

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

Data: 24/11/2021

Aluno: Lucas Antônio Ribeiro

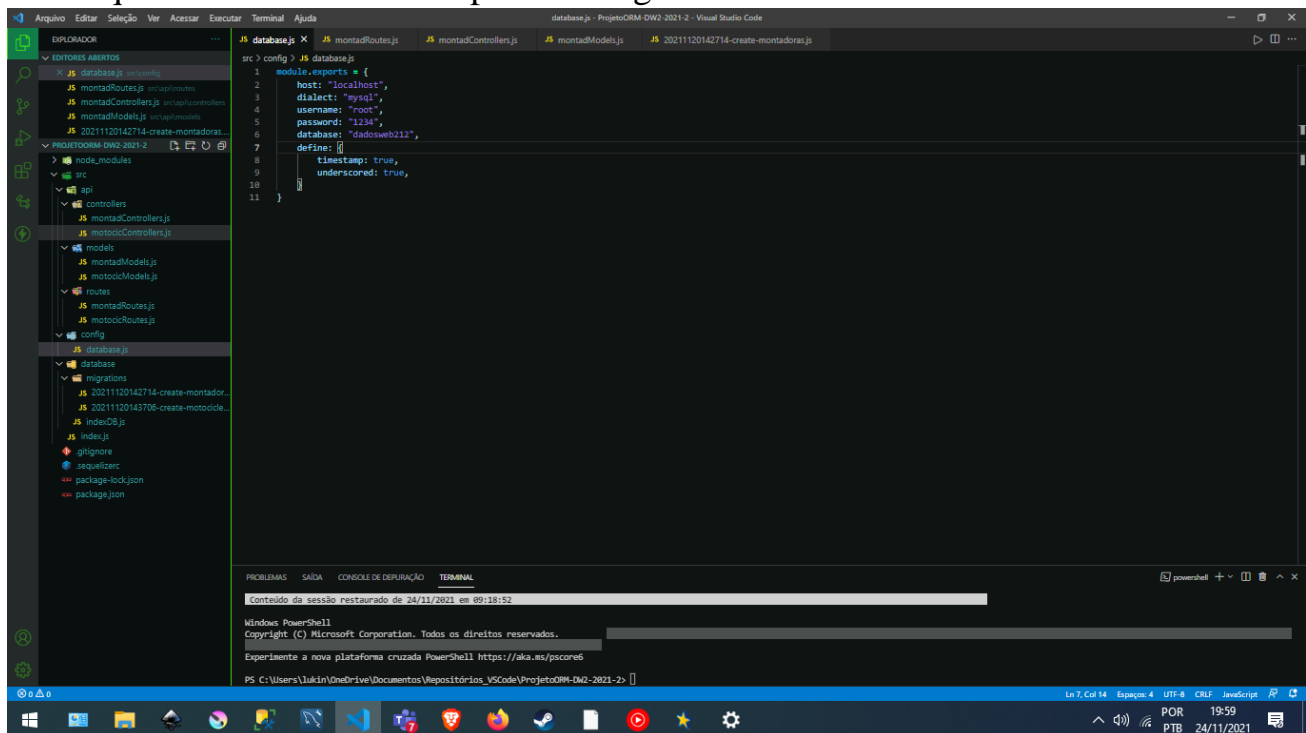
## Projeto 13 – Motocicleta x Fabricante

1) Figura 1: Estrutura das tabelas – ao lado

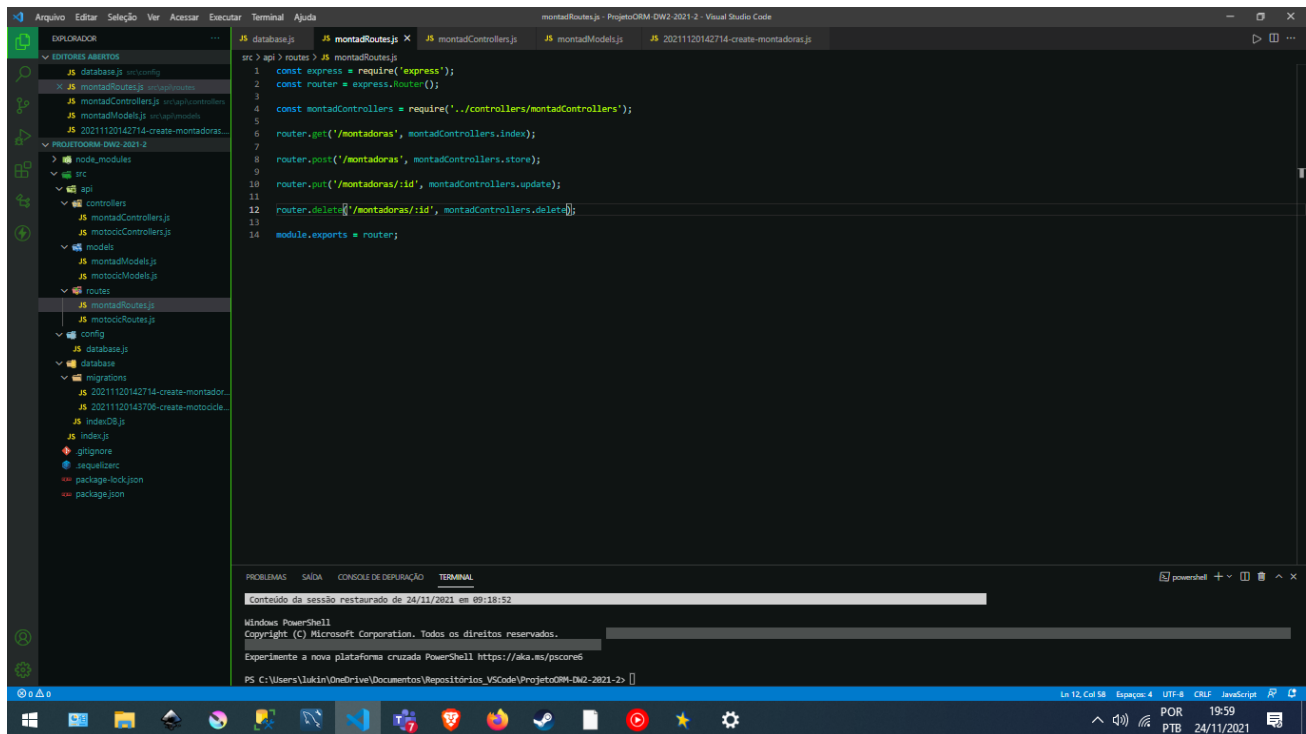
Especificação da Entidade – Tabela: MONTADORA - MON				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	mon_codigo		Chave primária da tabela
	varchar	mon_nome	30	Nome da montadora
	varchar	mon_fantasia	10	Nome fantasia da montadora
	varchar	mon_pais	20	País de origem da montadora

Especificação da Entidade – Tabela: MOTOCICLETA - MOT				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	mot_codigo		Chave primária da tabela
	varchar	mot_modelo	20	Identificação do modelo da motocicleta
	varchar	mot_cor	10	Descrição da cor da motocicleta
	varchar	mot_categoria	15	Categoria da motocicleta
	varchar	mot_motor	20	Identificação do motor
	inteiro	mot_ano		Ano de fabricação da motocicleta
	numerico	mot_valorcusto	12,2	Valor de custo junto a montadora
FK	inteiro	mon_codigo		Código da montadora – chave estrangeira

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

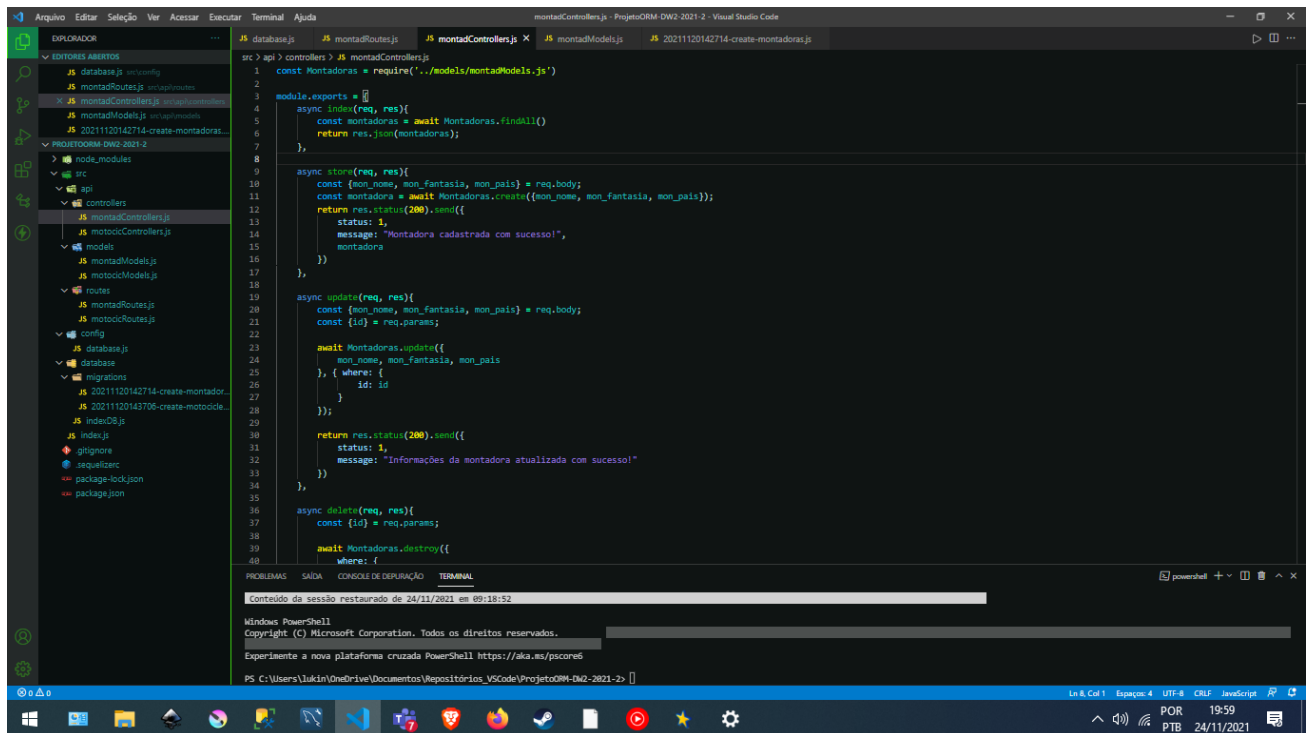


### 3.1) Figura 3: Routes.js da tabela montadoras – MON – montad



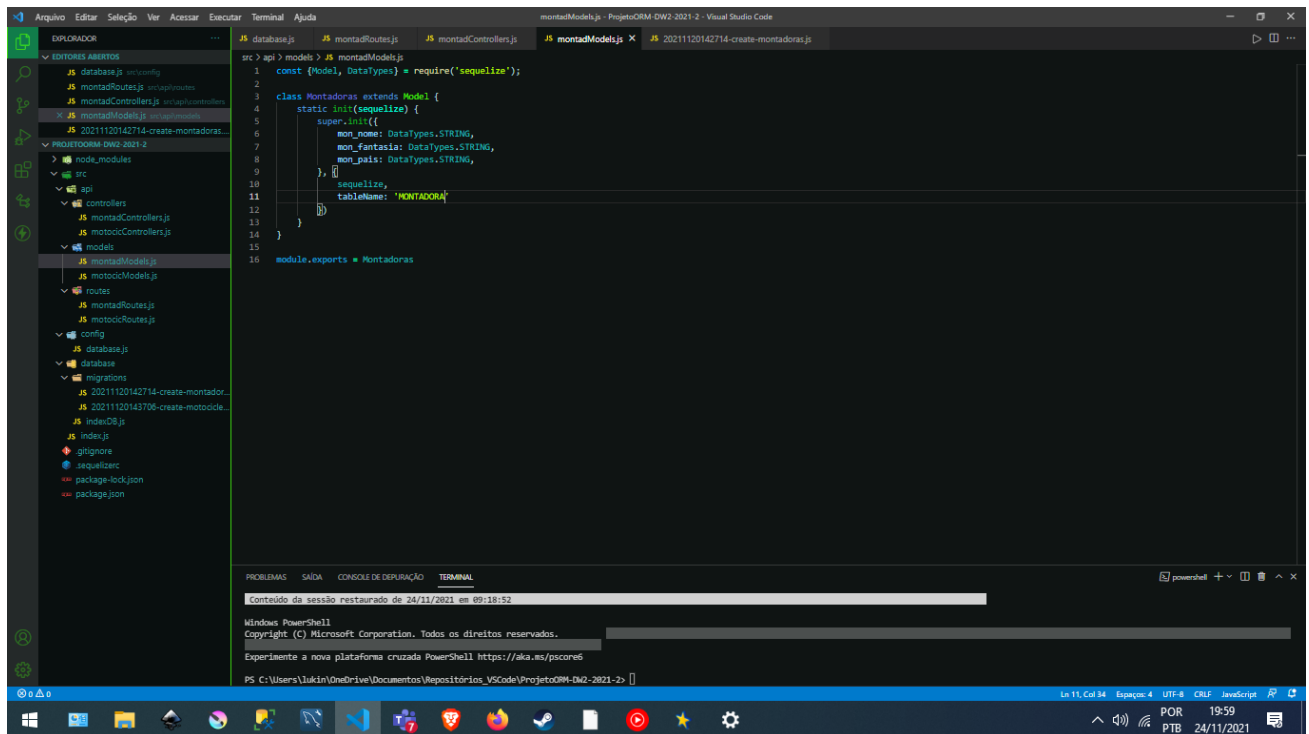
```
src > api > routes > .js montadRoutes.js
1 const express = require('express');
2 const router = express.Router();
3
4 const montadControllers = require('../controllers/montadControllers');
5
6 router.get('/montadoras', montadControllers.index);
7
8 router.post('/montadoras', montadControllers.store);
9
10 router.put('/montadoras/:id', montadControllers.update);
11
12 router.delete('/montadoras/:id', montadControllers.delete);
13
14 module.exports = router;
```

### 3.2) Figura 4: Controllers da tabela montadoras – MON – montad

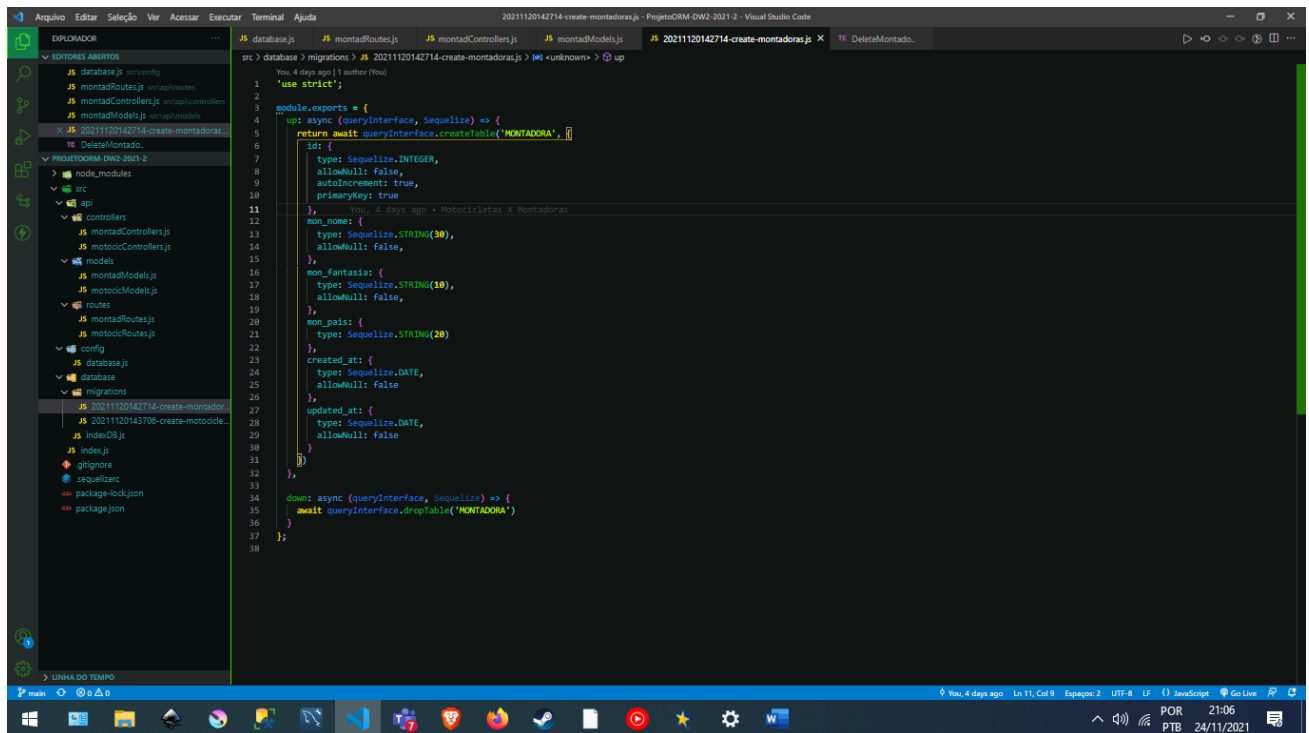


```
src > api > controllers > .js montadControllers.js
1 const Montadoras = require('../models/montadModels.js');
2
3 module.exports = {
4   async index(req, res){
5     const montadoras = await Montadoras.findAll();
6     return res.json(montadoras);
7   },
8
9   async store(req, res){
10    const {mon_nome, mon_fantasia, mon_pais} = req.body;
11    const montadora = await Montadoras.create({mon_nome, mon_fantasia, mon_pais});
12    return res.status(200).send({
13      status: 1,
14      message: "Montadora cadastrada com sucesso!",
15      montadora
16    });
17  },
18
19   async update(req, res){
20    const {mon_nome, mon_fantasia, mon_pais} = req.body;
21    const {id} = req.params;
22
23    await Montadoras.update({
24      mon_nome, mon_fantasia, mon_pais
25    }, { where: {
26      id: id
27    } });
28
29    return res.status(200).send({
30      status: 1,
31      message: "Informações da montadora atualizada com sucesso!"
32    });
33  },
34
35   async delete(req, res){
36    const {id} = req.params;
37
38    await Montadoras.destroy({
39      where: {
40        id: id
41      }
42    });
43  }
44 }
```

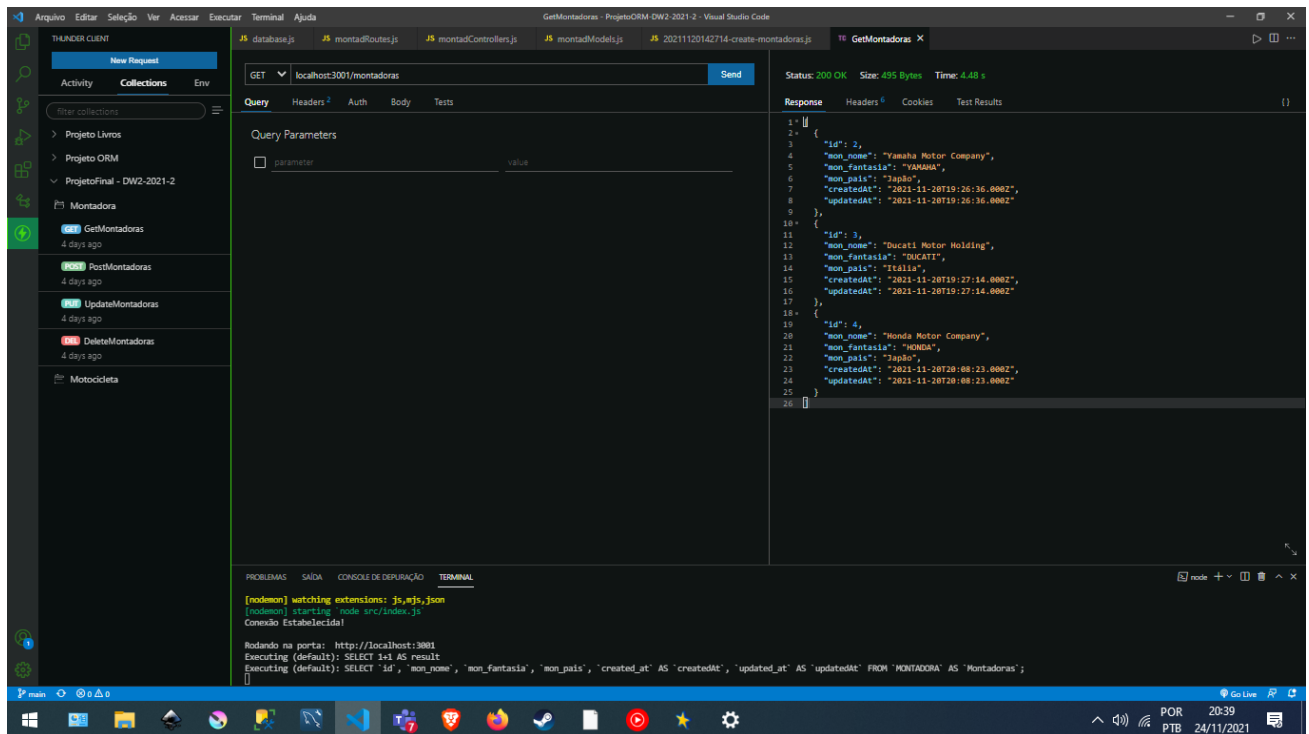
### 3.3) Figura 5: Models da tabela montadora – MON – montad



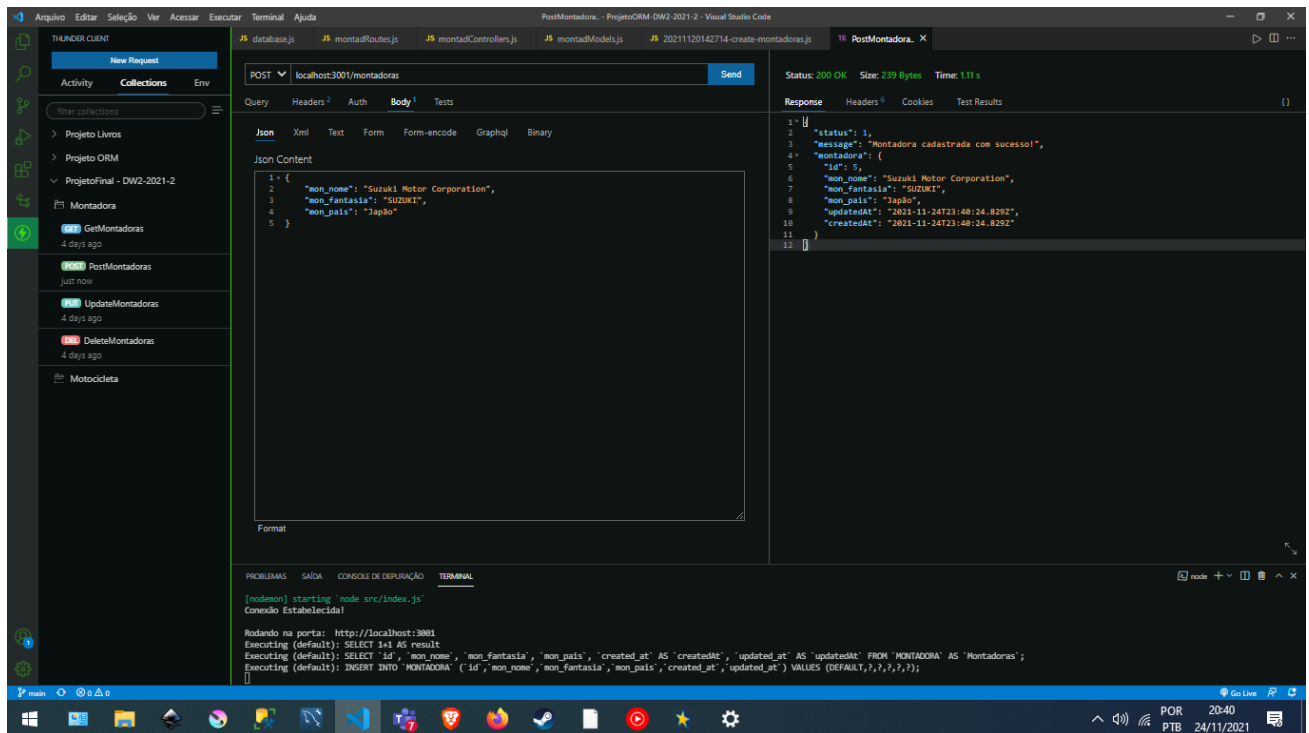
### 3.4) Figura 6: Migration Montadora



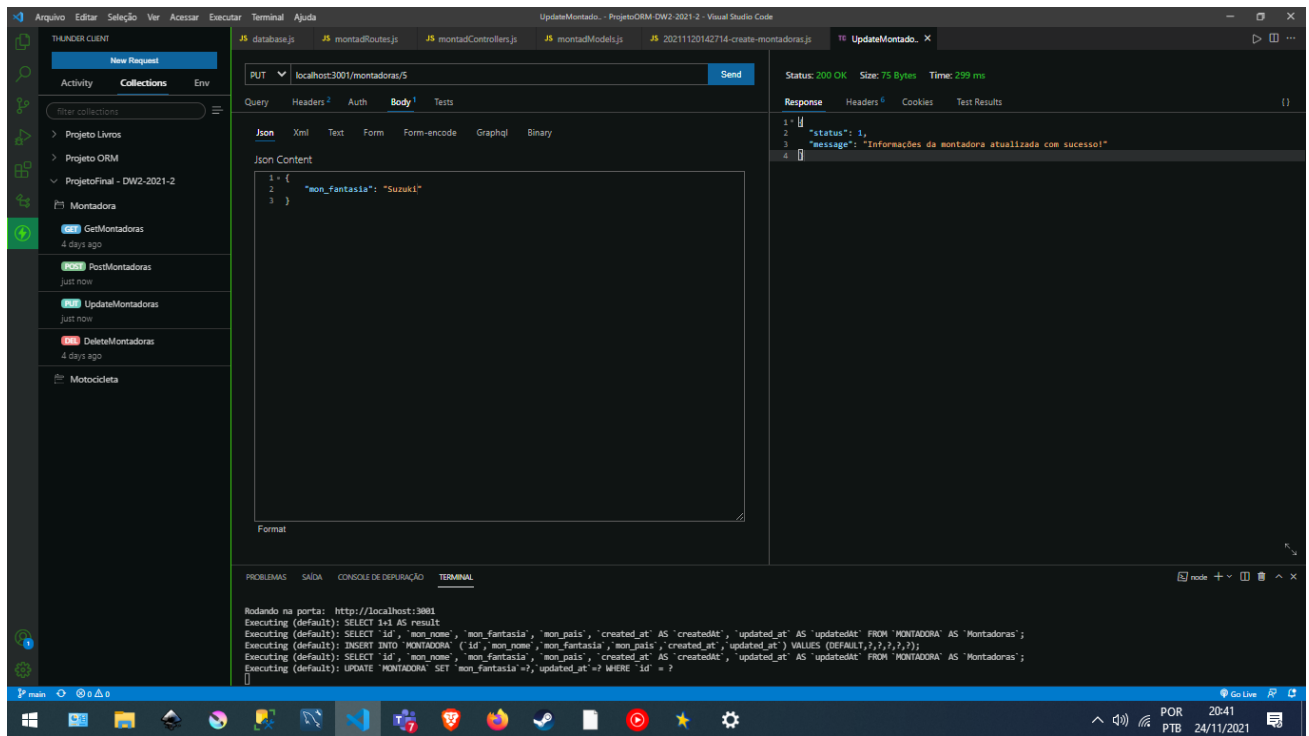
### 3.5) Figura 7: Protocolo GET método index



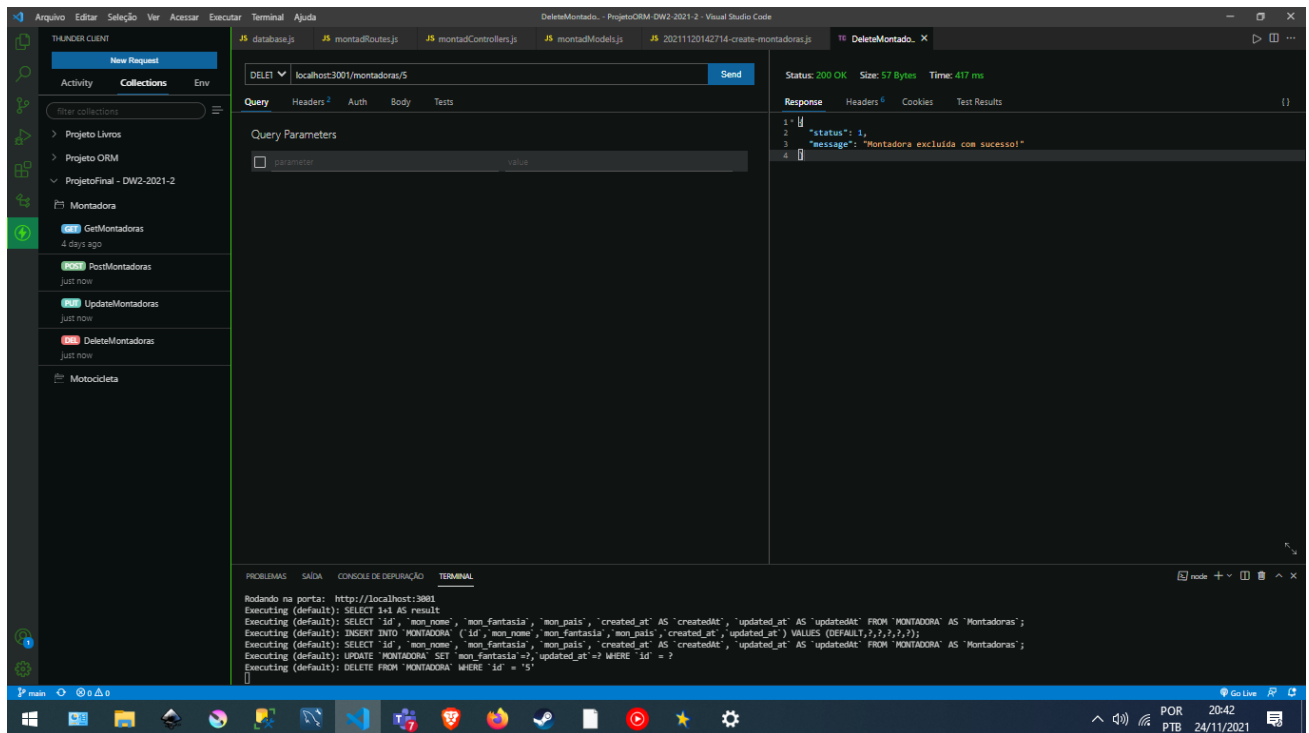
### 3.6) Figura 8: Protocolo POST m\u00e9todo store (adicionar registro).



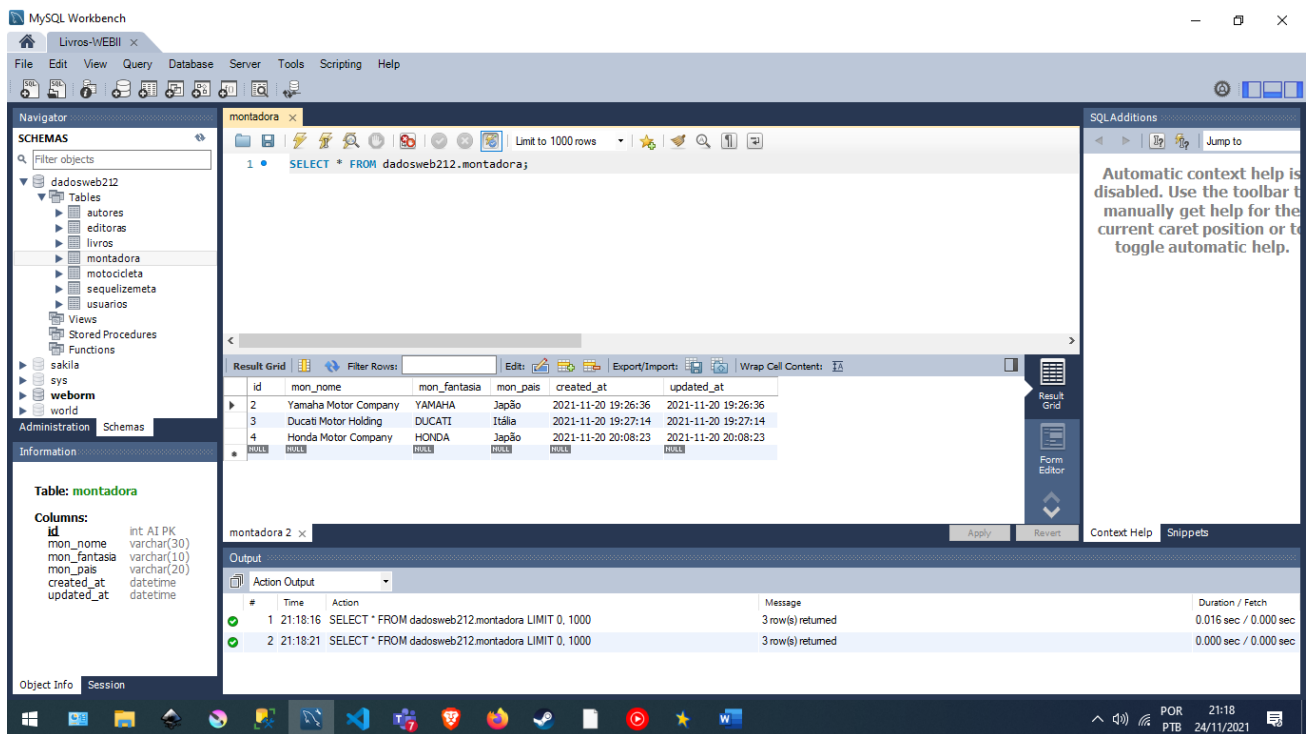
### 3.7) Figura 9: Protocolo PUT método update (alterar registro)



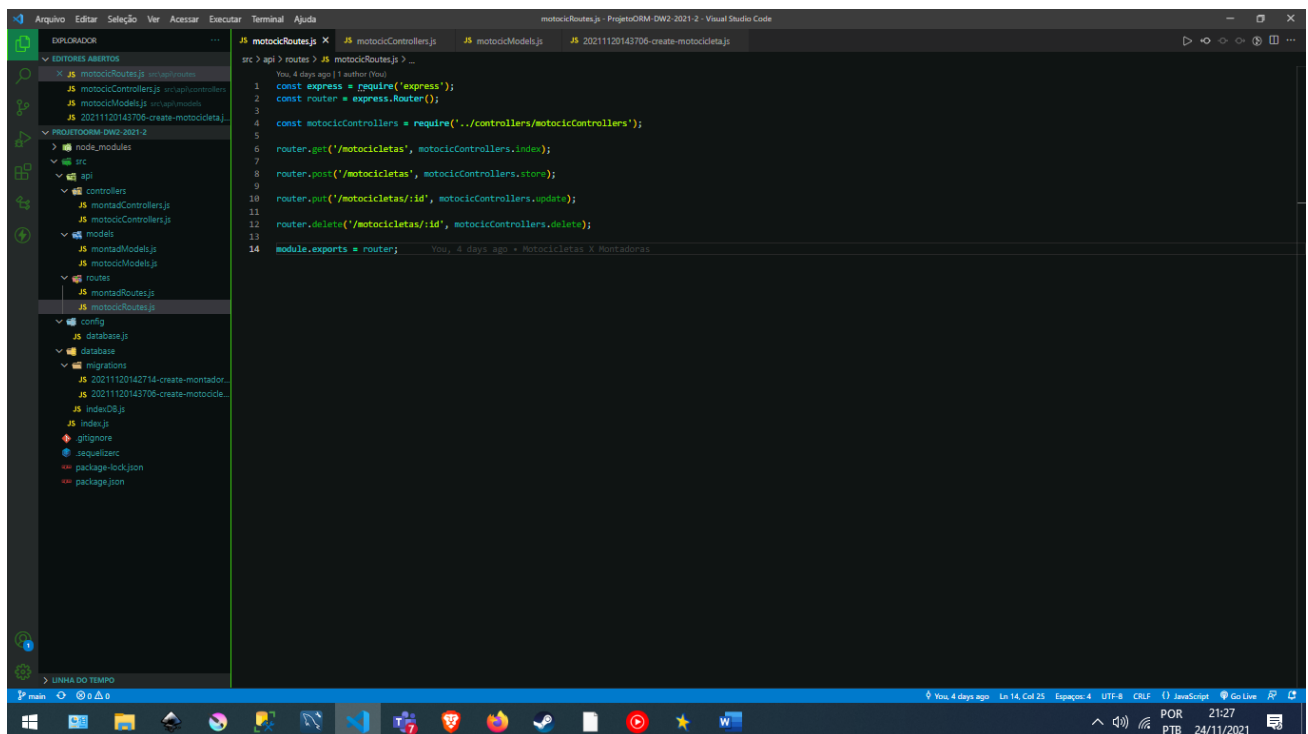
### 3.8) Figura 10: Protocolo DELETE método destroy (excluir registro).



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



4.1) Figura 12: Routes.js da tabela motocicleta – MOT – motocic



4.2) Figura 13: Controllers da tabela motocicleta – MOT – motocic

```

1 const Motocicletas = require('../models/motocicModels.js')
2
3 module.exports = {
4   async index(req, res){
5     const motocicleta = await Motocicletas.findAll()
6     return res.json(motocicleta);
7   },
8
9   async store(req, res){
10    const {mot_modelo, mot_cor, mot_categoria, mot_motor, mot_ano, mot_valorcusto, mon_codigo} = req.body;
11    const motocicleta = await Motocicletas.create({mot_modelo, mot_cor, mot_categoria, mot_motor, mot_ano, mot_valorcusto, mon_codigo});
12    return res.status(200).send({
13      status: 1,
14      message: "Motocicleta cadastrada com sucesso!",
15      motocicleta
16    });
17   },
18
19   async update(req, res){
20    const {mot_modelo, mot_cor, mot_categoria, mot_motor, mot_ano, mot_valorcusto, mon_codigo} = req.body;
21    const {id} = req.params;
22
23    await Motocicletas.update(
24      {
25        mot_modelo, mot_cor, mot_categoria, mot_motor, mot_ano, mot_valorcusto, mon_codigo
26      }, { where: {
27        id: id
28      } });
29
30    return res.status(200).send({
31      status: 1,
32      message: "Informações da motocicleta atualizada com sucesso!"
33    });
34   },
35
36   async delete(req, res){
37    const {id} = req.params;
38
39    await Motocicletas.destroy(
40      {
41        id: id
42      } );
43
44    return res.status(200).send({
45      status: 1,
46      message: "Motocicleta excluída com sucesso!"
47    });
48   }
49 }

```

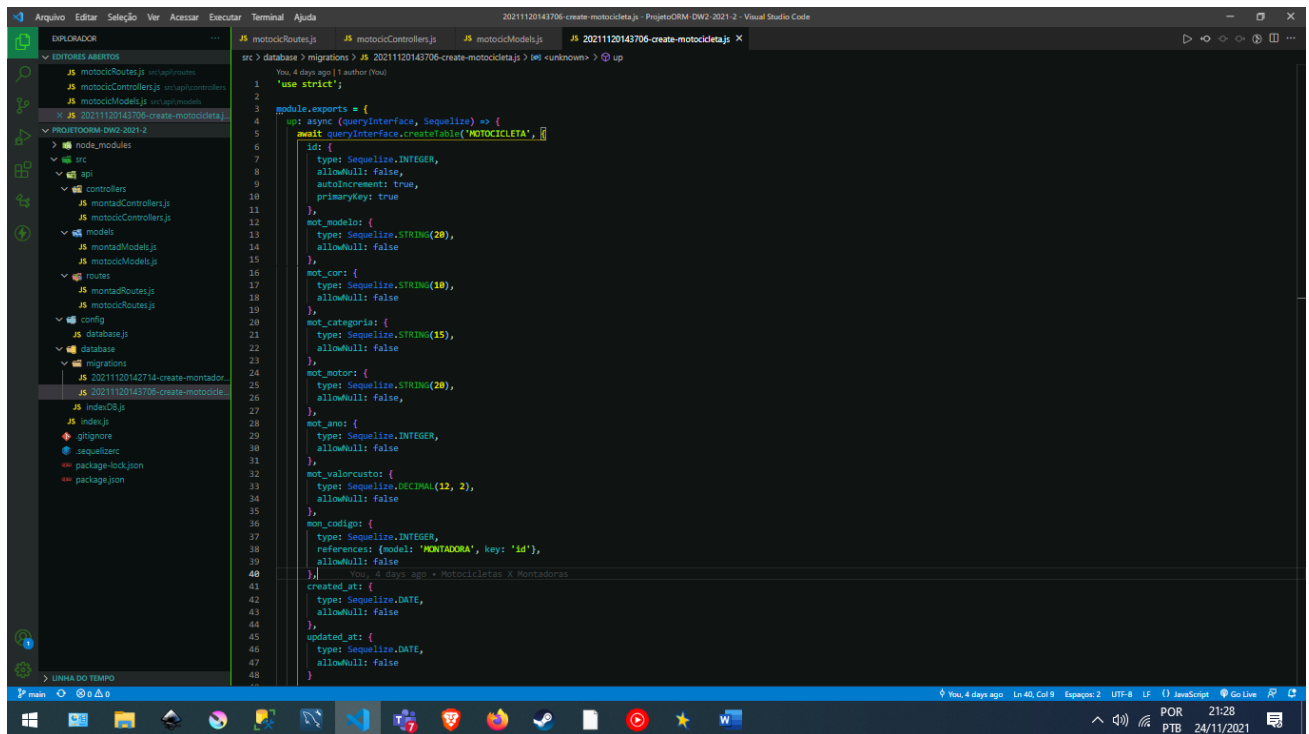
4.3) Figura 14: Models da tabela motocicleta – MOT – motocic

```

1 const {Model, DataTypes} = require('sequelize');
2
3 class Motocicletas extends Model {
4   static init(sequelize) {
5     super.init({
6       mot_modelo: DataTypes.STRING,
7       mot_cor: DataTypes.STRING,
8       mot_categoria: DataTypes.STRING,
9       mot_motor: DataTypes.STRING,
10      mot_ano: DataTypes.INTEGER,
11      mot_valorcusto: DataTypes.DECIMAL,
12      mon_codigo: DataTypes.INTEGER,
13    }, sequelize);
14    tableName: 'MOTOCICLETA'
15  }
16 }
17
18 module.exports = Motocicletas

```

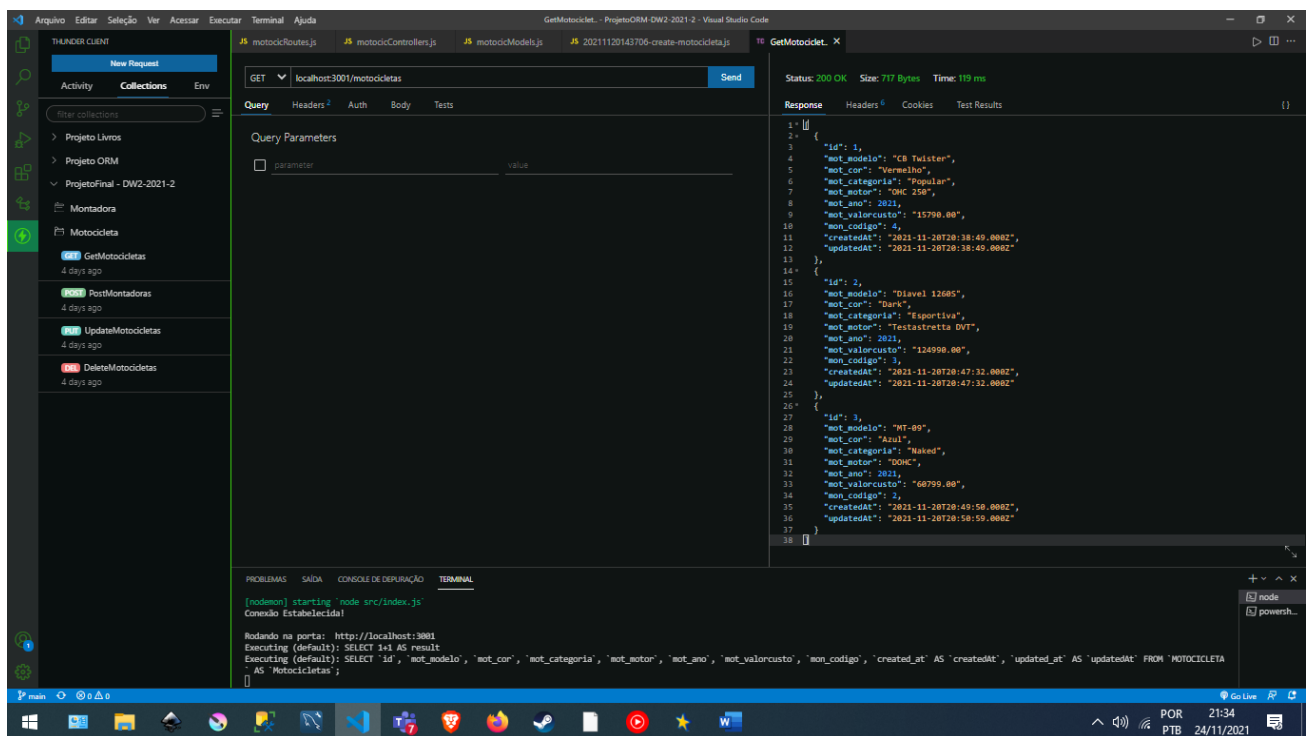
#### 4.4) Figura 15: Migration Motocicleta



The screenshot shows a Visual Studio Code editor with a file explorer on the left and a code editor in the center. The file explorer shows a project structure with folders like 'src', 'controllers', 'models', 'routes', 'config', 'database', and 'migrations'. The code editor displays a migration file named '20211120143706-create-motocicleta.js'. The code defines a Sequelize model for 'MOTOCICLETA' with the following attributes:

```
module.exports = {
  'use strict';
  module: {
    id: {
      type: Sequelize.INTEGER,
      allowNull: false,
      autoIncrement: true,
      primaryKey: true
    },
    mot_modelo: {
      type: Sequelize.STRING(20),
      allowNull: false
    },
    mot_cor: {
      type: Sequelize.STRING(10),
      allowNull: false
    },
    mot_categoria: {
      type: Sequelize.STRING(15),
      allowNull: false
    },
    mot_motor: {
      type: Sequelize.STRING(20),
      allowNull: false
    },
    mot_ano: {
      type: Sequelize.INTEGER,
      allowNull: false
    },
    mot_valorcusto: {
      type: Sequelize.DECIMAL(12, 2),
      allowNull: false
    },
    mon_codigo: {
      type: Sequelize.INTEGER,
      references: { model: 'MONTADORA', key: 'id' },
      allowNull: false
    },
    created_at: {
      type: Sequelize.DATE,
      allowNull: false
    },
    updated_at: {
      type: Sequelize.DATE,
      allowNull: false
    }
  }
};
```

#### 4.5) Figura 16: Protocolo GET método index



The screenshot shows a Visual Studio Code editor with a REST client window open. The REST client window displays a GET request to 'localhost:3001/motocicletas'. The response is a JSON array of three motorcycle objects. The terminal window shows the command 'node src/index.js' being executed, and the output displays the JSON response of the GET request.

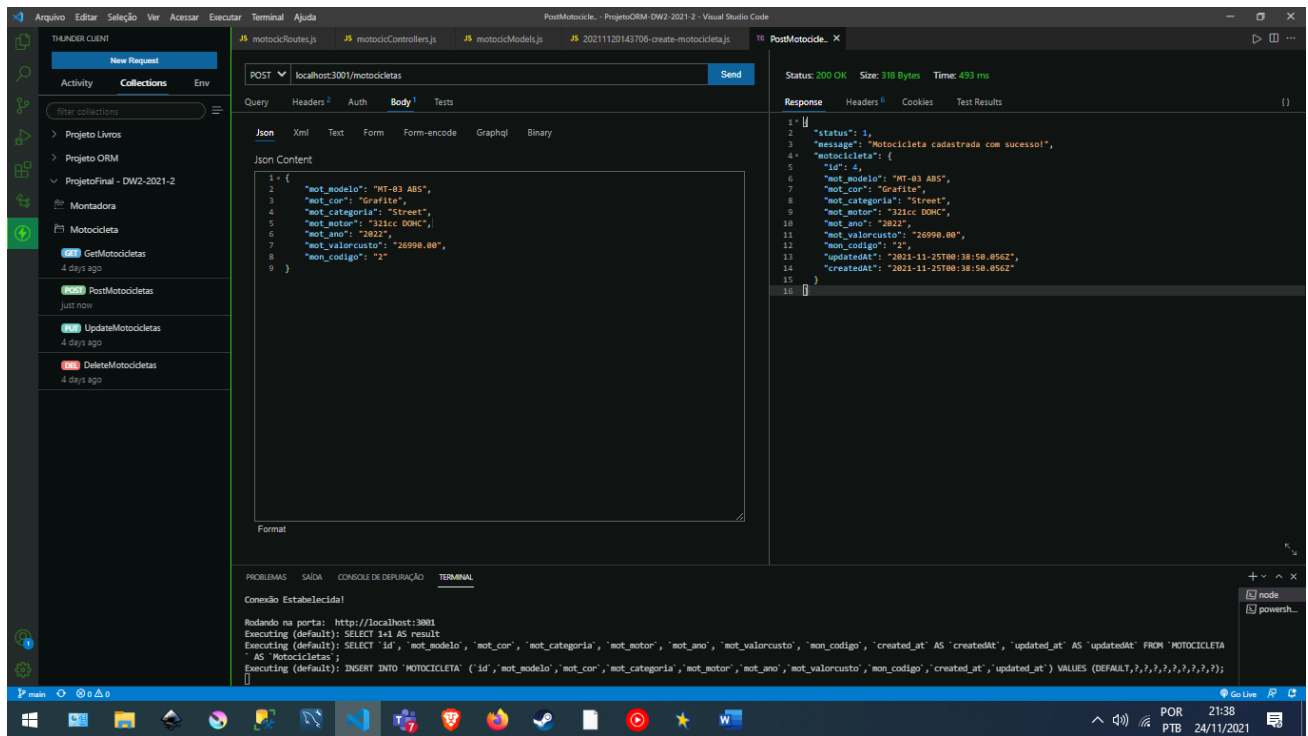
**Query:** GET localhost:3001/motocicletas

**Response:** Status: 200 OK, Size: 717 Bytes, Time: 119 ms

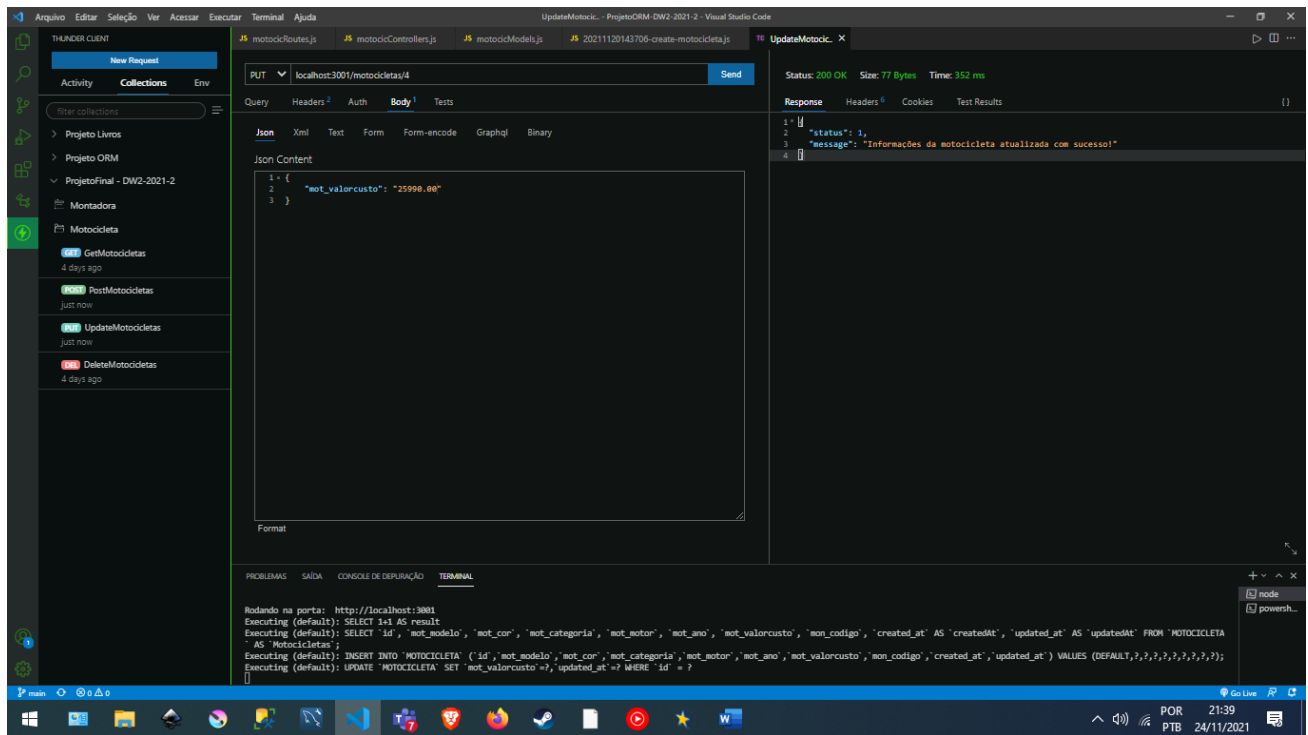
```
{
  "id": 1,
  "mot_modelo": "CB Twister",
  "mot_cor": "Verde",
  "mot_categoria": "Popular",
  "mot_motor": "150",
  "mot_ano": 2021,
  "mot_valorcusto": "15790.00",
  "mon_codigo": 4,
  "created_at": "2021-11-20T20:38:49.000Z",
  "updated_at": "2021-11-20T20:38:49.000Z"
},
{
  "id": 2,
  "mot_modelo": "Ducati 1260S",
  "mot_cor": "Branco",
  "mot_categoria": "Esportiva",
  "mot_motor": "1200",
  "mot_ano": 2021,
  "mot_valorcusto": "124990.00",
  "mon_codigo": 5,
  "created_at": "2021-11-20T20:47:32.000Z",
  "updated_at": "2021-11-20T20:47:32.000Z"
},
{
  "id": 3,
  "mot_modelo": "Honda 125",
  "mot_cor": "Azul",
  "mot_categoria": "Popular",
  "mot_motor": "125",
  "mot_ano": 2021,
  "mot_valorcusto": "68795.00",
  "mon_codigo": 2,
  "created_at": "2021-11-20T20:49:50.000Z",
  "updated_at": "2021-11-20T20:49:50.000Z"
}
```



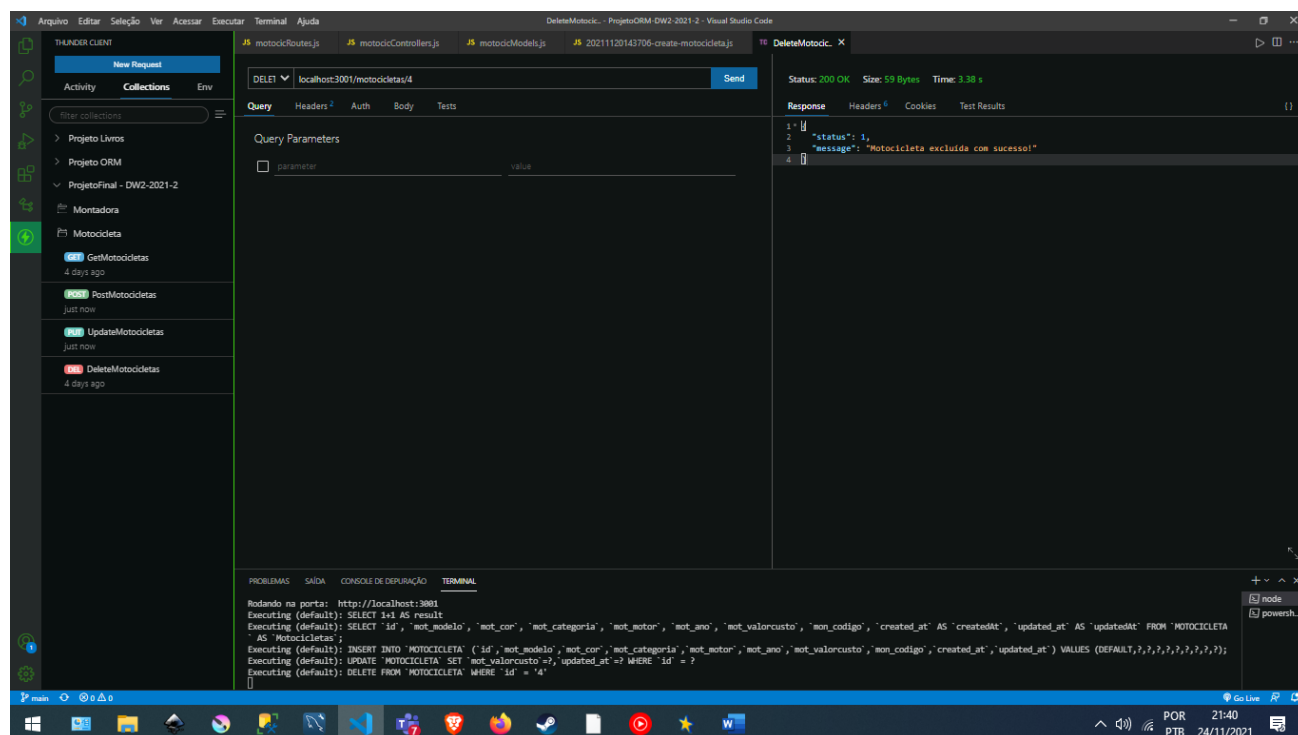
#### 4.6) Figura 17: Protocolo POST método store (adicionar registro).



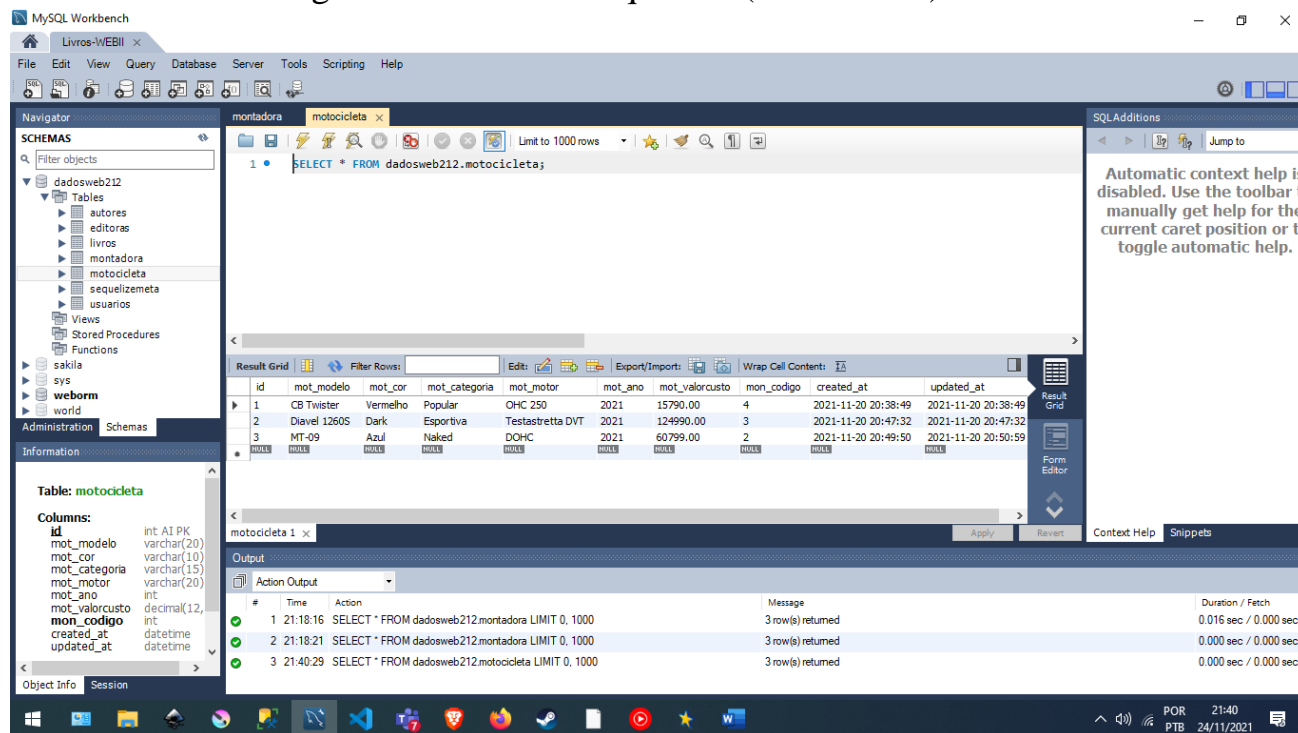
#### 4.7) Figura 18: Protocolo PUT método update (alterar registro)



#### 4.8) Figura 19: Protocolo DELETE método destroy (excluir registro).



#### 4.9) Figura 20: Imagem do SGDB utilizado, mostrando o banco de dados à esquerda aberto listando os registros da tabela em questão (motocicleta).



Obs: Alguns horários e datas podem estar desorganizados pelo fato de ter encontrado algum erro na hora de colar ao arquivo, pois já havia feito o projeto antes da aula do dia 24/11, portanto fui tirando as prints de uma vez e o upload após. Algumas imagens não possuem o destroy por opção.