

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

Data: 24/11/2021

Aluno: Tatiana Hitomi Miyazaki

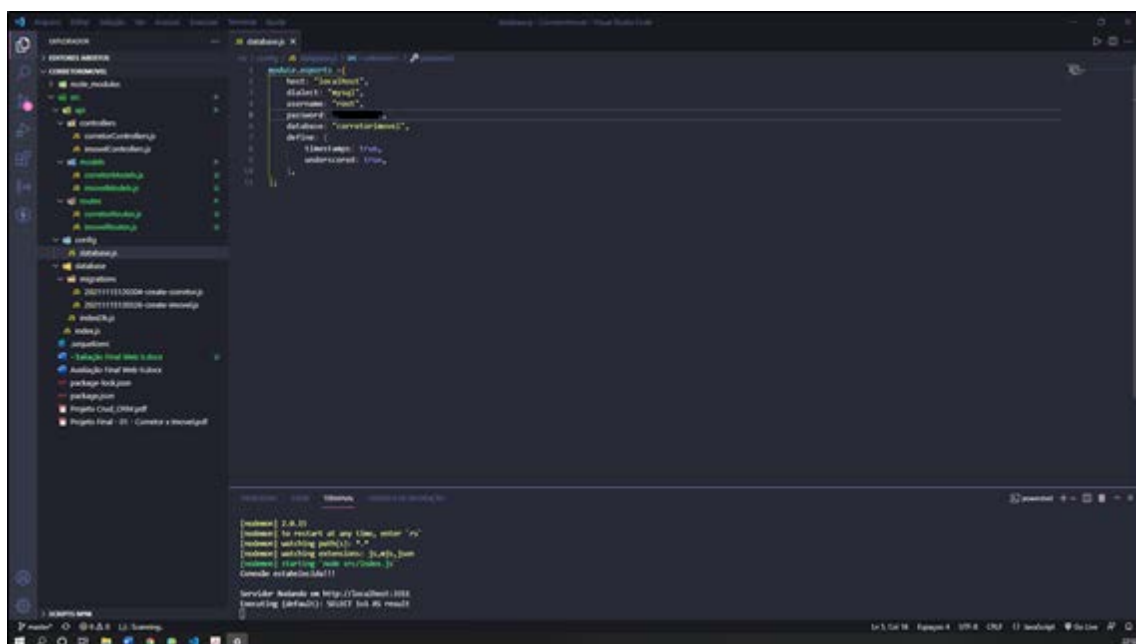
Projeto: 01 – Corretor x Imóvel

1) Figura 1: Estrutura das tabelas

Especificação da Entidade – Tabela: CARRETOR - CRT				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	crt_codigo		Chave primária da tabela
	varchar	crt_nome	10	Nome do corretor de imóveis
	varchar	crt_telefone	15	Número do telefone
	varchar	crt_crea	10	Número de registro no CREA

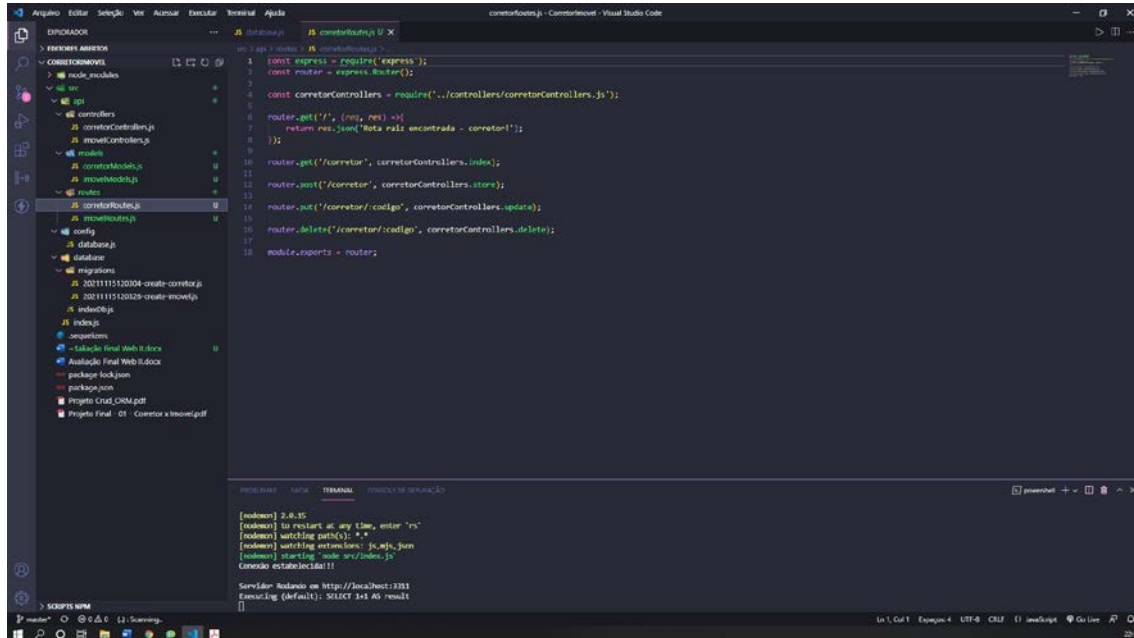
Especificação da Entidade – Tabela: IMOVEL - IMO				
#	Tipo	Nome	<->	Descrição do campo
PK	inteiro	imo_codigo		Chave primária da tabela
	varchar	imo_tipo	15	Tipo do imóvel – casa, apartamento, sítio, fazenda...
	varchar	imo_cidade	20	Nome da cidade
	inteiro	imo_area		Área em metros quadrados ou alqueires
	inteiro	imo_comodos		Número de comodoss do imóvel
FK	inteiro	crt_codigo		Código do corretor – 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.



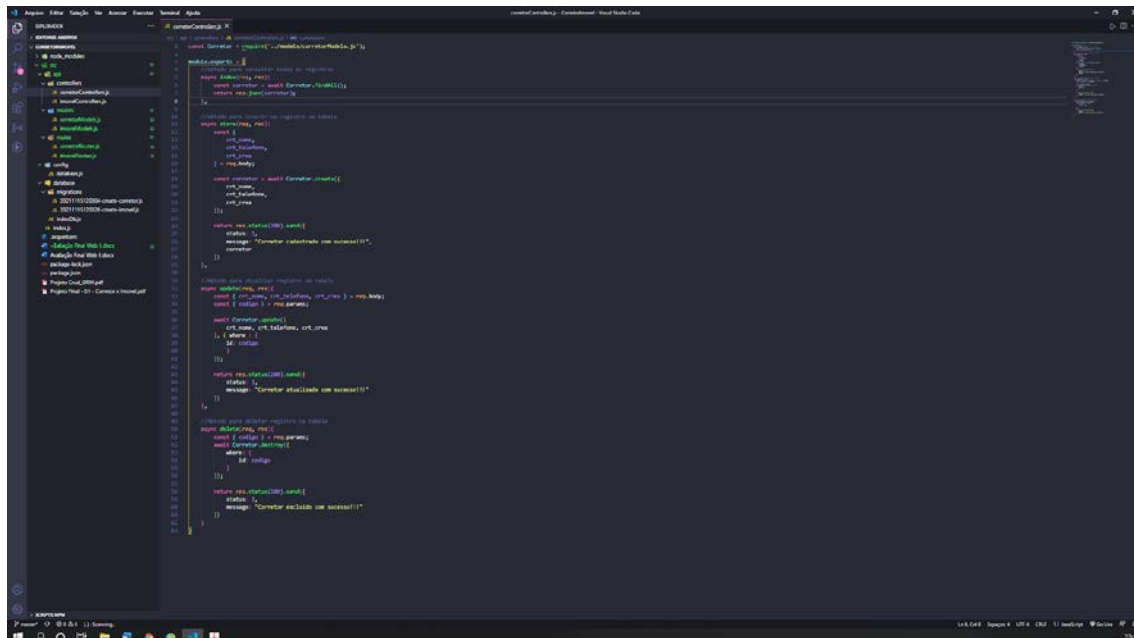
3) A partir de agora serão apresentados os prints sequenciais da primeira tabela informada no documento da tarefa. Corretor – crt

3.1) Figura 3: Routes.js da tabela corretor – crt



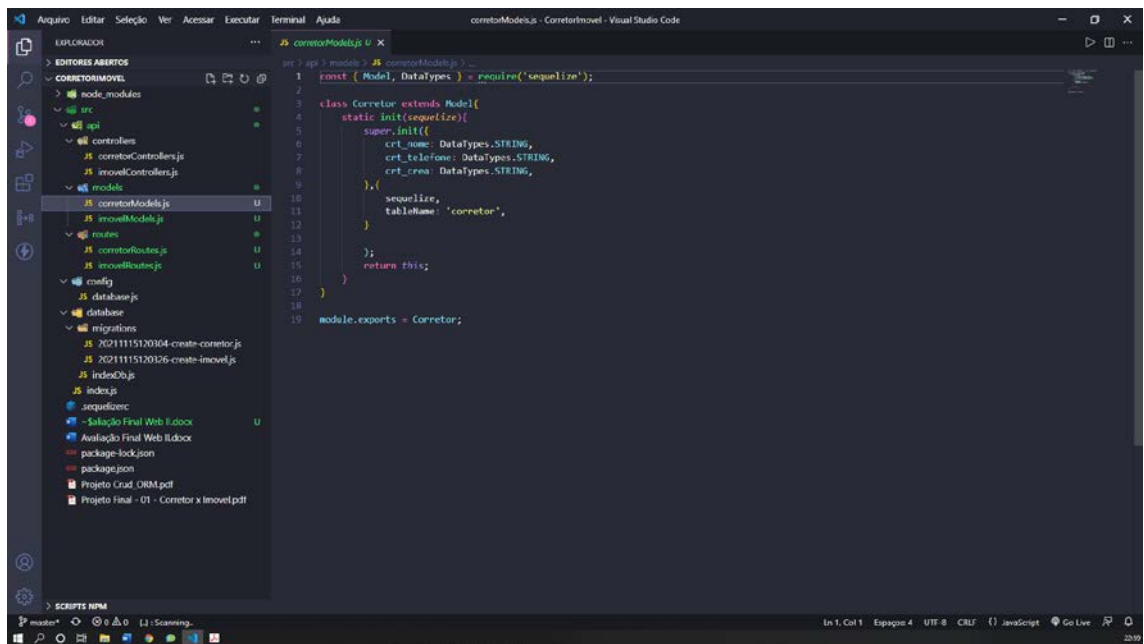
```
1 const express = require('express');
2 const router = express.Router();
3
4 const corretorControllers = require('../controllers/corretorControllers.js');
5
6 router.get('/', (req, res) => {
7   return res.json({nota: 'Nota não encontrada - corretor!'});
8 });
9
10 router.get('/corretor', corretorControllers.index);
11
12 router.post('/corretor', corretorControllers.store);
13
14 router.put('/corretor/:codigo', corretorControllers.update);
15
16 router.delete('/corretor/:codigo', corretorControllers.delete);
17
18 module.exports = router;
```

3.2) Figura 4: Controllers da tabela corretor – crt

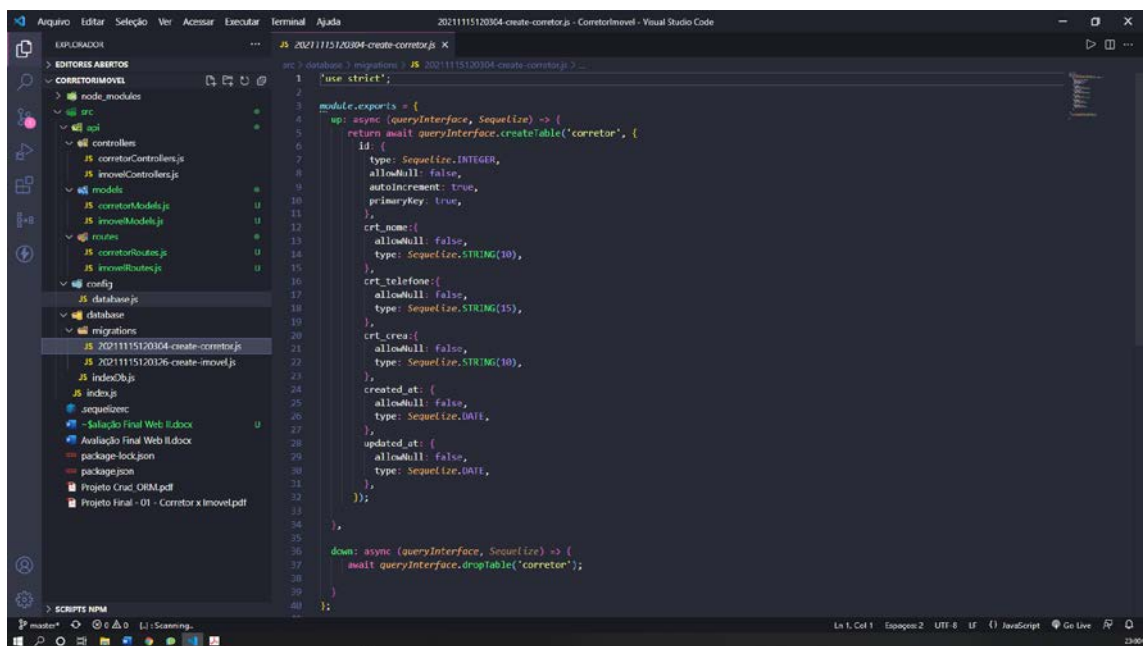


```
1 const { Router } = require('express');
2 const { Model } = require('mongoose');
3
4 const corretorModel = require('../models/corretorModel.js');
5
6 const corretorController = Router();
7
8 // Index
9 corretorController.get('/', async (req, res) => {
10   const corretor = await corretorModel.find().exec();
11   return res.json(corretor);
12 });
13
14 // Store
15 corretorController.post('/', async (req, res) => {
16   const { nome, crt_codigo, crt_titulo, crt_preco } = req.body;
17   const corretor = new corretorModel({ nome, crt_codigo, crt_titulo, crt_preco });
18   await corretor.save();
19   return res.status(201).json({ message: 'Corretor cadastrado com sucesso!!' });
20 });
21
22 // Update
23 corretorController.put('/:codigo', async (req, res) => {
24   const { crt_codigo, crt_titulo, crt_preco } = req.body;
25   const corretor = await corretorModel.findOneAndUpdate({ crt_codigo: req.params.codigo }, { crt_codigo, crt_titulo, crt_preco }, { new: true }).exec();
26   return res.status(200).json({ message: 'Corretor atualizado com sucesso!!' });
27 });
28
29 // Delete
30 corretorController.delete('/:codigo', async (req, res) => {
31   const { crt_codigo } = req.params;
32   await corretorModel.findOneAndDelete({ crt_codigo: crt_codigo }).exec();
33   return res.status(200).json({ message: 'Corretor excluído com sucesso!!' });
34 });
```

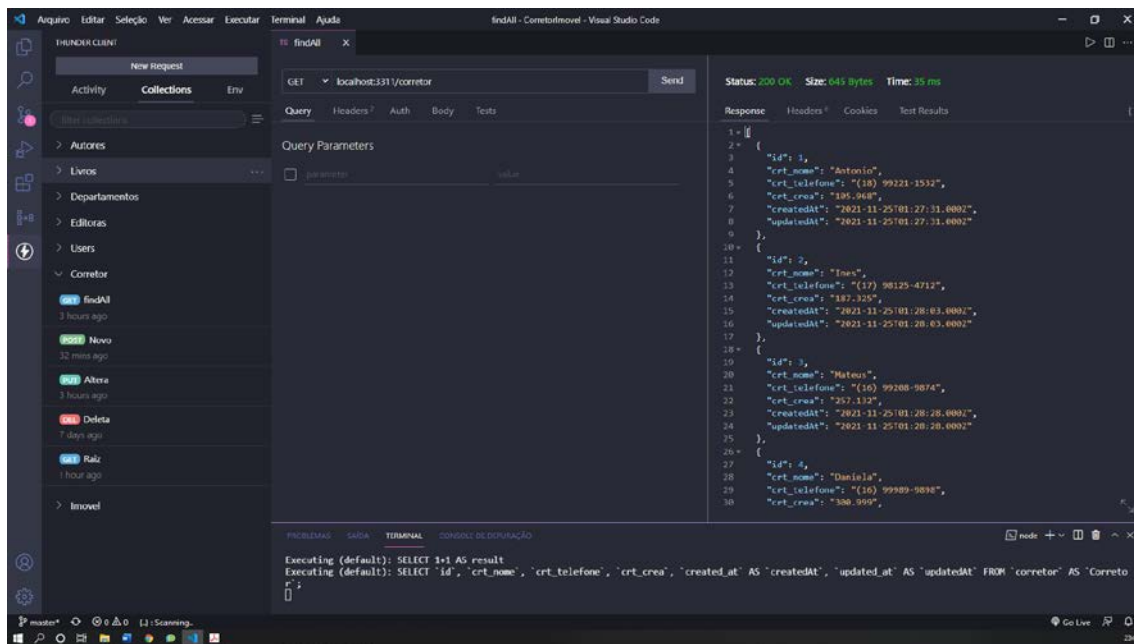
3.3) Figura 5: Models da tabela corretor – crt



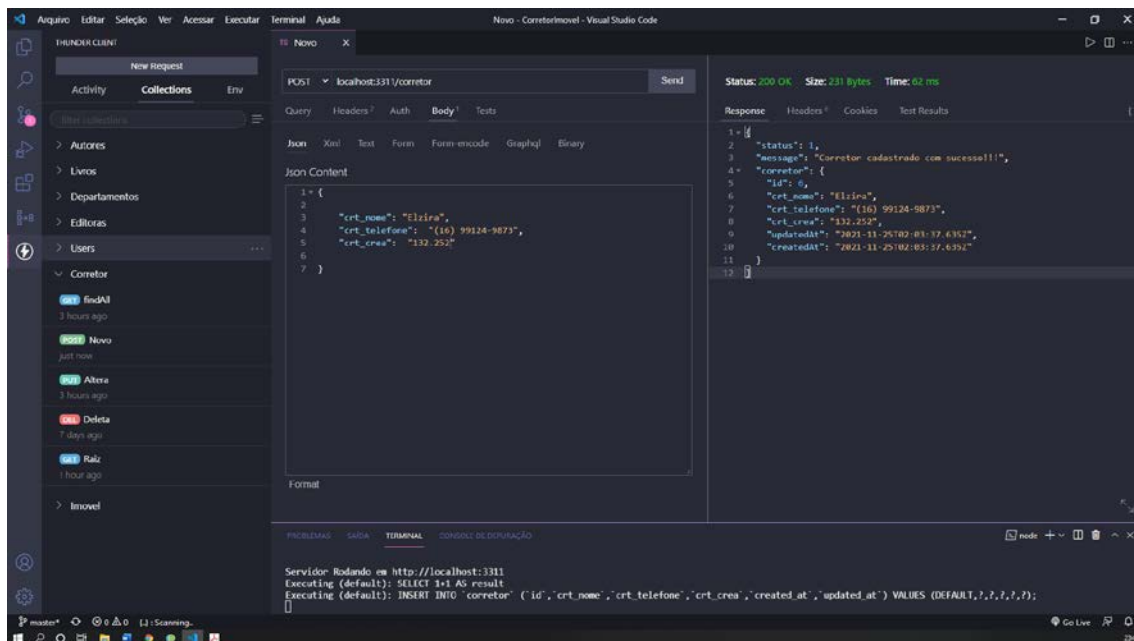
3.4) Figura 6: Migration corretor



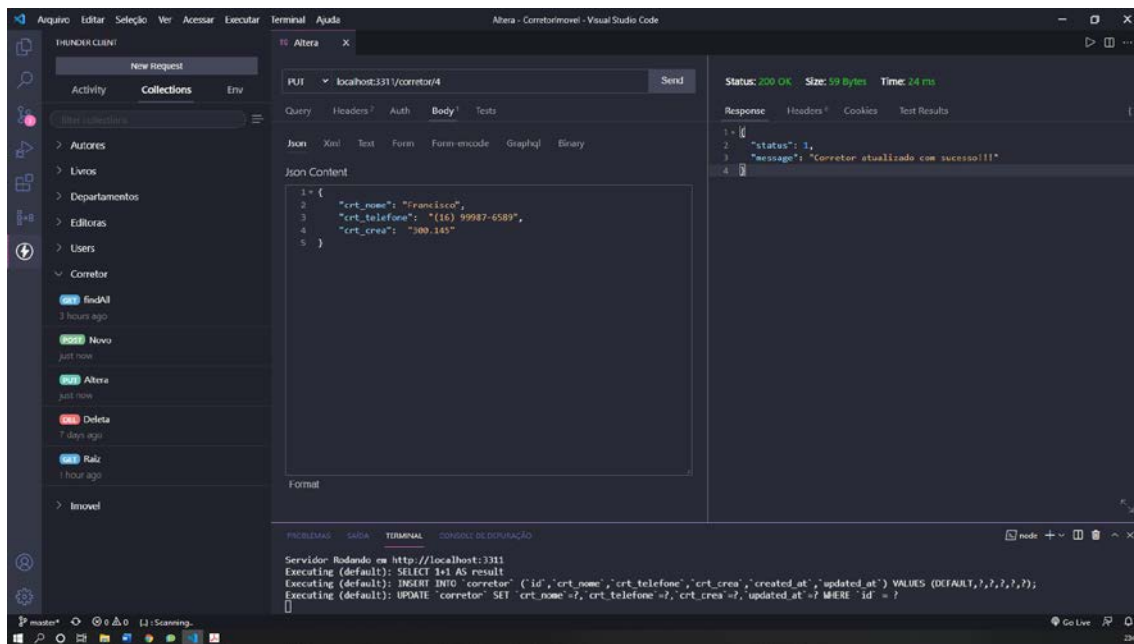
3.5) Figura 7: Protocolo GET método index



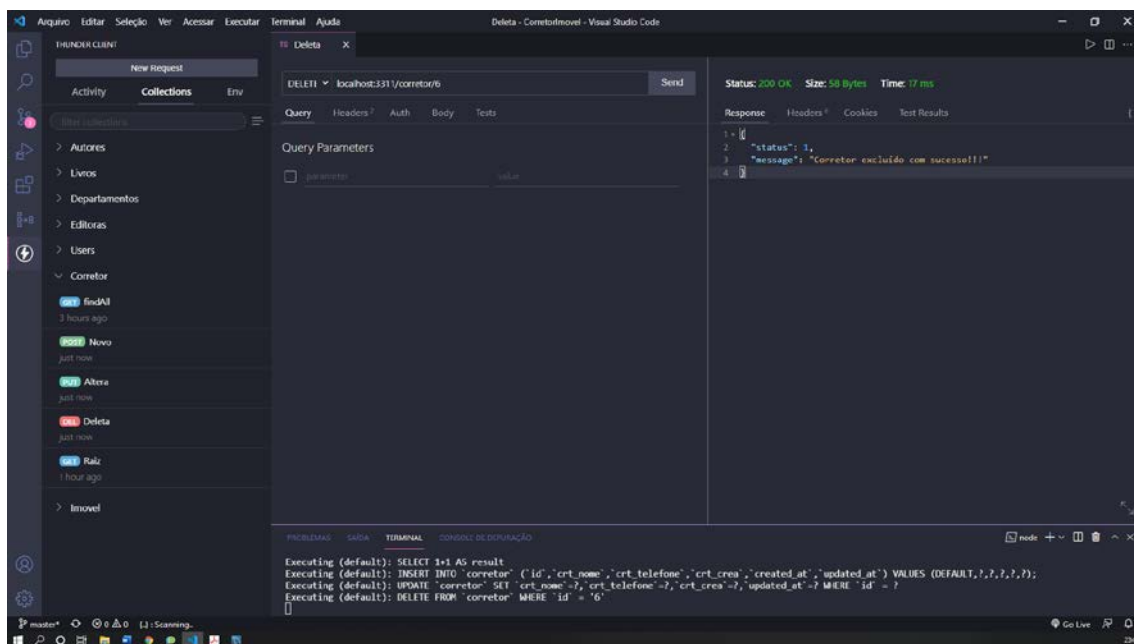
3.6) Figura 8: Protocolo POST método store (adicionar registro)



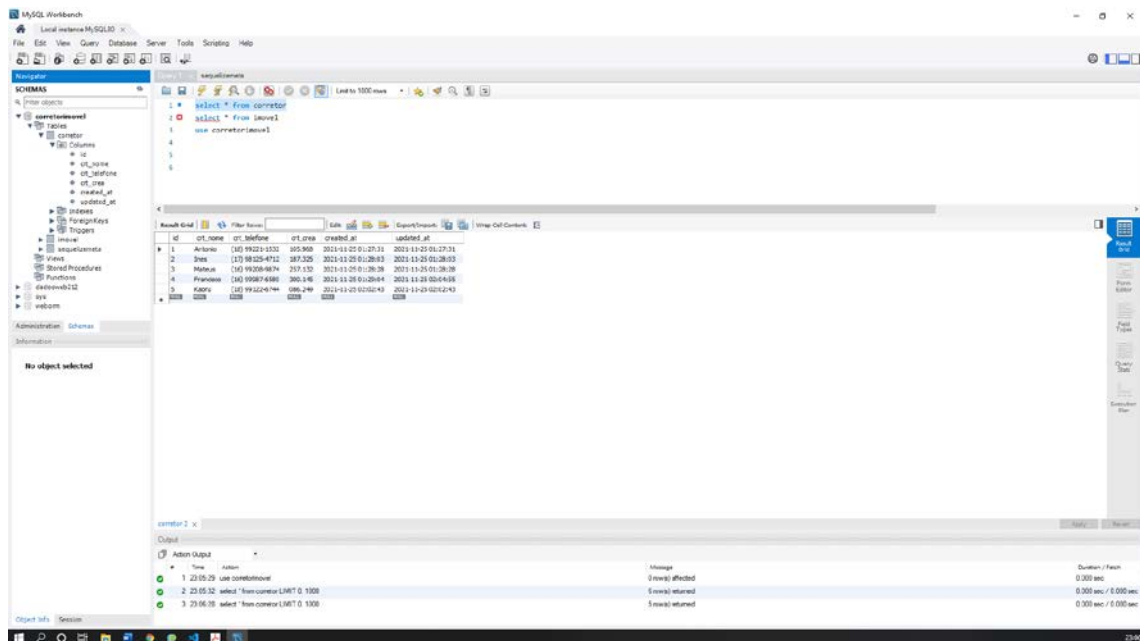
3.7) Figura 9: Protocolo PUT método update (alterar registro)



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

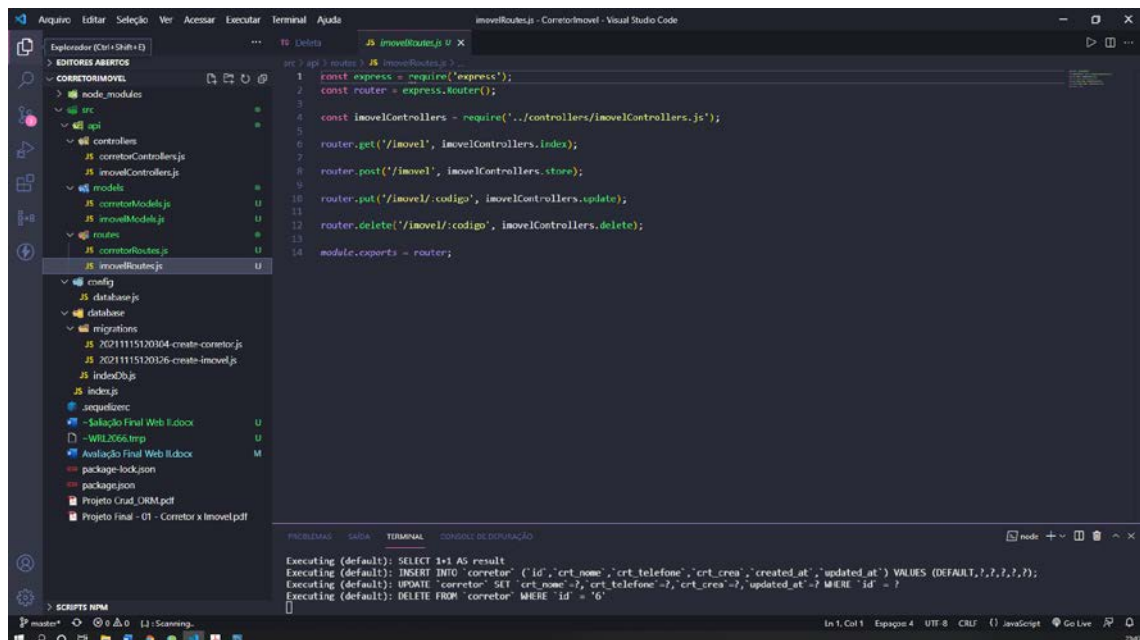


3.9) Figura 11: Imagem do SGDB utilizado, mostrando o banco de dados à esquerda aberto listando os registros da tabela em questão (crt).



4) A partir de agora serão apresentados os prints sequenciais da segunda tabela informada no documento da tarefa. Imovel – imo

4.1) Figura 12: Routes.js da tabela imóvel – imo



4.2) Figura 13: Controllers da tabela imóvel – imo

```

1 const express = require('express');
2 const router = express.Router();
3 const imovelController = require('../controllers/imovelController');
4 const { validationResult } = require('express-validator');
5 const { Op } = require('sequelize');
6
7 // Criar novo imóvel
8 router.post('/', async (req, res) => {
9   const errors = validationResult(req);
10  if (!errors.isEmpty()) {
11    return res.status(400).json({ errors: errors.array() });
12  }
13  const { imo_tipo, imo_cidade, imo_area, imo_comodos, crt_codigo } = req.body;
14  const novoImovel = await imovelController.createImovel(imo_tipo, imo_cidade, imo_area, imo_comodos, crt_codigo);
15  res.status(201).json(novoImovel);
16 });
17
18 // Atualizar imóvel
19 router.put('/:id', async (req, res) => {
20   const errors = validationResult(req);
21   if (!errors.isEmpty()) {
22     return res.status(400).json({ errors: errors.array() });
23   }
24   const { imo_tipo, imo_cidade, imo_area, imo_comodos, crt_codigo } = req.body;
25   const atualizadoImovel = await imovelController.updateImovel(req.params.id, imo_tipo, imo_cidade, imo_area, imo_comodos, crt_codigo);
26   res.status(200).json(atualizadoImovel);
27 });
28
29 // Deletar imóvel
30 router.delete('/:id', async (req, res) => {
31   const errors = validationResult(req);
32   if (!errors.isEmpty()) {
33     return res.status(400).json({ errors: errors.array() });
34   }
35   const deletadoImovel = await imovelController.deleteImovel(req.params.id);
36   res.status(200).json(deletadoImovel);
37 });
38
39 module.exports = router;

```

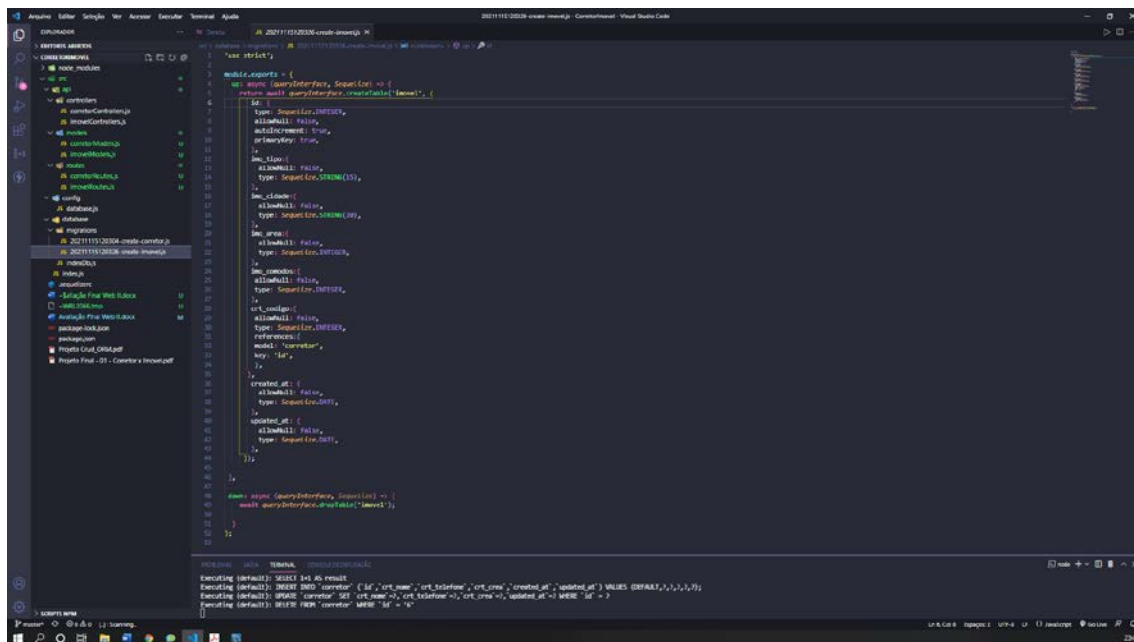
4.3) Figura 14: Models da tabela imóvel – imo

```

1 const { Model, DataTypes } = require('sequelize');
2
3 class Imovel extends Model {
4   static init(sequelize) {
5     super.init({
6       imo_tipo: DataTypes.STRING,
7       imo_cidade: DataTypes.STRING,
8       imo_area: DataTypes.INTEGER,
9       imo_comodos: DataTypes.INTEGER,
10      crt_codigo: DataTypes.INTEGER,
11    }, {
12      sequelize,
13      tableName: 'imovel',
14    });
15    return this;
16  }
17 }
18
19 module.exports = Imovel;

```

4.4) Figura 15: Migration imóvel



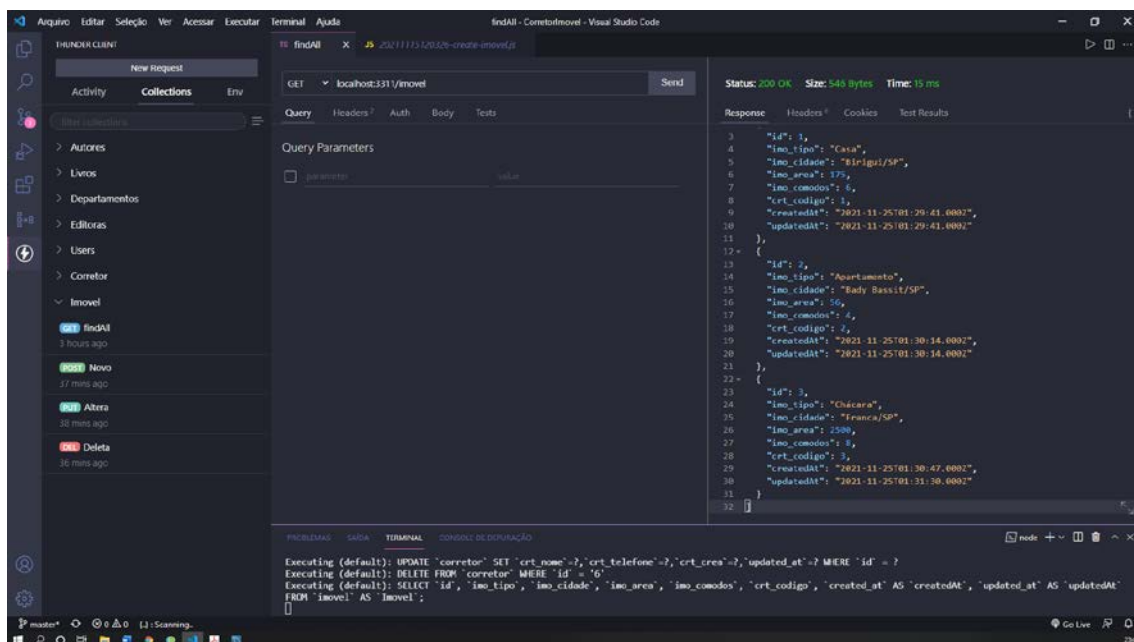
```
20211113125206-criar-imovel.ts
// ...
module.exports = {
  async up(queryInterface, Sequelize) => {
    return queryInterface.bulkCreate('imovel', [
      {
        id: 1,
        tipo: 'Casa',
        cidade: 'Birigui/SP',
        area: 175,
        comodo: 6,
        crt_codigo: 1,
        created_at: '2021-11-25T01:29:41.000Z',
        updated_at: '2021-11-25T01:29:41.000Z',
      },
      {
        id: 2,
        tipo: 'Apartamento',
        cidade: 'Rady Bassit/SP',
        area: 56,
        comodo: 4,
        crt_codigo: 2,
        created_at: '2021-11-25T01:30:14.000Z',
        updated_at: '2021-11-25T01:30:14.000Z',
      },
      {
        id: 3,
        tipo: 'Chácara',
        cidade: 'Franca/SP',
        area: 2500,
        comodo: 8,
        crt_codigo: 3,
        created_at: '2021-11-25T01:30:47.000Z',
        updated_at: '2021-11-25T01:31:30.000Z',
      },
    ], {});
  },
  async down(queryInterface, Sequelize) => {
    return queryInterface.bulkDrop('imovel', {});
  },
};
// ...

```

Terminal output:

```
Executing (default): CREATE TABLE IF NOT EXISTS "imovel" ("id" SERIAL, "tipo" VARCHAR(255), "cidade" VARCHAR(255), "area" NUMERIC(10,2), "comodo" NUMERIC(10,2), "crt_codigo" VARCHAR(255), "created_at" TIMESTAMP, "updated_at" TIMESTAMP);
Executing (default): INSERT INTO "imovel" ("id","tipo","cidade","area","comodo","crt_codigo","created_at","updated_at") VALUES (1,'Casa','Birigui/SP',175,6,1,'2021-11-25T01:29:41.000Z','2021-11-25T01:29:41.000Z');
Executing (default): INSERT INTO "imovel" ("id","tipo","cidade","area","comodo","crt_codigo","created_at","updated_at") VALUES (2,'Apartamento','Rady Bassit/SP',56,4,2,'2021-11-25T01:30:14.000Z','2021-11-25T01:30:14.000Z');
Executing (default): INSERT INTO "imovel" ("id","tipo","cidade","area","comodo","crt_codigo","created_at","updated_at") VALUES (3,'Chácara','Franca/SP',2500,8,3,'2021-11-25T01:30:47.000Z','2021-11-25T01:31:30.000Z');
```

4.5) Figura 16: Protocolo GET método index



Thunder Client interface showing a GET request to `localhost:3311/imovel`. The response is a JSON array of 3 objects, each representing an immovable property.

Request Details:

- Method: GET
- URL: localhost:3311/imovel
- Status: 200 OK
- Size: 548 Bytes
- Time: 15 ms

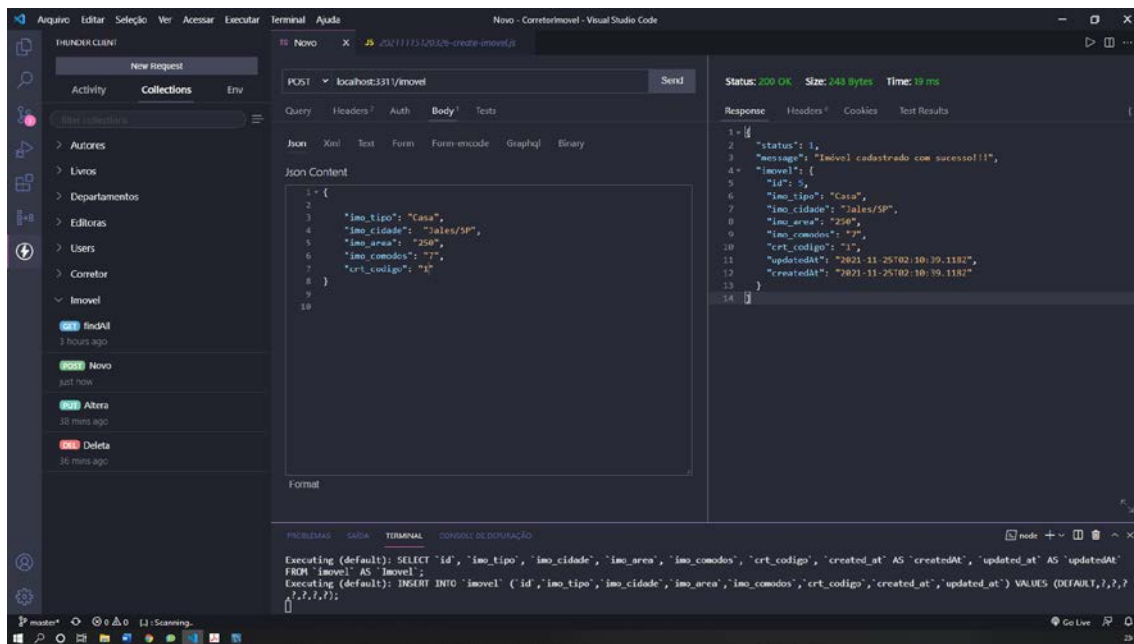
Response (JSON):

```
[
  {
    "id": 1,
    "tipo": "Casa",
    "cidade": "Birigui/SP",
    "area": 175,
    "comodo": 6,
    "crt_codigo": 1,
    "created_at": "2021-11-25T01:29:41.000Z",
    "updated_at": "2021-11-25T01:29:41.000Z"
  },
  {
    "id": 2,
    "tipo": "Apartamento",
    "cidade": "Rady Bassit/SP",
    "area": 56,
    "comodo": 4,
    "crt_codigo": 2,
    "created_at": "2021-11-25T01:30:14.000Z",
    "updated_at": "2021-11-25T01:30:14.000Z"
  },
  {
    "id": 3,
    "tipo": "Chácara",
    "cidade": "Franca/SP",
    "area": 2500,
    "comodo": 8,
    "crt_codigo": 3,
    "created_at": "2021-11-25T01:30:47.000Z",
    "updated_at": "2021-11-25T01:31:30.000Z"
  }
]
```

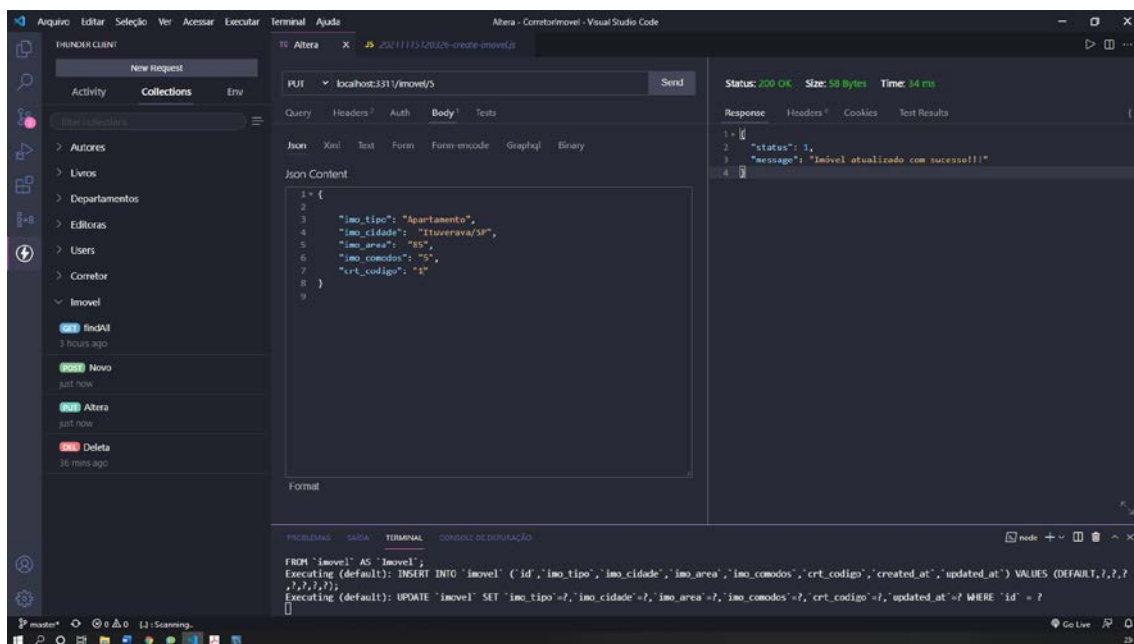
Terminal output (bottom):

```
Executing (default): UPDATE "corretor" SET "crt_nome"=?, "crt_telefone"=?, "crt_crea"=?, "updated_at"=? WHERE "id" = ?
Executing (default): DELETE FROM "corretor" WHERE "id" = ?
Executing (default): SELECT "id", "tipo", "cidade", "area", "comodo", "crt_codigo", "created_at" AS "createdAt", "updated_at" AS "updatedAt" FROM "imovel" AS "imovel";
```

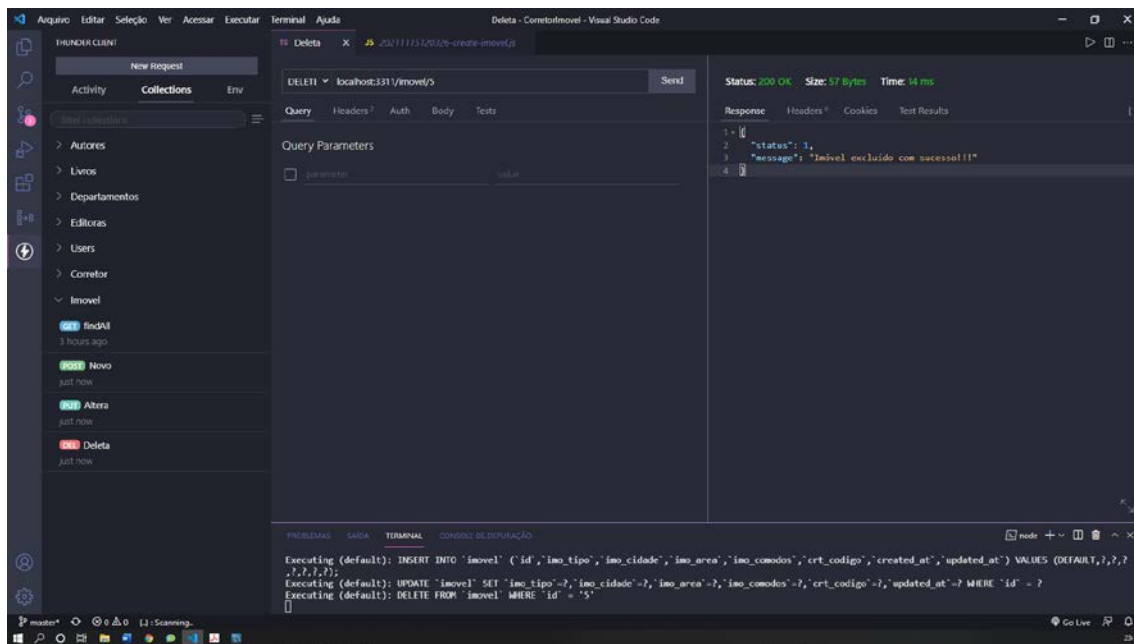

4.6) Figura 17: Protocolo POST método store (adicionar registro)



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 SGBD utilizado, mostrando o banco de dados à esquerda aberto listando os registros da tabela em questão (imo).

