

## 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. Revisão do package.json**

1.1. Scripts no package.json

1.2. Vídeo Aula

1.3. Criação de Servidor HTTP com Node.js

1.4. Vídeo Aula

1.5. HTTP e HTTPS com Node.js

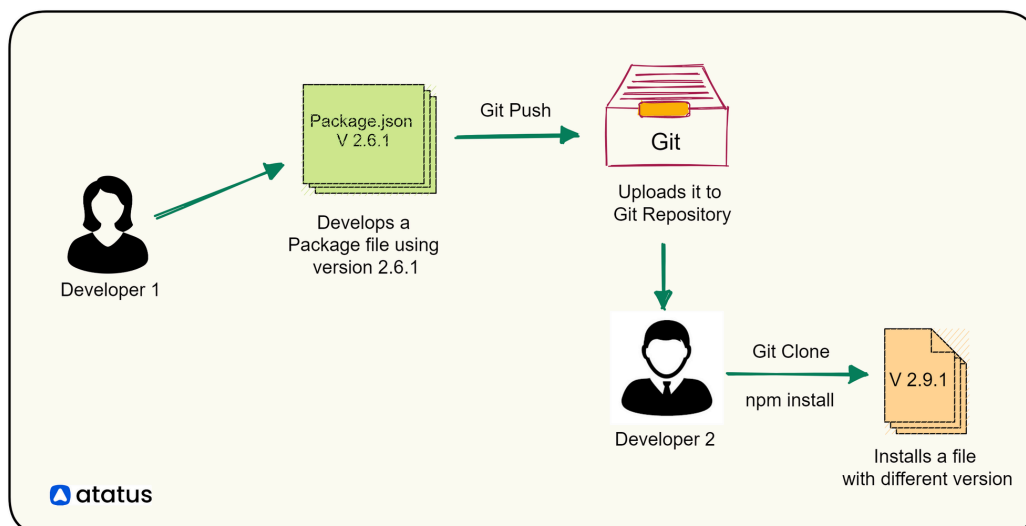
1.6. HTTP e HTTPS no Node.js

1.7. Vídeo Aula

# 1. Revisão do package.json

## Revisão do package.json

O `package.json` é o arquivo de configuração do projeto. Ele contém informações como o nome do projeto, versão, scripts, autor, licença e dependências. Vamos revisar e configurar alguns pontos importantes.




## 1.1. Scripts no package.json

Vamos usar o npm para gerenciar e executar nossos scripts. Primeiro, vamos configurar um script de start para nosso projeto.

1. Abra o arquivo `package.json` e encontre a seção de `scripts`:


json

 Copiar código

```
"scripts": { "test": "echo \"Error: no test specified\" && exit 1" }
```

2. Modifique para incluir um script de start:

json

 Copiar código

```
"scripts": { "start": "node primeiro-programa.js", "test": "echo \"Error: no test specified\" && exit 1" }
```

## Executando Scripts com npm

Agora, vamos executar nosso script de start usando npm:

1. No terminal, execute:

bash

 Copiar código

```
npm start
```

2. O npm buscará o script de start no `package.json` e executará o comando associado:

plaintext

 Copiar código

```
> primeiro-programa@1.0.0 start > node primeiro-programa.js Olá, sou JS A soma é: 30
```


## Configurando o Servidor com Node.js

Vamos criar um servidor básico com Node.js.

### 1. Criar o Arquivo do Servidor

- o Crie um novo arquivo chamado `server.js`:

javascript

 Copiar código

```
console.log("Olá, eu sou o server.js");
```

### 2. Modificar o package.json para Usar o server.js

- o Abra o `package.json` e atualize o script de start:

json

 Copiar código

```
"scripts": { "start": "node server.js", "test": "echo \"Error: no test specified\" && exit 1" }
```

- o Atualize também o campo `main`:

json


 Copiar código

```
"main": "server.js",
```

### 3. Executar o Servidor

- No terminal, execute:


bash

 Copiar código

```
npm start
```

- O npm executará o `server.js`:

plaintext

 Copiar código

```
> primeiro-programa@1.0.0 start > node server.js Olá, eu sou o server.js
```


## Criando um Servidor HTTP com Node.js

Vamos configurar um servidor HTTP simples utilizando o módulo `http` do Node.js.

### 1. Importar o Módulo HTTP e Criar o Servidor

- Edite o arquivo `server.js`:

javascript


 Copiar código

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

### 2. Executar o Servidor

- No terminal, execute:


bash

 Copiar código

```
npm start
```

- O output será:

plaintext

 Copiar código

```
> primeiro-programa@1.0.0 start > node server.js Server running at http://127.0.0.1:3000/
```

### 3. Acessar o Servidor no Navegador

- Abra o navegador e vá para <http://127.0.0.1:3000/>. Você verá a mensagem "Olá, mundo!".

## Conclusão

Nesta aula, configuramos nosso projeto para usar scripts do npm, criamos um servidor básico com Node.js e configuramos um servidor HTTP simples. Agora temos uma base sólida para continuar desenvolvendo nosso back-end.

Espero que tenham gostado da aula e nos vemos na próxima sessão!



## 1.2. Vídeo Aula

dia 02 nodejs 1 1



## 1.3. Criação de Servidor HTTP com Node.js

### Introdução

Olá pessoal, tudo bem? Na última aula, criamos um servidor básico em Node.js. Hoje, vamos aprofundar mais na criação e configuração do nosso servidor Node.js utilizando o módulo `http`.

### Estrutura do Servidor

#### Importando o Módulo HTTP

Primeiro, precisamos importar o módulo `http` do Node.js. Vamos criar uma constante para isso:

javascript

 Copiar código

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

#### Criando o Servidor

Vamos criar uma variável constante chamada `server` e usar a função `createServer` do módulo `http` para configurar nosso servidor. Dentro dessa função, passamos outra função de callback que recebe dois parâmetros: `req` (request) e `res` (response).

javascript


 Copiar código

```
const server = http.createServer((req, res) => { res.statusCode = 200; // Código de status HTTP 200 (OK)
res.setHeader('Content-Type', 'text/html'); // Definindo o tipo de conteúdo como HTML res.end('<h1>Olá, mundo!</h1>'); //
Enviando a resposta });
```

#### Definindo Porta e Host

Vamos definir a porta e o host onde nosso servidor será executado:

javascript

 Copiar código

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

#### Iniciando o Servidor

Por fim, usamos o método `listen` para iniciar o servidor na porta e host definidos. Adicionamos também um callback para confirmar que o servidor foi iniciado com sucesso:

javascript

 Copiar código

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

### Configurando o `package.json`

Para facilitar a execução do nosso servidor, vamos configurar o `package.json` para usar este arquivo:

json

 Copiar código


```
{ "name": "aula-backend", "version": "1.0.0", "main": "server.js", "scripts": { "start": "node server.js" }, "author":
"Marcio Ferreira", "license": "ISC", "description": "Aula de back-end com Node.js" }
```



## Executando o Servidor

Abra o terminal e execute o comando:


```
bash
```

 Copiar código

```
npm start
```

Se tudo estiver configurado corretamente, você verá a mensagem:

```
plaintext
```

 Copiar código


Servidor rodando em <http://127.0.0.1:3000/>

Abra o navegador e acesse <http://127.0.0.1:3000/>. Você verá a mensagem "Olá, mundo!" exibida na tela.

## Modificando o Conteúdo da Resposta

Vamos modificar o conteúdo da resposta para retornar um HTML mais complexo:

```
javascript
```

 Copiar código

```
const server = http.createServer((req, res) => { 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, mundo!</h1> <p>Este é um servidor Node.js</p> </body> </html> `); });
```

## Manipulando Parâmetros de URL

Podemos acessar os parâmetros passados na URL. Vamos usar o módulo `url` para isso:

```
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 aprendemos a criar e configurar um servidor básico com Node.js, entendendo como funcionam as requisições e respostas HTTP. Pratiquem essas lições criando seus próprios servidores e manipulando diferentes tipos de conteúdo e parâmetros.

Espero que tenham gostado e até a próxima aula!

## 1.4. Vídeo Aula

dia 02 nodejs 2 1



## 1.5. HTTP e HTTPS com Node.js

### Introdução

Olá a todos, espero que estejam bem. Hoje abordaremos os conceitos de HTTP e HTTPS, além de como utilizá-los com Node.js. Discutiremos noções básicas e demonstraremos como configurar um servidor HTTP utilizando Node.js.

### O que é HTTP e HTTPS?

#### HTTP

HTTP (HyperText Transfer Protocol) é um protocolo de transferência de texto que permite a comunicação entre cliente (navegador) e servidor. Ele é utilizado para o envio e recebimento de informações na web.

#### HTTPS

HTTPS é a versão segura do HTTP, onde o "S" significa "Secure" (Segurança). Este protocolo utiliza criptografia para proteger a comunicação entre o cliente e o servidor, garantindo que os dados não sejam interceptados ou alterados.

### Comunicação entre Cliente e Servidor

Na comunicação entre cliente e servidor, o cliente faz uma requisição (request) e o servidor responde (response). Este ciclo de requisição e resposta é a base do protocolo HTTP.

### Exemplo de Comunicação

Vamos imaginar um cliente acessando um servidor. O cliente faz uma requisição ao servidor, que responde com o conteúdo solicitado.

### Ferramentas de Desenvolvimento

Ao desenvolver aplicações web, podemos utilizar ferramentas de desenvolvimento no navegador (como Chrome, Edge, Firefox) para analisar requisições e respostas HTTP.

1. Abra o navegador.
2. Pressione F12 para abrir as ferramentas de desenvolvimento.
3. Navegue até a aba "Network" (Rede).
4. Faça uma requisição (atualize a página ou acesse uma URL).
5. Verifique as requisições e respostas na aba "Network".

### Métodos HTTP

Existem vários métodos HTTP (também conhecidos como verbos) que indicam a intenção da requisição. Alguns dos mais comuns são:

- **GET**: Solicita dados do servidor.
- **POST**: Envia dados para o servidor.
- **PUT**: Atualiza dados no servidor.
- **DELETE**: Remove dados do servidor.


### Criando um Servidor HTTP com Node.js

Vamos criar um servidor HTTP básico utilizando Node.js.

#### Passo 1: Importando o Módulo HTTP

Primeiro, vamos importar o módulo http do Node.js:

```
javascript
```

 Copiar código

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

#### Passo 2: Criando o Servidor

Vamos criar o servidor utilizando a função `createServer` do módulo `http`:

javascript


 Copiar código

```
const server = http.createServer((req, res) => { res.statusCode = 200; res.setHeader('Content-Type', 'text/html');  
res.end('<h1>Olá, mundo!</h1>'); });
```

## Passo 3: Definindo Porta e Host

Definimos a porta e o host onde nosso servidor será executado:

javascript


 Copiar código

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

## Passo 4: Iniciando o Servidor

Iniciamos o servidor utilizando o método listen:

javascript

 Copiar código

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

## Configurando o `package.json`

Para facilitar a execução do servidor, vamos configurar o `package.json`:

json

 Copiar código

```
{ "name": "aula-backend", "version": "1.0.0", "main": "server.js", "scripts": { "start": "node server.js" }, "author":  
"Marcio Ferreira", "license": "ISC", "description": "Aula de back-end com Node.js" }
```

## Executando o Servidor

Abra o terminal e execute o comando:

bash

 Copiar código

```
npm start
```

Se tudo estiver configurado corretamente, você verá a mensagem:

plaintext

 Copiar código


```
Servidor rodando em http://127.0.0.1:3000/
```

Abra o navegador e acesse <http://127.0.0.1:3000/>. Você verá a mensagem "Olá, mundo!" exibida na tela.

## Analisando Parâmetros de URL

Podemos acessar os parâmetros passados na URL utilizando 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 aprendemos sobre HTTP e HTTPS, como eles funcionam, e criamos um servidor HTTP básico utilizando Node.js. Pratiquem esses conceitos criando seus próprios servidores e manipulando diferentes tipos de conteúdo e parâmetros.

Espero que tenham gostado e até a próxima aula!

## 1.6. HTTP e HTTPS no Node.js

### Olá pessoal, tudo bem?

Hoje vamos continuar nossa aula de back-end, focando em HTTP e HTTPS. Vamos entender o que são, como funcionam e como podemos usá-los em nossas aplicações.

Imagine que você está navegando na internet, procurando informações sobre o próximo show de sua banda favorita. Você digita o endereço do site no navegador e, em poucos segundos, a página aparece na tela. Esse processo simples é facilitado por um protocolo chamado HTTP.

HTTP, que significa HyperText Transfer Protocol (Protocolo de Transferência de Hipertexto), é a linguagem que seu navegador usa para conversar com o servidor onde o site está hospedado. Quando você faz uma solicitação para acessar um site, o navegador (cliente) envia uma requisição HTTP ao servidor. O servidor então processa essa requisição e envia de volta uma resposta com as informações solicitadas, como o conteúdo da página web que você deseja ver.

Agora, imagine que você está comprando ingressos para o show. Você insere seus dados pessoais e informações de pagamento no site. É aqui que entra o HTTPS. HTTPS é uma versão segura do HTTP. O "S" significa Secure (Seguro), e isso é crucial quando se trata de proteger suas informações sensíveis. Com HTTPS, os dados transferidos entre seu navegador e o servidor são criptografados. Isso significa que, mesmo que alguém consiga interceptar os dados, eles não poderão lê-los ou usá-los para fins maliciosos.

Ao usar HTTPS, você tem a garantia de que suas informações de pagamento e dados pessoais estão protegidos contra interceptações e ataques maliciosos. Assim, você pode concluir a compra dos ingressos com tranquilidade, sabendo que seus dados estão seguros.

Em resumo, HTTP e HTTPS são fundamentais para a comunicação na internet. HTTP permite a transferência de informações entre o cliente e o servidor, enquanto HTTPS adiciona uma camada de segurança, criptografando os dados para protegê-los durante a transmissão.

### Comunicação entre Cliente e Servidor

Para ilustrar como funciona a comunicação entre o cliente e o servidor, vamos usar um pequeno diagrama:

- **Cliente (Front-End):** Pode ser um navegador ou qualquer outro tipo de aplicação que faz requisições HTTP.
- **Servidor (Back-End):** Processa as requisições HTTP e retorna as respostas apropriadas.

### Exemplo de Comunicação

Vamos imaginar que estamos executando nossa aplicação no navegador em <http://127.0.0.1:3000> (porta 3000).

- **Requisição HTTP (Request):** O cliente faz uma solicitação ao servidor.
- **Resposta HTTP (Response):** O servidor processa a solicitação e retorna uma resposta ao cliente.

### Estrutura de uma Requisição e Resposta HTTP

Ao acessar <http://127.0.0.1:3000>, o navegador faz uma requisição HTTP ao servidor. O servidor processa essa requisição e retorna uma resposta.

### Ferramentas de Desenvolvedor

No navegador, podemos usar as Ferramentas de Desenvolvedor para inspecionar as requisições e respostas HTTP:

1. Pressione F12 para abrir as Ferramentas de Desenvolvedor.
2. Vá para a aba "Rede" (ou "Network").
3. Atualize a página (F5) e veja todas as requisições sendo feitas.

### Cabeçalhos HTTP

Os cabeçalhos HTTP contêm informações importantes sobre a requisição e a resposta, como o método HTTP utilizado (GET, POST, etc.), o status da resposta (200 OK, 404 Not Found, etc.), e muito mais.

### Métodos HTTP

Existem vários métodos HTTP (também chamados de verbos) que indicam a ação a ser realizada:

- **GET:** Solicita dados do servidor.

- **POST:** Envia dados ao servidor para serem processados.
- **PUT:** Atualiza dados no servidor.
- **DELETE:** Remove dados do servidor.
- **PATCH:** Atualiza parcialmente dados no servidor.
- **OPTIONS:** Descreve as opções de comunicação para o recurso alvo.


## Implementação no Node.js

Vamos implementar um servidor HTTP simples usando Node.js para entender como esses conceitos são aplicados na prática.

### Criando o Servidor

Crie um arquivo chamado `server.js`:

javascript

 Copiar código

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

### Atualize o `package.json` para usar este arquivo:

json

 Copiar código

```
"main": "server.js", "scripts": { "start": "node server.js" }
```

### Execute o servidor:

bash

 Copiar código

```
npm start
```

### Testando o Servidor

Abra o navegador e vá para <http://127.0.0.1:3000/>. Verifique a resposta do servidor ("Olá, mundo!"). No terminal, você verá os detalhes da requisição:

plaintext


 Copiar código

```
Requisição recebida Método: GET URL: /
```

## Trabalhando com Parâmetros de URL

Podemos acessar os parâmetros passados na URL:

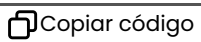
javascript

 Copiar código

```
const url = require('url'); const server = http.createServer((req, res) => { const parsedUrl = url.parse(req.url, true); console.log('Requisição recebida'); console.log(`Método: ${req.method}`); console.log(`URL: ${req.url}`); console.log(`Parâmetros: ${JSON.stringify(parsedUrl.query)}`); res.statusCode = 200; res.setHeader('Content-Type', 'text/plain'); res.end('Olá, mundo!\n'); });
```

Agora, ao acessar <http://127.0.0.1:3000/?nome=Maria>, veremos os parâmetros no console:

plaintext



Requisição recebida Método: GET URL: /?nome=Maria Parâmetros: {"nome":"Maria"}

## Conclusão

Hoje aprendemos sobre HTTP e HTTPS, métodos HTTP, e como criar um servidor simples com Node.js.

Pratiquem essas lições criando suas próprias requisições e respostas no servidor Node.js.

Na próxima aula, continuaremos explorando mais funcionalidades e melhores práticas de desenvolvimento de back-end com Node.js.

Espero que tenham gostado e até a próxima aula!



## 1.7. Vídeo Aula

dia 02 nodejs 3 1

