Passo 1: Configuração inicial

1. Abra o VS Code.
2. Crie uma nova pasta para o projeto (abrir no VS Code).
3. Abra o terminal integrado do VS Code (Menu "Terminal" -> "New Terminal").

Passo 2: Inicializar o projeto

1. No terminal, execute o comando `npm init -y` para inicializar um projeto Node.js padrão com as configurações padrão (O terminal deve estar dentro da pasta criada).
2. Execute o comando `npm install express` para instalar o pacote Express.
3. Execute o comando `npm install body-parser` para instalar o pacote body-parser.

Passo 3: Criar o arquivo principal

1. Crie um novo arquivo chamado `index.js` na pasta do projeto.
2. Abra o arquivo `index.js` no VS Code.

Passo 4: Importar as dependências

1. No arquivo `index.js`, adicione o seguinte código no topo do arquivo:

```
const express = require('express');
const bodyParser = require('body-parser');
```

Passo 5: Criar a aplicação Express

1. Ainda no arquivo `index.js`, adicione o seguinte código abaixo das importações:

```
const app = express();
app.use(bodyParser.json());
```

Passo 6: Definir o array para armazenar os veículos

1. Ainda no arquivo `index.js`, adicione o seguinte código abaixo da criação da aplicação Express:

```
let vehicles = [];
```

Passo 7: Criar a rota para obter todos os veículos

1. Ainda no arquivo index.js, adicione o seguinte código abaixo da definição dos dados de exemplo:

```
app.get('/vehicles', (req, res) => {
      res.json(vehicles);
});
```

Passo 8: Criar a rota para obter um veículo pelo número da placa

1. Ainda no arquivo index.js, adicione o seguinte código abaixo da rota para obter todos os veículos:

```
app.get('/vehicles/:placa', (req, res) => {
      const { placa } = req.params;
      const vehicle = vehicles.find(v => v.placa === placa);
      if (vehicle) {
      res.json(vehicle);
      } else {
            res.status(404).json({ message: 'Veículo não encontrado.' });
      }
});
```

Passo 9: Criar a rota para cadastrar um novo veículo

1. Ainda no arquivo index.js, adicione o seguinte código abaixo da rota para obter um veículo pelo número da placa:

```
app.post('/vehicles', (req, res) => {
      const { placa, marca, modelo, ano } = req.body;
      const vehicle = { placa, marca, modelo, ano };
      vehicles.push(vehicle);
      res.status(201).json({ message: 'Veículo cadastrado com sucesso.' });
 });
```

Passo 10: Criar a rota para atualizar as informações de um veículo pelo número da placa

1. Ainda no arquivo index.js, adicione o seguinte código abaixo da rota para cadastrar um novo veículo:

```
app.put('/vehicles/:placa', (req, res) => {
     const { placa } = req.params;
     const { marca, modelo, ano } = req.body;
     const vehicle = vehicles.find(v => v.placa === placa);
     if (vehicle) {
           vehicle.marca = marca || vehicle.marca;
           vehicle.modelo = modelo || vehicle.modelo;
           vehicle.ano = ano || vehicle.ano;
           res.json({ message: 'Informações do veículo atualizadas com sucesso.' });
     } else {
           res.status(404).json({ message: 'Veículo não encontrado.' });
      }
});
```

Passo 11: Criar a rota para excluir um veículo pelo número da placa

1. Ainda no arquivo index.js, adicione o seguinte código abaixo da rota para atualizar as informações de um veículo pelo número da placa:

```
app.delete('/vehicles/:placa', (req, res) => {
     const { placa } = req.params;
     const vehicleIndex = vehicles.findIndex(v => v.placa === placa);
     if (vehicleIndex !== -1) {
           vehicles.splice(vehicleIndex, 1);
           res.json({ message: 'Veículo excluído com sucesso.' });
      } else {
           res.status(404).json({ message: 'Veículo não encontrado.' });
      }
});
```

Passo 12: Iniciar o servidor

1. Ainda no arquivo index.js, adicione o seguinte código no final do arquivo:

```
const port = 3000;
app.listen(port, () => {
     console.log(`Servidor rodando em http://localhost:${port}`);
});
```

Passo 13: Executar o servidor

1. No terminal, execute o comando `node index.js` para iniciar o servidor.
2. Você verá a mensagem "Servidor rodando em http://localhost:3000" no terminal, indicando que o servidor está em execução.

Passo 14: Testar as rotas no Postman

1. Abra o Postman (ou qualquer outra ferramenta para testar APIs REST).
2. Configure uma nova requisição GET para a URL `http://localhost:3000/vehicles` para obter todos os veículos.
3. Configure uma nova requisição GET para a URL `http://localhost:3000/vehicles/{placa}` substituindo `{placa}` pelo número da placa de um veículo existente para obter um veículo específico.
4. Configure uma nova requisição POST para a URL `http://localhost:3000/vehicles` e adicione um objeto JSON no corpo da requisição contendo os dados do novo veículo para cadastrar um novo veículo.
5. Configure uma nova requisição PUT para a URL `http://localhost:3000/vehicles/{placa}` substituindo `{placa}` pelo número da placa de um veículo existente e adicione um objeto JSON no corpo da requisição contendo os dados atualizados do veículo para atualizar as informações de um veículo.
6. Configure uma nova requisição DELETE para a URL `http://localhost:3000/vehicles/{placa}` substituindo `{placa}` pelo número da placa de um veículo existente para excluir um veículo.

TESTANDO COM O REACT (Front-End)

Passo 1: Separar os arquivos do servidor

1. Instale o pacote `cors`:

```
npm install cors
```

2. Crie três arquivos: `routes.js`, `controller.js` e `server.js`

routes.js:

```
const express = require('express');
const router = express.Router();
const controller = require('./controller');
router.get('/vehicles', controller.getVehicles);
router.get('/vehicles/:placa', controller.getVehicleByPlaca);
router.post('/vehicles', controller.createVehicle);
router.put('/vehicles/:placa', controller.updateVehicle);
router.delete('/vehicles/:placa', controller.deleteVehicle);
module.exports = router;
```

controller.js:

```
let vehicles = [];
function getVehicles(req, res) {
res.json(vehicles);
}
function getVehicleByPlaca(req, res) {
const { placa } = req.params;
const vehicle = vehicles.find(v => v.placa === placa);
if (vehicle) {
res.json(vehicle);
} else {
res.status(404).json({ message: 'Veículo não encontrado.' });
}
}
function createVehicle(req, res) {
const { placa, marca, modelo, ano } = req.body;
const vehicle = { placa, marca, modelo, ano };
vehicles.push(vehicle);
res.status(201).json({ message: 'Veículo cadastrado com sucesso.' });
```

```
}
function updateVehicle(req, res)
{ const { placa } = req.params;
const { marca, modelo, ano } = req.body;
const vehicle = vehicles.find(v => v.placa === placa);
if (vehicle) {
vehicle.marca = marca || vehicle.marca;
vehicle.modelo = modelo || vehicle.modelo;
vehicle.ano = ano || vehicle.ano;
res.json({ message: 'Informações do veículo atualizadas com sucesso.' });
} else {
res.status(404).json({ message: 'Veículo não encontrado.' });
}
} function deleteVehicle(req, res) {
const { placa } = req.params;
const vehicleIndex = vehicles.findIndex(v => v.placa === placa);
if (vehicleIndex !== -1) {
vehicles.splice(vehicleIndex, 1);
res.json({ message: 'Veículo excluído com sucesso.' });
} else {
res.status(404).json({ message: 'Veículo não encontrado.' });
}
} module.exports = { getVehicles, getVehicleByPlaca, createVehicle,
updateVehicle, deleteVehicle };
```

server.js:

```
const express = require('express');
const bodyParser = require('body-parser');
const routes = require('./routes');
const cors = require('cors');
const app = express();
app.use(bodyParser.json());
app.use(cors());
app.use('/', routes);
const port = 3000;
app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${port}`);
```

```
});
```

Passo 2: Criar projeto REACT

Passo 1: Criar o projeto React

1. Abra o Visual Studio Code (VS Code) e crie um novo diretório para o seu projeto.
2. Abra o terminal no VS Code (Ctrl + `) e navegue até o diretório do projeto.
3. Execute o seguinte comando para criar um novo projeto React: `npx create-react-app nome-do-projeto`

   (tudo com letras minúsculas)

(Substitua "nome-do-projeto" pelo nome desejado para o seu projeto)

OBS: Caso apareçam erros, reinstalar o pacote npm com os comandos: `npm init -y` e `npm install -g`

4. Aguarde o processo de criação do projeto ser concluído.

Passo 2: Instalar o Axios

1. No terminal, navegue até o diretório do projeto React (se ainda não estiver nele).
2. Execute o seguinte comando para instalar o Axios:

```
npm install axios
```

(A instalação do Axios será realizada no projeto React)

Passo 3: Criar o arquivo .env

1. No diretório do projeto, crie um novo arquivo chamado ".env".
2. Abra o arquivo .env no VS Code.
3. Adicione a seguinte linha ao arquivo .env: `PORT=3001`

(Esta configuração define a porta 3001 para o servidor React)

Passo 4: Atualizar o código do App.js

1. Abra o arquivo App.js no VS Code.

2. Substitua o conteúdo existente pelo seguinte código:

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

const App = () => {
  const [vehicles, setVehicles] = useState([]);
  const [formData, setFormData] = useState({
    placa: '',
    marca: '',
    modelo: '',
    ano: ''
  });

  useEffect(() => {
    // Carrega os veículos ao montar o componente
    fetchVehicles();
  }, []);

  const fetchVehicles = async () => {
    try {
      // Faz uma requisição GET para obter a lista de veículos
      const response = await axios.get(`http://localhost:3000/vehicles`);
      // Atualiza o estado com os veículos obtidos
      setVehicles(response.data);
    } catch (error) {
      console.error(error);
    }
  };

  const handleInputChange = e => {
    // Atualiza o estado do formulário quando o valor de um input é alterado
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleCreateVehicle = async e => {
    e.preventDefault();
    try {
```

```
    // Faz uma requisição POST para criar um novo veículo
    await axios.post(`http://localhost:3000/vehicles`, formData);
    // Limpa o formulário após o cadastro
    setFormData({
      placa: '',
      marca: '',
      modelo: '',
      ano: ''
    });
    // Atualiza a lista de veículos
    fetchVehicles();
  } catch (error) {
    console.error(error);
  }
};

const handleUpdateVehicle = async placa => {
  try {
    // Faz uma requisição PUT para atualizar as informações de um veículo
    await axios.put(`http://localhost:3000/vehicles/${placa}`, formData);
    // Limpa o formulário após a atualização
    setFormData({
      placa: '',
      marca: '',
      modelo: '',
      ano: ''
    });
    // Atualiza a lista de veículos
    fetchVehicles();
  } catch (error) {
    console.error(error);
  }
};

const handleDeleteVehicle = async placa => {
  try {
    // Faz uma requisição DELETE para excluir um veículo
    await axios.delete(`http://localhost:3000/vehicles/${placa}`);
```

```
    // Atualiza a lista de veículos
    fetchVehicles();
  } catch (error) {
    console.error(error);
  }
};

return (
  <div>
    <h1>Veículos</h1>
    <form onSubmit={handleCreateVehicle}>
      <label>
        Placa:
        <input
          type="text"
          name="placa"
          value={formData.placa}
          onChange={handleInputChange}
        />
      </label>
      <label>
        Marca:
        <input
          type="text"
          name="marca"
          value={formData.marca}
          onChange={handleInputChange}
        />
      </label>
      <label>
        Modelo:
        <input
          type="text"
          name="modelo"
          value={formData.modelo}
          onChange={handleInputChange}
        />
      </label>
```

```
        <label>
          Ano:
          <input
            type="text"
            name="ano"
            value={formData.ano}
            onChange={handleInputChange}
          />
        </label>
        <button type="submit">Cadastrar</button>
      </form>
      <ul>
        {vehicles.map(vehicle => (
          <li key={vehicle.placa}>
            {vehicle.placa} - {vehicle.marca} - {vehicle.modelo} - {vehicle.ano}
            <button onClick={() => handleUpdateVehicle(vehicle.placa)}>Atualizar</button>
            <button onClick={() => handleDeleteVehicle(vehicle.placa)}>Excluir</button>
          </li>
        ))}
      </ul>
    </div>
  );
};

export default App;
```

O código do App.js utilizará as rotas http://localhost:3000/vehicles do servidor Node na porta 3000. Certifique-se de que o servidor Node esteja em execução na porta 3000 para receber as solicitações do React.

Passo 5: Executar o servidor React

1. No terminal do VS Code, certifique-se de estar no diretório do projeto React.
2. Execute o seguinte comando para iniciar o servidor React:

```
npm start
```

(O servidor será executado na porta 3001, conforme definido no arquivo .env)

Agora o seu projeto React estará em execução na porta 3001 e se comunicando com o servidor Node na porta 3000 usando o Axios.

Caso queira entender como foi construído o código em REACT.js, abaixo está o passo a passo:

Passo 1: Importar as dependências

- Importe o React, useState, useEffect do pacote 'react'.
- Importe o axios para fazer requisições HTTP.

Passo 2: Definir o componente App

- Crie uma função chamada 'App' que retorna o componente principal.

Passo 3: Configurar os estados

- Declare o estado 'vehicles' utilizando useState e atribua um array vazio como valor inicial.
- Declare o estado 'formData' utilizando useState e atribua um objeto com as chaves 'placa', 'marca', 'modelo' e 'ano' com valores vazios como valor inicial.

Passo 4: Carregar veículos ao montar o componente

- Utilize o useEffect com um array vazio como segundo argumento para carregar os veículos assim que o componente for montado.
- Dentro do useEffect, chame a função 'fetchVehicles'.

Passo 5: Criar a função fetchVehicles

- Crie uma função chamada 'fetchVehicles' que é assíncrona.

- Utilize o try-catch para tratar erros.
- Faça uma requisição GET utilizando o axios para obter a lista de veículos a partir da URL 'http://localhost:3000/vehicles'.
- Atualize o estado 'vehicles' com os dados obtidos da resposta.

Passo 6: Criar a função handleInputChange

- Crie uma função chamada 'handleInputChange' que recebe um evento como parâmetro.
- Dentro da função, atualize o estado 'formData' utilizando o spread operator para manter os valores antigos e atualizar apenas o campo correspondente ao nome do input alterado.

Passo 7: Criar a função handleCreateVehicle

- Crie uma função chamada 'handleCreateVehicle' que recebe um evento como parâmetro.
- Previna o comportamento padrão do formulário utilizando o método 'preventDefault' do evento.
- Dentro da função, utilize o try-catch para tratar erros.
- Faça uma requisição POST utilizando o axios para criar um novo veículo na URL 'http://localhost:3000/vehicles', passando o objeto 'formData' como dados da requisição.
- Limpe o formulário, atualizando o estado 'formData' com valores vazios.
- Atualize a lista de veículos chamando a função 'fetchVehicles'.

Passo 8: Criar a função handleUpdateVehicle

- Crie uma função chamada 'handleUpdateVehicle' que recebe a placa do veículo como parâmetro.
- Dentro da função, utilize o try-catch para tratar erros.
- Faça uma requisição PUT utilizando o axios para atualizar as informações do veículo na URL 'http://localhost:3000/vehicles/placa', passando o objeto 'formData' como dados da requisição.
- Limpe o formulário, atualizando o estado 'formData' com valores vazios.

- Atualize a lista de veículos chamando a função 'fetchVehicles'.

Passo 9: Criar a função handleDeleteVehicle

- Crie uma função chamada 'handleDeleteVehicle' que recebe a placa do veículo como parâmetro.
- Dentro da função, utilize o try-catch para tratar erros.
- Faça uma requisição DELETE utilizando o axios para excluir o veículo na URL 'http://localhost:3000/vehicles/placa'.
- Atualize a lista de veículos chamando a função 'fetchVehicles'.

Passo 10: Renderizar o componente

- No retorno do componente 'App', coloque o código HTML JSX para exibir o título 'Veículos', um formulário com inputs para os campos 'placa', 'marca', 'modelo' e 'ano', um botão para cadastrar e uma lista de veículos.
- No formulário, defina o atributo 'onSubmit' como 'handleCreateVehicle' para chamar a função de cadastro ao enviar o formulário.
- Nos inputs, defina o atributo 'value' como o valor correspondente do estado 'formData' e o atributo 'onChange' como 'handleInputChange' para atualizar o estado do formulário quando o valor é alterado.
- Na lista de veículos, utilize um loop para mapear cada veículo para um elemento 'li'.
- Para cada veículo, exiba as informações da placa, marca, modelo e ano, e adicione botões para atualizar e excluir o veículo, chamando as respectivas funções 'handleUpdateVehicle' e 'handleDeleteVehicle' ao serem clicados.

Passo 11: Exportar o componente

- No final do arquivo, exporte o componente 'App' utilizando 'export default App'.