

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

Data: 25/11/2021

Aluno: Fabricio Rangel de Sousa

Projeto: 32 – Proprietário x Veículos

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

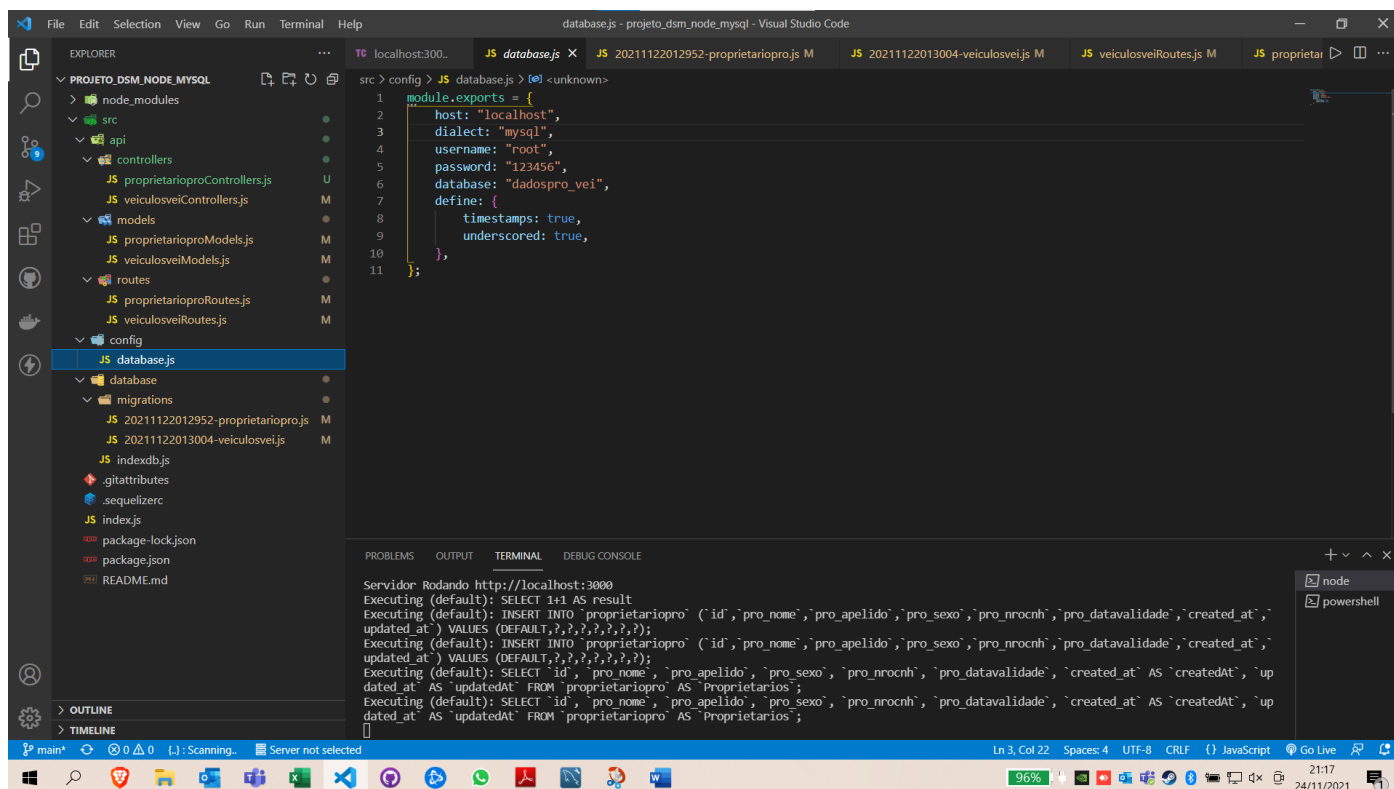
Link do github do projeto: https://github.com/fabriciorangel07/projeto_final_dsm_webII ORM

1) Figura 1: Estrutura das tabelas – ao lado

Especificação da Entidade – Tabela: PROPRIETARIO - PRO				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	pro_codigo		Chave primária da tabela
	varchar	pro_nome	20	Nome do proprietário
	varchar	pro_apelido	10	Apelido do proprietário
	char	pro_sexo	1	Sexo do proprietário
	varchar	pro_nrocnh	10	Número da CNH
	varchar	pro_datavalidade		Data de validade da CNH

Especificação da Entidade – Tabela: VEICULOS - VEI				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	vei_codigo		Chave primária da tabela
	varchar	vei_marca	20	Marca do veículo
	char	vei_modelo	20	Modelo do veículo
	varchar	vei_cor	15	Cor do veículo
	varchar	vei_anomodelo	10	Ano e modelo – Ex.: 2010/2011
FK	inteiro	pro_codigo		Código do Proprietário – 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**. É importante que o rodapé da tela esteja visível na imagem.



3) A partir de agora serão apresentados os print's 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 Proprietários – proprietáriospro

```

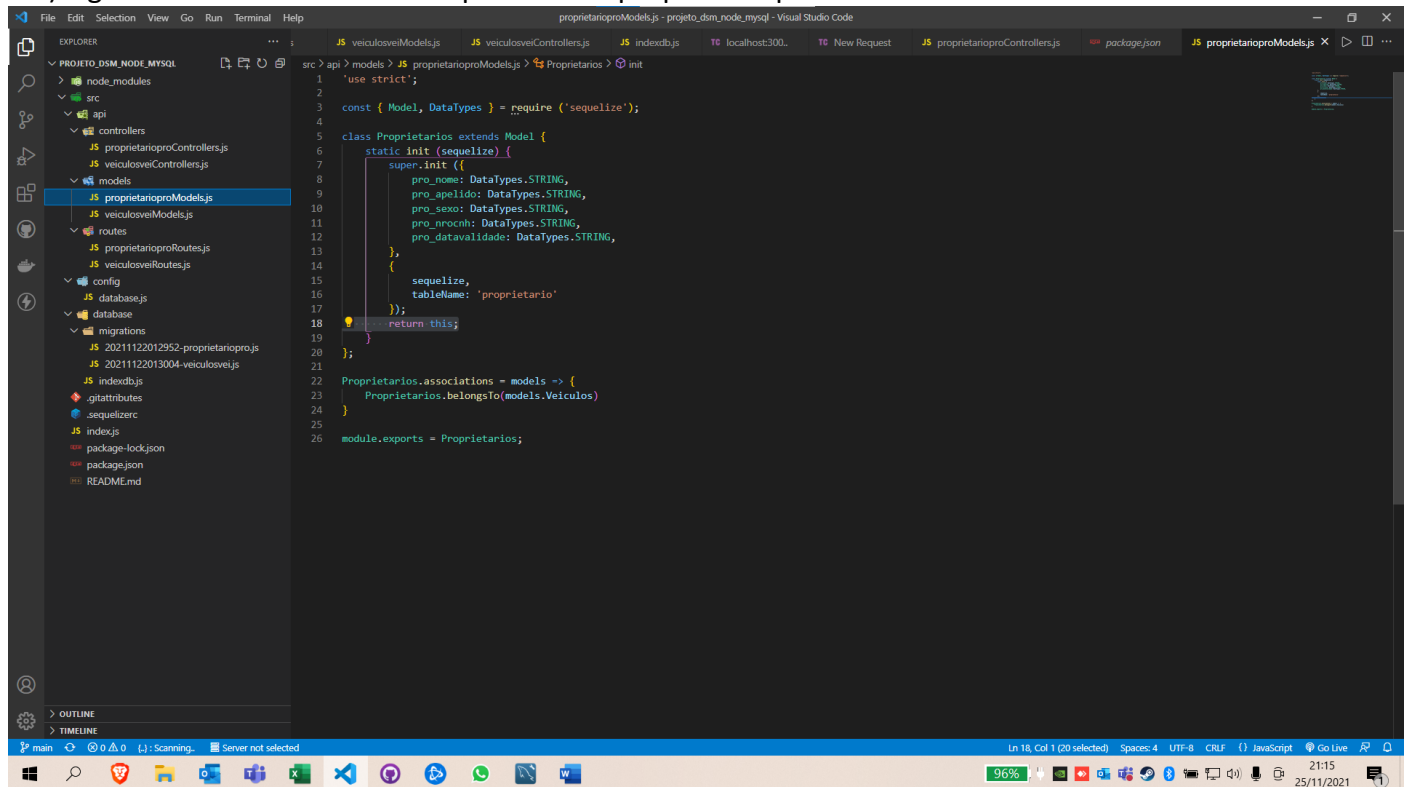
src > api > routes > JS proprietarioproRoutes.js > ...
1  const express = require('express');
2  const router = express.Router();
3
4  const proprietarioproControllers = require('../controllers/proprietarioproControllers');
5
6  router.get('/proprietario', proprietarioproControllers.index);
7
8  router.post('/proprietario',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

3.2) Figura 4: Controllers da tabela Proprietários – proprietáriospro

```

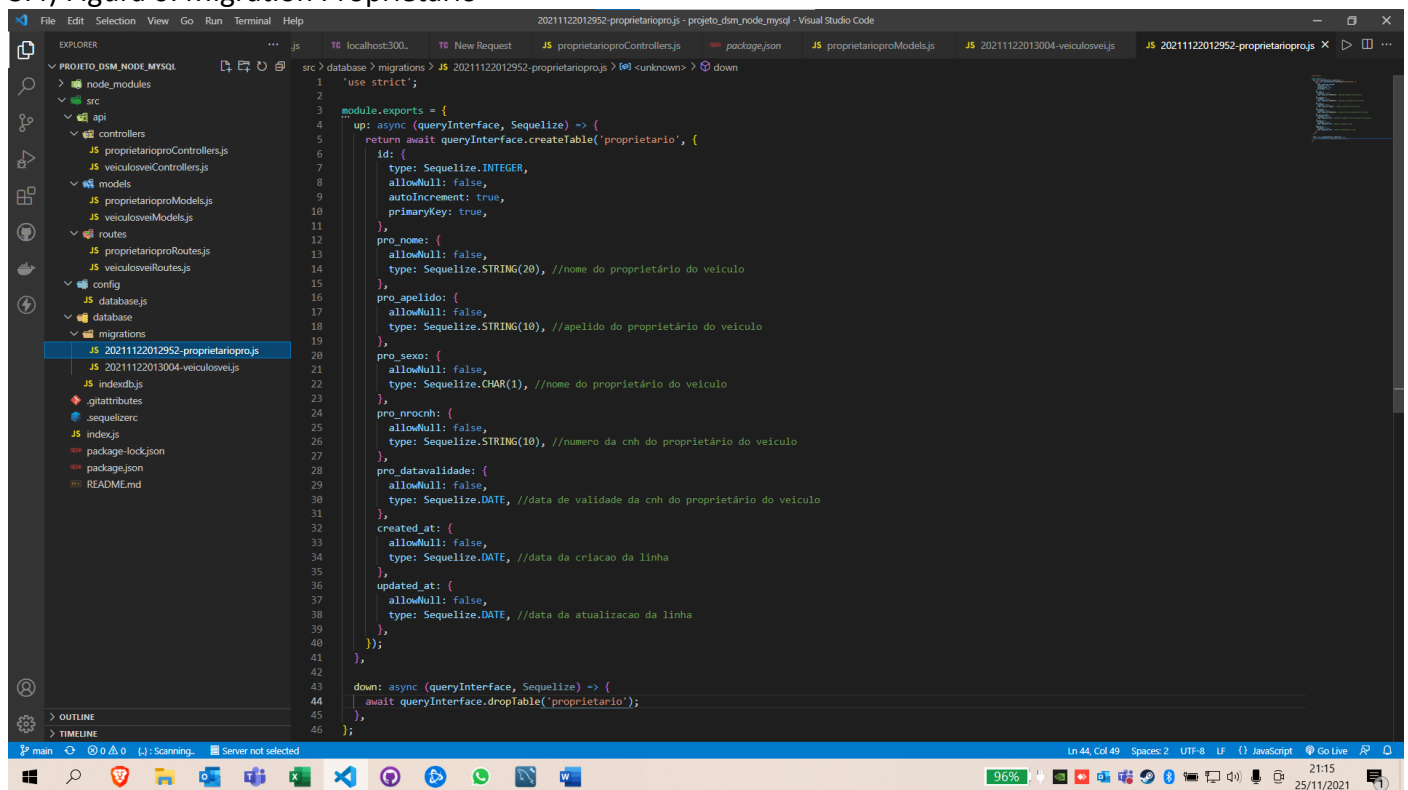
src > api > controllers > JS proprietarioproControllers.js > [unknown]
1  const Proprietarios = require('../models/proprietarioproModels');
2
3  module.exports = {
4    async index(req, res) {
5      const proprietarios = await Proprietarios.findAll();
6      return res.json(proprietarios);
7    },
8
9    async store(req, res) {
10     const { pro_nome, pro_apelido, pro_sexo, pro_nrocnh, pro_datavalidade } = req.body;
11     const proprietarios = await Proprietarios.create ({ pro_nome, pro_apelido, pro_sexo, pro_nrocnh, pro_datavalidade });
12     return res.status(200).send({
13       status: 1,
14       message: "Proprietário cadastrado com sucesso!",
15       proprietarios
16     });
17   },
18
19   async update(req, res) {
20     const { codigo } = req.params;
21     const { pro_nrocnh } = req.body;
22     await Proprietarios.update ({ pro_nrocnh }, {where: { id: codigo}});
23     return res.status(200).send({
24       status: 1,
25       message: "Proprietário atualizado com sucesso!",
26     });
27   },
28
29   async delete(req, res) {
30     const { codigo } = req.params;
31     await Proprietarios.destroy({where: { id: codigo }});
32     return res.status(200).send ({
33       status: 1,
34       message: "Proprietário removido com sucesso!",
35     });
36   },
37 }
  
```

3.3) Figura 5: Models da tabela Proprietários – proprietariopro



```
1 'use strict';
2
3 const { Model, DataTypes } = require('sequelize');
4
5 class Proprietarios extends Model {
6   static init(sequelize) {
7     super.init({
8       pro_nome: DataTypes.STRING,
9       pro_apellido: DataTypes.STRING,
10      pro_seso: DataTypes.STRING,
11      pro_nrocnh: DataTypes.STRING,
12      pro_datavalidade: DataTypes.STRING,
13    }, {
14      sequelize,
15      tableName: 'proprietario'
16    });
17    return this;
18  }
19
20  static associations() {
21    Proprietarios.associations = models => {
22      Proprietarios.belongsTo(models.Veiculos)
23    }
24  }
25
26  module.exports = Proprietarios;
27 }
```

3.4) Figura 6: Migration Proprietário

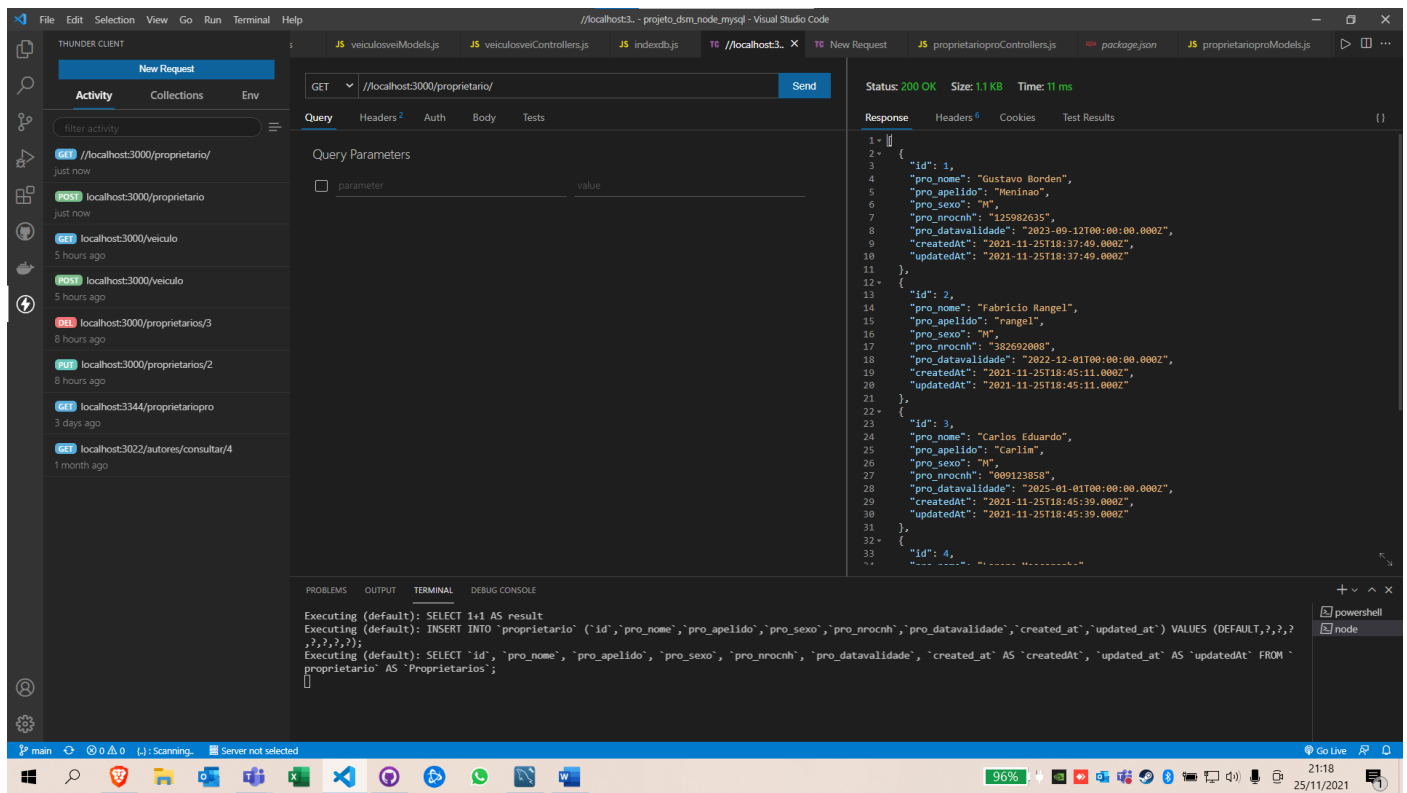


```
1 'use strict';
2
3 module.exports = {
4   up: async (queryInterface, Sequelize) => {
5     return await queryInterface.createTable('proprietario', {
6       id: {
7         type: Sequelize.INTEGER,
8         allowNull: false,
9         autoIncrement: true,
10        primaryKey: true,
11      },
12      pro_nome: {
13        allowNull: false,
14        type: Sequelize.STRING(20), //nome do proprietário do veículo
15      },
16      pro_apellido: {
17        allowNull: false,
18        type: Sequelize.STRING(10), //apellido do proprietário do veículo
19      },
20      pro_seso: {
21        allowNull: false,
22        type: Sequelize.CHAR(1), //nome do proprietário do veículo
23      },
24      pro_nrocnh: {
25        allowNull: false,
26        type: Sequelize.STRING(10), //numero da cnh do proprietário do veículo
27      },
28      pro_datavalidade: {
29        allowNull: false,
30        type: Sequelize.DATE, //data de validade da cnh do proprietário do veículo
31      },
32      created_at: {
33        allowNull: false,
34        type: Sequelize.DATE, //data da criação da linha
35      },
36      updated_at: {
37        allowNull: false,
38        type: Sequelize.DATE, //data da atualização da linha
39      },
40    });
41  },
42
43   down: async (queryInterface, Sequelize) => {
44     await queryInterface.dropTable('proprietario');
45   },
46 }
```

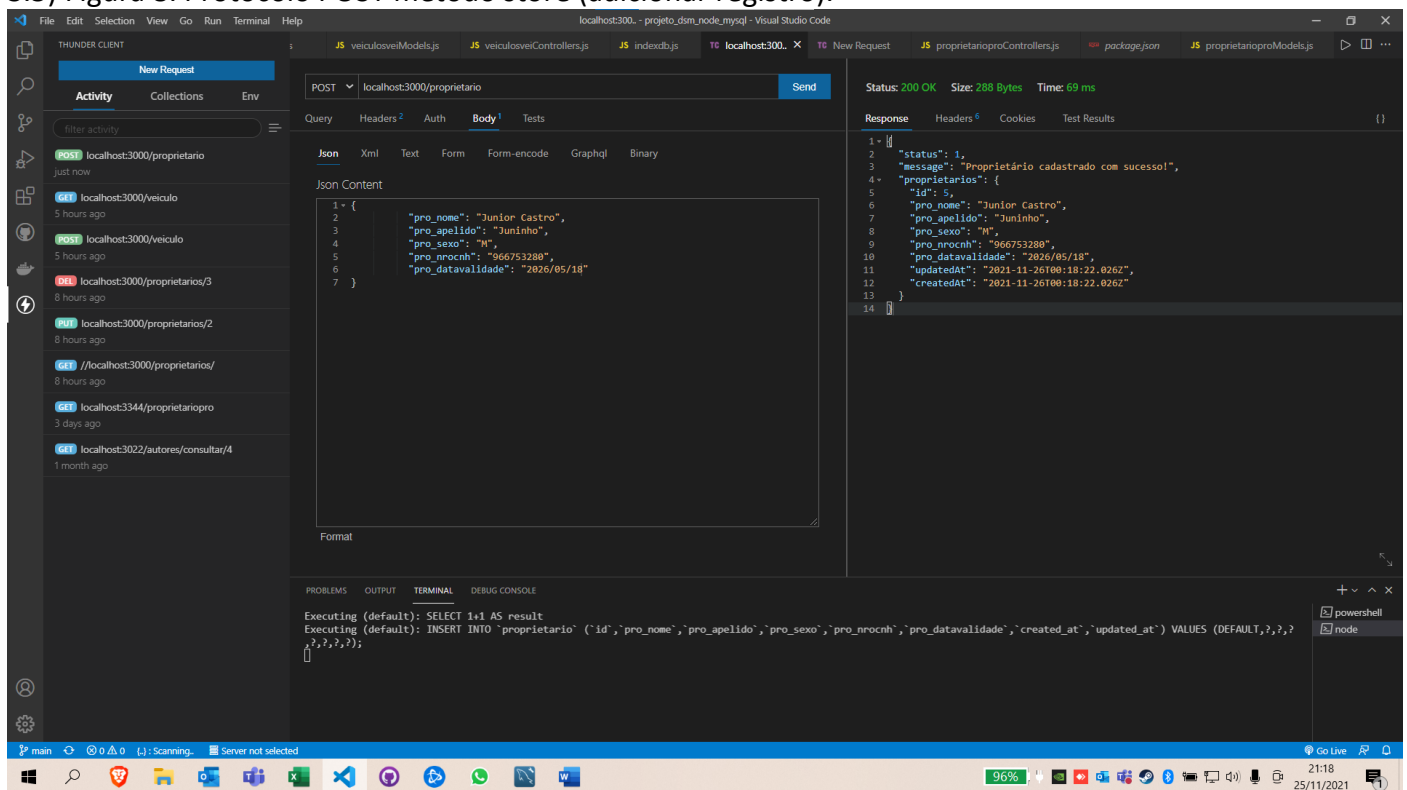
IMPORTANTE:

É necessário que as operações realizadas no backend sejam mostradas no terminal conforme pode ser vista 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).

The screenshot shows the Thunder Client interface with a PUT request to `http://localhost:3000/proprietario/2`. The request body is a JSON object: `{ "pro_nrocnh": "008652898" }`. The response is a 200 OK status with a JSON body: `{ "status": 1, "message": "Proprietário atualizado com sucesso!" }`. The terminal shows the execution of SQL queries for inserting, selecting, updating, and deleting records.

```
PUT http://localhost:3000/proprietario/2

Query Headers Auth Body Tests

Json Content

1 {
2   "pro_nrocnh": "008652898"
3 }

Format

Status: 200 OK Size: 62 Bytes Time: 18 ms

Response Headers Cookies Test Results

1 {
2   "status": 1,
3   "message": "Proprietário atualizado com sucesso!"
4 }
```

Problems Output Terminal Debug Console

```
Executing (default): SELECT 1+1 AS result
Executing (default): INSERT INTO `proprietario` (`id`,`pro_nome`,`pro_apelido`,`pro_sexo`,`pro_nrocnh`,`pro_datavalidade`,`created_at`,`updated_at`) VALUES (DEFAULT,?,?,?, ?,?,?,?);
Executing (default): SELECT `id`,`pro_nome`,`pro_apelido`,`pro_sexo`,`pro_nrocnh`,`pro_datavalidade`,`created_at` AS `createdAt`,`updated_at` AS `updatedAt` FROM `proprietario` AS `Proprietarios`;
Executing (default): UPDATE `proprietario` SET `pro_nrocnh`=?,`updated_at`=? WHERE `id` = ?
Executing (default): DELETE FROM `proprietario` WHERE `id` = '4'
Executing (default): UPDATE `proprietario` SET `pro_nrocnh`=?,`updated_at`=? WHERE `id` = ?
```

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

The screenshot shows the Thunder Client interface with a DELETE request to `http://localhost:3000/proprietario/4`. The response is a 200 OK status with a JSON body: `{ "status": 1, "message": "Proprietário removido com sucesso!" }`. The terminal shows the execution of SQL queries for inserting, selecting, updating, and deleting records.

```
DELETE http://localhost:3000/proprietario/4

Query Headers Auth Body Tests

Json Content

1 {
2 }

Format

Status: 200 OK Size: 60 Bytes Time: 14 ms

Response Headers Cookies Test Results

1 {
2   "status": 1,
3   "message": "Proprietário removido com sucesso!"
4 }
```

Problems Output Terminal Debug Console

```
Executing (default): SELECT 1+1 AS result
Executing (default): INSERT INTO `proprietario` (`id`,`pro_nome`,`pro_apelido`,`pro_sexo`,`pro_nrocnh`,`pro_datavalidade`,`created_at`,`updated_at`) VALUES (DEFAULT,?,?,?, ?,?,?,?);
Executing (default): SELECT `id`,`pro_nome`,`pro_apelido`,`pro_sexo`,`pro_nrocnh`,`pro_datavalidade`,`created_at` AS `createdAt`,`updated_at` AS `updatedAt` FROM `proprietario` AS `Proprietarios`;
Executing (default): UPDATE `proprietario` SET `pro_nrocnh`=?,`updated_at`=? WHERE `id` = ?
Executing (default): DELETE FROM `proprietario` WHERE `id` = '4'
```

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

The screenshot displays the MySQL Workbench interface. On the left, the 'SCHEMAS' pane shows a tree view of the database structure, including tables like 'proprietario' and 'veiculo'. The 'Query' pane in the center contains the SQL statement: `select * from proprietario`. Below the query, the 'Result Grid' shows the data from the 'proprietario' table. The table has columns: id, pro_nome, pro_apellido, pro_sexo, pro_nrocnh, pro_datavalidade, created_at, and updated_at. The data is as follows:

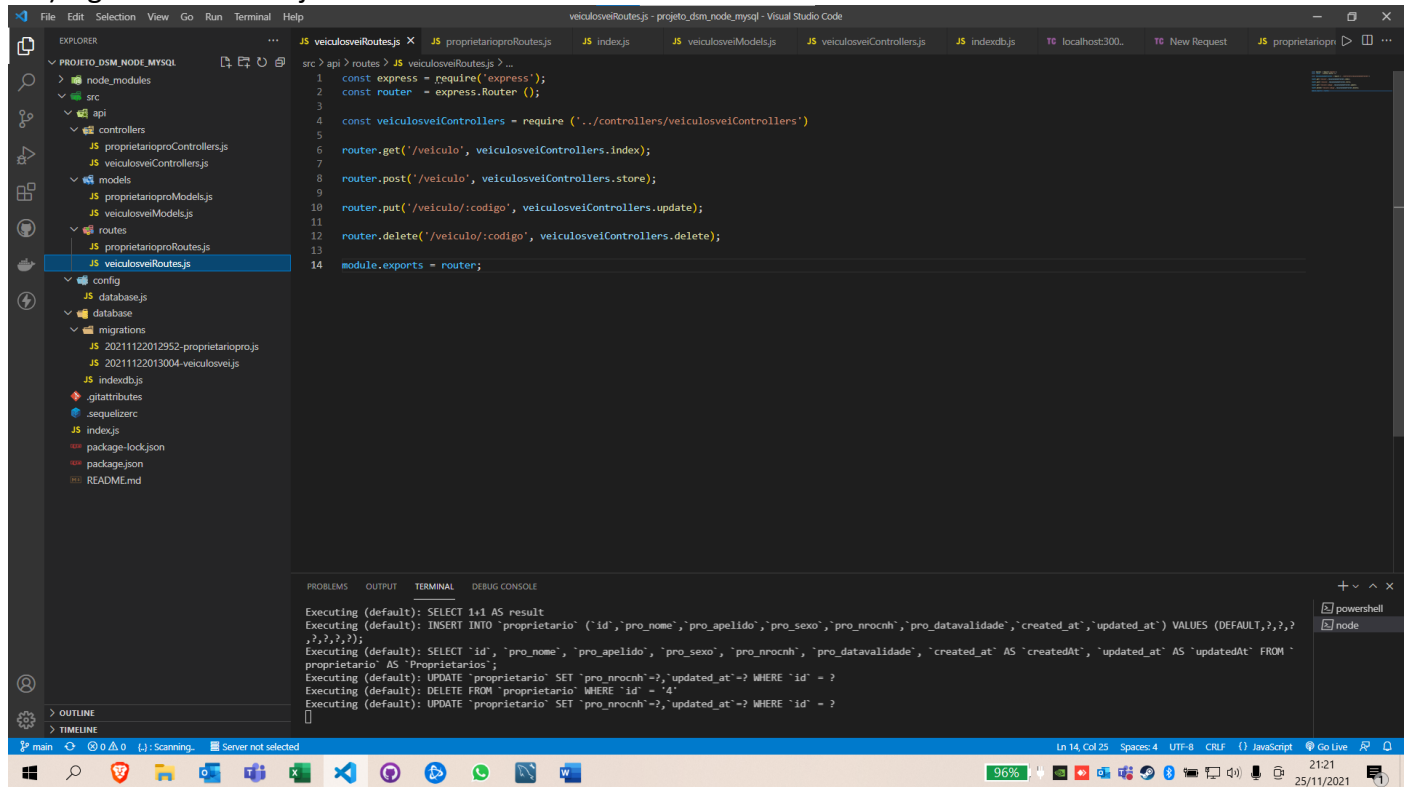
id	pro_nome	pro_apellido	pro_sexo	pro_nrocnh	pro_datavalidade	created_at	updated_at
1	Gustavo Borden	Meninao	M	125982635	2023-09-12 00:00:00	2021-11-25 18:37:49	2021-11-25 18:37:49
2	Fabricao Rangel	rangel	M	008652898	2022-12-01 00:00:00	2021-11-25 18:45:11	2021-11-26 00:20:42
3	Carlos Eduardo	Carlim	M	009123858	2025-01-01 00:00:00	2021-11-25 18:45:39	2021-11-25 18:45:39
5	Junior Castro	Juninho	M	966753280	2026-05-18 00:00:00	2021-11-26 00:18:22	2021-11-26 00:18:22

At the bottom, the 'Output' pane shows the execution log of the query, indicating that 1 row was returned for the first query and 4 rows were returned for the subsequent queries. The system tray at the bottom shows the date and time as 21:21 on 25/11/2021.

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

- routes
- controllers
- models
- migrations
- workbanch

4.1) Figura 12: Routes.js da tabela Veículos – veículo

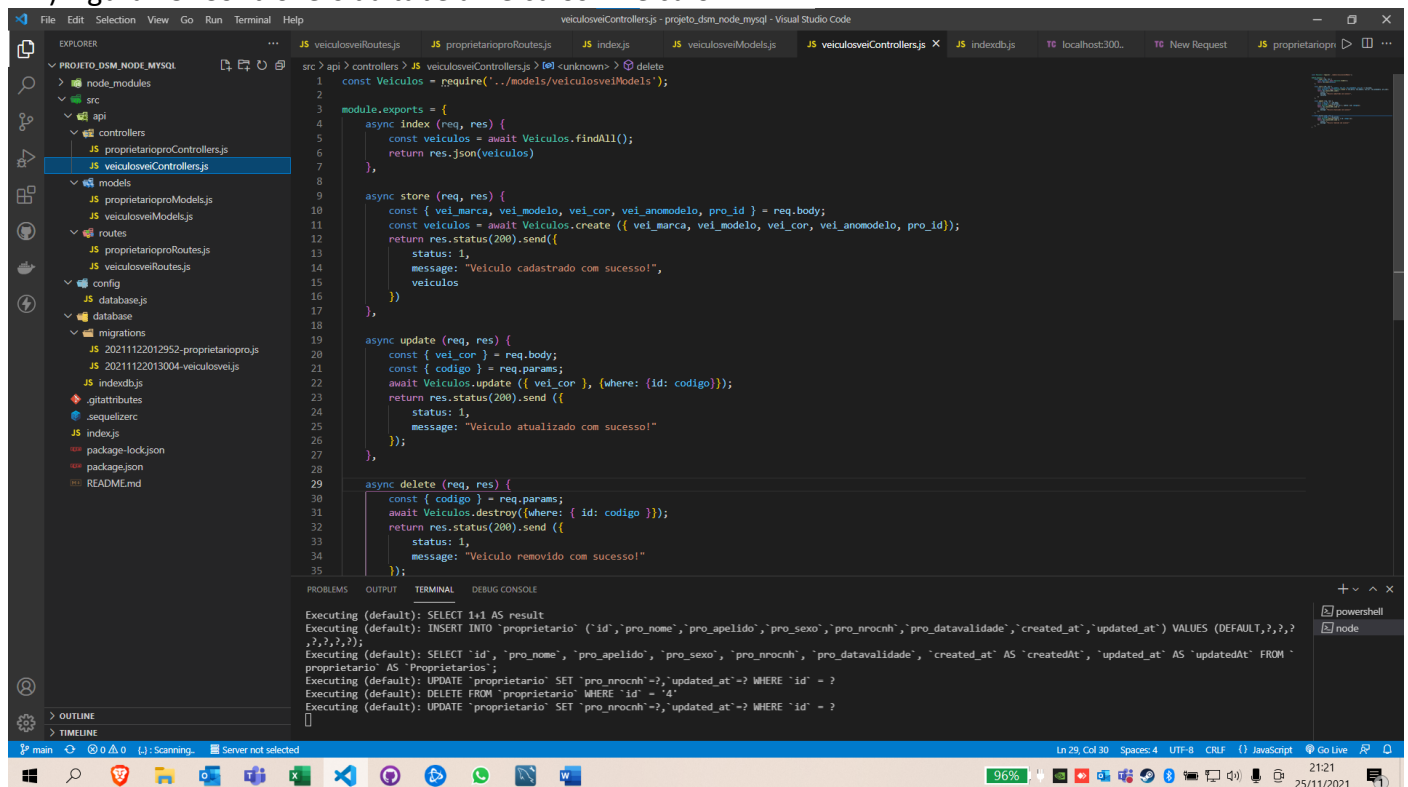


```
1 const express = require('express');
2 const router = express.Router();
3
4 const veiculosveicontrollers = require('../controllers/veiculosveicontrollers')
5
6 router.get('/veiculo', veiculosveicontrollers.index);
7
8 router.post('/veiculo', veiculosveicontrollers.store);
9
10 router.put('/veiculo/:codigo',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

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Executing (default): SELECT 1+1 AS result
Executing (default): INSERT INTO "proprietario" ("id","pro_nome","pro_apelido","pro_sexo","pro_nrocnh","pro_datavalidade","created_at","updated_at") VALUES (DEFAULT,?,?,?,?,?)
Executing (default): SELECT "id", "pro_nome", "pro_apelido", "pro_sexo", "pro_nrocnh", "pro_datavalidade", "created_at" AS "createdAt", "updated_at" AS "updatedAt" FROM "proprietario" AS "Proprietarios";
Executing (default): UPDATE "proprietario" SET "pro_nrocnh"=?,"updated_at"=? WHERE "id" = ?
Executing (default): DELETE FROM "proprietario" WHERE "id" = '4'
Executing (default): UPDATE "proprietario" SET "pro_nrocnh"=?,"updated_at"=? WHERE "id" = ?

4.2) Figura 13: Controllers da tabela Veículos – veículo

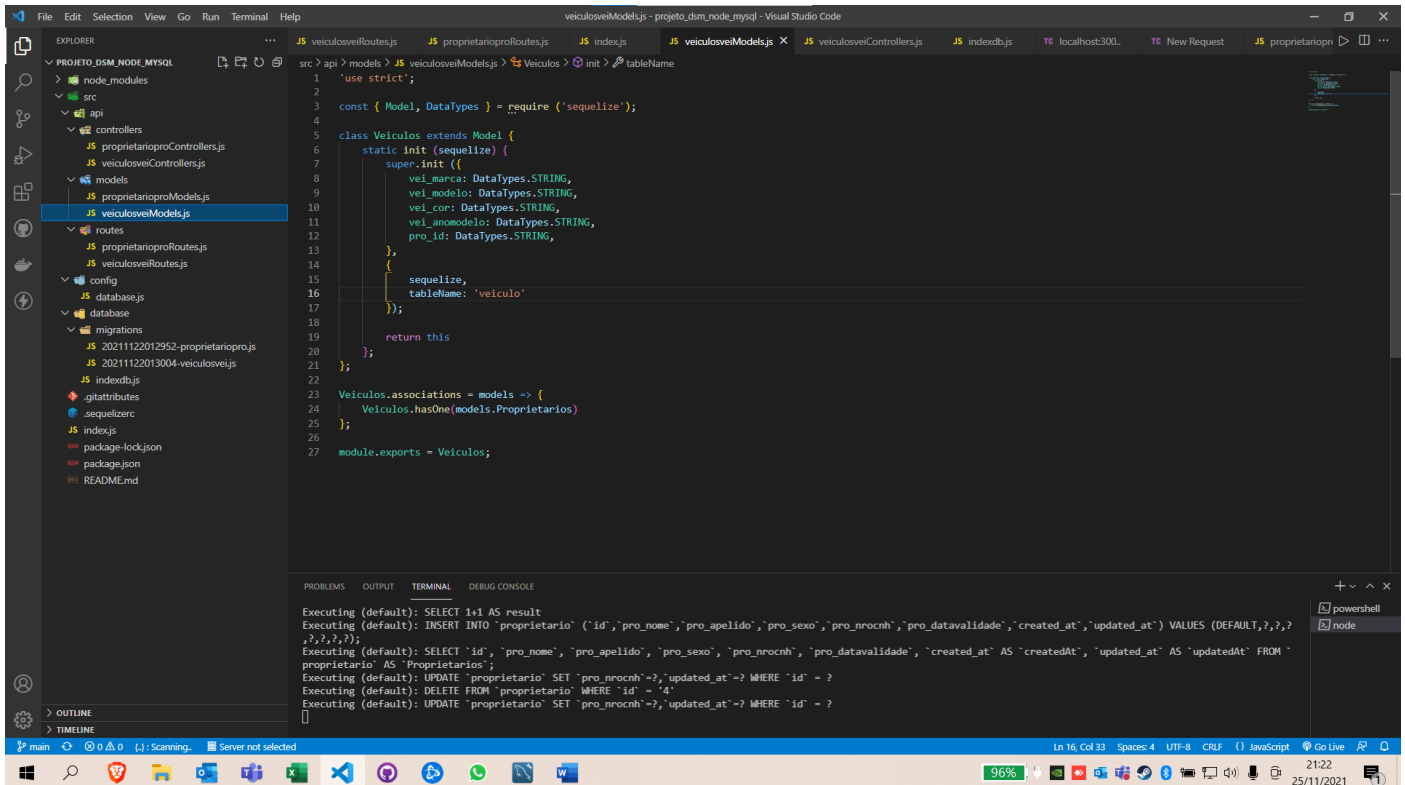


```
1 const Veiculos = require('../models/veiculosveimodels');
2
3 module.exports = {
4   async index (req, res) {
5     const veiculos = await Veiculos.findAll();
6     return res.json(veiculos)
7   },
8
9   async store (req, res) {
10     const { vei_marca, vei_modelo, vei_cor, vei_anomodelo, pro_id } = req.body;
11     const veiculos = await Veiculos.create ({ vei_marca, vei_modelo, vei_cor, vei_anomodelo, pro_id });
12     return res.status(200).send({
13       status: 1,
14       message: "Veículo cadastrado com sucesso!",
15       veiculos
16     })
17   },
18
19   async update (req, res) {
20     const { vei_cor } = req.body;
21     const { codigo } = req.params;
22     await Veiculos.update ({ vei_cor }, { where: { id: codigo } });
23     return res.status(200).send ({
24       status: 1,
25       message: "Veículo atualizado com sucesso!"
26     });
27   },
28
29   async delete (req, res) {
30     const { codigo } = req.params;
31     await Veiculos.destroy({ where: { id: codigo } });
32     return res.status(200).send ({
33       status: 1,
34       message: "Veículo removido com sucesso!"
35     });
36   }
37 };
38
```

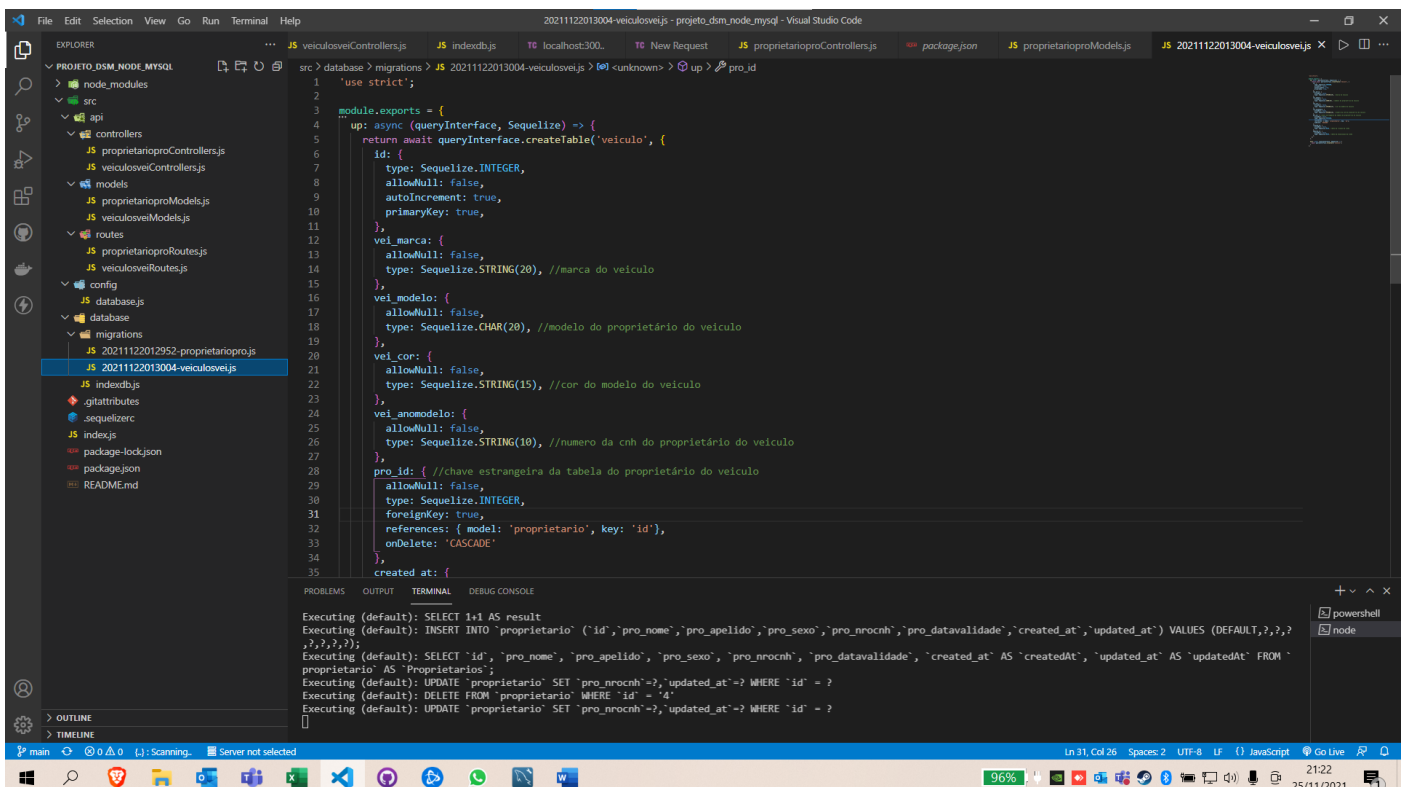
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

Executing (default): SELECT 1+1 AS result
Executing (default): INSERT INTO "proprietario" ("id","pro_nome","pro_apelido","pro_sexo","pro_nrocnh","pro_datavalidade","created_at","updated_at") VALUES (DEFAULT,?,?,?,?,?)
Executing (default): SELECT "id", "pro_nome", "pro_apelido", "pro_sexo", "pro_nrocnh", "pro_datavalidade", "created_at" AS "createdAt", "updated_at" AS "updatedAt" FROM "proprietario" AS "Proprietarios";
Executing (default): UPDATE "proprietario" SET "pro_nrocnh"=?,"updated_at"=? WHERE "id" = ?
Executing (default): DELETE FROM "proprietario" WHERE "id" = '4'
Executing (default): UPDATE "proprietario" SET "pro_nrocnh"=?,"updated_at"=? WHERE "id" = ?

4.3) Figura 14: Models da tabela Veículos – veículo



4.4) Figura 15: Migration Veiculos



IMPORTANTE:

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

4.5) Figura 16: Protocolo GET método index.

The screenshot shows the Thunder Client interface in Visual Studio Code. A GET request is sent to `localhost:3000/veiculo`. The response is a JSON array of three vehicle records, each with fields like `id`, `vei_marca`, `vei_modelo`, `vei_cor`, `vei_anomodelo`, `pro_id`, `createdAt`, and `updatedAt`.

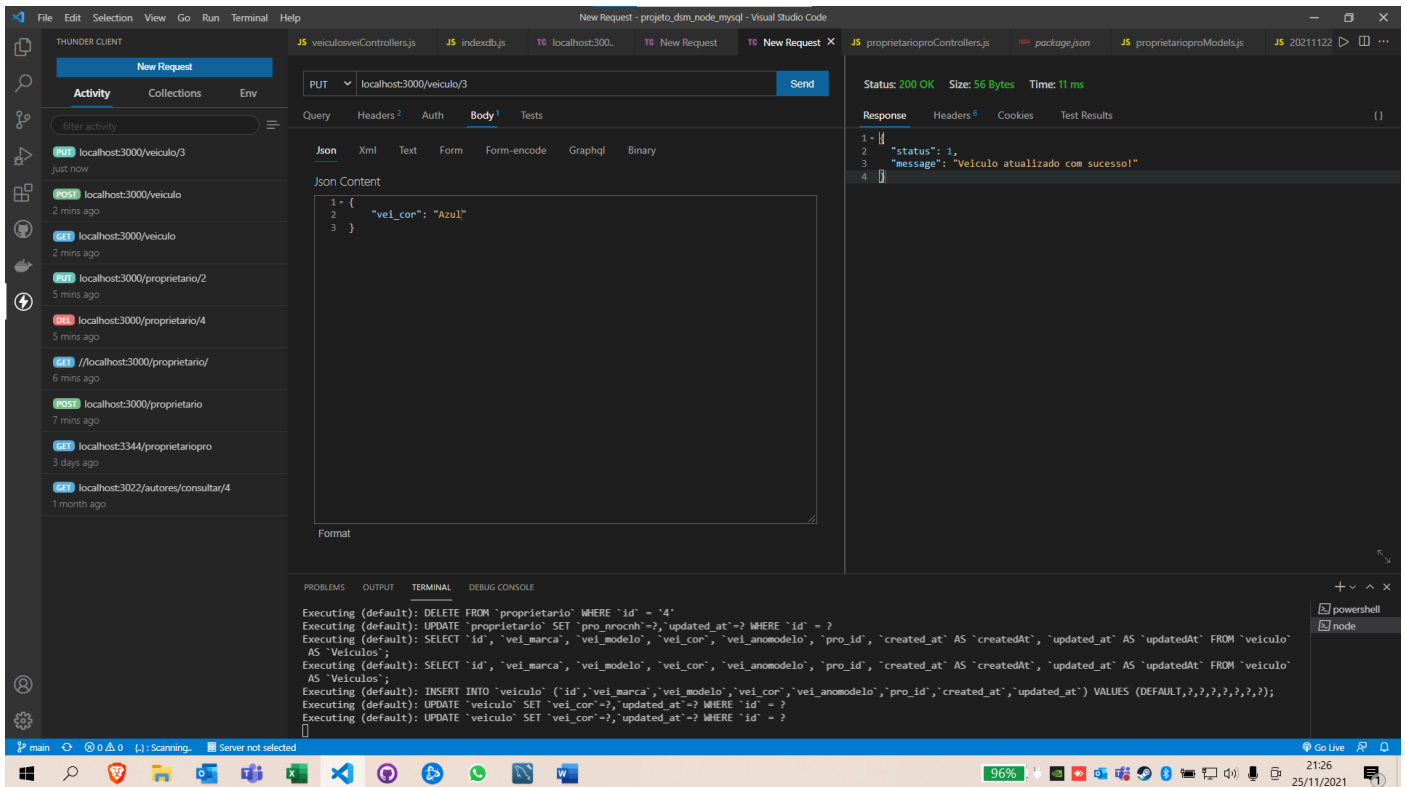
```
1 {
2   {
3     "id": 1,
4     "vei_marca": "Ford",
5     "vei_modelo": "Mustang",
6     "vei_cor": "Vermelho",
7     "vei_anomodelo": "2018/2019",
8     "pro_id": 1,
9     "createdAt": "2021-11-25T18:41:57.000Z",
10    "updatedAt": "2021-11-25T18:41:57.000Z"
11  },
12  {
13    "id": 2,
14    "vei_marca": "Fiat",
15    "vei_modelo": "Touro",
16    "vei_cor": "Prata",
17    "vei_anomodelo": "2020/2020",
18    "pro_id": 3,
19    "createdAt": "2021-11-25T18:46:48.000Z",
20    "updatedAt": "2021-11-25T18:46:48.000Z"
21  },
22  {
23    "id": 3,
24    "vei_marca": "Porsche",
25    "vei_modelo": "910 Cayman",
26    "vei_cor": "Preto",
27    "vei_anomodelo": "2020/2021",
28    "pro_id": 2,
29    "createdAt": "2021-11-25T18:47:31.000Z",
30    "updatedAt": "2021-11-25T18:47:31.000Z"
31  }
32 }
```

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

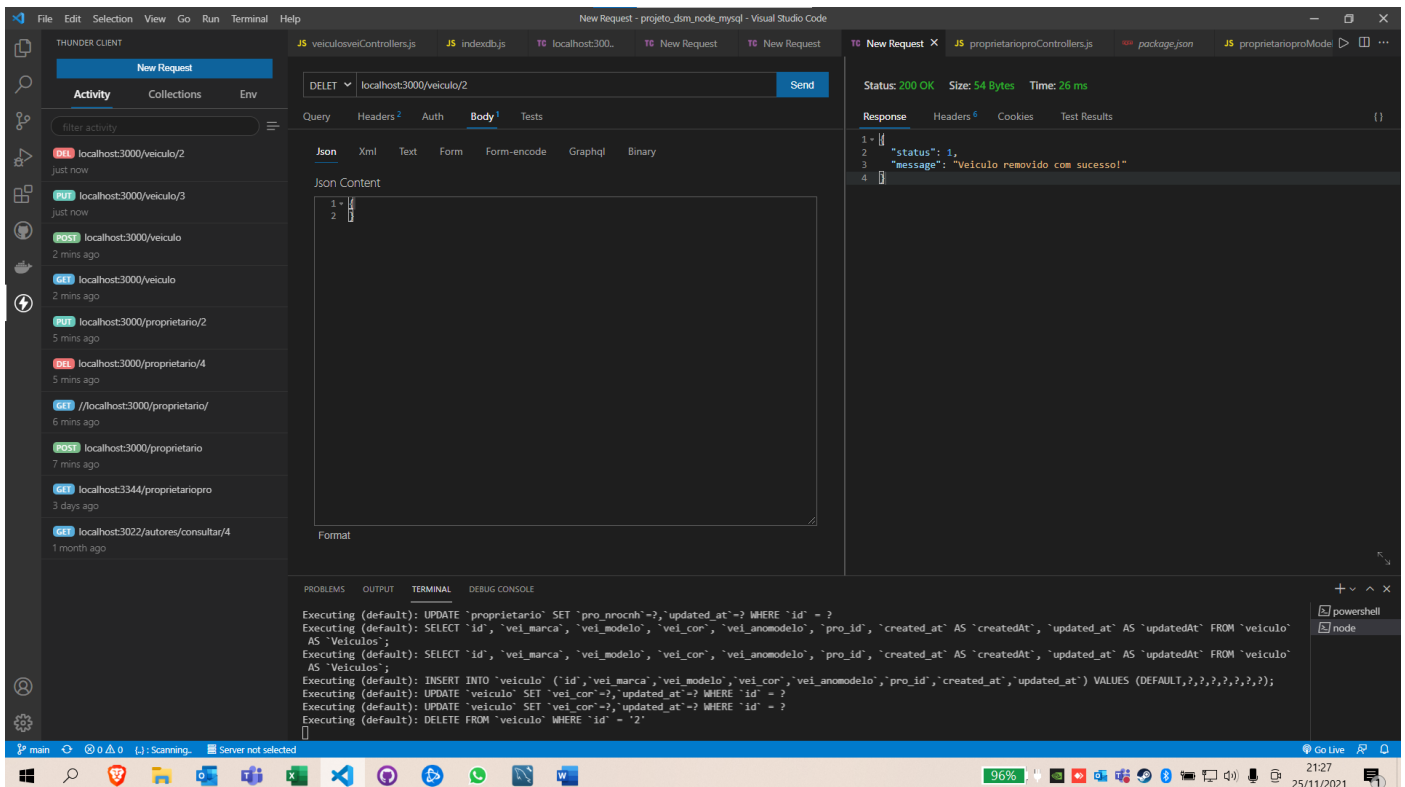
The screenshot shows the Thunder Client interface in Visual Studio Code. A POST request is sent to `localhost:3000/veiculo` with a JSON body containing vehicle details. The response is a JSON object with `status`, `message`, and the created vehicle record.

```
1 {
2   "status": 1,
3   "message": "Veiculo cadastrado com sucesso!",
4   "veiculos": {
5     "id": 4,
6     "vei_marca": "Jaguar",
7     "vei_modelo": "A210",
8     "vei_cor": "Branco",
9     "vei_anomodelo": "2018/2018",
10    "pro_id": "2",
11    "updatedAt": "2021-11-26T00:23:40.106Z",
12    "createdAt": "2021-11-26T00:23:40.106Z"
13  }
14 }
```

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 (Veiculo).

The screenshot displays the MySQL Workbench interface. On the left, the 'Schemas' tree shows the 'veiculo' table selected. The main editor shows a query: `select * from veiculo`. The results are displayed in a grid with the following data:

id	vei_marca	vei_modelo	vei_cor	vei_anomodelo	pro_id	created_at	updated_at
1	Ford	Mustang	Vermelho	2018/2019	1	2021-11-25 18:41:57	2021-11-25 18:41:57
3	Porsche	910 Cayman	Azul	2020/2021	2	2021-11-25 18:47:31	2021-11-26 00:25:58
4	Jaguar	A210	Branco	2018/2018	2	2021-11-26 00:23:40	2021-11-26 00:23:40

The bottom panel shows the 'Table: veiculo' structure with columns: `id` (int AI PK), `vei_marca` (varchar(20)), `vei_modelo` (char(20)), `vei_cor` (varchar(1)), `vei_anomodelo` (int), and `pro_id` (int). The 'Action Output' log shows the execution of the query, indicating that 4 rows were returned for the first three queries and 3 rows for the last one.