

Material de apoio

Site: [Geração Tech](#)
Curso: Formação em Desenvolvedor Web - Online
Livro: Material de apoio

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: sexta-feira, 2 ago. 2024, 08:37

Índice

1. Back-End

1.1. Implementando a Classe de Produtos

1.2. Vídeo Aula

1.3. REST API - Implementação

1.4. Vídeo Aula

1. Back-End

Revisão e Preparação

Revisão de Conceitos

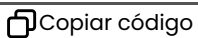
Na última aula, vimos:

- **Classes e Objetos:** Estruturas que permitem organizar dados e comportamentos.
- **Atributos e Métodos:** Dados e funções que pertencem a uma classe.
- **Instanciação:** Processo de criação de um objeto a partir de uma classe.
- **Herança:** Mecanismo pelo qual uma classe pode herdar atributos e métodos de outra.
- **Métodos Estáticos:** Funções que pertencem à classe e podem ser chamadas sem a necessidade de instanciar a classe.

Preparação do Ambiente

Voltando ao nosso projeto, vamos trabalhar na pasta das aulas passadas. O arquivo `server.js` já contém uma configuração básica de servidor com Node.js, incluindo a criação de um servidor, listagem, adição e remoção de produtos.

javascript



Copiar código

```
const http = require('http'); const produtos = require('./produtos'); const server = http.createServer((req, res) => { const { url, method } = req; res.setHeader('Content-Type', 'application/json'); if (url === '/') { res.statusCode = 200; res.end('Servidor Node.js'); } else if (url === '/produtos') { res.statusCode = 200; res.end(JSON.stringify(produtos.listar())); } else if (url === '/produtos/adicionar') { produtos.adicionar({ id: 3, nome: 'Mouse', valor: 50 }); res.statusCode = 201; res.end('Produto adicionado com sucesso'); } else if (url === '/produtos/remover') { produtos.remover(1); res.statusCode = 200; res.end('Produto removido com sucesso'); } else { res.statusCode = 404; res.end('Not Found'); } }); const PORT = 3000; const HOST = '127.0.0.1'; server.listen(PORT, HOST, () => { console.log(`Servidor rodando em http://${HOST}:${PORT}`); });
```


1.1. Implementando a Classe de Produtos

Refatorando com Orientação a Objetos

Vamos criar uma classe `Produto` e mover as funções relacionadas aos produtos para dentro dessa classe.

Criando a Classe `Produto`

javascript

 Copiar código

```
class Produto { static produtos = []; static listar() { return this.produtos; } static adicionar(produto) {
  this.produtos.push(produto); } static remover(id) { this.produtos = this.produtos.filter(produto => produto.id !== id); } }
module.exports = Produto;
```

Refatorando o Servidor

Agora, vamos ajustar nosso servidor para utilizar a nova classe `Produto`.

javascript

 Copiar código

```
const http = require('http'); const Produto = require('./produto'); const server = http.createServer((req, res) => { const {
  url, method } = req; res.setHeader('Content-Type', 'application/json'); if (url === '/') { res.statusCode = 200;
  res.end('Servidor Node.js'); } else if (url === '/produtos') { res.statusCode = 200;
  res.end(JSON.stringify(Produto.listar())); } else if (url === '/produtos/adicionar') { Produto.adicionar({ id: 3, nome:
  'Mouse', valor: 50 }); res.statusCode = 201; res.end('Produto adicionado com sucesso'); } else if (url ===
  '/produtos/remover') { Produto.remover(1); res.statusCode = 200; res.end('Produto removido com sucesso'); } else {
  res.statusCode = 404; res.end('Not Found'); } }); const PORT = 3000; const HOST = '127.0.0.1'; server.listen(PORT, HOST, ()
=> { console.log(`Servidor rodando em http://${HOST}:${PORT}`); });
```

Testando a Aplicação

Rodando o Servidor

Execute o comando para iniciar o servidor:

bash

 Copiar código

```
npm start
```

Verificando as Rotas

Abra o navegador e teste as seguintes URLs:

- <http://127.0.0.1:3000/>: Deve retornar "Servidor Node.js".
- <http://127.0.0.1:3000/produtos>: Deve listar os produtos.
- <http://127.0.0.1:3000/produtos/adicionar>: Deve adicionar um novo produto.
- <http://127.0.0.1:3000/produtos/remover>: Deve remover um produto.

Conferindo o Funcionamento

Verifique se as operações de listar, adicionar e remover produtos estão funcionando conforme esperado.

Conclusão

Hoje, vimos como aplicar conceitos de orientação a objetos em nosso projeto de back-end.

Criamos uma classe `Produto` e refatoramos nosso servidor para utilizar essa classe, tornando o código mais organizado e modular.

Pratiquem esses conceitos e experimentem criar novas funcionalidades utilizando classes e métodos estáticos.

Até a próxima aula!

1.2. Vídeo Aula

dia 05 nodejs 1 converted



1.3. REST API – Implementação

Olá pessoal, tudo bem?

Vamos continuar com nossa aula de back-end.

Na aula passada, criamos a classe `Produto` e implementamos seus métodos estáticos para listar, adicionar, editar e excluir produtos.

Hoje, vamos completar a implementação dessas funcionalidades e aprender sobre REST API.

Revisão da Aula Anterior

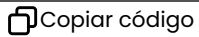
Na última aula, vimos como:

- Criar uma classe `Produto` com métodos estáticos.
- Exportar a classe `Produto` para poder ser utilizada em outras partes da aplicação.
- Configurar rotas no servidor para listar, adicionar e remover produtos.

Implementação da Rota de Edição

Primeiro, vamos adicionar a rota para editar produtos no nosso servidor.

javascript



```
const http = require('http'); const Produto = require('./produto'); const server = http.createServer((req, res) => { const { url, method } = req; res.setHeader('Content-Type', 'application/json'); if (url === '/produtos' && method === 'GET') { res.statusCode = 200; res.end(JSON.stringify(Produto.listar())); } else if (url === '/produtos/adicionar' && method === 'POST') { Produto.adicionar({ id: 3, nome: 'Mouse', valor: 50 }); res.statusCode = 201; res.end('Produto adicionado com sucesso'); } else if (url === '/produtos/remover' && method === 'DELETE') { Produto.remover(1); res.statusCode = 200; res.end('Produto removido com sucesso'); } else if (url === '/produtos/editar' && method === 'PUT') { Produto.editar(2, { nome: 'Teclado Mecânico', valor: 200 }); res.statusCode = 200; res.end('Produto editado com sucesso'); } else { res.statusCode = 404; res.end('Not Found'); } }); const PORT = 3000; const HOST = '127.0.0.1'; server.listen(PORT, HOST, () => { console.log(`Servidor rodando em http://${HOST}:${PORT}`); });
```

Introdução à REST API

O Que é REST API?

REST (Representational State Transfer) é uma arquitetura para comunicação entre sistemas, especialmente web services. Foi introduzida por Roy Fielding em 2000. A arquitetura REST permite que front-end e back-end se comuniquem de forma eficiente e padronizada.

Métodos HTTP

Os métodos HTTP são utilizados para indicar a ação a ser realizada:

- **GET**: Solicitar dados do servidor.
- **POST**: Enviar dados para o servidor.
- **PUT**: Atualizar dados no servidor.
- **DELETE**: Excluir dados do servidor.

Estrutura de URL

Na arquitetura REST, as URLs são estruturadas de forma intuitiva. Por exemplo:

- **GET /produtos**: Lista todos os produtos.
- **POST /produtos**: Adiciona um novo produto.
- **PUT /produtos/**

: Atualiza o produto com o ID especificado.

- **DELETE /produtos/**


: Remove o produto com o ID especificado.

Implementando REST API

Configurando as Rotas

Vamos ajustar nosso servidor para seguir os padrões REST.

javascript

 Copiar código

```
const http = require('http'); const Produto = require('./produto'); const server = http.createServer((req, res) => { const { url, method } = req; res.setHeader('Content-Type', 'application/json'); if (url === '/produtos' && method === 'GET') { res.statusCode = 200; res.end(JSON.stringify(Produto.listar())); } else if (url === '/produtos' && method === 'POST') { Produto.adicionar({ id: 3, nome: 'Mouse', valor: 50 }); res.statusCode = 201; res.end('Produto adicionado com sucesso'); } else if (url.match(/\/produtos\/\d+/) && method === 'PUT') { const id = parseInt(url.split('/')[2]); Produto.editar(id, { nome: 'Teclado Mecânico', valor: 200 }); res.statusCode = 200; res.end('Produto editado com sucesso'); } else if (url.match(/\/produtos\/\d+/) && method === 'DELETE') { const id = parseInt(url.split('/')[2]); Produto.remover(id); res.statusCode = 200; res.end('Produto removido com sucesso'); } else { res.statusCode = 404; res.end('Not Found'); } }); const PORT = 3000; const HOST = '127.0.0.1'; server.listen(PORT, HOST, () => { console.log(`Servidor rodando em http://${HOST}:${PORT}`); });
```

Testando a Aplicação

Rodando o Servidor

Execute o comando para iniciar o servidor:

bash

 Copiar código

```
npm start
```

Verificando as Rotas

Abra o navegador e teste as seguintes URLs e métodos utilizando uma ferramenta como Postman ou Insomnia:

- **GET <http://127.0.0.1:3000/produtos>**: Deve listar os produtos.
- **POST <http://127.0.0.1:3000/produtos>**: Deve adicionar um novo produto.
- **PUT <http://127.0.0.1:3000/produtos/2>**: Deve atualizar o produto com ID 2.
- **DELETE <http://127.0.0.1:3000/produtos/2>**: Deve remover o produto com ID 2.

Conclusão

Nesta aula, vimos como aplicar os conceitos de REST API em nosso projeto de back-end com Node.js.

Criamos rotas para listar, adicionar, editar e remover produtos seguindo os padrões REST.

Essa abordagem facilita a comunicação entre o front-end e o back-end e organiza melhor nosso código.

Pratiquem esses conceitos e experimentem adicionar novas funcionalidades. Até a próxima aula!

1.4. Vídeo Aula

dia 05 nodejs 2 converted

