

# Documetação:

URL: <http://localhost:3001/api/veiculos-eletricos>

Com este sistema, podemos monitorar e analisar o desempenho dos veículos elétricos em tempo real. Ele permite a coleta de dados essenciais sobre a eficiência energética, a autonomia e o uso dos veículos elétricos. Esses dados são cruciais para otimizar a gestão de frotas, planejar a manutenção preventiva e melhorar a infraestrutura de recarga.

JSON:

```
{
  "id": "veic001",
  "tipo": "carro",
  "modelo": "Modelo X",
  "bateriaCapacidade": 75,
  "bateriaNivel": 85,
  "autonomia": 400,
  "consumoEnergia": 0.18,
  "estacaoRecargaId": "st123",
  "dataUltimaRecarga": "2024-11-21T20:35:00Z",
  "distanciaPercorrida": 15000
}
```

```
const VeiculoEletricoSchema = new mongoose.Schema({
  id: { type: String, required: true },
  tipo: { type: String, required: true },
  modelo: { type: String, required: true },
  bateriaCapacidade: { type: Number, required: true },
  bateriaNivel: { type: Number, required: true },
  autonomia: { type: Number, required: true },
  consumoEnergia: { type: Number, required: true },
  estacaoRecargaId: { type: String, required: true },
  dataUltimaRecarga: { type: Date, required: true },
  distanciaPercorrida: { type: Number, required: true },
});

const VeiculoEletrico = mongoose.model('VeiculoEletrico', VeiculoEletricoSchema);

export default VeiculoEletrico;
```

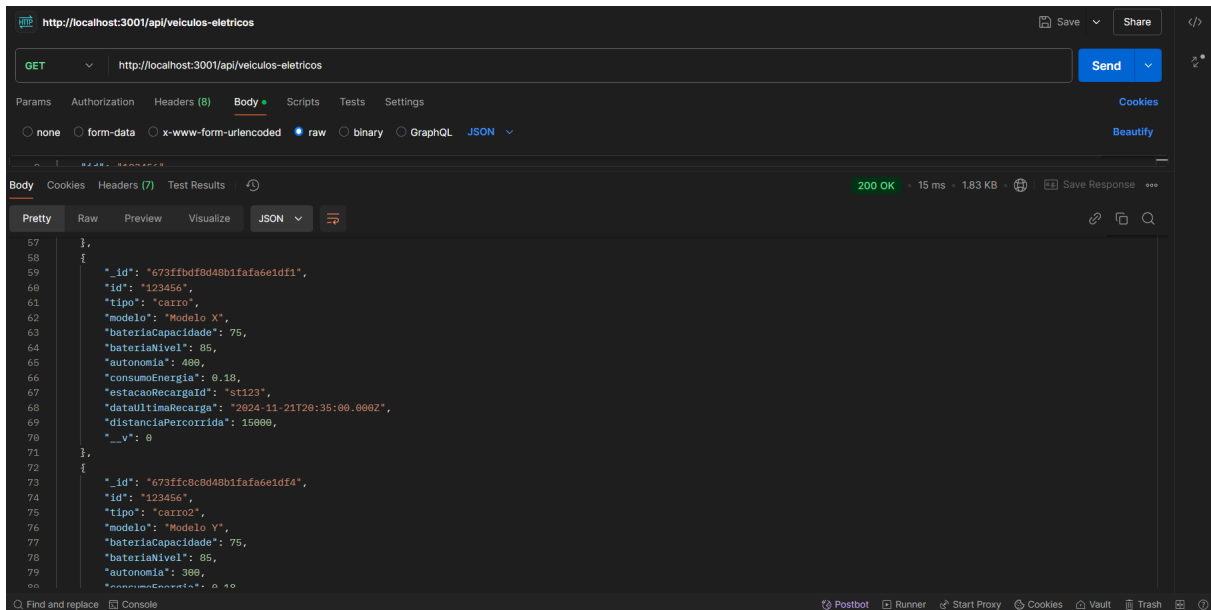
The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:3001/api/veiculos-eletricos`
- Method:** `POST`
- Body (raw):**

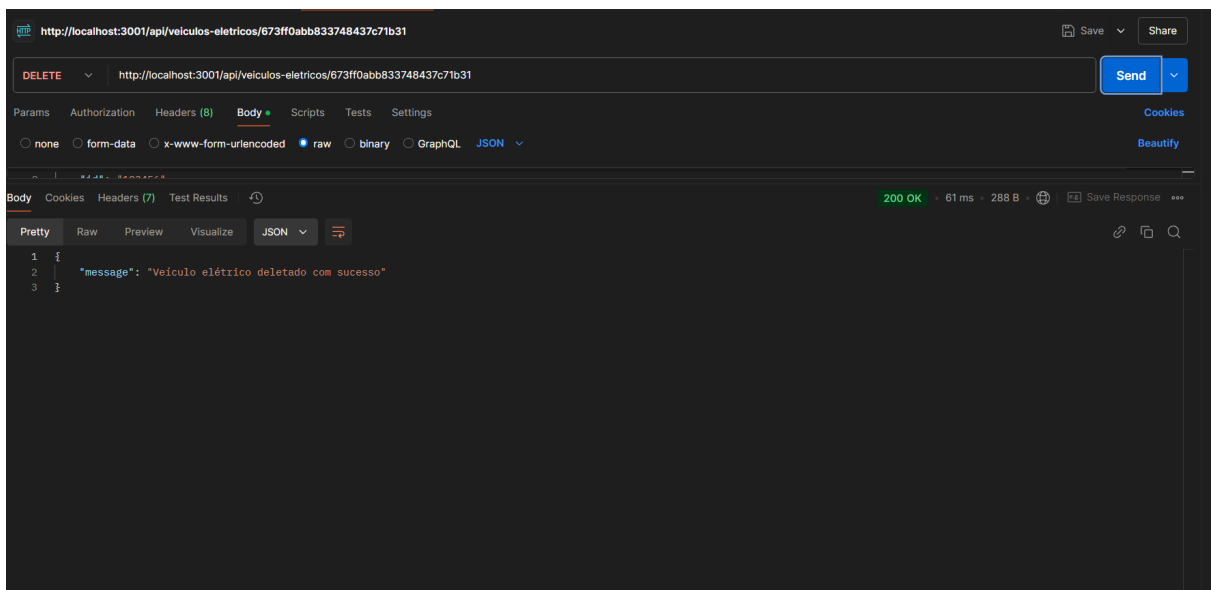
```
{
  "id": "123456",
  "tipo": "carro",
  "modelo": "Modelo X",
  "bateriaCapacidade": 75, // em kwh
  "bateriaNivel": 85, // em %
  "autonomia": 400, // em km
  "consumoEnergia": 0.18, // kwh/km
  "estacaoRecargaId": "st123",
  "dataUltimaRecarga": "2024-11-21T20:35:00Z", // Formato ISO 8601
}
```
- Response (JSON):**

```
{
  "id": "123456",
  "tipo": "carro",
  "modelo": "Modelo X",
  "bateriaCapacidade": 75,
  "bateriaNivel": 85,
  "autonomia": 400,
  "consumoEnergia": 0.18,
  "estacaoRecargaId": "st123",
  "dataUltimaRecarga": "2024-11-21T20:35:00.000Z",
  "distanciaPercorrida": 15000,
  "_id": "673ffbf8d48b1fa6e1df1",
  "__v": 0
}
```
- Status:** 201 Created
- Time:** 97 ms
- Size:** 514 B

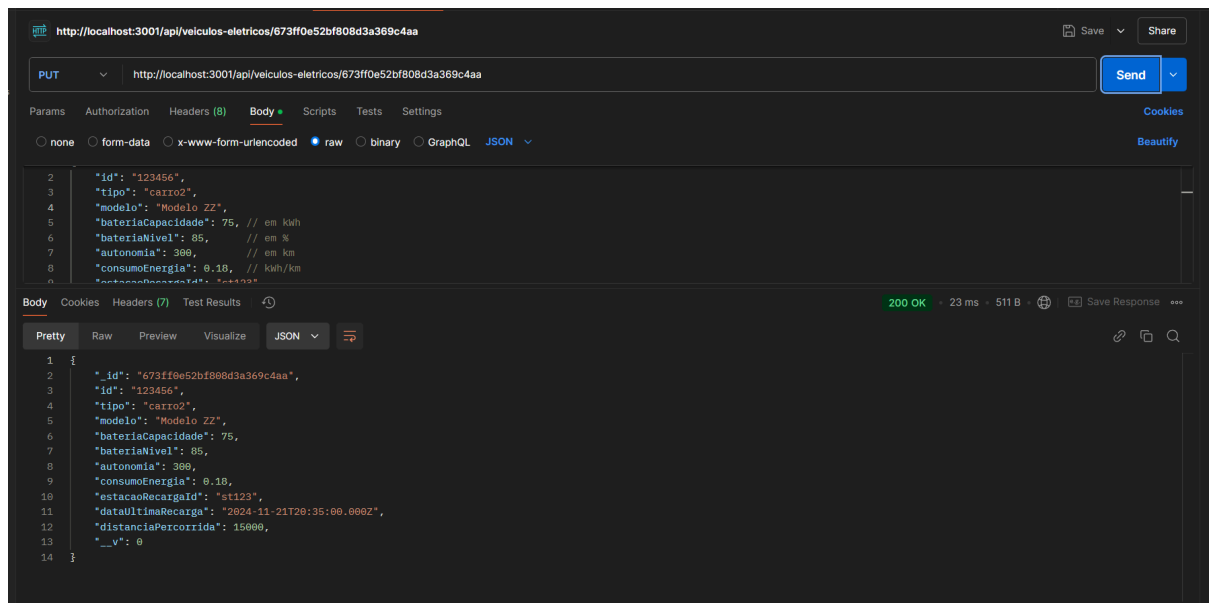
POST: criação de um carro da Frota elétrica - com uso único e exclusivo de MongoDB e JS.



GET: Pegando dados de um carro da Frota elétrica - com uso único e exclusivo de MongoDB e JS.



Delete: Apagando dados de um carro da Frota elétrica - com uso único e exclusivo de MongoDB e JS.



PUT: Atualizando dados de um carro da Frota elétrica - com uso único e exclusivo de MongoDB e JS.

## Postos de recarga:

URL: <http://localhost:3001/api/estacoes-recarga>

Através deste sistema, podemos monitorar e analisar os postos de carregamento com maior movimento. Isso nos permite coletar dados e gerar métricas detalhadas sobre o uso das estações de recarga em toda a cidade. Essas informações são essenciais para otimizar a distribuição dos recursos e planejar futuras expansões de forma eficiente.

JSON

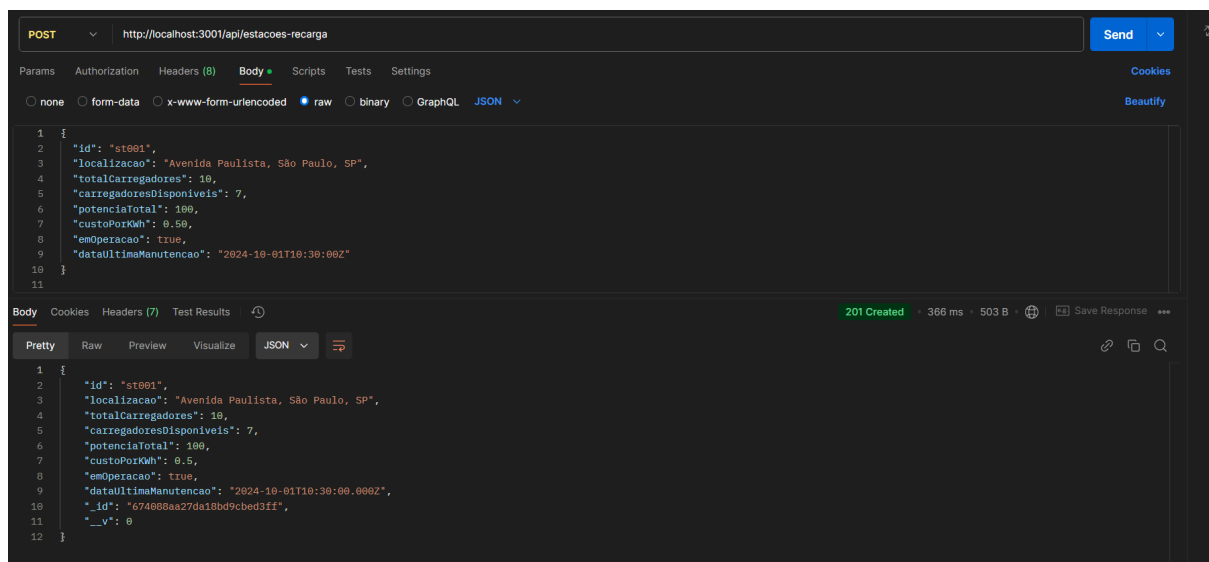
```
{  
  "id": "st002",
```

```
"localizacao": "Rua das Flores, Rio de Janeiro, RJ",
"totalCarregadores": 8,
"carregadoresDisponiveis": 7,
"potenciaTotal": 80,
"custoPorKWh": 0.45,
"emOperacao": false,
"dataUltimaManutencao": "2024-11-10T14:20:00Z"
}
```

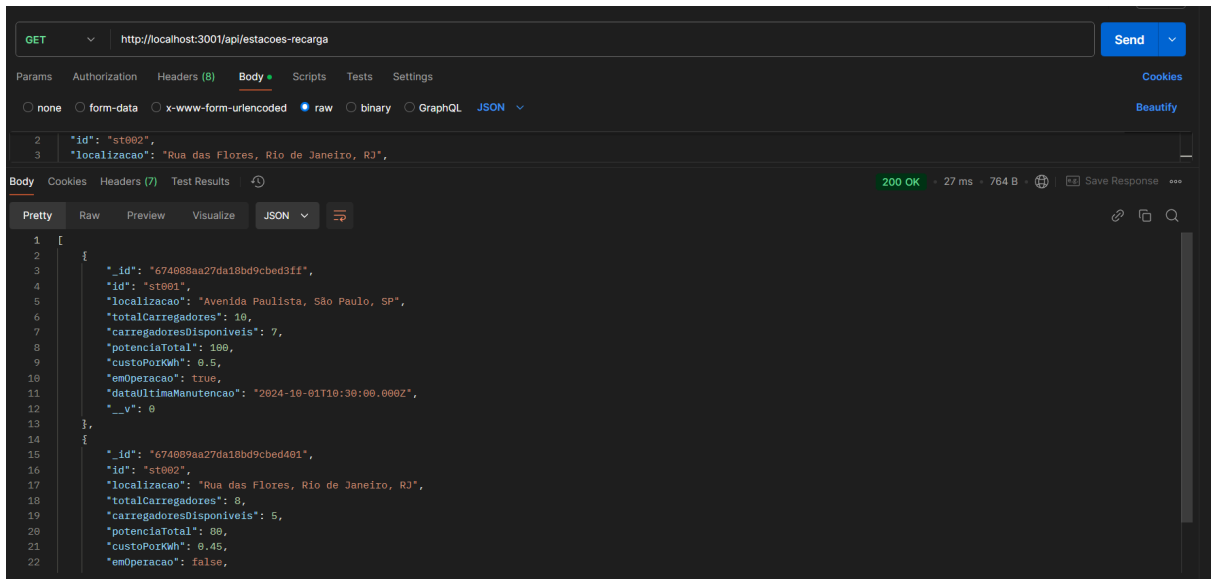
```
const EstacaoRecargaSchema = new mongoose.Schema({
  id: { type: String, required: true },
  localizacao: { type: String, required: true },
  totalCarregadores: { type: Number, required: true },
  carregadoresDisponiveis: { type: Number, required: true },
  potenciaTotal: { type: Number, required: true },
  custoPorKWh: { type: Number, required: true },
  emOperacao: { type: Boolean, required: true },
  dataUltimaManutencao: { type: Date, required: true },
});

const EstacaoRecarga = mongoose.model('EstacaoRecarga', EstacaoRecargaSchema);

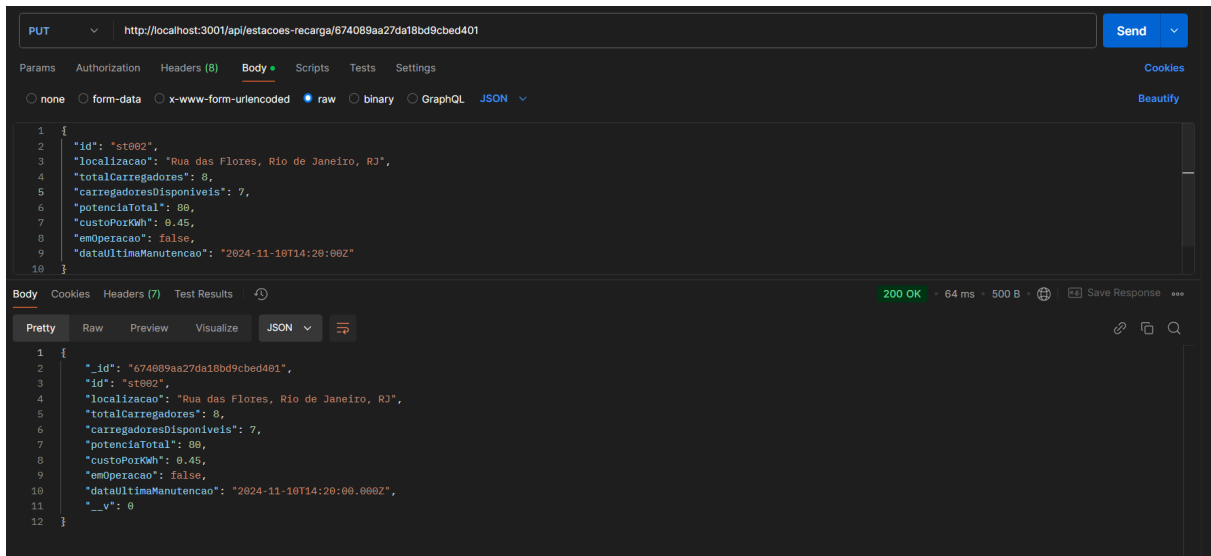
export default EstacaoRecarga;
```



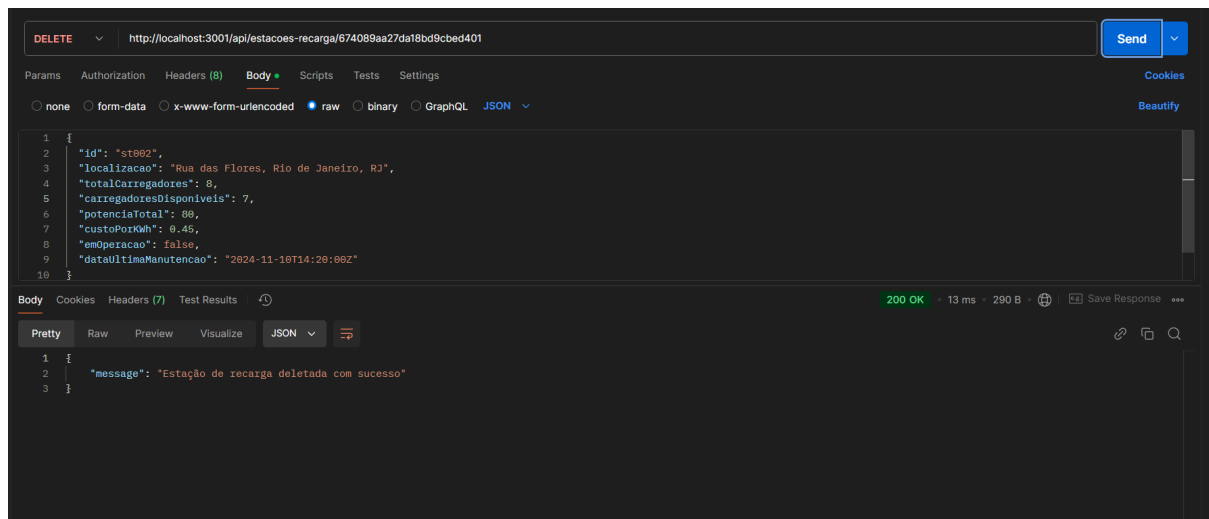
POST: criação de um Postos de recarregamento- com uso único e exclusivo de MongoDB e JS.



GET: busca de Postos de recarregamento- com uso único e exclusivo de MongoDB e JS.



PUT: Atualização do Postos de recarregamento- com uso único e exclusivo de MongoDB e JS



Delete: deletar o Postos de recarregamento- com uso único e exclusivo de MongoDB e JS

## Aplicando Redis banco em cash:

O Redis pode ser uma ferramenta extremamente valiosa para sua aplicação, atuando como um cache de dados em memória. Isso permite a rápida recuperação de dados frequentemente acessados, reduzindo significativamente a carga sobre seu banco de dados principal, como o MongoDB. Ao armazenar em memória respostas frequentes, o Redis oferece baixa latência e alto desempenho, resultando em uma experiência de usuário mais fluida e rápida. Esse benefício é especialmente útil para operações que requerem acessos rápidos e repetidos a grandes volumes de dados.

Além disso, o Redis pode melhorar a gestão de sessões de usuário, fornecendo um mecanismo eficiente e escalável para armazenar informações de sessão. Isso é crucial para manter a persistência e continuidade da experiência do usuário em aplicações web. O Redis também suporta a implementação de filas de mensagens e sistemas de publicação/assinatura, que permitem a comunicação assíncrona entre diferentes partes da aplicação. Essa capacidade de comunicação eficiente entre componentes desacoplados melhora a escalabilidade e a modularidade da sua aplicação, tornando-a mais robusta e preparada para lidar com grandes volumes de tráfego.

## Plicando o Redis:

```
import EstacaoRecargaService from '../services/EstacaoRecargaService.js';
import redisClient from '../connections/redis.js';

class EstacaoRecargaController {
  static async listarEstacoes(req, res) {
    try {
      const cacheKey = 'estacoes-recarga';
      const cacheData = await redisClient.get(cacheKey);

      if (cacheData) {
        console.log('Dados retornados do cache');
        return res.json(JSON.parse(cacheData));
      }

      const estacoes = await EstacaoRecargaService.listarEstacoes();
      await redisClient.setex(cacheKey, 3600, JSON.stringify(estacoes));
      console.log('Dados armazenados no cache');
      res.json(estacoes);
    } catch (err) {
      res.status(500).json({ error: 'Erro ao buscar estações de recarga' });
    }
  }

  static async criarEstacao(req, res) {

```

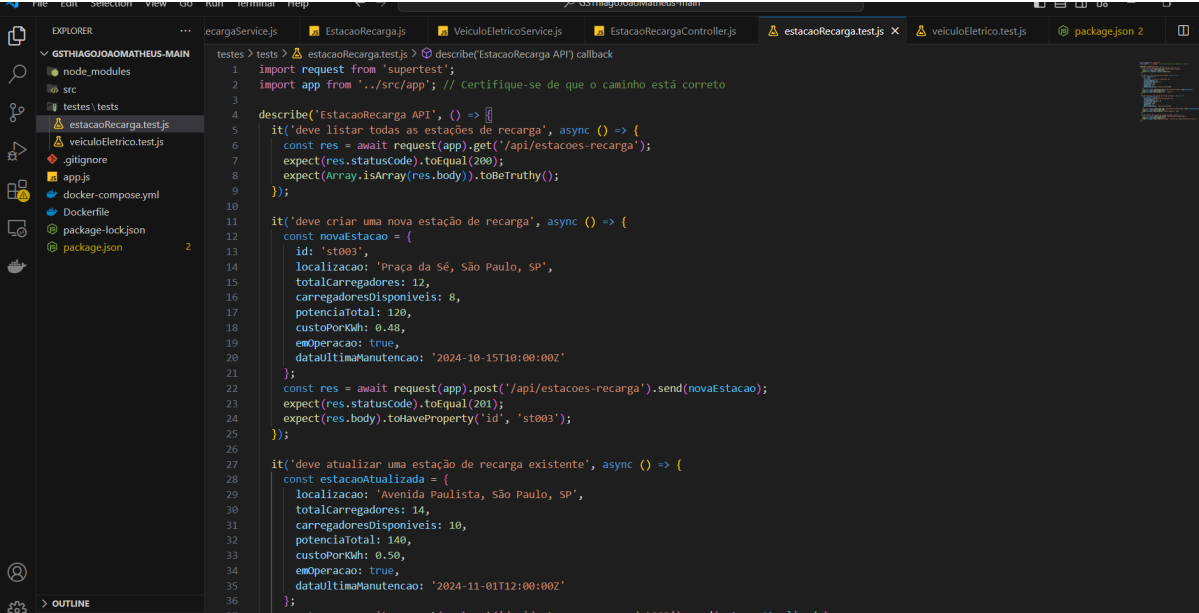
O uso do Redis neste caso pode melhorar significativamente o desempenho e a eficiência da aplicação. Quando a função `listarEstacoes` é chamada, ela primeiro verifica se os dados das estações de recarga estão armazenados no cache do Redis. Se os dados estiverem no cache, eles são retornados diretamente, o que reduz o tempo de resposta e a carga no banco de dados. Isso é especialmente útil para dados que não mudam com frequência e são acessados repetidamente, proporcionando um acesso rápido e eficiente.

## Testes



npm install --save-dev jest

npm install --save-dev supertest



The screenshot shows a VS Code editor window with a project named 'GSTHAGIOJOAO-MATHEUS-MAIN'. The Explorer sidebar on the left shows the file structure with folders for 'node\_modules', 'src', and 'testes'. The 'testes' folder is expanded, showing 'estacaoRecarga.test.js' and 'veiculoElettrico.test.js'. The 'estacaoRecarga.test.js' file is selected and its content is displayed in the main editor. The code is a Jest test file for an API, using 'supertest' for HTTP requests. It includes three test cases: a 'describe' block for the API, an 'it' block for listing all charging stations, and another 'it' block for creating a new charging station. The code is written in JavaScript with ES6 syntax, including 'import' and 'export' statements. The test cases use 'expect' to verify the status code and the body of the responses. The 'package.json' file is also visible in the Explorer sidebar, showing the project's dependencies and scripts.

```
1 import request from 'supertest';
2 import app from '../src/app'; // Certifique-se de que o caminho está correto
3
4 describe('EstacaoRecarga API', () => {
5   it('deve listar todas as estações de recarga', async () => {
6     const res = await request(app).get('/api/estacoes-recarga');
7     expect(res.statusCode).toEqual(200);
8     expect(Array.isArray(res.body)).toBeTruthy();
9   });
10
11   it('deve criar uma nova estação de recarga', async () => {
12     const novaEstacao = {
13       id: 'st003',
14       localizacao: 'Praça da Sé, São Paulo, SP',
15       totalCarregadores: 12,
16       carregadoresDisponiveis: 8,
17       potenciaTotal: 120,
18       custoPorKwh: 0.48,
19       emOperacao: true,
20       dataUltimaManutencao: '2024-10-15T10:00:00Z'
21     };
22     const res = await request(app).post('/api/estacoes-recarga').send(novaEstacao);
23     expect(res.statusCode).toEqual(201);
24     expect(res.body).toHaveProperty('id', 'st003');
25   });
26
27   it('deve atualizar uma estação de recarga existente', async () => {
28     const estacaoAtualizada = {
29       localizacao: 'Avenida Paulista, São Paulo, SP',
30       totalCarregadores: 14,
31       carregadoresDisponiveis: 10,
32       potenciaTotal: 140,
33       custoPorKwh: 0.50,
34       emOperacao: true,
35       dataUltimaManutencao: '2024-11-01T12:00:00Z'
36     };
37     const res = await request(app).put('/api/estacoes-recarga/st003').send(estacaoAtualizada);
```