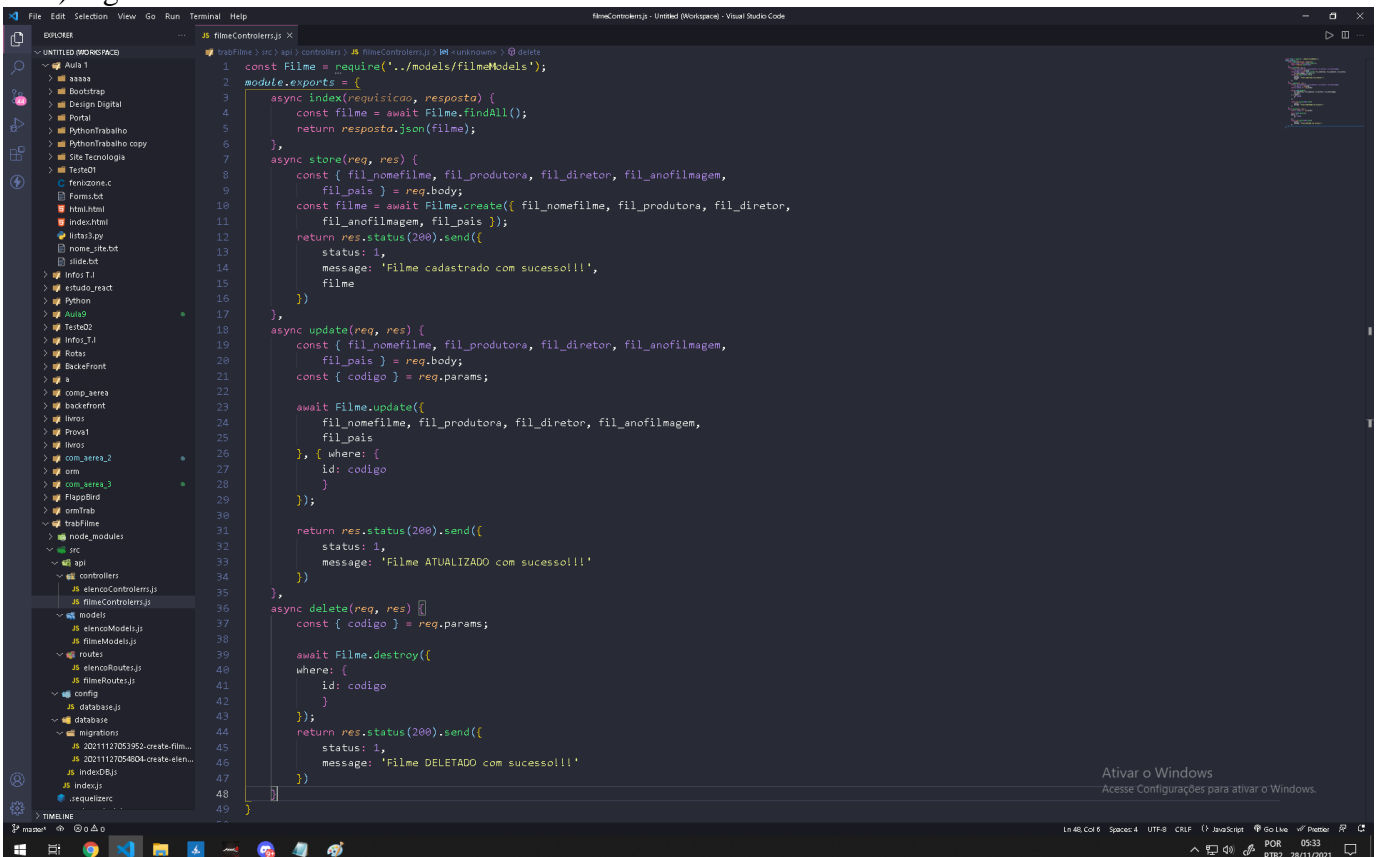
—

3.1) Figura 3: Routes.js da tabela filme – fil



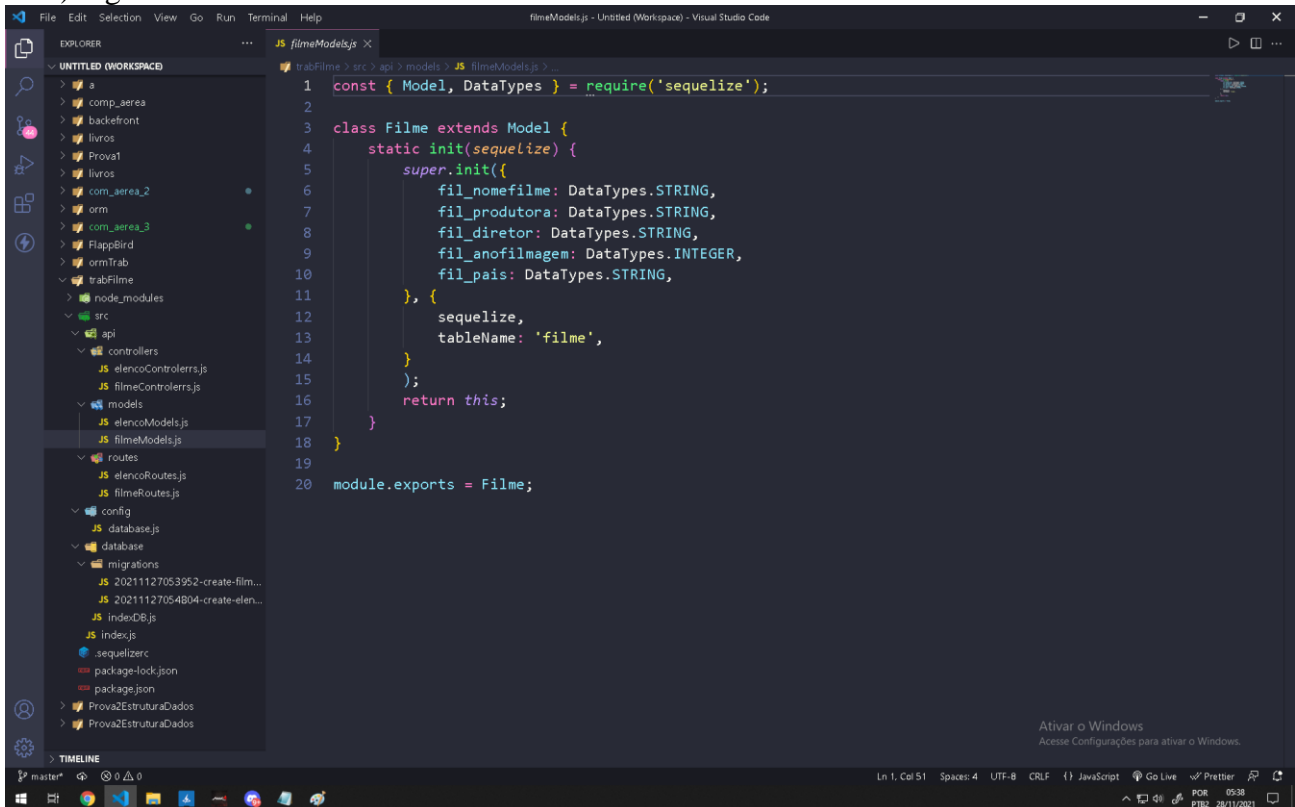
```
1 const express = require('express');
2 const router = express.Router();
3
4 const filmeControllers = require('../controllers/filmeControlerrrs');
5
6 router.get('/filme', filmeControllers.index);
7
8 router.post('/filme', filmeControllers.store);
9
10 router.put('/filme/:codigo', filmeControllers.update);
11
12 router.delete('/filme/:codigo', filmeControllers.delete);
13
14 module.exports = router;
```

3.2) Figura 4: Controllers da tabela filme - fil



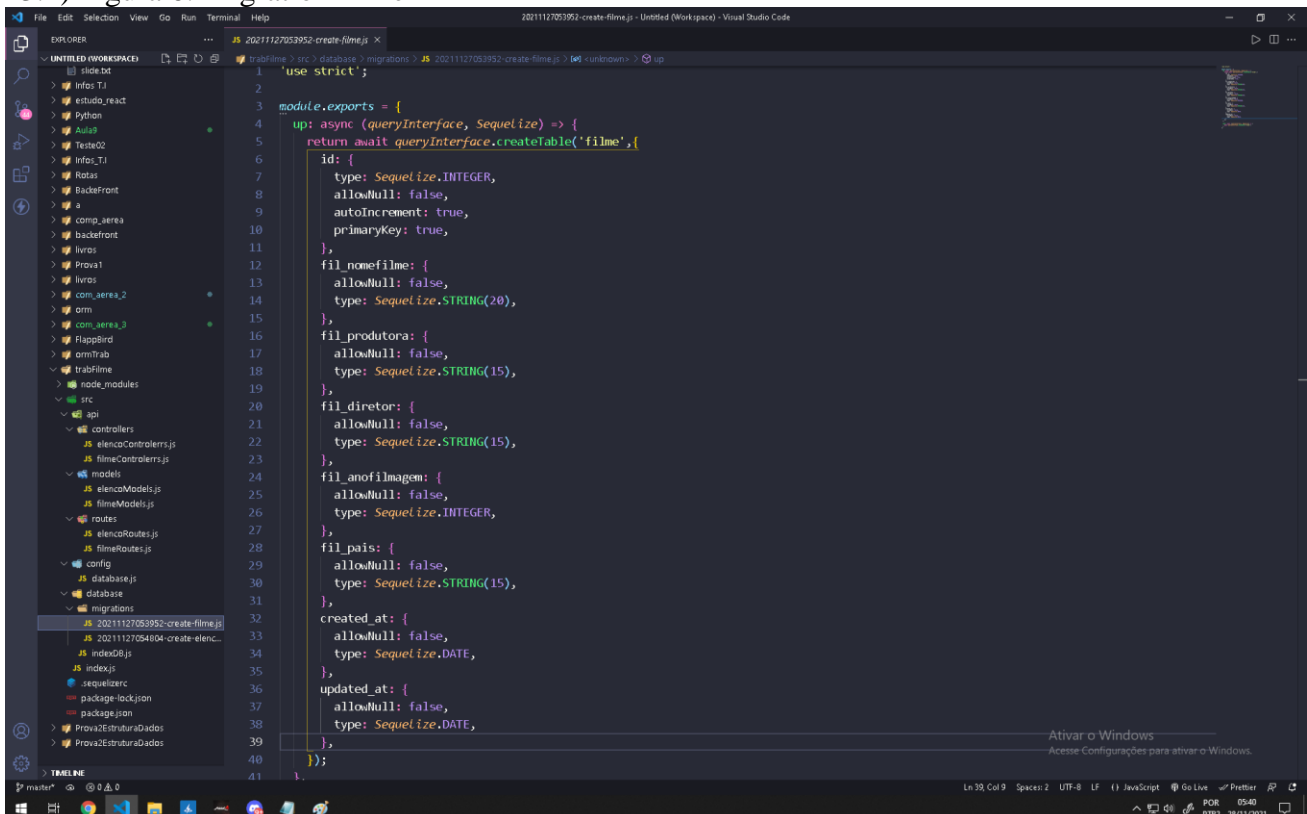
```
1 const Filme = require('../models/filmeModels');
2 module.exports = {
3   async index(req, res) {
4     const filme = await Filme.findAll();
5     return res.json(filme);
6   },
7   async store(req, res) {
8     const { fil_nomefilme, fil_produtores, fil_diretor, fil_anofilagem,
9           fil_pais } = req.body;
10    const filme = await Filme.create({ fil_nomefilme, fil_produtores, fil_diretor,
11          fil_anofilagem, fil_pais });
12    return res.status(200).send({
13      status: 1,
14      message: 'Filme cadastrado com sucesso!!!',
15      filme
16    });
17  },
18  async update(req, res) {
19    const { fil_nomefilme, fil_produtores, fil_diretor, fil_anofilagem,
20          fil_pais } = req.body;
21    const { codigo } = req.params;
22
23    await Filme.update({
24      fil_nomefilme, fil_produtores, fil_diretor, fil_anofilagem,
25      fil_pais
26    }, { where: {
27      id: codigo
28    } });
29
30    return res.status(200).send({
31      status: 1,
32      message: 'Filme ATUALIZADO com sucesso!!!'
33    });
34  },
35  async delete(req, res) {
36    const { codigo } = req.params;
37
38    await Filme.destroy({
39      where: {
40        id: codigo
41      }
42    });
43
44    return res.status(200).send({
45      status: 1,
46      message: 'Filme DELETADO com sucesso!!!'
47    });
48  }
49 }
```

3.3) Figura 5: Models da tabela filme - fil



```
1 const { Model, DataTypes } = require('sequelize');
2
3 class Filme extends Model {
4   static init(sequelize) {
5     super.init({
6       fil_nomefilme: DataTypes.STRING,
7       fil_produtores: DataTypes.STRING,
8       fil_diretor: DataTypes.STRING,
9       fil_anofilmagem: DataTypes.INTEGER,
10      fil_pais: DataTypes.STRING,
11    }, {
12      sequelize,
13      tableName: 'filme',
14    });
15  }
16  return this;
17 }
18
19 module.exports = Filme;
```

3.4) Figura 6: Migration filme

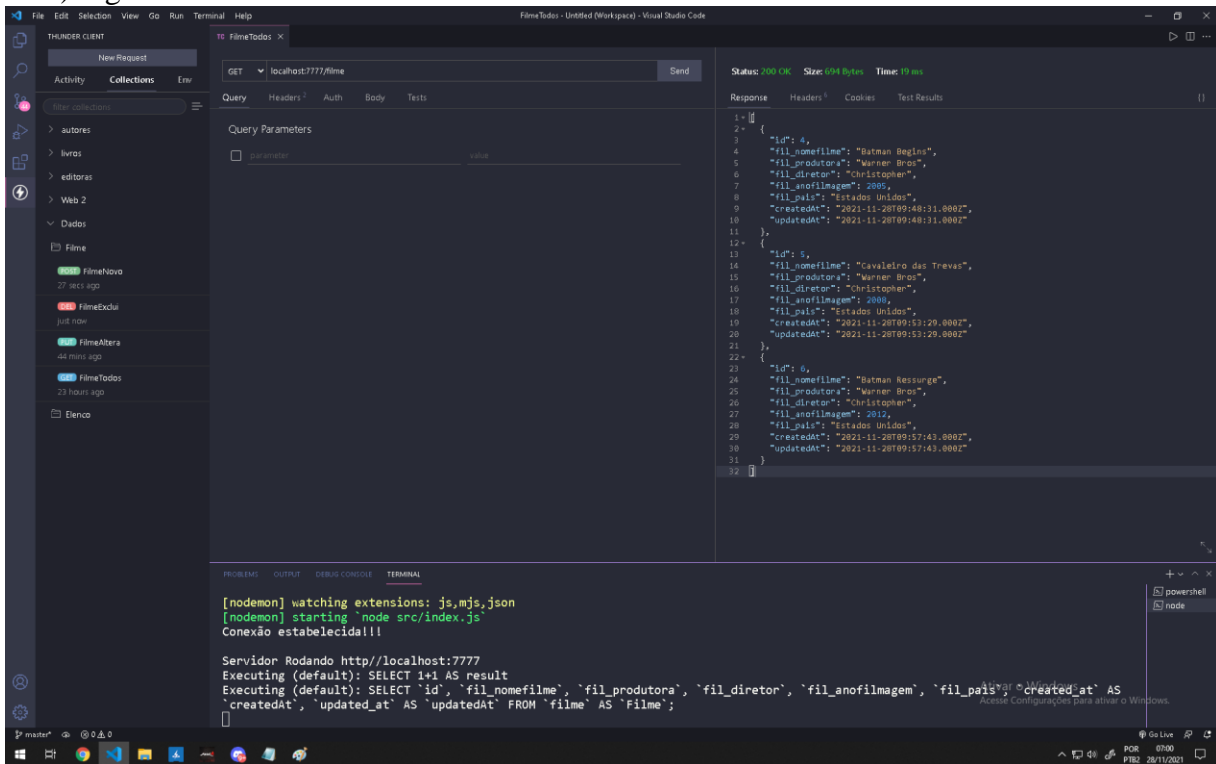


```
1 'use strict';
2
3 module.exports = {
4   up: async (queryInterface, Sequelize) => {
5     return await queryInterface.createTable('filme', {
6       id: {
7         type: Sequelize.INTEGER,
8         allowNull: false,
9         autoIncrement: true,
10        primaryKey: true,
11      },
12      fil_nomefilme: {
13        allowNull: false,
14        type: Sequelize.STRING(20),
15      },
16      fil_produtores: {
17        allowNull: false,
18        type: Sequelize.STRING(15),
19      },
20      fil_diretor: {
21        allowNull: false,
22        type: Sequelize.STRING(15),
23      },
24      fil_anofilmagem: {
25        allowNull: false,
26        type: Sequelize.INTEGER,
27      },
28      fil_pais: {
29        allowNull: false,
30        type: Sequelize.STRING(15),
31      },
32      created_at: {
33        allowNull: false,
34        type: Sequelize.DATE,
35      },
36      updated_at: {
37        allowNull: false,
38        type: Sequelize.DATE,
39      },
40    });
41  },
42 };
```

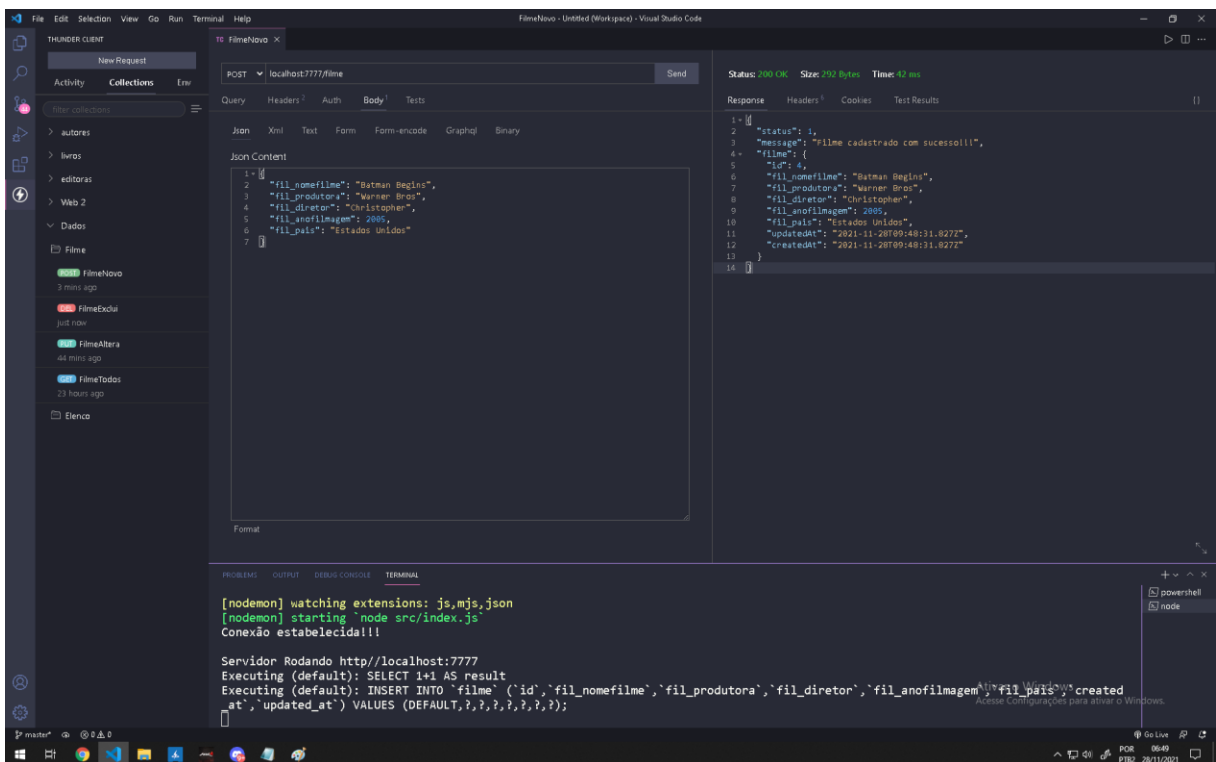
IMPORTANTE:

É necessário que as operações realizadas no backend sejam mostradas no terminal conforme pode ser vistas nas imagens a seguir abaixo da área do Thunder Client

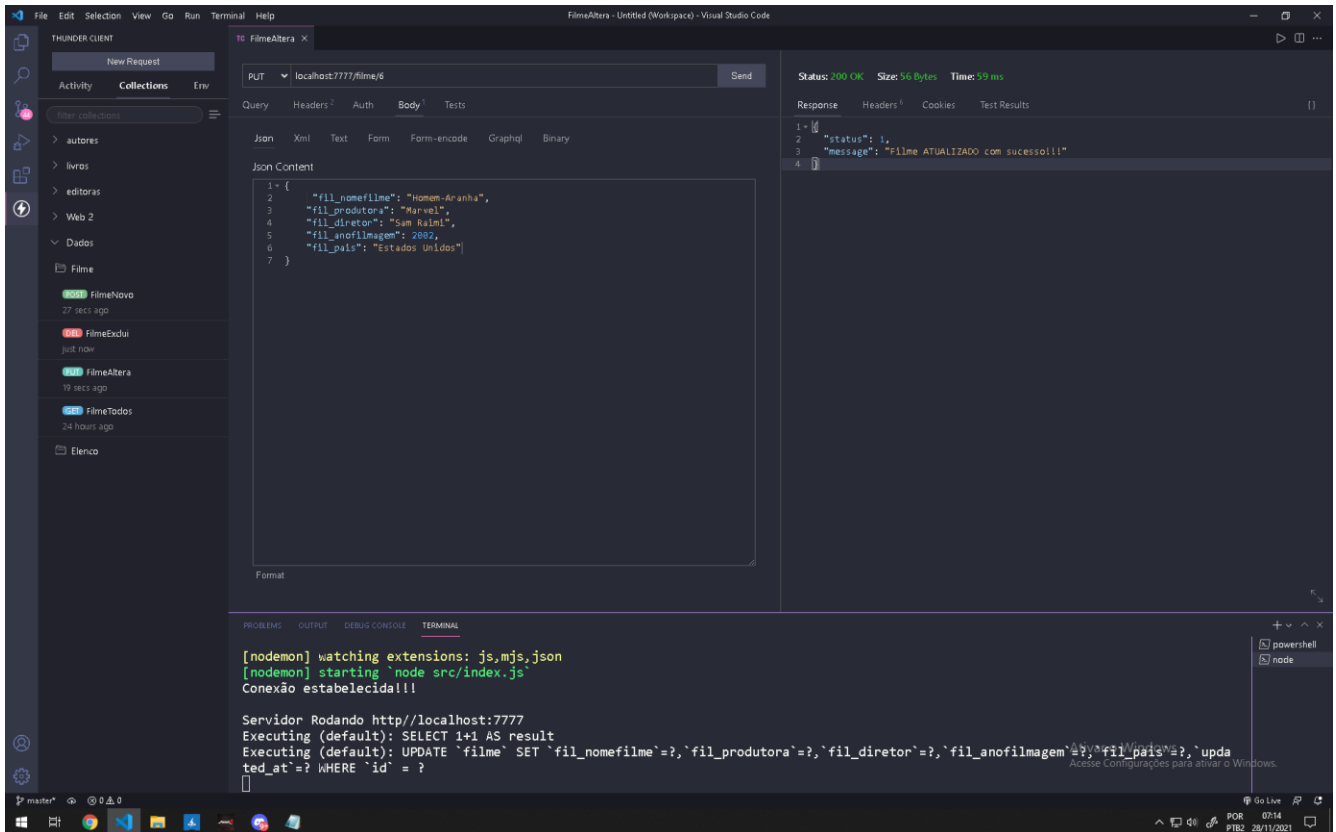
3.5) Figura 7: Protocolo GET método index.



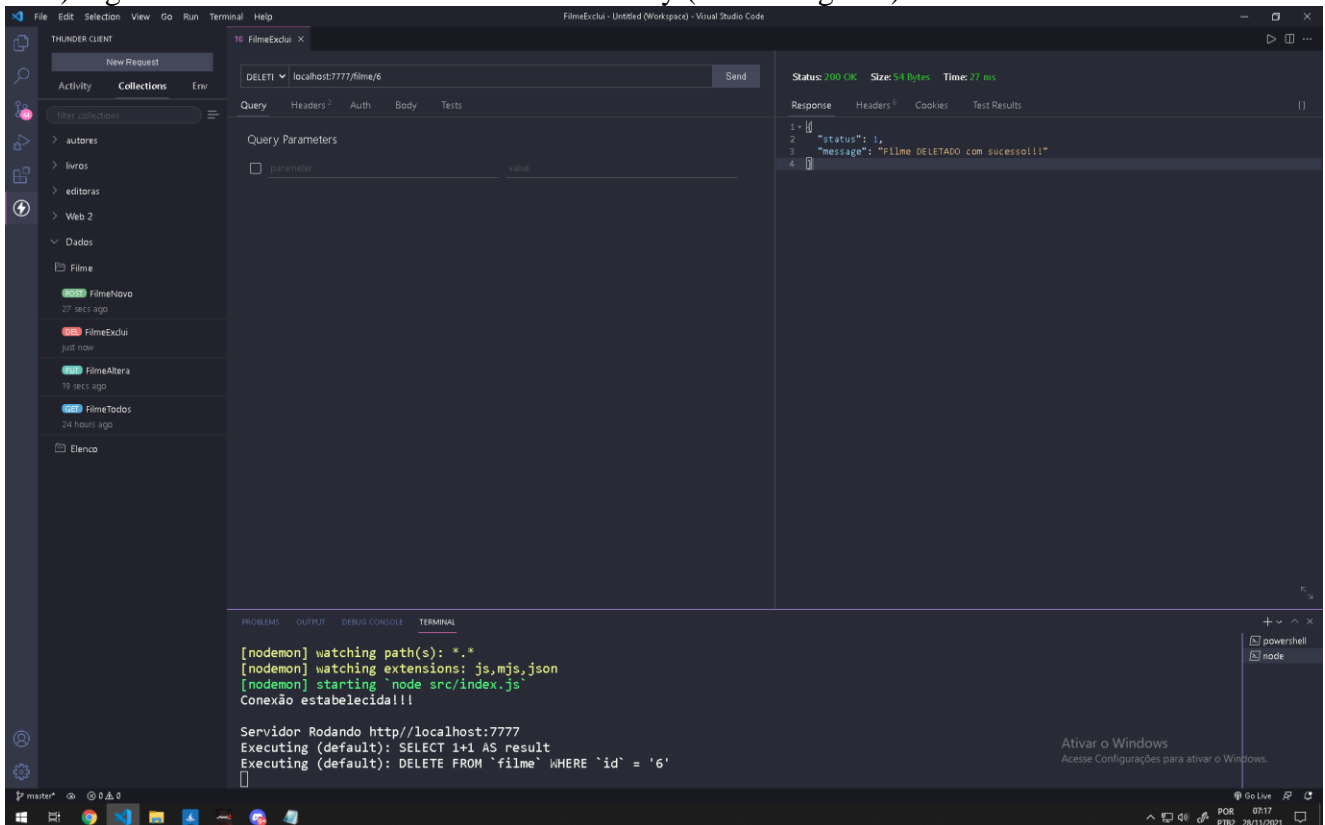
3.5) Figura 8: Protocolo POST método store (adicionar registro).



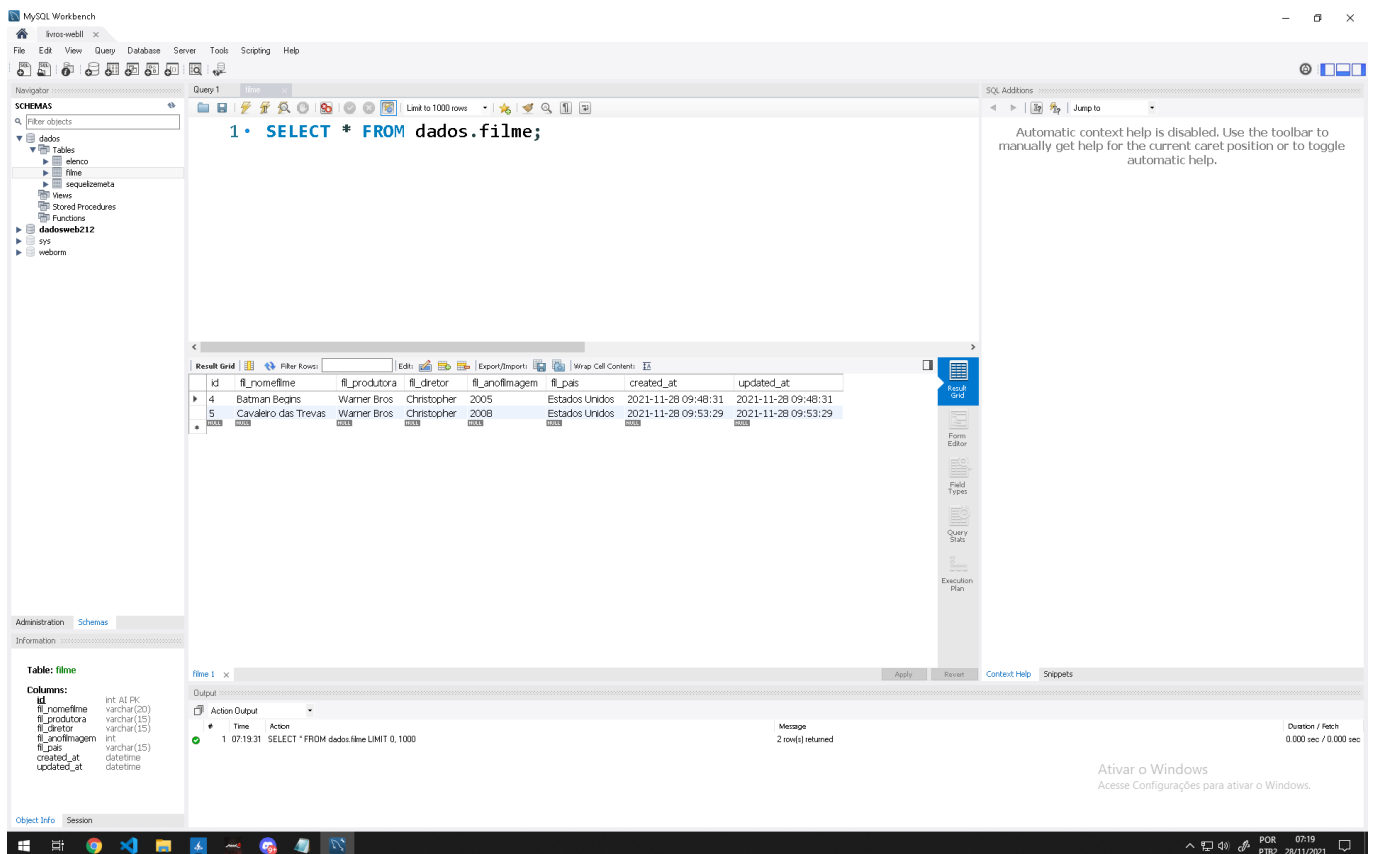
3.6) Figura 9: Protocolo PUT método update (alterar registro).



3.7) Figura 10: Protocolo DELETE método destroy (excluir registro).



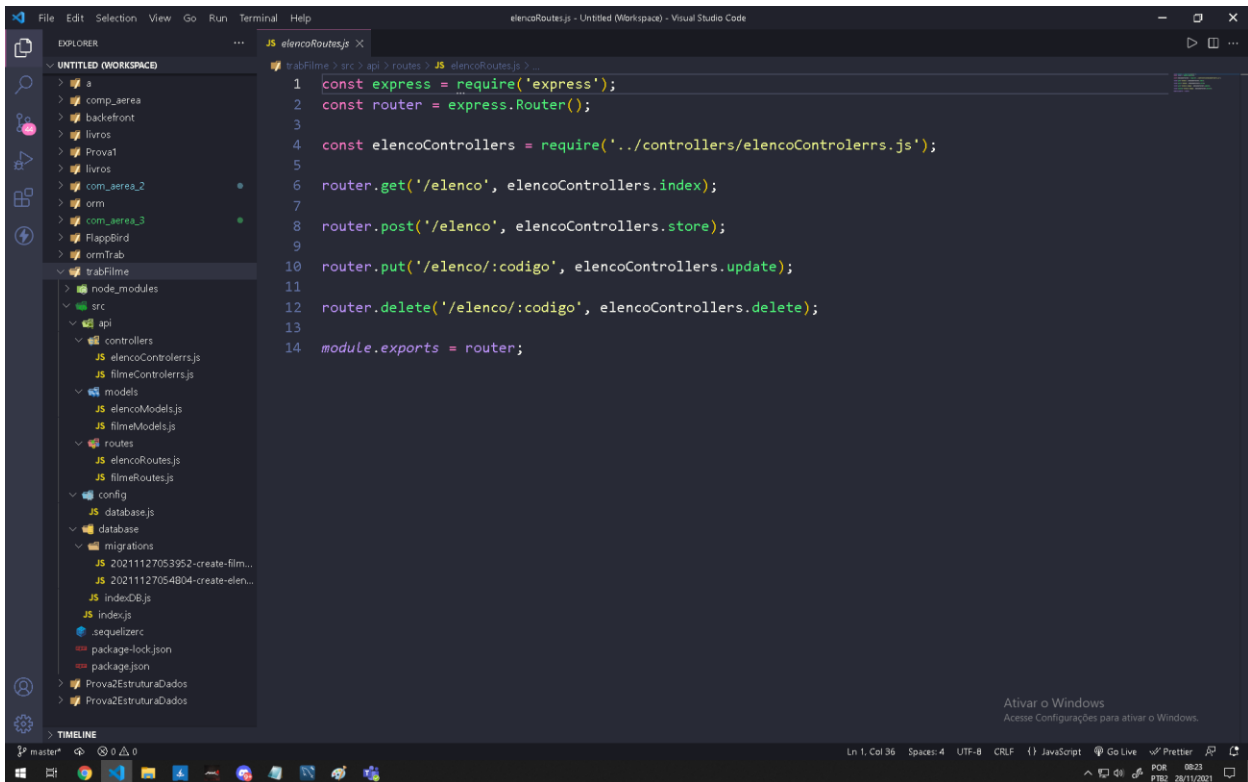
3.8) Figura 11: Imagem do SGDB utilizado, mostrando o banco de dados à esquerda aberto listando os registros da tabela em questão (filme).



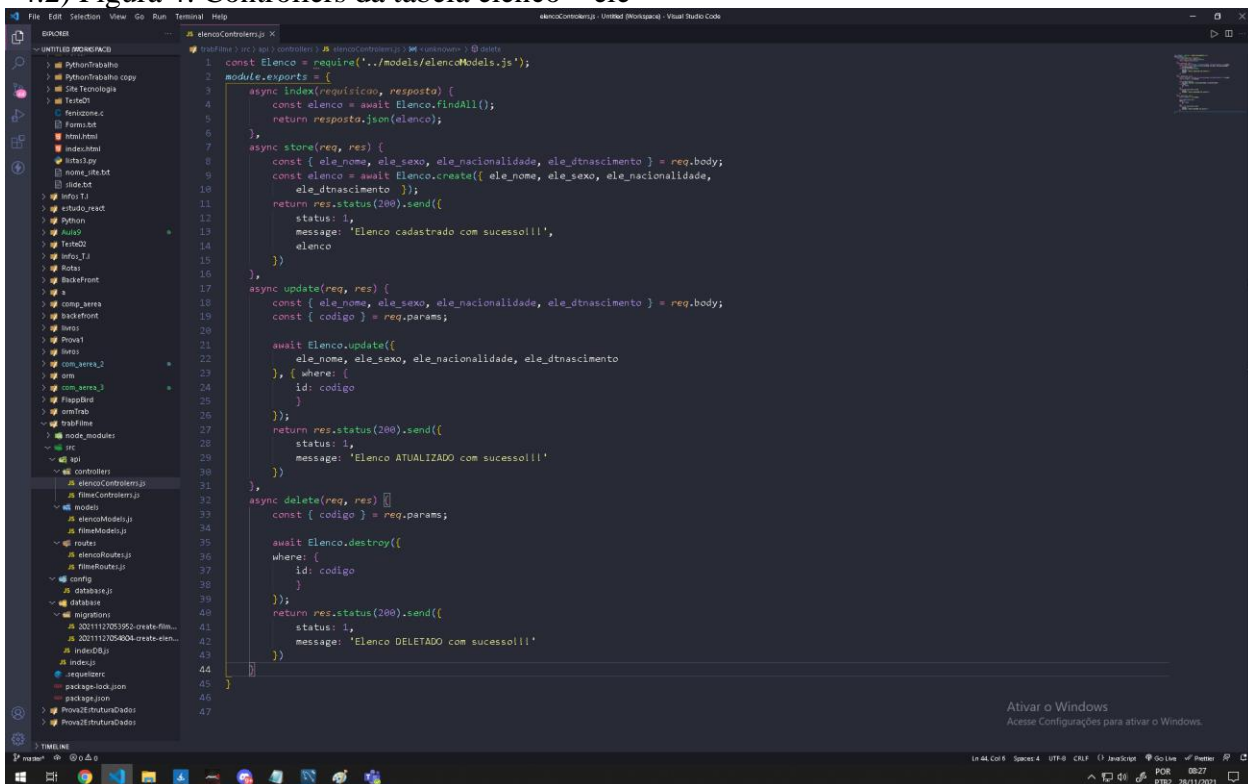
4) A partir de agora serão apresentados os print's sequenciais da segunda tabela informada no documento da tarefa. Neste exemplo apresentaremos a sequência dos códigos na seguinte ordem:

- routes
- controllers
- models
- migrations
- Workbench

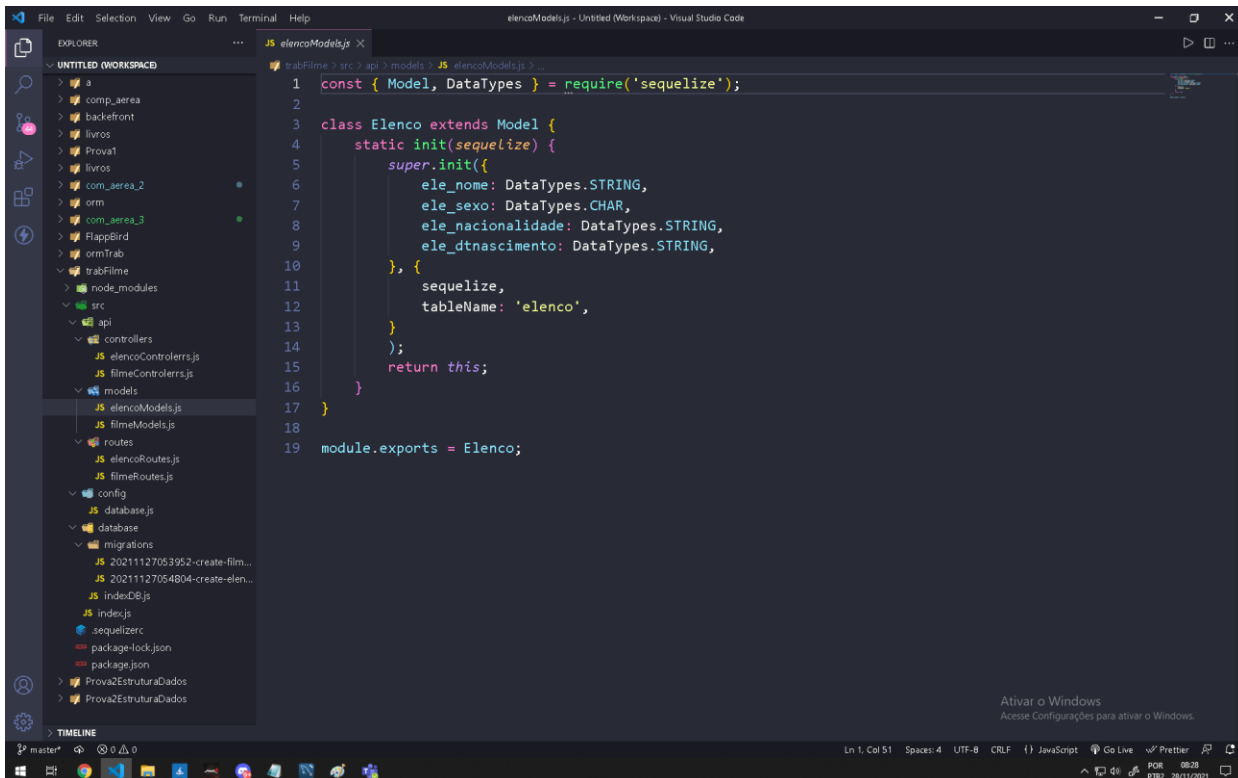
4.1) Figura 3: Routes.js da tabela elenco – ele



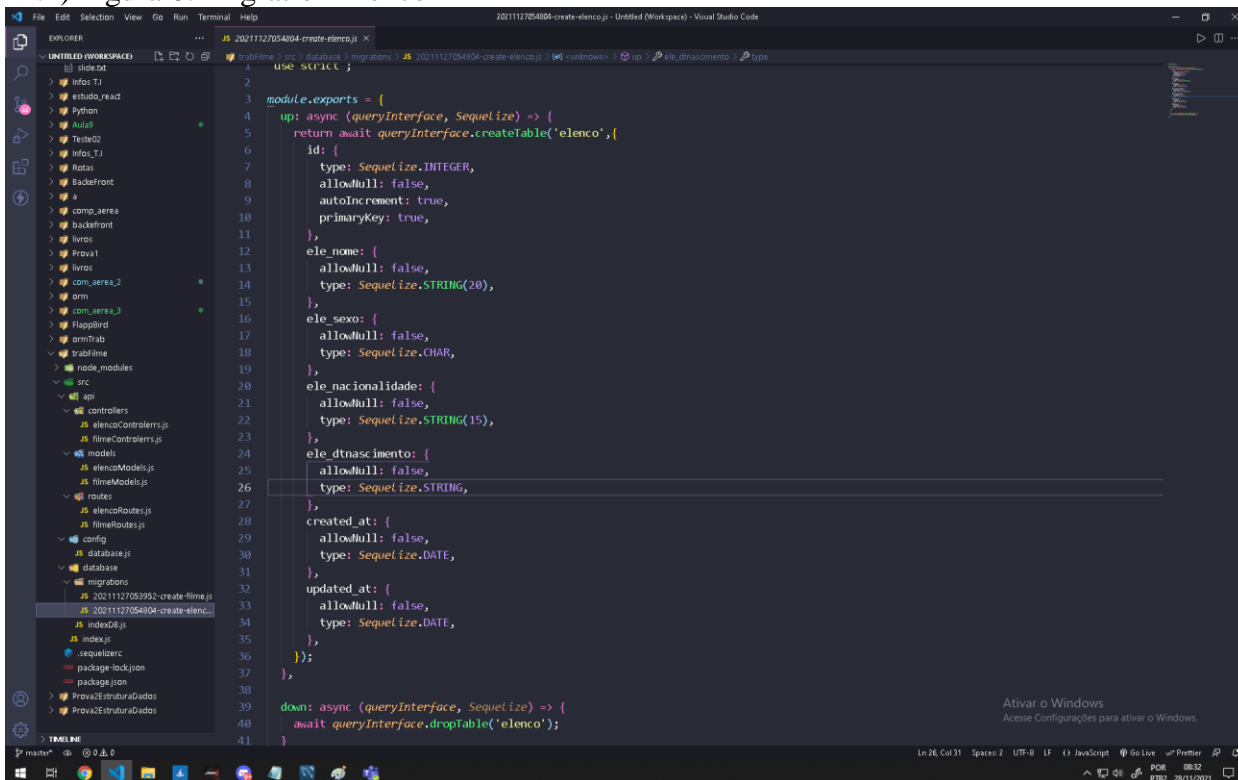
4.2) Figura 4: Controllers da tabela elenco – ele



4.3) Figura 5: Models da tabela elenco – ele



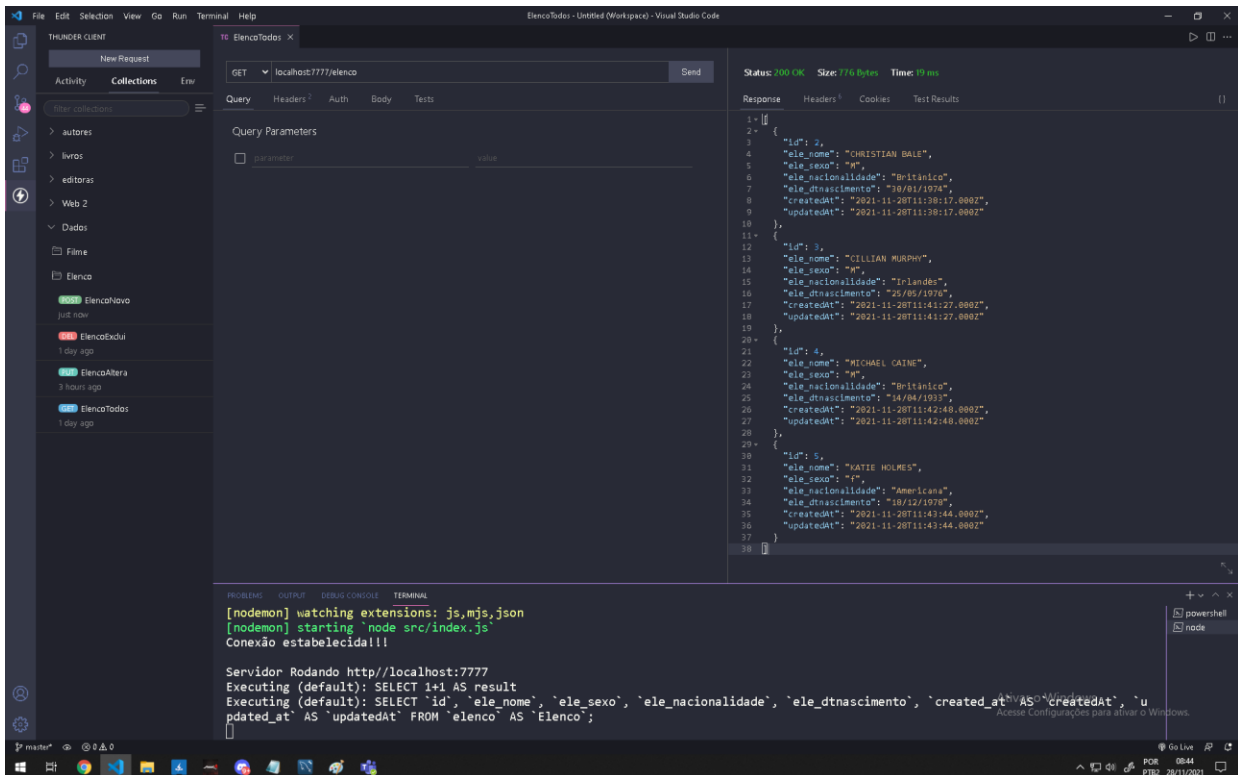
4.4) Figura 6: Migration Elenco



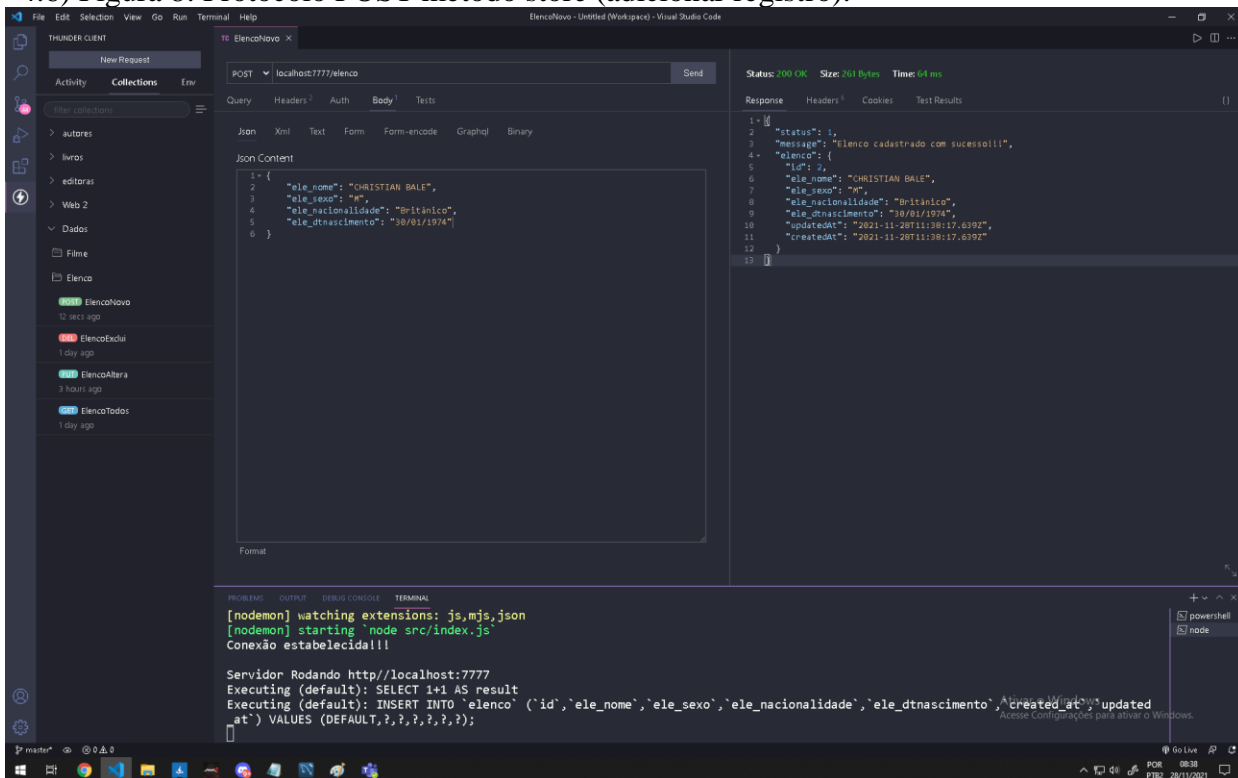
IMPORTANTE:

É necessário que as operações realizadas no backend sejam mostradas no terminal conforme pode ser vistas nas imagens a seguir abaixo da área do Thunder Client

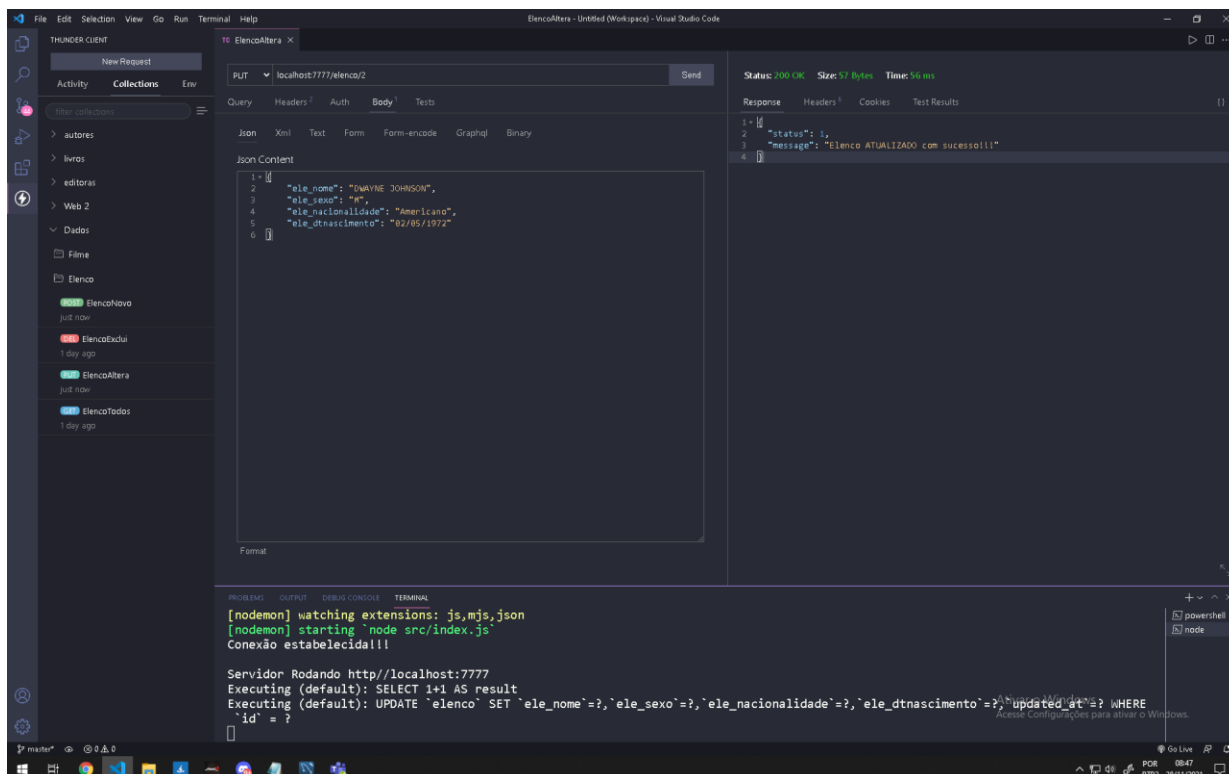
4.5) Figura 7: Protocolo GET método index.



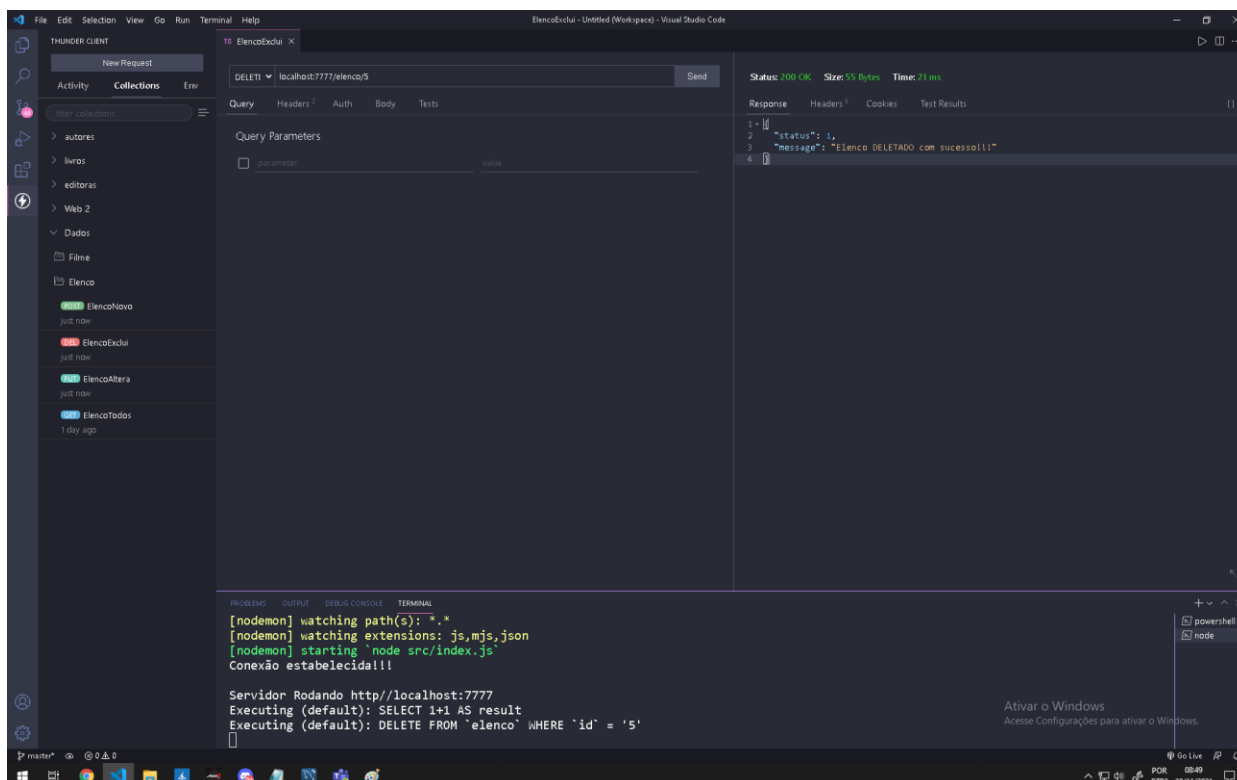
4.6) Figura 8: Protocolo POST método store (adicionar registro).



4.7) Figura 9: Protocolo PUT método update (alterar registro).



4.8) Figura 10: Protocolo DELETE método destroy (excluir registro).



4.9) Figura 11: Imagem do SGDB utilizado, mostrando o banco de dados à esquerda aberto listando os registros da tabela em questão (elenco).

MySQL Workbench

File Edit View Query Database Server Tools Scripting Help

Navigation

SCHEMAS

Filter objects

Filter

Tables

dados

elenco

filme

sequelizemeta

Views

Stored Procedures

Functions

dadosweb212

err

weborn

1. SELECT * FROM dados.elenco;

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Table: elenco

Columns:

id int(4) PK

ele_nome varchar(255)

ele_sexo char(1)

ele_nacionalidade varchar(255)

ele_data_nascimento varchar(255)

created_at datetime

updated_at datetime

Output

Action Output

#	Time	Action	Message	Duration / Rows
1	07:19:31	SELECT * FROM dados.elenco LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
2	07:21:23	SELECT * FROM dados.elenco LIMIT 0, 1000	2 row(s) returned	0.000 sec / 0.000 sec
3	08:50:41	SELECT * FROM dados.elenco LIMIT 0, 1000	3 row(s) returned	0.000 sec / 0.000 sec

Ativar o Windows

Acesse Configurações para ativar o Windows.

PCB 08:50

PTB2 28/11/2021