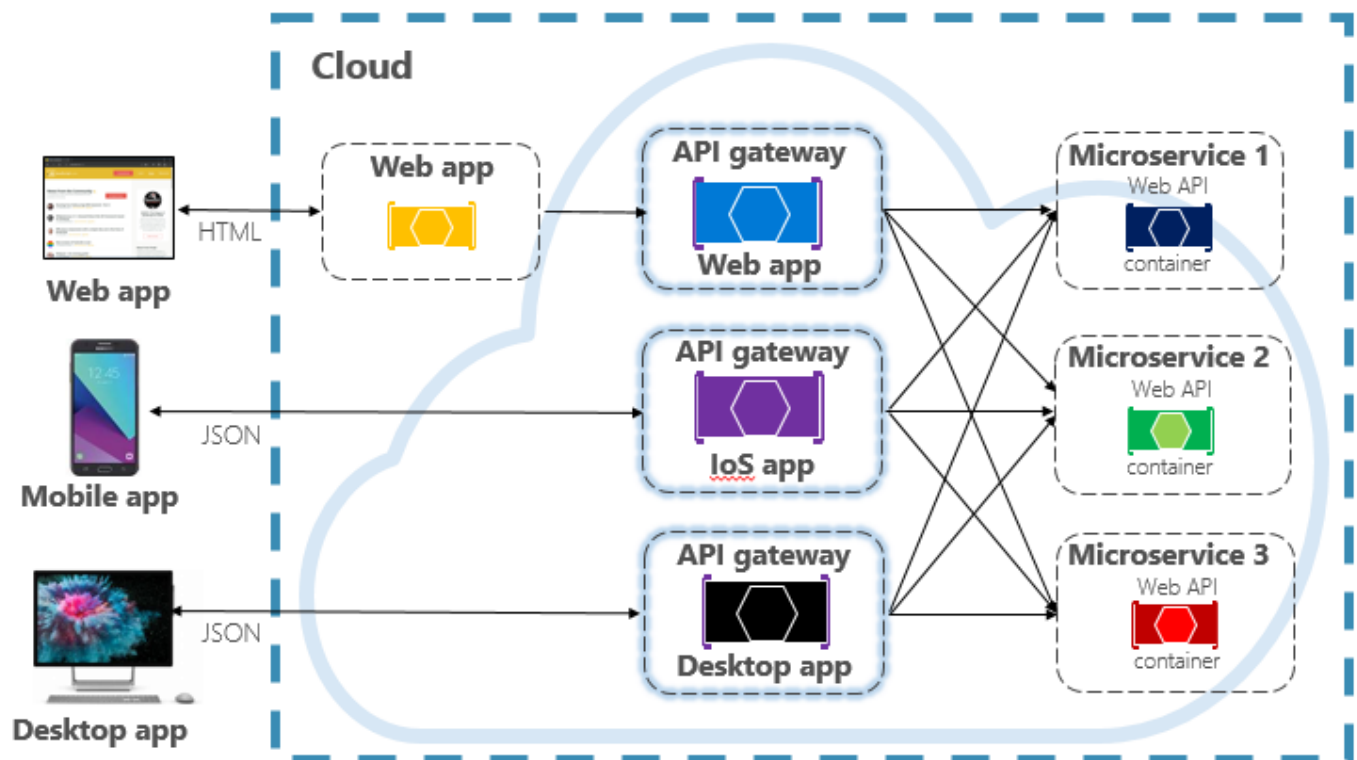
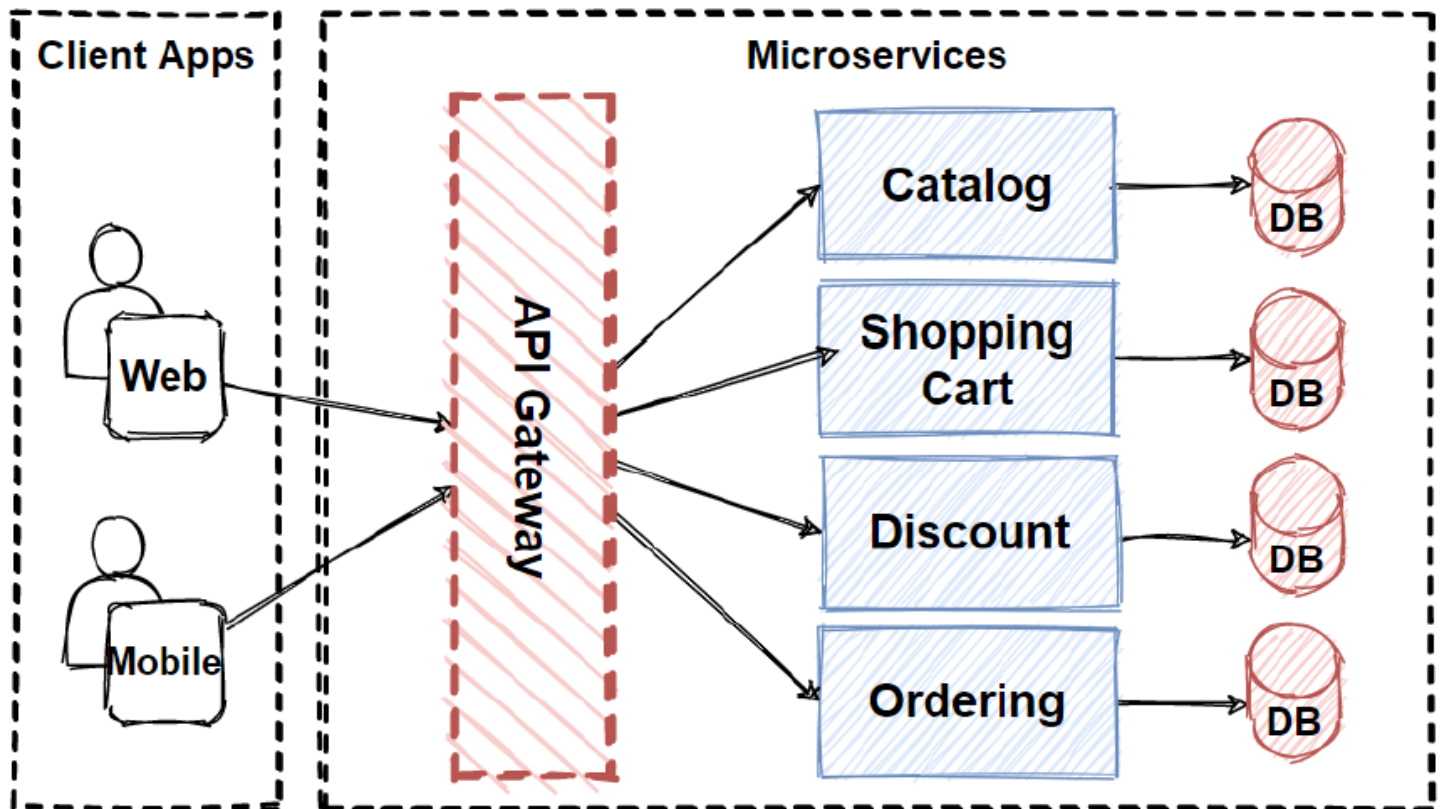
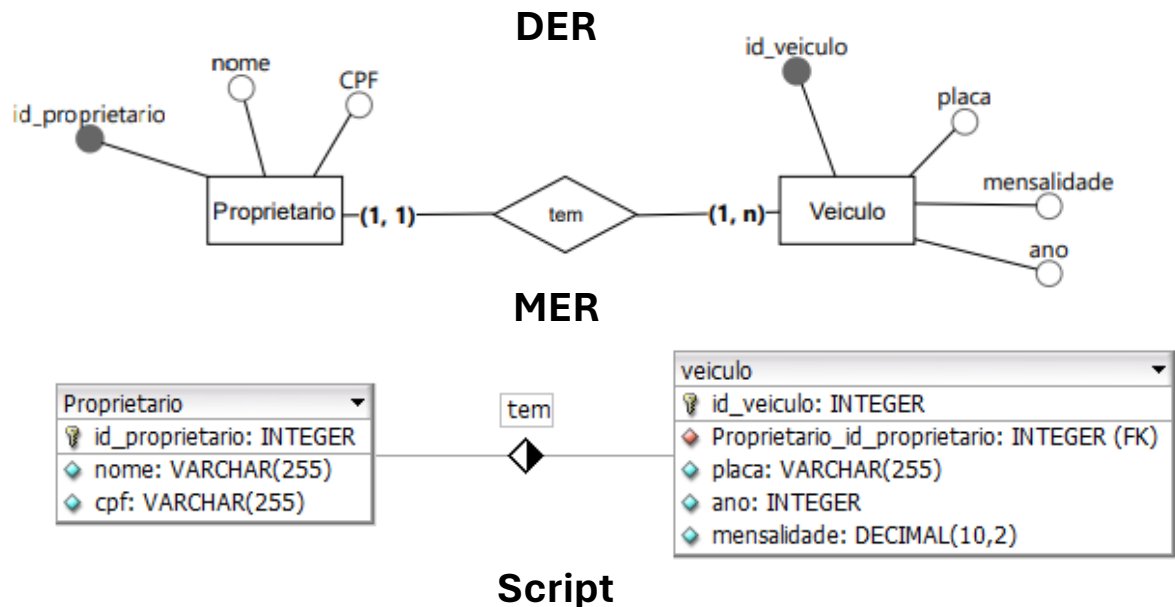


Professor Mário de Jesus



Situação Problema

Um Estacionamento começou a cobrar uma mensalidade os clientes que utilizam diariamente o estacionamento. Para isso ele precisa cadastrar o proprietário e o veículo que vai ficar no estacionamento junto com a mensalidade do mês. Vamos criar o **projeto de banco de dados**.



```
CREATE database estacionamento;
use estacionamento;
CREATE TABLE Proprietario (
  Id_proprietario INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  nome VARCHAR(255) NULL ,
  cpf VARCHAR(255) NULL ,
  PRIMARY KEY(id));

CREATE TABLE veiculo (
  Id_veiculo INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
  fk_proprietario INTEGER UNSIGNED NOT NULL ,
  placa VARCHAR(255) NULL ,
  ano INTEGER UNSIGNED NULL ,
  mensalidade DECIMAL(10,2) NULL ,
  PRIMARY KEY(id) ,
  INDEX veiculo_FKIndex1(fk_proprietario),
  FOREIGN KEY(fk_proprietario)
  REFERENCES Proprietario(id)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION);
```

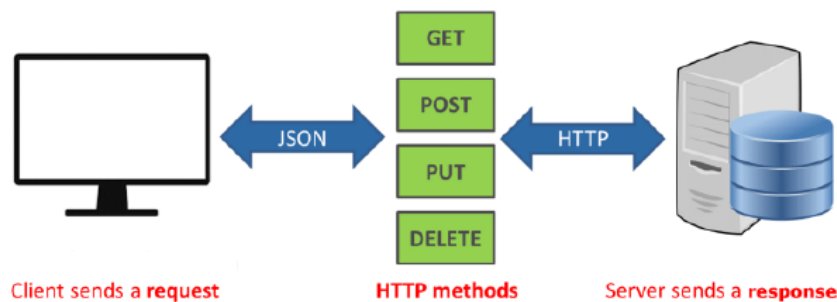
O que é arquitetura REST?

O acrônimo API é a abreviação de *Application Programming Interface*, que significa “Interface de Programação de Aplicações”. Representa um conjunto de rotinas e padrões estabelecidos e documentados para que uma determinada aplicação de software tenha autorização para utilizar as funcionalidades oferecidas por essa aplicação, sem precisar conhecer as anuências dessa implementação.

REST significa Transferência de Estado Representacional. REST é uma arquitetura baseada em padrões da web e usa protocolo HTTP. Ele gira em torno de recurso onde cada componente é um recurso acessado por uma interface comum usando métodos padrão HTTP.

API REST é uma abstração de arquitetura de software que fornece dados em um formato padronizado para modelos de requisições HTTP.

Um servidor REST simplesmente fornece acesso a recursos, e o cliente REST acessa e modifica os recursos usando o protocolo HTTP. Aqui, cada recurso é identificado por URIs / IDs globais. REST usa várias representações para apresentar um recurso como texto, JSON, XML, mas JSON é o mais popular.



Métodos HTTP

Os **quatro métodos HTTP** a seguir são comumente usados na arquitetura baseada em REST.

GET - É utilizado para ler registros no banco de dados. Status Code **200** – retorna os registros no formato solicitado.

POST - É utilizado para criar um novo registro no banco de dados. Status Code **201** – registro criado

PUT - É utilizado para atualizar um registro no banco de dados. Status Code **200** – registro atualizado.

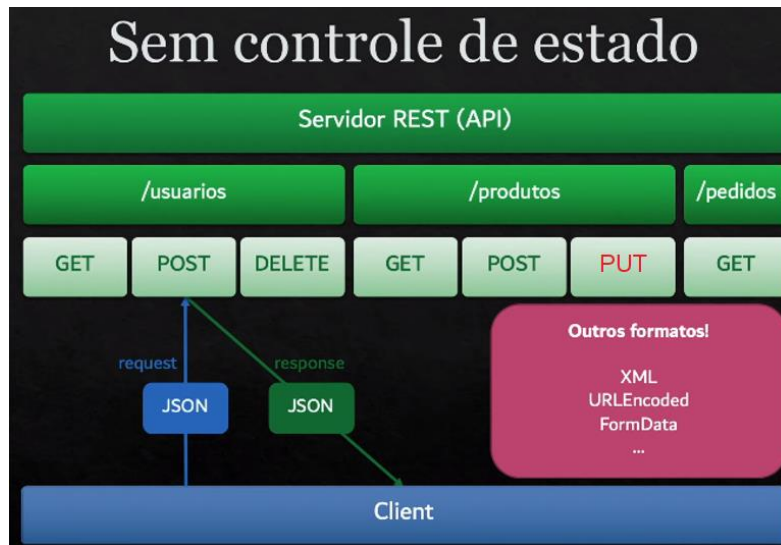
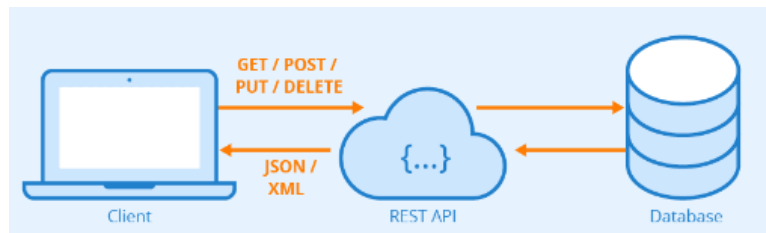
DELETE - É utilizado para deletar um registro no banco de dados. Status Code **204** – registro deletado.

Criando nosso servidor

O CORS **Cross-origin Resource Sharing** (Compartilhamento de recursos com origens diferentes) é um mecanismo utilizado pelos navegadores para compartilhar recursos entre diferentes origens. O CORS é uma especificação do W3C e faz uso de headers do HTTP para informar aos navegadores se determinado recurso pode ser ou não acessado.

OBS: O comando `$ npm init` nos permite iniciar um pacote, criando o arquivo `package.json` de acordo com certas respostas que damos às perguntas feitas. Mas você pode **pular as perguntas**, fazendo com que o arquivo `package.json` seja criado imediatamente. Basta adicionar `-y` ao comando: `$ npm init -y`

Exemplo de API Rest sem controle de estado



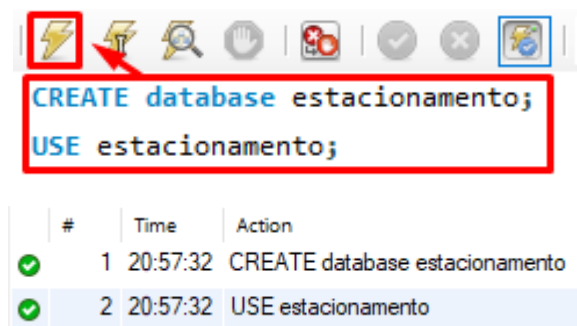
Teste de API

O **Insomnia** é um aplicativo gratuito e é uma das ferramentas populares usadas em testes de API (Interfaces de Programação de Aplicativos). Essas ferramentas permitem que desenvolvedores da web possam testar um conjunto específico de dados para o aplicativo e determinar se eles atendem às expectativas de funcionalidade, confiabilidade, desempenho e segurança. Temos também o postman, SoapUI, entre outros. Faça download do insomnia no site abaixo.

<https://insomnia.rest/download>

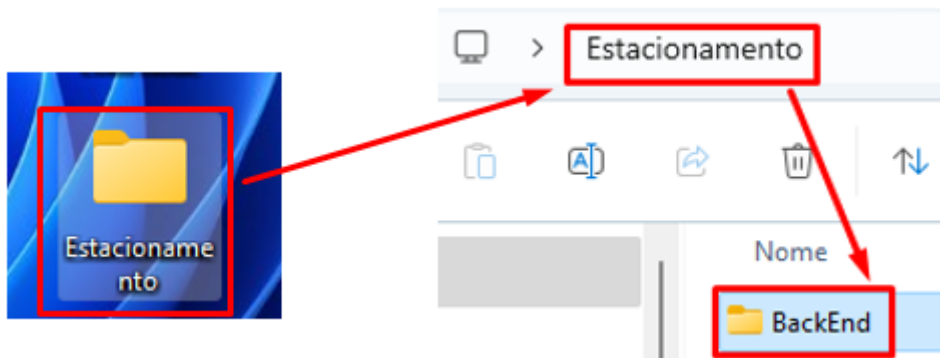
Criando o banco de dados

Crie um **banco de dados** no mysql chamado **estacionamento**

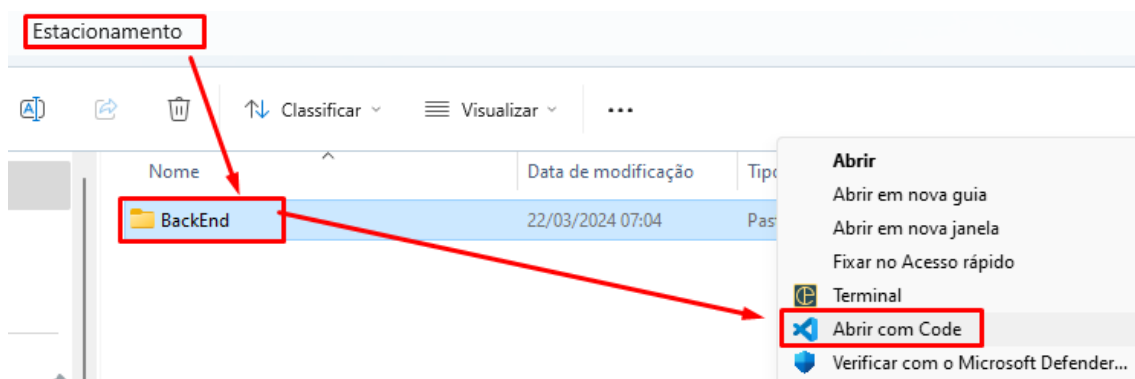


Criando as pastas do projeto

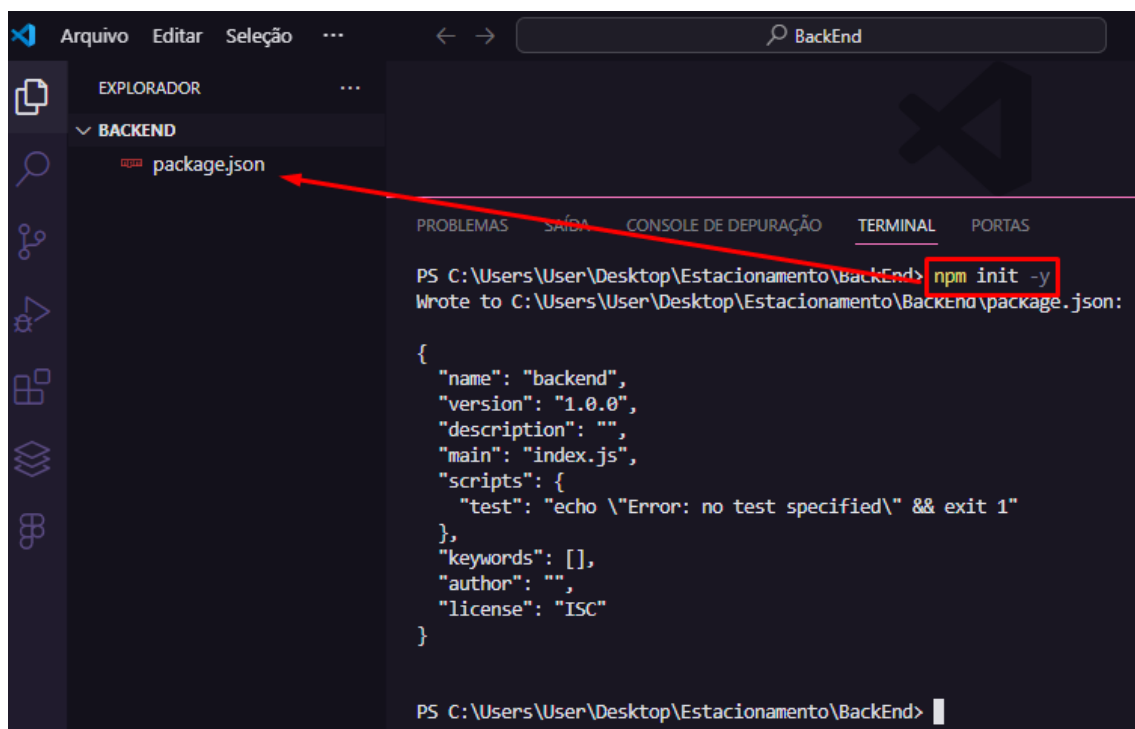
Crie a pasta estacionamento e dentro dela crie a pasta BackEnd



Abra essa pasta BackEnd com o VS Code

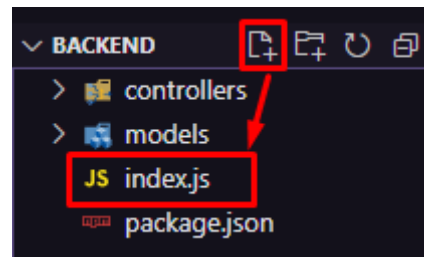
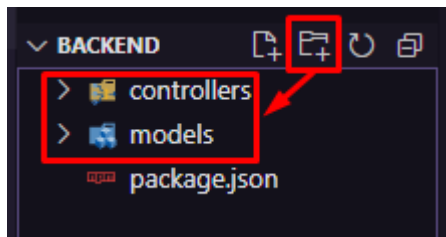


Abra o terminal e digite npm init -y



Instalando as dependências

Crie a pasta **models**, **controllers** e o arquivo **index.js**



Agora **instale** os **módulos** abaixo no terminal

```
npm install express --save
```

```
npm install nodemon -g
```

```
npm install sequelize --save
```

```
npm install body-parser --save
```

```
npm install mysql2 --save
```

```
npm install cors --save
```

```
PROBLEMAS  SAÍDA  CONSOLE DE DEPURACÃO  TERMINAL  PORTAS

PS C:\Users\User\Desktop\React Native com node\BackEnd> npm install express --save

added 64 packages, and audited 65 packages in 6s

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

found 0 vulnerabilities
PS C:\Users\User\Desktop\React Native com node\BackEnd> npm install nodemon -g

changed 33 packages in 31s

4 packages are looking for funding
  run `npm fund` for details
PS C:\Users\User\Desktop\React Native com node\BackEnd> npm install sequelize --save

added 23 packages, and audited 88 packages in 18s

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

found 0 vulnerabilities
PS C:\Users\User\Desktop\React Native com node\BackEnd> npm install body-parser --save

up to date, audited 88 packages in 1s

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

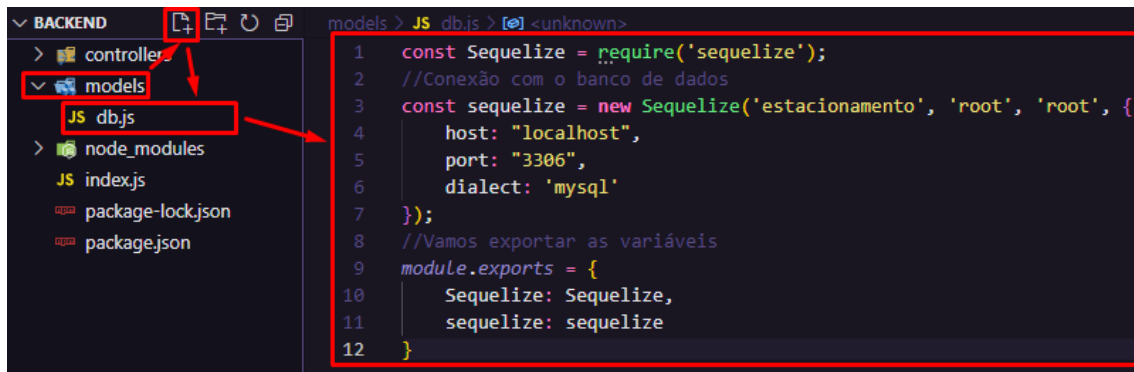
found 0 vulnerabilities
PS C:\Users\User\Desktop\React Native com node\BackEnd> npm install mysql2 --save

added 2 packages, and audited 101 packages in 1s

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

Estabelecendo a Conexão com o banco de dados

Dentro de **models** crie o arquivo **db.js** e digite o código abaixo:



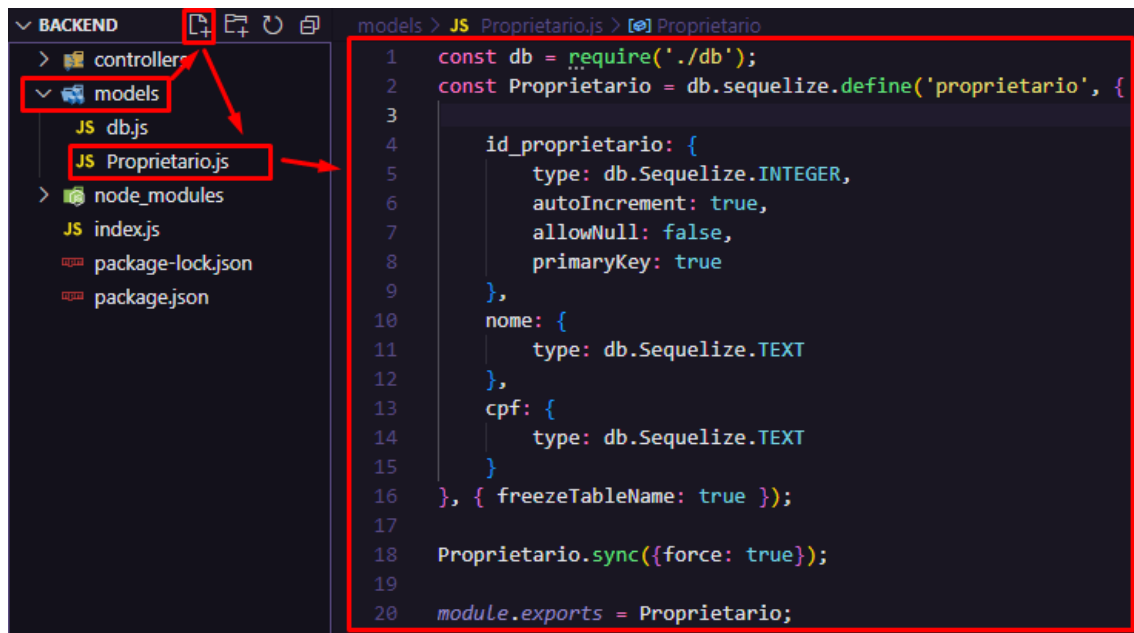
```
1  const Sequelize = require('sequelize');
2  //Conexão com o banco de dados
3  const sequelize = new Sequelize('estacionamento', 'root', 'root', {
4      host: "localhost",
5      port: "3306",
6      dialect: 'mysql'
7  });
8  //Vamos exportar as variáveis
9  module.exports = {
10      Sequelize: Sequelize,
11      sequelize: sequelize
12  }
```

Código:

```
const Sequelize = require('sequelize');
//Conexão com o banco de dados
const sequelize = new Sequelize('estacionamento', 'root', 'root', {
  host: "localhost",
  port: "3306",
  dialect: 'mysql'
});
//Vamos exportar as variáveis
module.exports = {
  Sequelize: Sequelize,
  sequelize: sequelize
}
```

Criando o Modelo de Proprietário

Dentro **models** crie o **arquivo Proprietario.js** e digite o código abaixo:



```
1 const db = require('./db');
2 const Proprietario = db.sequelize.define('proprietario', {
3
4   id_proprietario: {
5     type: db.Sequelize.INTEGER,
6     autoIncrement: true,
7     allowNull: false,
8     primaryKey: true
9   },
10  nome: {
11    type: db.Sequelize.TEXT
12  },
13  cpf: {
14    type: db.Sequelize.TEXT
15  }
16 }, { freezeTableName: true });
17
18 Proprietario.sync({force: true});
19
20 module.exports = Proprietario;
```

Código

```
const db = require('./db');
const Proprietario = db.sequelize.define('proprietario', {

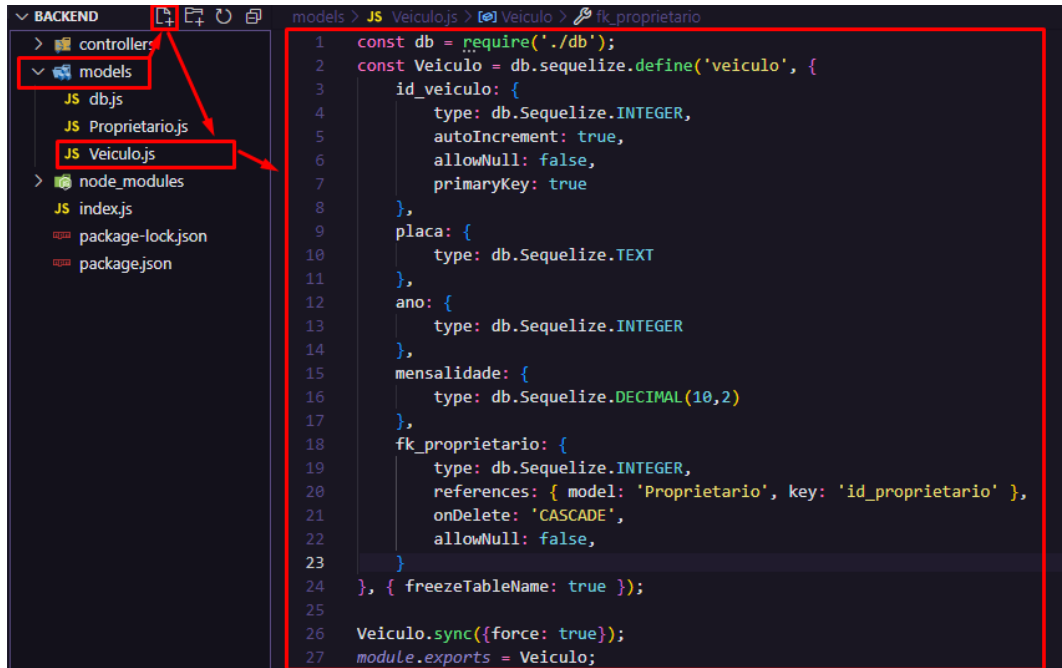
  id_proprietario: {
    type: db.Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  nome: {
    type: db.Sequelize.TEXT
  },
  cpf: {
    type: db.Sequelize.TEXT
  }
}, { freezeTableName: true });

Proprietario.sync({force: true});

module.exports = Proprietario;
```


Criando o Modelo de Veículo

Dentro **models** crie o arquivo **Veiculo.js** e digite o código abaixo:



```
1  const db = require('./db');
2  const Veiculo = db.sequelize.define('veiculo', {
3
4      id_veiculo: {
5          type: db.Sequelize.INTEGER,
6          autoIncrement: true,
7          allowNull: false,
8          primaryKey: true
9      },
10     placa: {
11         type: db.Sequelize.TEXT
12     },
13     ano: {
14         type: db.Sequelize.INTEGER
15     },
16     mensalidade: {
17         type: db.Sequelize.DECIMAL(10,2)
18     },
19     fk_proprietario: {
20         type: db.Sequelize.INTEGER,
21         references: { model: 'Proprietario', key: 'id_proprietario' },
22         onDelete: 'CASCADE',
23         allowNull: false,
24     },
25 }, { freezeTableName: true });
26
27 Veiculo.sync({force: true});
28 module.exports = Veiculo;
```

Código

```
const db = require('./db');
const Veiculo = db.sequelize.define('veiculo', {
  id_veiculo: {
    type: db.Sequelize.INTEGER,
    autoIncrement: true,
    allowNull: false,
    primaryKey: true
  },
  placa: {
    type: db.Sequelize.TEXT
  },
  ano: {
    type: db.Sequelize.INTEGER
  },
  mensalidade: {
    type: db.Sequelize.DECIMAL(10,2)
  },
  fk_proprietario: {
    type: db.Sequelize.INTEGER,
    references: { model: 'Proprietario', key: 'id_proprietario' },
    onDelete: 'CASCADE',
    allowNull: false,
  }
}, { freezeTableName: true });
Veiculo.sync({force: true});
module.exports = Veiculo;
```

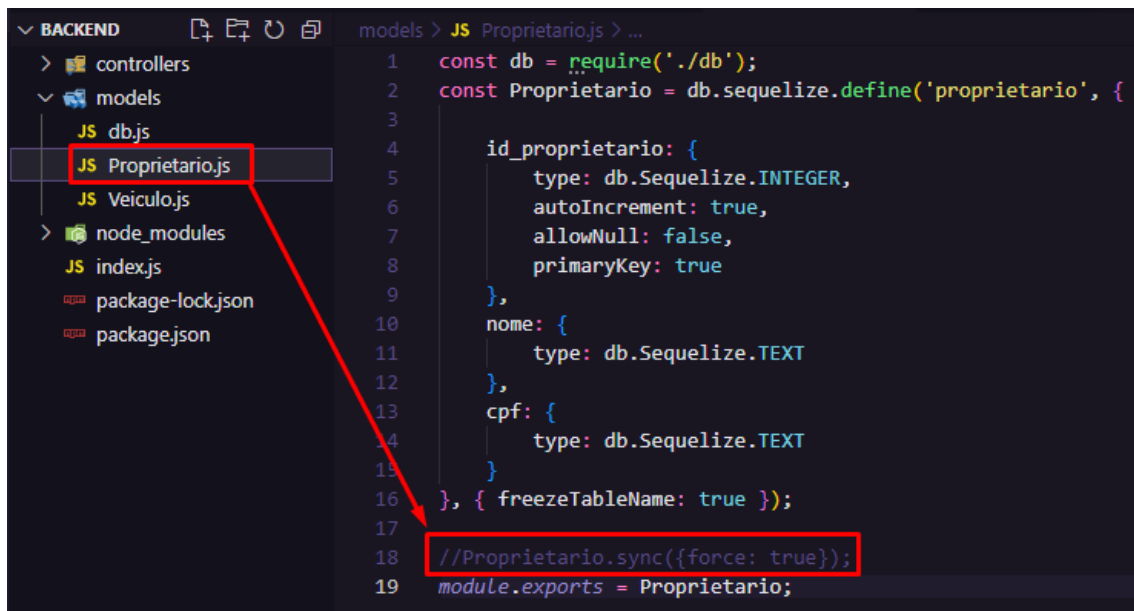
Criando as tabelas de Proprietário e Veículo no banco de dados

Entre na pasta models e crie as tabelas com o comando node depois retorne para a pasta BackEnd

```
cd models
node Proprietario.js
node Veiculo.js
cd..
```

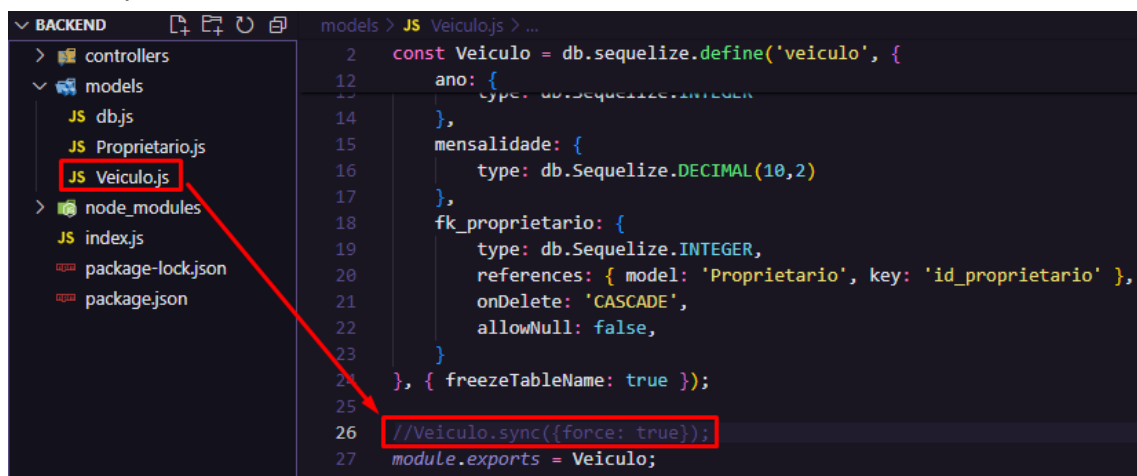
```
PS C:\Users\User\Desktop\Estacionamento\Backend> cd models
PS C:\Users\User\Desktop\Estacionamento\Backend\models> node Proprietario.js
Executing (default): DROP TABLE IF EXISTS `proprietario`;
Executing (default): CREATE TABLE IF NOT EXISTS `proprietario` (`id_proprietario` INTEGER NOT NULL auto_increment, `nome` TEXT, `cpf` TEXT, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id_proprietario`)) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `proprietario`
PS C:\Users\User\Desktop\Estacionamento\Backend\models> node Veiculo.js
Executing (default): DROP TABLE IF EXISTS `veiculo`;
Executing (default): CREATE TABLE IF NOT EXISTS `veiculo` (`id_veiculo` INTEGER NOT NULL auto_increment, `placa` TEXT, `ano` INTEGER, `mensalidade` DECIMAL(10,2), `fk_proprietario` INTEGER NOT NULL, `createdAt` DATETIME NOT NULL, `updatedAt` DATETIME NOT NULL, PRIMARY KEY (`id_veiculo`), FOREIGN KEY (`fk_proprietario`) REFERENCES `Proprietario` (`id_proprietario`) ON DELETE CASCADE) ENGINE=InnoDB;
Executing (default): SHOW INDEX FROM `veiculo`
PS C:\Users\User\Desktop\Estacionamento\Backend\models> cd..
```

Volte para o Arquivo **Proprietario.js** e comente a linha de sincronização com o banco de dados para ele não criar a tabela novamente.



```
1 const db = require('./db');
2 const Proprietario = db.sequelize.define('proprietario', {
3
4   id_proprietario: {
5     type: db.Sequelize.INTEGER,
6     autoIncrement: true,
7     allowNull: false,
8     primaryKey: true
9   },
10  nome: {
11    type: db.Sequelize.TEXT
12  },
13  cpf: {
14    type: db.Sequelize.TEXT
15  }
16 }, { freezeTableName: true });
17 //Proprietario.sync({force: true});
18 module.exports = Proprietario;
```

Volte para o Arquivo **Veiculo.js** e comente a linha de sincronização com o banco de dados para ele não criar a tabela novamente.



```
2 const Veiculo = db.sequelize.define('veiculo', {
12  ano: {
13    type: db.Sequelize.INTEGER
14  },
15  mensalidade: {
16    type: db.Sequelize.DECIMAL(10,2)
17  },
18  fk_proprietario: {
19    type: db.Sequelize.INTEGER,
20    references: { model: 'Proprietario', key: 'id_proprietario' },
21    onDelete: 'CASCADE',
22    allowNull: false,
23  }
24 }, { freezeTableName: true });
25 //Veiculo.sync({force: true});
26 module.exports = Veiculo;
```

Testando o banco de dados

Volte para o **mysql**, verifique se as tabelas **foram criadas**, faça uma **inserção** e uma **seleção** de dados em **cada** uma das **tabelas**.

```
1 • CREATE database estacionamento;
2 • USE estacionamento;
3 • select * from proprietario;
4 • select * from veiculo;
5
6 • insert into proprietario(nome, cpf, createdAt, updatedAt) values("Mario", "332323", now(), now());
7 • insert into veiculo(placa, ano, mensalidade, fk_proprietario, createdAt, updatedAt) values("gav-99", 2017, "1423.58",1,now(), now());
8
9 • select * from proprietario;
10 • select * from veiculo;
```

Result Grid						Filter Rows:	Edit:	Export/Import:
	id_proprietario	nome	cpf	createdAt	updatedAt			
▶	1	Mario	332323	2024-03-22 08:01:50	2024-03-22 08:01:50			
*	NULL	NULL	NULL	NULL	NULL			

Result Grid								Filter Rows:	Edit:	Export/Import:	Wrap Cell
	id_veiculo	placa	ano	mensalidade	fk_proprietario	createdAt	updatedAt				
▶	1	gav-99	2017	1423.58	1	2024-03-22 08:01:53	2024-03-22 08:01:53				
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL				

Script

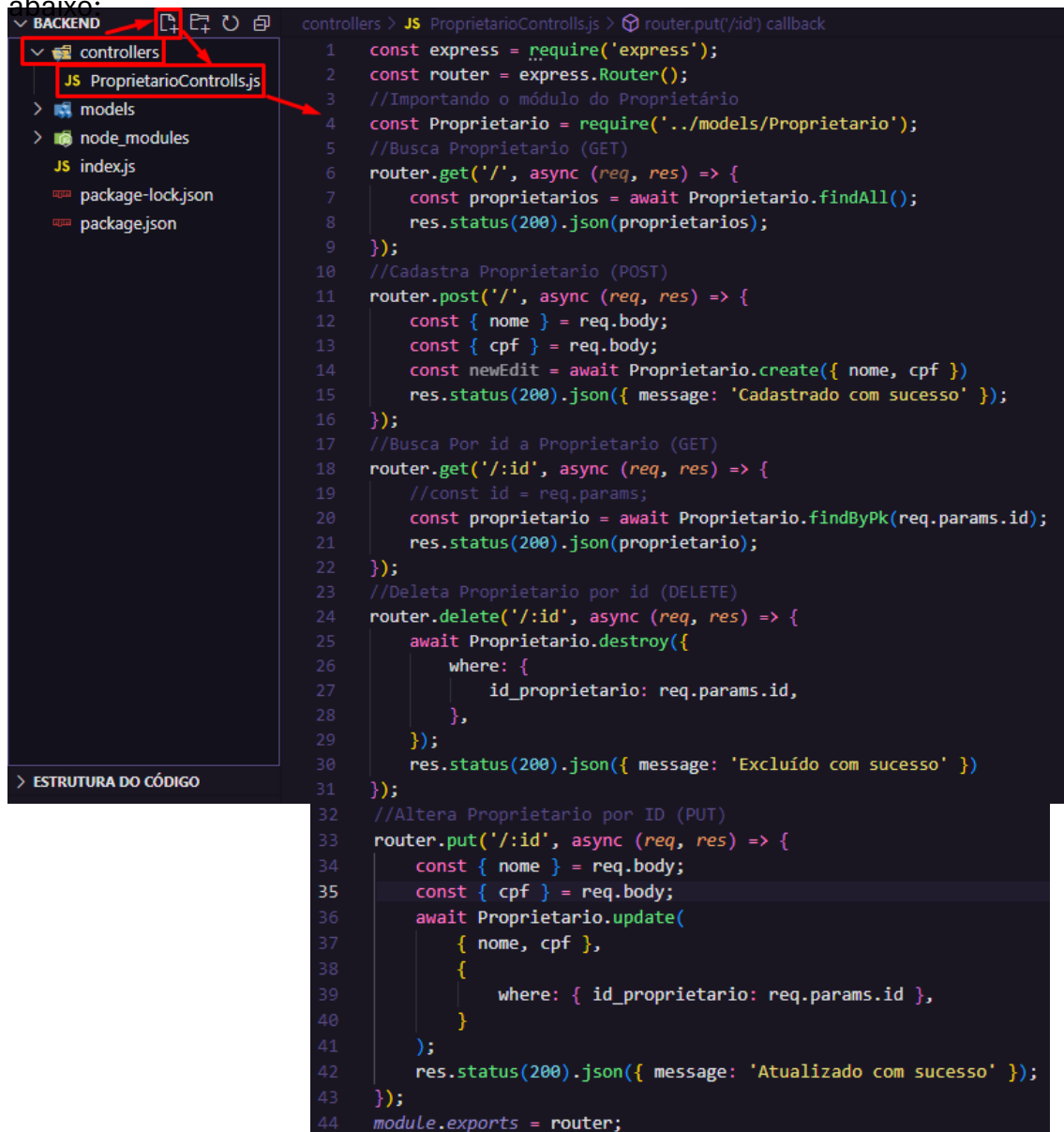
```
CREATE database estacionamento;
USE estacionamento;
select * from proprietario;
select * from veiculo;

insert into proprietario(nome, cpf, createdAt, updatedAt) values("Mario",
"332323", now(), now());
insert into veiculo(placa, ano, mensalidade, fk_proprietario, createdAt,
updatedAt) values("gav-99", 2017, "1423.58",1,now(), now());

select * from proprietario;
select * from veiculo;
```

Criando o Controle (Rotas) para Proprietário

Crie o arquivo **ProprietarioControlls** dentro da pasta **controllers** e digite o código abaixo:



The image shows a code editor interface. On the left, a file explorer shows a project structure with a 'BACKEND' folder containing 'controllers', 'models', 'node_modules', 'index.js', 'package-lock.json', and 'package.json'. The 'controllers' folder is expanded, and a new file 'JS ProprietarioControlls.js' is being created, highlighted with a red box and a red arrow. The main editor area shows the code for this file, which implements a REST API for a 'Proprietario' resource using Express.js and Sequelize. The code includes routes for GET (list), POST (create), GET (find by ID), DELETE (delete by ID), and PUT (update by ID). Red arrows point from the file explorer to the file name and from the code editor to the 'router.put' method call in the header.

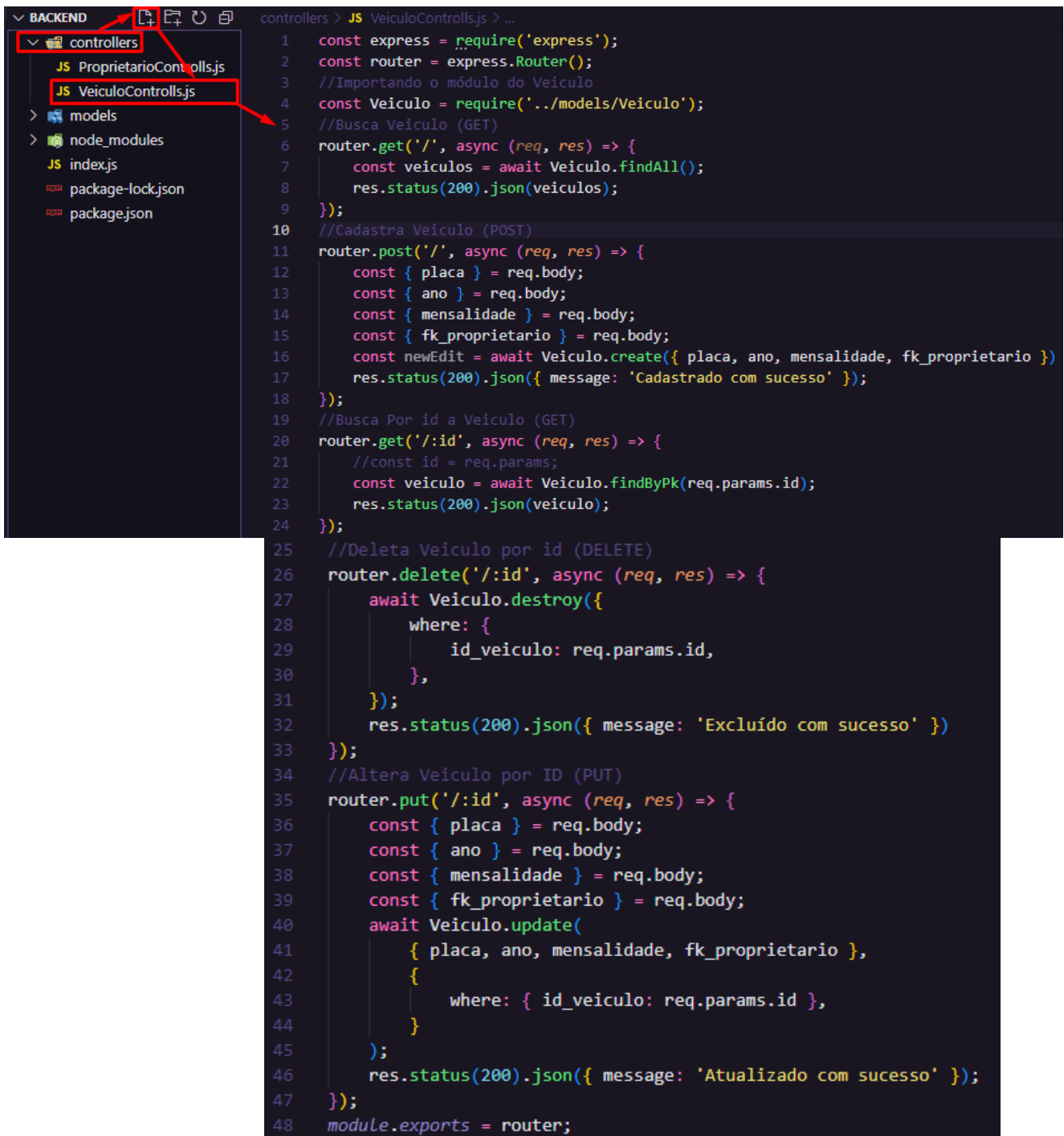
```
1  const express = require('express');
2  const router = express.Router();
3  //Importando o módulo do Proprietário
4  const Proprietario = require('../models/Proprietario');
5  //Busca Proprietario (GET)
6  router.get('/', async (req, res) => {
7    const proprietarios = await Proprietario.findAll();
8    res.status(200).json(proprietarios);
9  });
10 //Cadastra Proprietario (POST)
11 router.post('/', async (req, res) => {
12   const { nome } = req.body;
13   const { cpf } = req.body;
14   const newEdit = await Proprietario.create({ nome, cpf });
15   res.status(200).json({ message: 'Cadastrado com sucesso' });
16 });
17 //Busca Por id a Proprietario (GET)
18 router.get('/:id', async (req, res) => {
19   //const id = req.params;
20   const proprietario = await Proprietario.findByPk(req.params.id);
21   res.status(200).json(proprietario);
22 });
23 //Deleta Proprietario por id (DELETE)
24 router.delete('/:id', async (req, res) => {
25   await Proprietario.destroy({
26     where: {
27       id_proprietario: req.params.id,
28     },
29   });
30   res.status(200).json({ message: 'Excluído com sucesso' });
31 });
32 //Altera Proprietario por ID (PUT)
33 router.put('/:id', async (req, res) => {
34   const { nome } = req.body;
35   const { cpf } = req.body;
36   await Proprietario.update(
37     { nome, cpf },
38     {
39       where: { id_proprietario: req.params.id },
40     }
41   );
42   res.status(200).json({ message: 'Atualizado com sucesso' });
43 });
44 module.exports = router;
```

Código do **ProprietarioControlls.js**

```
const express = require('express');
const router = express.Router();
//Importando o módulo do Proprietário
const Proprietario = require('../models/Proprietario');
//Busca Proprietario (GET)
router.get('/', async (req, res) => {
  const proprietarios = await Proprietario.findAll();
  res.status(200).json(proprietarios);
});
//Cadastra Proprietario (POST)
router.post('/', async (req, res) => {
  const { nome } = req.body;
  const { cpf } = req.body;
  const newEdit = await Proprietario.create({ nome, cpf })
  res.status(200).json({ message: 'Cadastrado com sucesso' });
});
//Busca Por id a Proprietario (GET)
router.get('/:id', async (req, res) => {
  //const id = req.params;
  const proprietario = await Proprietario.findByPk(req.params.id);
  res.status(200).json(proprietario);
});
//Deleta Proprietario por id (DELETE)
router.delete('/:id', async (req, res) => {
  await Proprietario.destroy({
    where: {
      id_proprietario: req.params.id,
    },
  });
  res.status(200).json({ message: 'Excluído com sucesso' });
});
//Altera Proprietario por ID (PUT)
router.put('/:id', async (req, res) => {
  const { nome } = req.body;
  const { cpf } = req.body;
  await Proprietario.update(
    { nome, cpf },
    {
      where: { id_proprietario: req.params.id },
    }
  );
  res.status(200).json({ message: 'Atualizado com sucesso' });
});
module.exports = router;
```

Criando o Controle (Rotas) para Veículo

Crie o arquivo **VeiculoControlls** dentro da pasta **controllers** e digite o código abaixo:



```
1  const express = require('express');
2  const router = express.Router();
3  //Importando o módulo do Veiculo
4  const Veiculo = require('../models/Veiculo');
5  //Busca Veiculo (GET)
6  router.get('/', async (req, res) => {
7    const veiculos = await Veiculo.findAll();
8    res.status(200).json(veiculos);
9  });
10 //Cadastra Veiculo (POST)
11 router.post('/', async (req, res) => {
12   const { placa } = req.body;
13   const { ano } = req.body;
14   const { mensalidade } = req.body;
15   const { fk_proprietario } = req.body;
16   const newEdit = await Veiculo.create({ placa, ano, mensalidade, fk_proprietario });
17   res.status(200).json({ message: 'Cadastrado com sucesso' });
18 });
19 //Busca Por id a Veiculo (GET)
20 router.get('/:id', async (req, res) => {
21   //const id = req.params;
22   const veiculo = await Veiculo.findByPk(req.params.id);
23   res.status(200).json(veiculo);
24 });
25 //Deleta Veiculo por id (DELETE)
26 router.delete('/:id', async (req, res) => {
27   await Veiculo.destroy({
28     where: {
29       id_veiculo: req.params.id,
30     },
31   });
32   res.status(200).json({ message: 'Excluído com sucesso' });
33 });
34 //Altera Veiculo por ID (PUT)
35 router.put('/:id', async (req, res) => {
36   const { placa } = req.body;
37   const { ano } = req.body;
38   const { mensalidade } = req.body;
39   const { fk_proprietario } = req.body;
40   await Veiculo.update(
41     { placa, ano, mensalidade, fk_proprietario },
42     {
43       where: { id_veiculo: req.params.id },
44     }
45   );
46   res.status(200).json({ message: 'Atualizado com sucesso' });
47 });
48 module.exports = router;
```

Código do **VeiculoControlls.js**

```
const express = require('express');
const router = express.Router();
//Importando o módulo do Veiculo
const Veiculo = require('../models/Veiculo');
//Busca Veiculo (GET)
router.get('/', async (req, res) => {
  const veiculos = await Veiculo.findAll();
  res.status(200).json(veiculos);
});
//Cadastra Veiculo (POST)
router.post('/', async (req, res) => {
  const { placa } = req.body;
  const { ano } = req.body;
  const { mensalidade } = req.body;
  const { fk_proprietario } = req.body;
  const newEdit = await Veiculo.create({ placa, ano, mensalidade,
fk_proprietario })
  res.status(200).json({ message: 'Cadastrado com sucesso' });
});
//Busca Por id a Veiculo (GET)
router.get('/:id', async (req, res) => {
  //const id = req.params;
  const veiculo = await Veiculo.findByPk(req.params.id);
  res.status(200).json(veiculo);
});
//Deleta Veiculo por id (DELETE)
router.delete('/:id', async (req, res) => {
  await Veiculo.destroy({
    where: { id_veiculo: req.params.id },
  });
  res.status(200).json({ message: 'Excluído com sucesso' })
});
//Altera Veiculo por ID (PUT)
router.put('/:id', async (req, res) => {
  const { placa } = req.body;
  const { ano } = req.body;
  const { mensalidade } = req.body;
  const { fk_proprietario } = req.body;
  await Veiculo.update(
    { placa, ano, mensalidade, fk_proprietario },
    {
      where: { id_veiculo: req.params.id },
    }
  );
  res.status(200).json({ message: 'Atualizado com sucesso' });
});
module.exports = router;
```

Criando o código do index.js

Vamos criar nosso **servidor** que **aponta** as **rotas** da pasta **Controllers**.

Dentro do arquivo **index.js** digite o **código** abaixo:



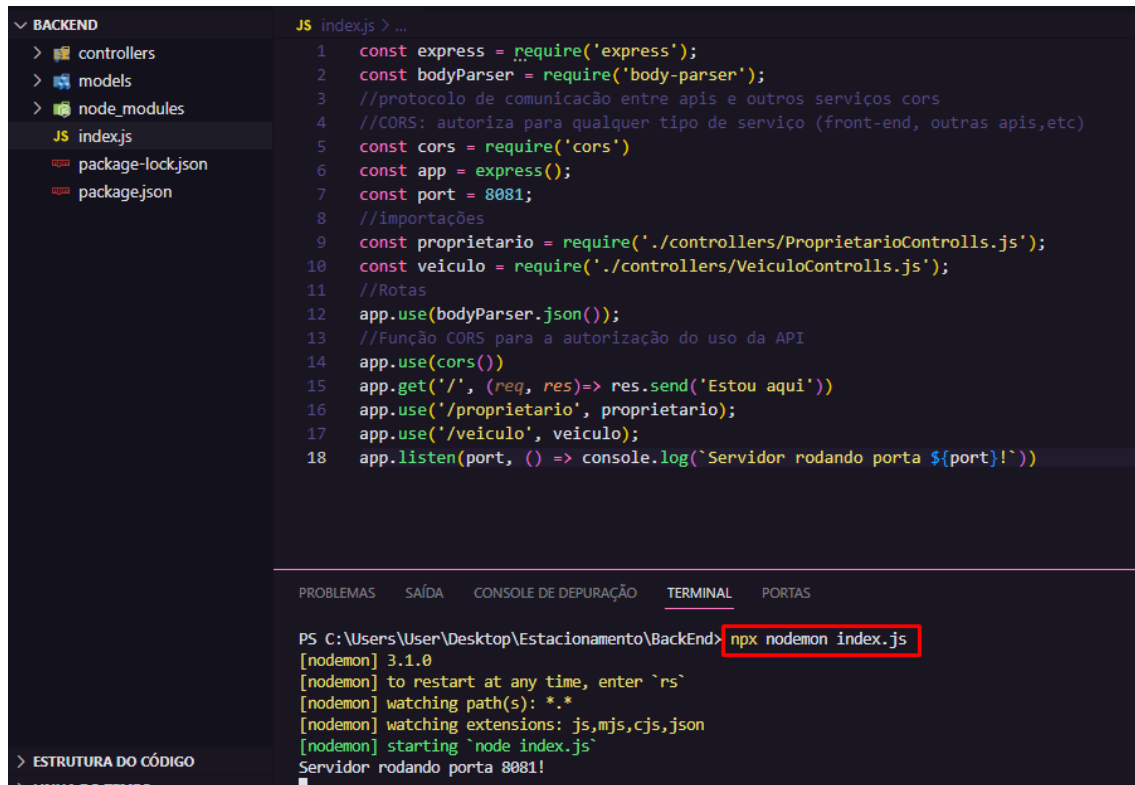
```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  //protocolo de comunicação entre apis e outros serviços cors
4  //CORS: autoriza para qualquer tipo de serviço (front-end, outras apis,etc)
5  const cors = require('cors')
6  const app = express();
7  const port = 8081;
8  //importações
9  const proprietario = require('./controllers/ProprietarioControlls.js');
10 const veiculo = require('./controllers/VeiculoControlls.js');
11 //Rotas
12 app.use(bodyParser.json());
13 //Função CORS para a autorização do uso da API
14 app.use(cors())
15 app.get('/', (req, res)=> res.send('Estou aqui'))
16 app.use('/proprietario', proprietario);
17 app.use('/veiculo', veiculo);
18 app.listen(port, () => console.log(`Servidor rodando porta ${port}!`))
```

Código do index.js

```
const express = require('express');
const bodyParser = require('body-parser');
//protocolo de comunicação entre apis e outros serviços cors
//CORS: autoriza para qualquer tipo de serviço (front-end, outras
apis,etc)
const cors = require('cors')
const app = express();
const port = 8081;
//importações
const proprietario = require('./controllers/ProprietarioControlls.js');
const veiculo = require('./controllers/VeiculoControlls.js');
//Rotas
app.use(bodyParser.json());
//Função CORS para a autorização do uso da API
app.use(cors())
app.get('/', (req, res)=> res.send('Estou aqui'))
app.use('/proprietario', proprietario);
app.use('/veiculo', veiculo);
app.listen(port, () => console.log(`Servidor rodando porta ${port}!`))
```


Vamos rodar nosso servidor

No terminal na pasta raiz do seu projeto digite **npx nodemon index.js**



The image shows a code editor with a file explorer on the left and a terminal at the bottom. The file explorer shows a project structure with a 'BACKEND' folder containing 'controllers', 'models', 'node_modules', 'index.js', 'package-lock.json', and 'package.json'. The 'index.js' file is open in the editor, showing the following code:

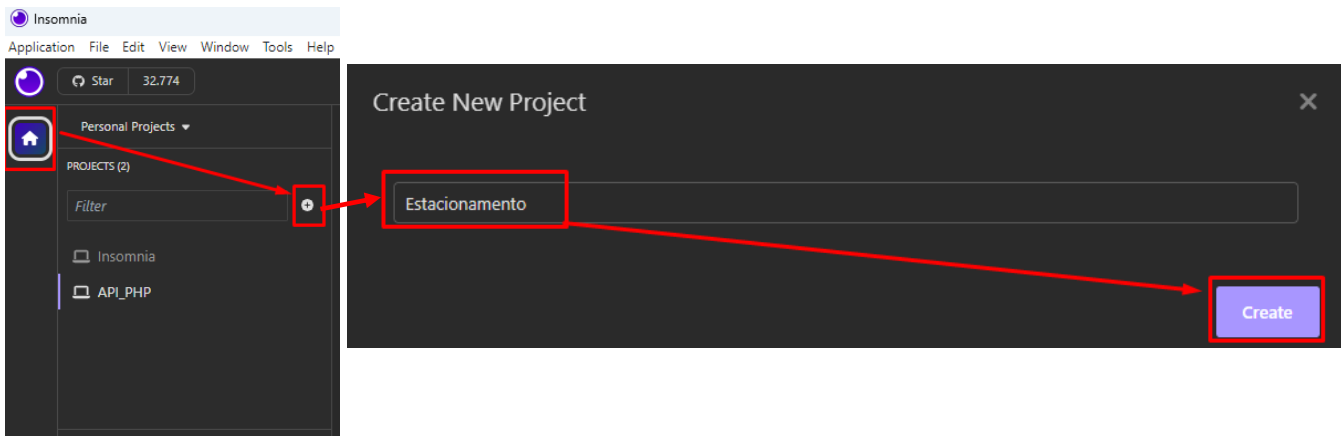
```
1  const express = require('express');
2  const bodyParser = require('body-parser');
3  //protocolo de comunicação entre apis e outros serviços cors
4  //CORS: autoriza para qualquer tipo de serviço (front-end, outras apis,etc)
5  const cors = require('cors')
6  const app = express();
7  const port = 8081;
8  //importações
9  const proprietario = require('./controllers/ProprietarioControlls.js');
10 const veiculo = require('./controllers/VeiculoControlls.js');
11 //Rotas
12 app.use(bodyParser.json());
13 //Função CORS para a autorização do uso da API
14 app.use(cors())
15 app.get('/', (req, res)=> res.send('Estou aqui'))
16 app.use('/proprietario', proprietario);
17 app.use('/veiculo', veiculo);
18 app.listen(port, () => console.log(`Servidor rodando porta ${port}!`))
```

The terminal window at the bottom shows the command `npx nodemon index.js` being executed. The output of the command is:

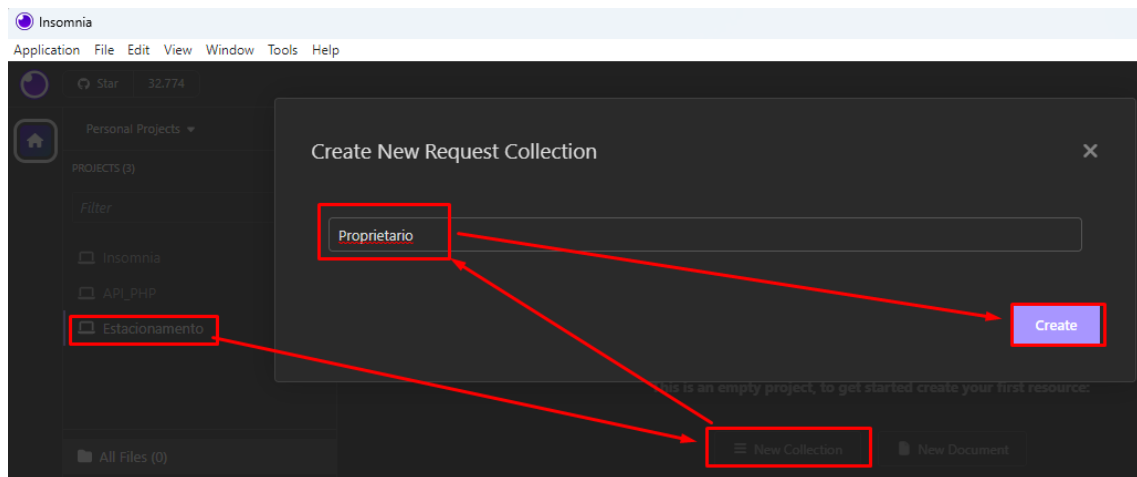
```
PS C:\Users\User\Desktop\Estacionamento\Backend> npx nodemon index.js
[nodemon] 3.1.0
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,cjs,json
[nodemon] starting `node index.js`
Servidor rodando porta 8081!
```

Testando a API

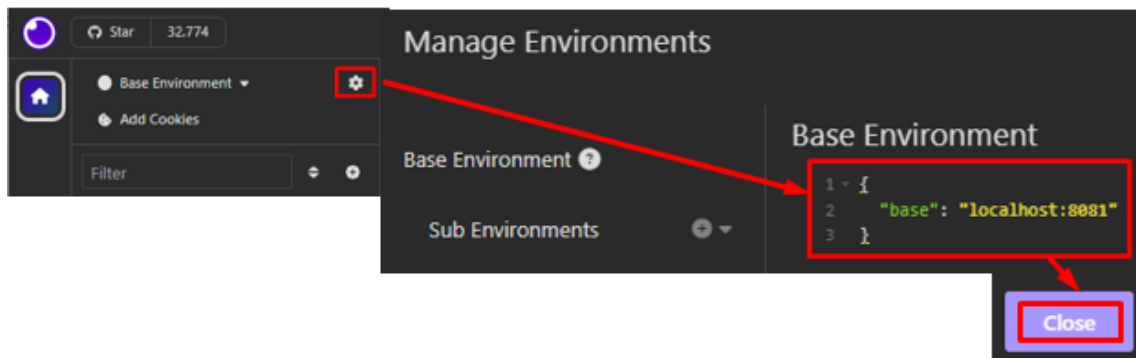
Abra o insomnia e crie um projeto chamado Estacionamento



Dentro de estacionamento crie a Collection Proprietário

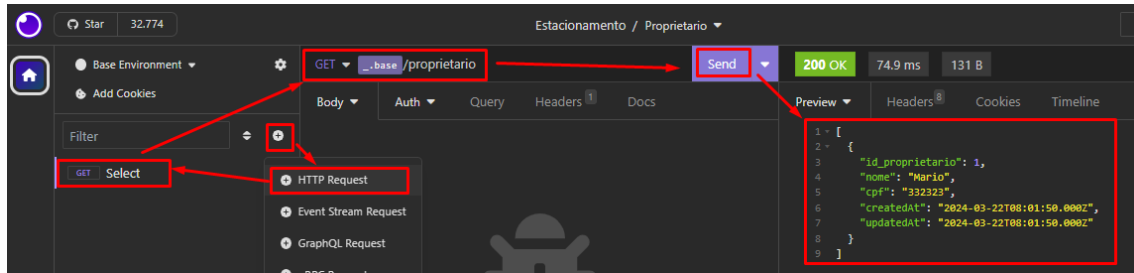


Adicione o base com o nosso ip do servidor localhost:8081

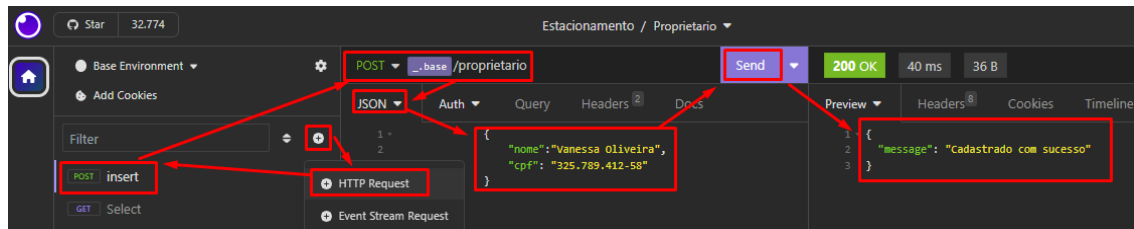


Testando as operações CRUD de Proprietário

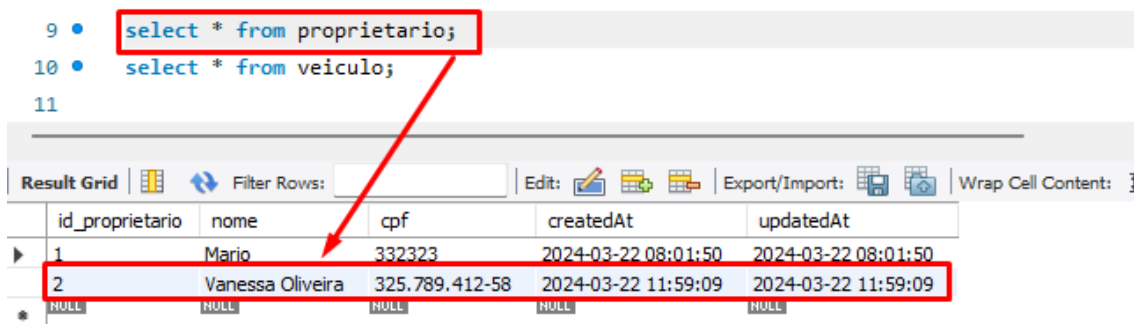
Selecionando todos os proprietários



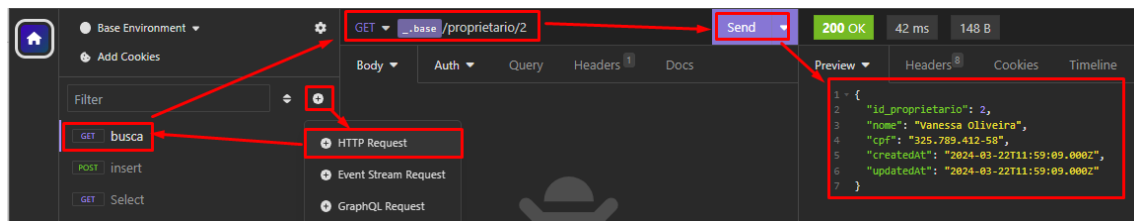
Cadastrando proprietário



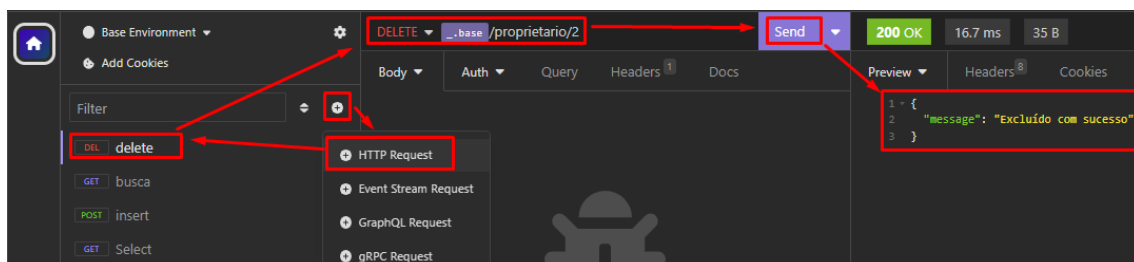
Verificando no banco de dados se aconteceu a inserção



Efetuando a busca por ID



Deletando informações pelo ID



Verificando no banco de dados se a instrução foi deletada

```
9 • select * from proprietario;
10 • select * from veiculo;
11
```

id_proprietario	nome	cpf	createdAt	updatedAt
1	Mario	332323	2024-03-22 08:01:50	2024-03-22 08:01:50

Alterando o proprietário

The screenshot shows a REST client interface with a PUT request to `base/proprietario/1`. The request body is a JSON object: `{ "id": 1, "nome": "Mario de Jesus", "cpf": "222.222.222-22" }`. The response is a 200 OK status with a message: `{ "message": "Atualizado com sucesso" }`. Red arrows indicate the flow from the request configuration to the response.

Verificando se a informação foi alterada

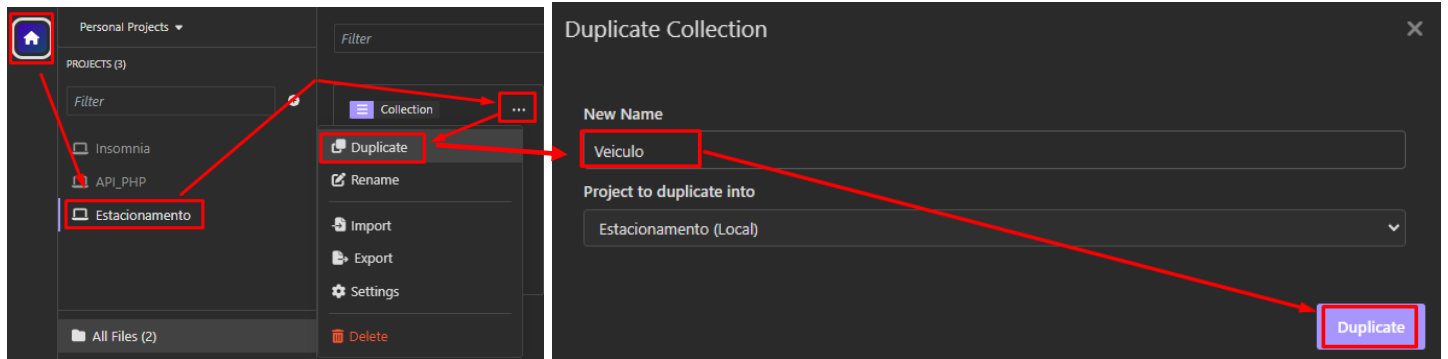
```
9 • select * from proprietario;
10 • select * from veiculo;
11
```

id_proprietario	nome	cpf	createdAt	updatedAt
1	Mario de Jesus	222.222.222-22	2024-03-22 08:01:50	2024-03-22 12:11:36

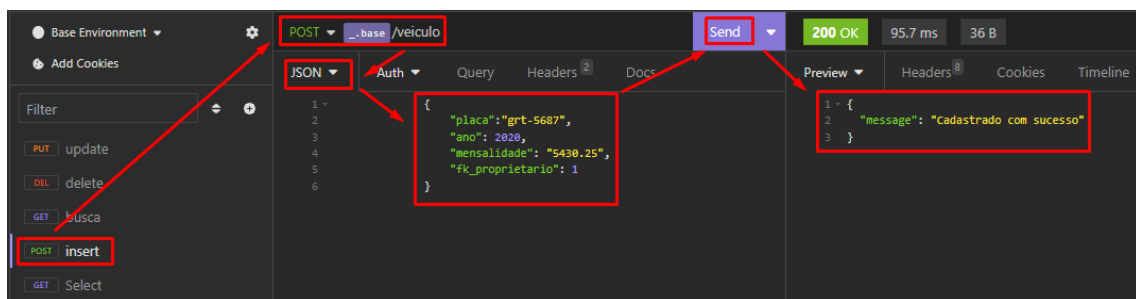
Testando as operações CRUD do Veículo

Volte para a home do insomnia, clique no estacionamento e duplique a collection proprietário, adicione o nome de Veiculo e clique em duplicar.

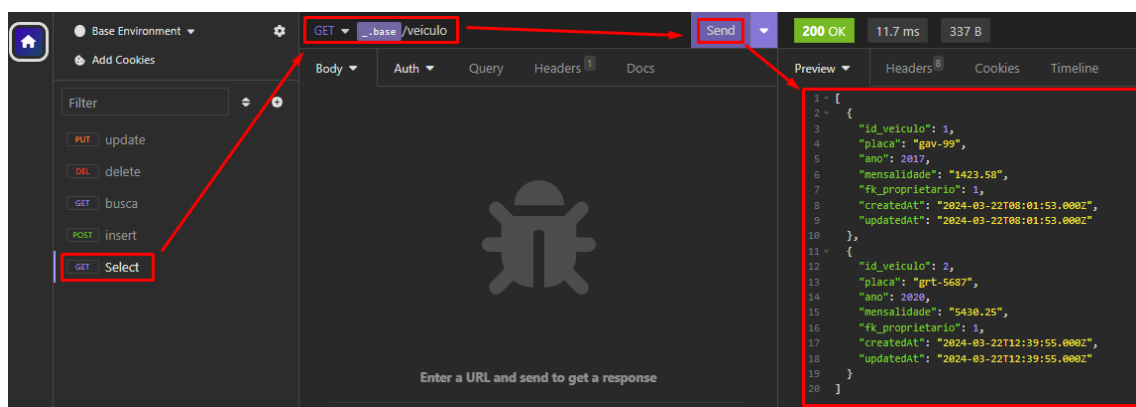
Como duplicamos a Collection todos os métodos criados também foram duplicados, então vai ficar mais fácil de fazer os testes fazendo alterações básicas.



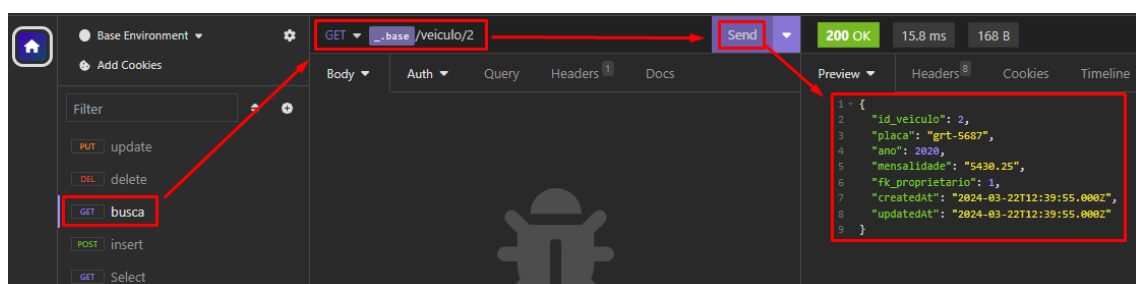
Clique em insert e faça uma inserção de veículo



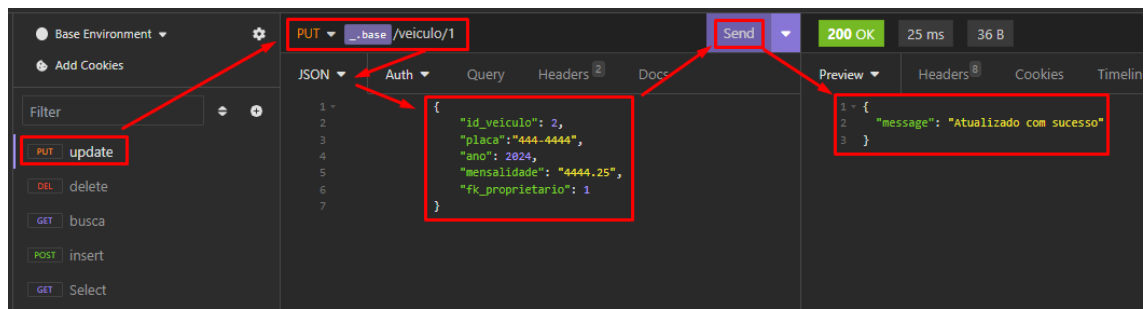
Clique em select e mostre todos os veículos



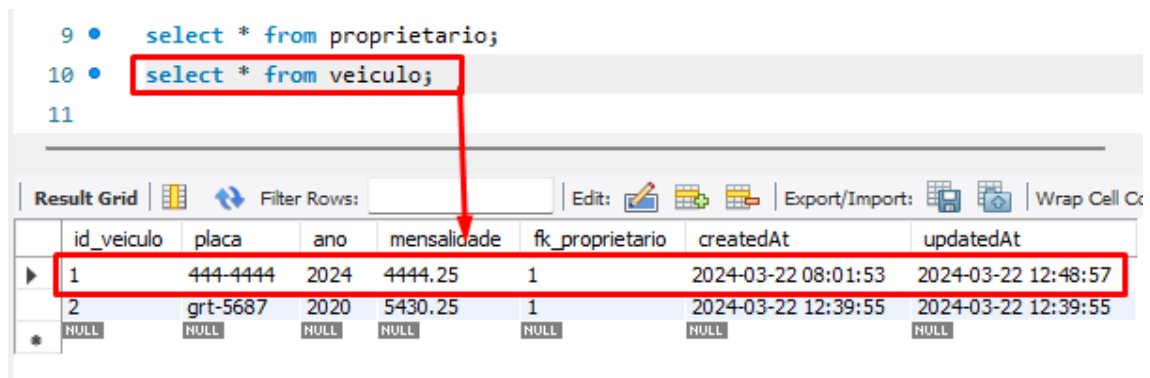
Clique em busca e faça uma pesquisa pelo ID do veículo



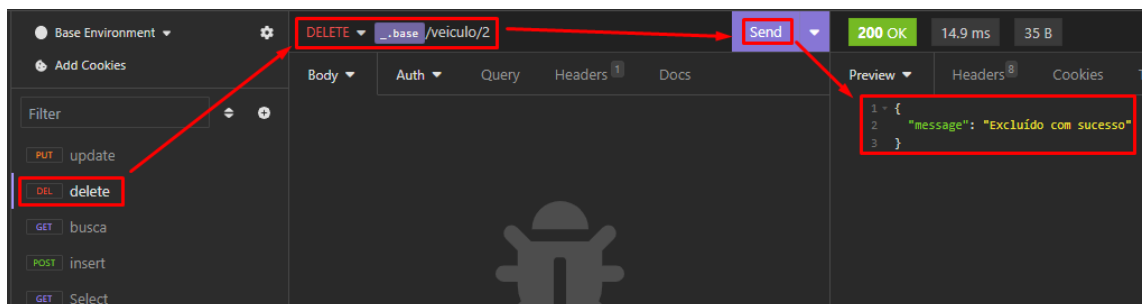
Clique em **update** e faça a **alteração** de um **veículo**



Verifique no banco de dados se a alteração aconteceu



Clique em **delete** e faça a **exclusão** de um dos **veículos**.



Depois verifique no **banco** de dados.