

Material de apoio

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

Impresso por: JOÃO VITOR DE MELO FREITAS
Data: quinta-feira, 18 jul. 2024, 23:06

Índice

1. Node.js e HTTP/HTTPS

1.1. Vídeo Aula

2. Configuração do Servidor Node.js

2.1. Vídeo Aula

3. Configuração do Servidor Node.js

3.1. Vídeo Aula

1. Node.js e HTTP/HTTPS

Revisão da Aula Anterior

Configuração do Ambiente

Vamos abrir o VS Code e criar uma estrutura de pastas adequada para nosso projeto:

1. Crie uma pasta chamada `modulo-backend`.
2. Dentro desta pasta, crie uma subpasta chamada `aula-node`.
3. No VS Code, vá até o menu **File** > **Open Folder** e selecione a pasta `aula-node`.

Configuração do Terminal

Para garantir que o Node.js e o npm estão corretamente instalados, abra o terminal integrado do VS Code (Ctrl + J) e execute os seguintes comandos:

```
bash
```

 Copiar código

```
node -v npm -v
```

Inicialização do Projeto

Inicialize um novo projeto Node.js utilizando o comando:

```
bash
```

 Copiar código

```
npm init -y
```

Este comando criará um arquivo `package.json` com as configurações básicas do projeto. Altere o arquivo conforme necessário:

- Nome do projeto
- Versão
- Arquivo principal (`server.js`)
- Scripts
- Autor
- Licença
- Descrição

Exemplo:

```
json
```

 Copiar código

```
{ "name": "servidor-js", "version": "1.0.0", "main": "server.js", "scripts": { "start": "node server.js" }, "author": "Marcio Ferreira", "license": "ISC", "description": "Servidor Node.js básico" }
```

Criação do Servidor Node.js

Crie um arquivo chamado `server.js` e adicione o seguinte código:

```
javascript
```

 Copiar código

```
const http = require('http'); const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Olá, mundo!'); }); const hostname = '127.0.0.1'; const port = 3000; server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Para iniciar o servidor, execute:

bash


 Copiar código

```
npm start
```

Explicação Detalhada

Importando o Módulo HTTP

javascript


 Copiar código

```
const http = require('http');
```

Utilizamos o módulo `http` do Node.js para criar o servidor.

Criando o Servidor

javascript


 Copiar código

```
const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain');  
res.end('Olá, mundo!'); });
```

A função `createServer` cria um novo servidor HTTP. O callback recebe dois parâmetros: `req` (request) e `res` (response). Configuramos o status da resposta como 200 (OK), definimos o tipo de conteúdo como `text/plain` e enviamos a mensagem "Olá, mundo!".

Definindo Porta e Host

javascript

 Copiar código

```
const hostname = '127.0.0.1'; const port = 3000;
```

Definimos o endereço do host (localhost) e a porta (3000) onde o servidor será executado.

Iniciando o Servidor

javascript

 Copiar código

```
server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Utilizamos o método `listen` para iniciar o servidor na porta e host definidos. O callback exibe uma mensagem no console indicando que o servidor está em execução.

Tipos de Conteúdo

Podemos responder com diferentes tipos de conteúdo. Além de `text/plain`, podemos utilizar:

- `text/html` para HTML
- `application/json` para JSON
- `application/pdf` para PDF

Status Codes

Os códigos de status HTTP indicam o resultado da requisição:

- `200`: OK
- `201`: Created
- `400`: Bad Request

- 401: Unauthorized
- 404: Not Found
- 500: Internal Server Error

Parâmetros de URL

Para acessar parâmetros passados na URL, utilizamos o módulo `url`:

javascript

 Copiar código

```
const url = require('url'); const server = http.createServer((req, res) => { const parsedUrl = url.parse(req.url, true);
const query = parsedUrl.query; res.statusCode = 200; res.setHeader('Content-Type', 'text/html'); res.end(` <!DOCTYPE html>
<html lang="pt-BR"> <head> <meta charset="UTF-8"> <title>Servidor Node.js</title> </head> <body> <h1>Olá, ${query.nome ||
'mundo'}!</h1> <p>Este é um servidor Node.js</p> </body> </html> `); });
```

Agora, ao acessar <http://127.0.0.1:3000/?nome=Marcio>, a resposta será "Olá, Marcio!".

Conclusão

Hoje, revisamos os conceitos de HTTP e HTTPS, configuramos um servidor HTTP básico utilizando Node.js e exploramos diferentes tipos de conteúdo e códigos de status. Pratiquem esses conceitos criando seus próprios servidores e manipulando diferentes tipos de conteúdo e parâmetros.

Espero que tenham gostado e até a próxima aula!

1.1. Vídeo Aula

dia 03 nodejs 1 converted



2. Configuração do Servidor Node.js

Introdução

Olá, pessoal. Tudo bem? Vamos dar continuidade ao nosso curso de Back-end com Node.js. Na aula passada, mostramos os códigos de status HTTP, explicando como padronizá-los para indicar erros ou informações provenientes do back-end, e como esses status podem ser utilizados para informar o front-end sobre o que está acontecendo.

Configuração do Servidor Node.js

Problema com Atualizações Manuais

Sempre que fazemos uma alteração no código do servidor, é necessário parar e reiniciar o servidor manualmente para que as mudanças sejam aplicadas. Isso pode ser bastante trabalhoso e ineficiente.


Utilizando o npm

O npm oferece bibliotecas que facilitam esse processo. Vamos começar explorando como podemos usar o npm para simplificar a execução de scripts.

Configuração do package.json

Primeiro, vamos parar o serviço do servidor atual e editar o arquivo `package.json` para adicionar scripts de execução. O `package.json` é um arquivo de configuração do npm que gerencia as dependências do projeto e facilita a execução de scripts.

json


 Copiar código

```
{ "name": "servidor-js", "version": "1.0.0", "main": "server.js", "scripts": { "start": "node server.js" }, "author": "Marcio Ferreira", "license": "ISC", "description": "Servidor Node.js básico" }
```

Executando Scripts com npm

Podemos executar o script de start configurado no `package.json` utilizando o comando:

bash

 Copiar código

```
npm start
```

Isso fará com que o npm busque o script de start no `package.json` e execute o comando especificado.

Utilizando Nodemon

Introdução ao Nodemon

O `nodemon` é uma ferramenta que monitora as mudanças nos arquivos e reinicia automaticamente o servidor quando uma alteração é detectada.

Instalando o Nodemon

Para instalar o `nodemon` como uma dependência de desenvolvimento, utilize o comando:

bash

 Copiar código

```
npm install --save-dev nodemon
```

Configurando o Nodemon no package.json

Altere o script de start no `package.json` para utilizar o `nodemon`:

json



Copiar código

```
"scripts": { "start": "nodemon server.js" }
```

Testando o Nodemon

Agora, ao executar `npm start`, o `nodemon` monitorará as alterações no código e reiniciará o servidor automaticamente. Faça uma alteração no arquivo `server.js` e salve para ver o `nodemon` em ação.

Configuração de Rotas no Node.js

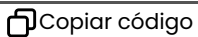
Introdução às Rotas

Rotas são URLs específicas que o servidor pode responder. Vamos configurar diferentes rotas no nosso servidor.

Adicionando Rotas

Modifique o arquivo `server.js` para incluir diferentes rotas:

javascript



Copiar código

```
const http = require('http'); const server = http.createServer((req, res) => { const { url, method } = req; if (url === '/' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Servidor Node.js'); } else if (url === '/produtos' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Listagem de Produtos'); } else { res.statusCode = 404; res.setHeader('Content-Type', 'text/plain'); res.end('Página não encontrada'); } }); const hostname = '127.0.0.1'; const port = 3000; server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Testando as Rotas

Abra o navegador e acesse as seguintes URLs para testar:

- <http://127.0.0.1:3000/> – Deverá exibir "Servidor Node.js".
- <http://127.0.0.1:3000/produtos> – Deverá exibir "Listagem de Produtos".
- Qualquer outra URL – Deverá exibir "Página não encontrada".

Capturando Parâmetros e Métodos

Podemos capturar a URL e o método da requisição para realizar diferentes ações baseadas nesses parâmetros:

javascript



Copiar código

```
const http = require('http'); const server = http.createServer((req, res) => { const { url, method } = req; if (url === '/' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Servidor Node.js'); } else if (url === '/produtos' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Listagem de Produtos'); } else { res.statusCode = 404; res.setHeader('Content-Type', 'text/plain'); res.end('Página não encontrada'); } }); const hostname = '127.0.0.1'; const port = 3000; server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Agora, sempre que uma requisição for feita, podemos ver a URL e o método no console.

Conclusão

Hoje aprendemos a configurar o `nodemon` para facilitar o desenvolvimento, evitando a necessidade de reiniciar o servidor manualmente. Também exploramos a criação de rotas no Node.js, permitindo que nosso servidor responda a diferentes URLs de forma adequada. Pratiquem esses conceitos criando suas próprias rotas e manipulando diferentes tipos de resposta.

Espero que tenham gostado e até a próxima aula!

2.1. Vídeo Aula

dia 03 nodejs 2 converted



3. Configuração do Servidor Node.js

Introdução

Olá, pessoal. Tudo bem? Vamos dar continuidade ao nosso curso de back-end com Node.js. Na aula passada, vimos como funcionam os códigos de status HTTP, como padronizá-los para indicar erros ou informações provenientes do back-end e como esses status podem ser utilizados para informar o front-end sobre o que está acontecendo. Hoje, continuaremos a explorar as rotas no Node.js e aprenderemos a retornar respostas em JSON.

Configuração do Servidor Node.js

Configurando Novas Rotas

Vamos configurar mais rotas no nosso servidor e aprender a retornar dados em formato JSON. Primeiramente, vamos criar um array de produtos para usarmos como exemplo.

javascript



```
const produtos = [ { id: 1, nome: 'Teclado', valor: 50 }, { id: 2, nome: 'Mouse', valor: 30 } ];
```

Retornando JSON

Vamos adicionar uma rota que retorna a lista de produtos em formato JSON.

javascript



```
const http = require('http'); const produtos = [ { id: 1, nome: 'Teclado', valor: 50 }, { id: 2, nome: 'Mouse', valor: 30 } ]; const server = http.createServer((req, res) => { const { url, method } = req; if (url === '/' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Servidor Node.js'); } else if (url === '/produtos' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'application/json'); res.end(JSON.stringify(produtos)); } else { res.statusCode = 404; res.setHeader('Content-Type', 'text/plain'); res.end('Página não encontrada'); } }); const hostname = '127.0.0.1'; const port = 3000; server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Testando a Rota de Produtos

Para testar, abra o navegador e acesse <http://127.0.0.1:3000/produtos>. Você deverá ver a lista de produtos em formato JSON.

Manipulando Produtos

Vamos adicionar rotas para adicionar e remover produtos.

Adicionando Produtos

javascript



```
else if (url === '/produtos/adicionar' && method === 'POST') { produtos.push({ id: 3, nome: 'Monitor', valor: 800 }); res.statusCode = 201; res.setHeader('Content-Type', 'application/json'); res.end(JSON.stringify({ mensagem: 'Produto adicionado com sucesso' })); }
```

Removendo Produtos

javascript



```
else if (url === '/produtos/remover' && method === 'DELETE') { produtos.pop(); res.statusCode = 200; res.setHeader('Content-Type', 'application/json'); res.end(JSON.stringify({ mensagem: 'Produto removido com sucesso' })); }
```

Testando as Rotas de Adicionar e Remover

Para testar, utilize ferramentas como Postman ou curl para enviar requisições POST e DELETE para as rotas configuradas.

Separando Rotas em Módulos

Vamos organizar nosso código separando as rotas em módulos.

Criando o Módulo de Produtos

Crie um arquivo chamado `produtos.js` dentro de uma pasta `rotas`.

javascript


 Copiar código

```
const produtos = [ { id: 1, nome: 'Teclado', valor: 50 }, { id: 2, nome: 'Mouse', valor: 30 } ]; function listarProdutos() {
return JSON.stringify(produtos); } function adicionarProduto(produto) { produtos.push(produto); return JSON.stringify({
mensagem: 'Produto adicionado com sucesso' }); } function removerProduto() { produtos.pop(); return JSON.stringify({
mensagem: 'Produto removido com sucesso' }); } module.exports = { listarProdutos, adicionarProduto, removerProduto };
```

Utilizando o Módulo de Produtos

No arquivo `server.js`, importe e utilize as funções do módulo `produtos`.

javascript

 Copiar código

```
const http = require('http'); const { listarProdutos, adicionarProduto, removerProduto } = require('./rotas/produtos');
const server = http.createServer((req, res) => { const { url, method } = req; if (url === '/' && method === 'GET') {
res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Servidor Node.js'); } else if (url ===
'/produtos' && method === 'GET') { res.statusCode = 200; res.setHeader('Content-Type', 'application/json');
res.end(listarProdutos()); } else if (url === '/produtos/adicionar' && method === 'POST') { const novoProduto = { id: 3,
nome: 'Monitor', valor: 800 }; res.statusCode = 201; res.setHeader('Content-Type', 'application/json');
res.end(adicionarProduto(novoProduto)); } else if (url === '/produtos/remover' && method === 'DELETE') { res.statusCode =
200; res.setHeader('Content-Type', 'application/json'); res.end(removerProduto()); } else { res.statusCode = 404;
res.setHeader('Content-Type', 'text/plain'); res.end('Página não encontrada'); } }); const hostname = '127.0.0.1'; const
port = 3000; server.listen(port, hostname, () => { console.log(`Servidor rodando em http://${hostname}:${port}/`); });
```

Conclusão

Hoje aprendemos a configurar mais rotas no nosso servidor Node.js, retornar respostas em formato JSON e a organizar nosso código separando as rotas em módulos. Pratiquem esses conceitos criando suas próprias rotas e manipulando diferentes tipos de resposta.

Espero que tenham gostado e até a próxima aula!

3.1. Vídeo Aula

dia 03 nodejs 3 converted

