



INFRAESTRUTURA PARA DESENVOLVIMENTO WEB

UNIDADE III NODE.JS & NPM

Elaboração

Miriã Falcão Freitas

Produção

Equipe Técnica de Avaliação, Revisão Linguística e Editoração

SUMÁRIO

UNIDADE III

NODE.JS & NPM.....	5
--------------------	---

CAPÍTULO 1

ASPECTOS GERAIS.....	7
----------------------	---

CAPÍTULO 2

NOÇÕES BÁSICAS DO NODE.JS	11
---------------------------------	----

CAPÍTULO 3

CRIANDO UM APLICATIVO.....	25
----------------------------	----

REFERÊNCIAS	36
-------------------	----

O NPM (*Node Package Manager*) consiste em duas coisas: primeiro, é um repositório on-line para publicação de projetos de código aberto para o Node.js; segundo, ele é um utilitário de linha de comando que interage com o referido repositório que auxilia na instalação de pacotes, gerenciamento de versões e gerenciamento de dependências.

O Node.js é um ambiente de tempo de execução do lado do servidor de código aberto criado no mecanismo JavaScript V8 (Mecanismo JavaScript e WebAssembly de alto desempenho e código-fonte do Google, escrito em C++) do Chrome. Ele fornece um ambiente de tempo de execução de E/S sem bloqueio (assíncrono) e de plataforma cruzada para criar aplicativos altamente escaláveis do lado do servidor usando JavaScript.

O Node.js pode ser usado para criar diferentes tipos de aplicativos, como aplicativo de linha de comando, aplicativo Web, aplicativo de bate-papo em tempo real, servidor de API REST, etc. No entanto, ele é usado principalmente para criar programas de rede, como servidores *web*, semelhantes ao PHP, Java ou ASP.NET.

Originalmente, o Node.js foi concebido como um ambiente de servidor para aplicativos, mas os desenvolvedores começaram a usá-lo para criar ferramentas para ajudá-los na automação de tarefas locais. Desde então, todo um novo ecossistema de ferramentas baseadas em Node (como *Grunt*, *Gulp* e *webpack*) evoluiu para transformar a cara do desenvolvimento *front-end*.



Site oficial do Node.js. Disponível em: <https://nodejs.org>. (NODEJS, 2020).

Node.js no GitHub. Disponível em: <https://github.com/nodejs/node>. (GITHUB, 2020).

Grunt: o executor de tarefas JavaScript. Disponível em: <https://gruntjs.com/>. (GRUNT, 2020)

Gulp: automatize tarefas difíceis e longas. Disponível em: <https://gulpjs.com/>. (GULP, 2020).

Webpack: junte seus scripts. Disponível em: <https://webpack.js.org/>. (WEBPACK, 2020).

Vantagens do Node.js:

- » O Node.js é uma estrutura de código-fonte aberto sob licença MIT. (A licença do MIT é uma licença de *software* livre originada no Massachusetts Institute of Technology).
- » Usa JavaScript para criar aplicativos inteiros do lado do servidor.
- » Estrutura leve que inclui módulos mínimos. Outros módulos podem ser incluídos conforme a necessidade de um aplicativo.
- » Assíncrono por padrão. Portanto, ele executa mais rapidamente do que outras estruturas.
- » Estrutura de plataforma cruzada que roda em Windows, MAC ou Linux.

Nesta Unidade, veremos em detalhes como funciona o Node.js e o NPM.

CAPÍTULO 1

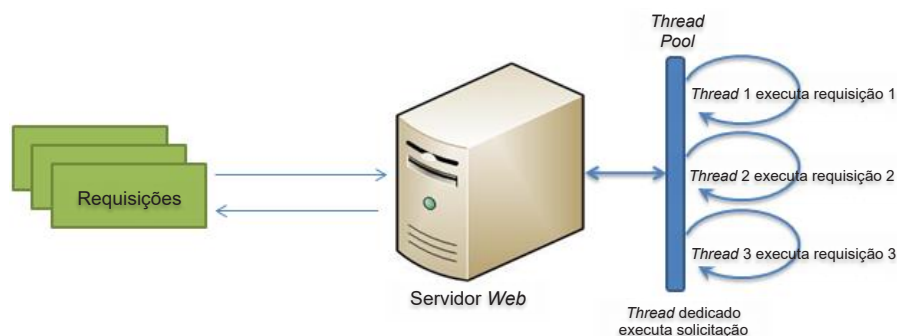
ASPECTOS GERAIS

No modelo tradicional de servidor da *web*, cada solicitação é tratada por um encadeamento dedicado do conjunto de encadeamentos. Se nenhum encadeamento estiver disponível no conjunto de encadeamentos a qualquer momento, a solicitação aguardará até o próximo encadeamento disponível. O encadeamento dedicado executa uma solicitação específica e não retorna ao conjunto de encadeamentos até concluir a execução e retornar uma resposta. A Figura 9 ilustra o modelo tradicional de um servidor *web*.

O Node.js processa solicitações de usuário de maneira diferente quando comparado a um modelo tradicional de servidor da *web*. O Node.js é executado em um único processo e o código do aplicativo é executado em um único encadeamento e, portanto, precisa de menos recursos que outras plataformas. Todas as solicitações do usuário para seu aplicativo da *web* serão tratadas por um único encadeamento e todo o trabalho de E/S ou de longa execução é executado de forma assíncrona para uma solicitação específica. Portanto, esse único encadeamento não precisa aguardar a conclusão da solicitação e é gratuito para lidar com a próxima solicitação. Quando o trabalho de E/S assíncrona é concluído, ele processa a solicitação ainda mais e envia a resposta.

Um *loop* de eventos está constantemente observando os eventos serem gerados para um trabalho assíncrono e executando a função de retorno de chamada quando o trabalho é concluído. Internamente, o Node.js usa libev para o *loop* de eventos que, por sua vez, usa o *pool* de encadeamentos interno do C++ para fornecer E/S assíncronas. A Figura 10 ilustra o modelo de servidor da *web* assíncrono usando no Node.js.

Figura 9. Modelo tradicional de um Servidor *Web*



Fonte: Adaptado de TutorialsTeacher.com (2020).

O modelo de processo do Node.js aumenta o desempenho e a escalabilidade com algumas ressalvas. O Node.js não é adequado para aplicativos que executam operações que exigem muita CPU, como processamento de imagens ou outros trabalhos pesados de computação, porque leva tempo para processar uma solicitação e, assim, bloqueia o encadeamento único.

1.1. Ambiente de Desenvolvimento Node.js

O ambiente de desenvolvimento do Node.js pode ser configurado no Windows, Mac, Linux e Solaris. As seguintes ferramentas/SDK são necessárias para o desenvolvimento de um aplicativo Node.js em qualquer plataforma:

- » Node.js.
- » Gerenciador de Pacotes de Nós – *Node Package Manager* (NPM).
- » IDE (Ambiente de desenvolvimento integrado) ou um Editor de Texto.

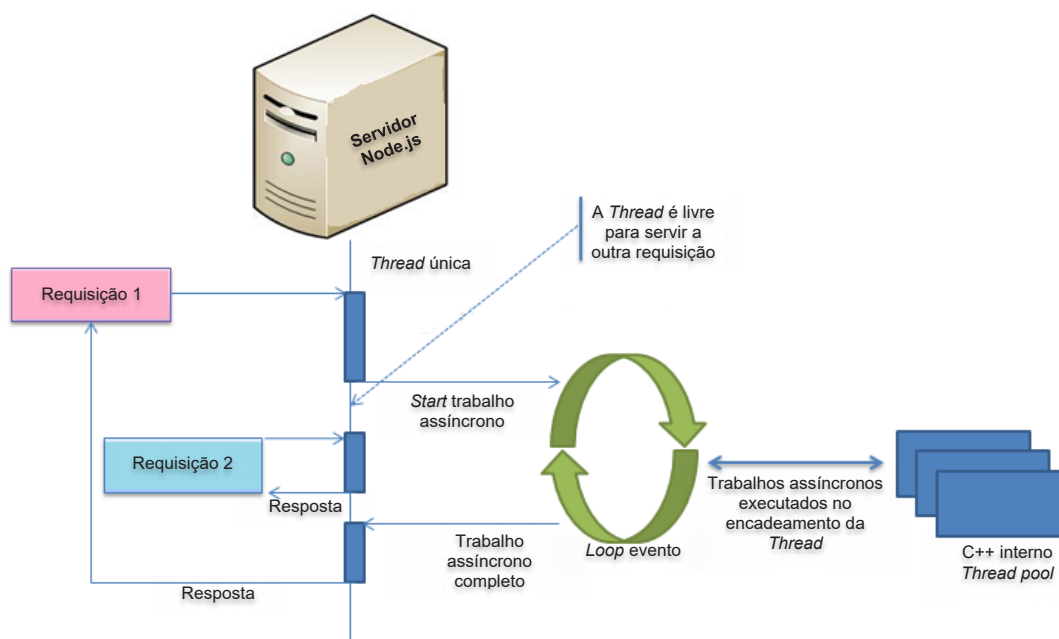
O NPM está incluído na instalação do Node.js desde o Node versão 0.6.0. Portanto, não há necessidade de instalá-lo separadamente.

1.2. Instalação do Node.js no Windows

Visite o *site* oficial do Node.js. (<https://nodejs.org>) (NODEJS, 2020). Ele detectará automaticamente o SO e exibirá o *link* de *download* conforme o seu sistema operacional.

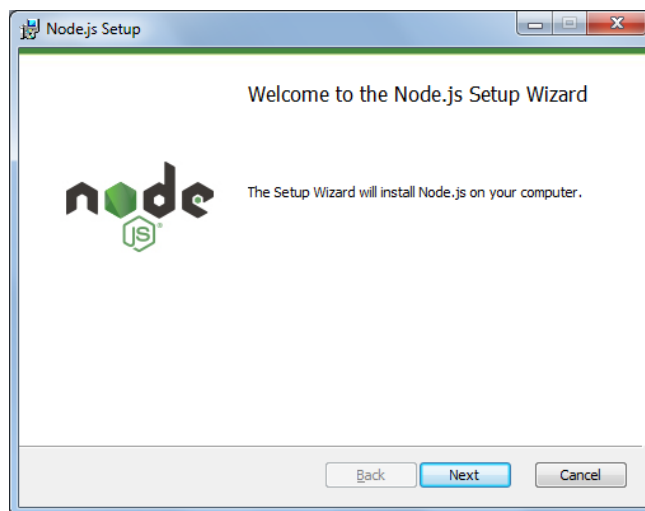
Faça o *download* do nó MSI para Windows clicando no botão 12.16.1 LTS ou 13.12.0 Current. Usaremos neste capítulo a versão 13.12.0 para Windows. Depois de baixar o MSI, clique duas vezes nele para iniciar a instalação, como mostrado na Figura 11.

Figura 10. Modelo de Processo do Node.js



Fonte: Adaptado de TutorialsTeacher.com (2020).

Figura 11. Instalação do Node.js

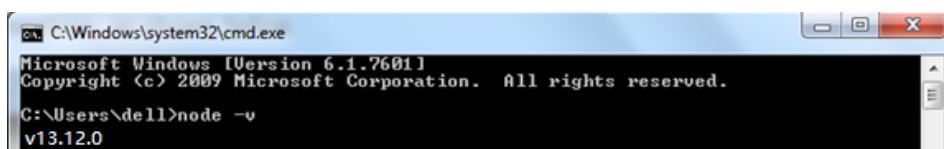


Fonte: Elaborado pela autora.

Clique em Avançar para ler e aceitar o Contrato de licença e clique em Instalar. Ele instalará o Node.js rapidamente no seu computador. Por fim, clique em Concluir para concluir a instalação.

Depois de instalar o Node.js no seu computador, você pode verificá-lo abrindo o *prompt* de comando e digitando `node -v`. Se o Node.js foi instalado com sucesso, ele exibirá a versão do Node.js instalado em sua máquina, conforme mostrado na Figura 12.

Figura 12. Verificando a instalação do Node.js



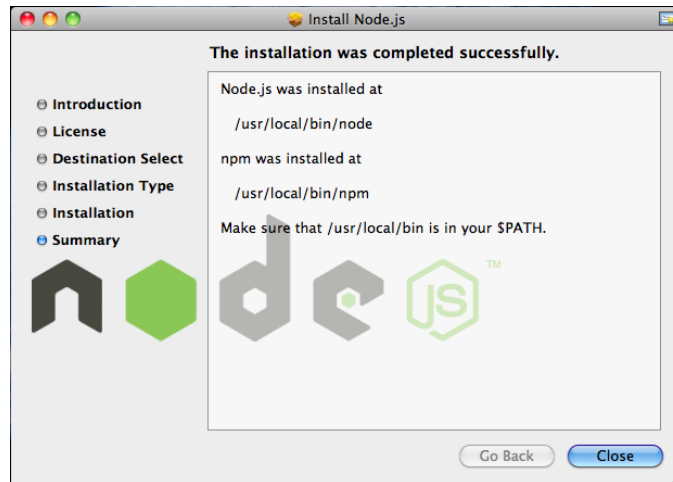
Fonte: Elaborado pela autora.

1.3. Instalação do Node.js no Mac/Linux

Visite o *site* oficial do Node.js. (<https://nodejs.org/en/download>) (NodeJs, 2020). Clique no instalador apropriado para Mac (.pkg ou .tar.gz) ou Linux para baixar o instalador do Node.js.

Após o *download*, clique no instalador para iniciar o assistente de instalação do Node.js. Clique em *Continuar* e siga as etapas. Após a instalação bem-sucedida, ele exibirá um resumo da instalação sobre o local em que instalou o Node.js e o NPM, conforme ilustrado na Figura 13.

Figura 13. Instalação do Node.js no OS X



Fonte: Elaborado pela autora.

Após a instalação, verifique a instalação do Node.js usando a janela do terminal e digite o comando `$ node -v`. Ele exibirá o número da versão do Node.js instalado no seu Mac.

Opcionalmente, para usuários de Mac ou Linux, é possível instalar diretamente o Node.js a partir da linha de comando, usando o gerenciador de pacotes *Homebrew* para Mac OS ou o gerenciador de pacotes *Linuxbrew* para Linux Operating System. Para Linux, você precisará instalar dependências adicionais.

1.4. IDE

O aplicativo Node.js usa JavaScript para desenvolver um aplicativo. Portanto, você pode usar qualquer ferramenta IDE ou editor de texto que suporte a sintaxe JavaScript. No entanto, é recomendado um IDE que suporte recursos de preenchimento automático para a API Node.js, por exemplo, *Visual Studio*, *Sublime Text*, *Eclipse*, *Aptana*, etc.

CAPÍTULO 2

NOÇÕES BÁSICAS DO NODE.JS

O Node.js suporta JavaScript. Portanto, a sintaxe JavaScript no Node.js é semelhante à sintaxe JavaScript do navegador.



Aprenda mais sobre a sintaxe do JavaScript. Disponível em:

<https://www.tutorialsteacher.com/javascript/javascript-tutorials>.

2.1. Tipos primitivos

O Node.js inclui os seguintes tipos primitivos:

- » *String*.
- » *Number*.
- » *Boolean*.
- » *Undefined*.
- » *Null*.
- » *RegExp*.

Todo o resto é um objeto no Node.js.

O JavaScript no Node.js suporta digitação livre como o JavaScript do navegador. Use a palavra-chave `var` para declarar uma variável de qualquer tipo.

2.1.1. Objeto literal

A sintaxe literal do objeto é igual ao JavaScript do navegador.

Exemplo: Objeto

```
var obj = {  
  authorName: 'Maria',  
  language: 'Node.js' }
```

2.1.2. Funções

As funções são cidadãos de primeira classe no JavaScript do Node, semelhante ao JavaScript do navegador. Uma função também pode ter atributos e propriedades. Pode ser tratado como uma classe em JavaScript.

Exemplo: Função

```
function Display(x) {  
    console.log(x); }  
Display(100);
```

2.1.3. Buffer

O Node.js inclui um tipo de dados adicional chamado *Buffer* (não disponível no JavaScript do navegador). O *buffer* é usado principalmente para armazenar dados binários, durante a leitura de um arquivo ou o recebimento de pacotes pela rede.

2.1.4. Objeto de processo

Cada *script* Node.js é executado em um processo. Ele inclui o objeto de processo para obter todas as informações sobre o processo atual do aplicativo Node.js.

O exemplo abaixo mostra como obter informações do processo no REPL usando o objeto de processo.

```
> process.execPath  
'C:\\Program Files\\nodejs\\node.exe'  
> process.pid  
1652  
> process.cwd()  
'C:\\'
```

2.1.5. Default é local

O JavaScript do nó é diferente do JavaScript do navegador quando se trata do escopo global. No JavaScript do navegador, as variáveis declaradas sem a palavra-chave `var` tornam-se globais. No Node.js, tudo se torna `default` por padrão.

2.1.6. Escopo global

Em um navegador, o escopo global é o objeto da janela. No Node.js, o objeto global representa o escopo global.

Para adicionar algo no escopo global, você precisa exportá-lo usando `export` ou `module.export`. Da mesma forma, importe módulos/objetos usando a função `require ()` para acessá-lo no escopo global.

Por exemplo, para exportar um objeto no Node.js, use `export.name = object`.

```
exports.log = {  
  console: function(msg) {  
    console.log(msg);  
  },  
  file: function(msg) {  
    // log to file here  
  }  
}
```

Você pode importar o objeto de `log` usando a função `require()` e usá-lo em qualquer lugar do seu projeto Node.js.

2.2. Módulo Node.js

O módulo no Node.js é uma funcionalidade simples ou complexa organizada em um ou mais arquivos JavaScript que podem ser reutilizados em todo o aplicativo Node.js.

Cada módulo no Node.js possui seu próprio contexto, portanto, não pode interferir com outros módulos ou poluir o escopo global. Além disso, cada módulo pode ser colocado em um arquivo `.js` em uma pasta separada.

O Node.js implementa o módulo padrão `CommonJS`. O CommonJS é um grupo de voluntários que define os padrões JavaScript para servidor da *web*, *desktop* e aplicativo de console.

O Node.js inclui três tipos de módulos:

- » Módulos principais.
- » Módulos locais.
- » Módulos de terceiros.

2.2.1. Módulos principais do Node.js

O Node.js é uma estrutura leve. Os módulos principais incluem funcionalidades mínimas básicas do Node.js. Esses módulos principais são compilados em sua distribuição binária e carregados automaticamente quando o processo do Node.js é iniciado. No entanto, você precisa importar o módulo principal primeiro para usá-lo em seu aplicativo.

A Tabela 1 lista alguns dos módulos principais do Node.js.

Tabela 1. Módulos principais do Node.js

Módulo principal	Descrição
http	O módulo http inclui classes, métodos e eventos para criar o servidor http Node.js.
url	O módulo url inclui métodos para resolução e análise de URL.
QueryString	O módulo <i>querystring</i> inclui métodos para lidar com a <i>string</i> de consulta.
Path	O módulo <i>path</i> inclui métodos para lidar com caminhos de arquivo.
Fs	O módulo fs inclui classes, métodos e eventos para trabalhar com E / S de arquivo.
Útil	O módulo útil inclui funções utilitárias úteis para programadores.

Fonte: Elaborado pela autora.

2.2.2. Carregando os módulos principais

Para usar os módulos principais ou NPM do Node.js, primeiro é necessário importá-lo usando a função `require()`, como mostrado abaixo.

```
var module = require('module_name');
```

Conforme a sintaxe acima, especifique o nome do módulo na função `require()`. A função `require()` retornará um objeto, função, propriedade ou qualquer outro tipo de JavaScript, dependendo do que o módulo especificado retornar.

O exemplo abaixo demonstra como usar o módulo http Node.js para criar um servidor *web*.

Exemplo: Carregar e usar o módulo principal http.

```
var http = require('http');  
var server = http.createServer(function(req, res){  
    //write code here  
});  
server.listen(5000);
```

No exemplo acima, a função `require()` retorna um objeto porque o módulo `http` retorna sua funcionalidade como um objeto. Você pode usar suas propriedades e métodos usando a notação de ponto, por exemplo, `http.createServer()`. Dessa forma, você pode carregar e usar os módulos principais do Node.js no seu aplicativo.

2.2.3. Módulos locais do Node.js

Módulos locais são módulos criados localmente no seu aplicativo Node.js. Esses módulos incluem diferentes funcionalidades do seu aplicativo em arquivos e pastas separados. Você também pode empacotá-los e distribuí-los via NPM, para que a comunidade do Node.js possa usá-los. Por exemplo, se você precisar se conectar ao MongoDB e buscar dados, poderá criar um módulo para ele, que poderá ser reutilizado no seu aplicativo.

Vamos escrever um módulo de registro simples que registra as informações, avisos ou erros no console.

No Node.js, o módulo deve ser colocado em um arquivo JavaScript separado. Portanto, crie um arquivo `Log.js` e escreva o seguinte código:

Arquivo = `Log.js`

```
var log = {
  info: function (info) {
    console.log('Info: ' + info);
  },
  warning: function (warning) {
    console.log('Warning: ' + warning);
  },
  error: function (error) {
    console.log('Error: ' + error);
  }
};
module.exports = log
```

No exemplo acima do módulo de `log`, criamos um objeto com três funções – `info()`, `warning()` e `error()`. No final, atribuímos esse objeto a `module.exports`. O `module.exports` no exemplo acima expõe um objeto de `log` como um módulo.

O `module.exports` é um objeto especial que é incluído em todos os arquivos JS no aplicativo Node.js. Por padrão, use `module.exports` ou **exportações** para expor uma função, objeto ou variável como um módulo no Node.js.

Agora, vamos ver como usar o módulo de registro acima em nosso aplicativo.

2.2.4. Carregando módulo local

Para usar módulos locais em seu aplicativo, você precisa carregá-lo usando a função `require()` da mesma maneira que o módulo principal. No entanto, você precisa especificar o caminho do arquivo JavaScript do módulo.

O exemplo abaixo demonstra como usar o módulo de *log* acima contido em `Log.js`.

Arquivo = `app.js`

```
var myLogModule = require('./Log.js');  
myLogModule.info('Node.js started');
```

No exemplo acima, `app.js` está usando o módulo de *log*. Primeiro, ele carrega o módulo de registro usando a função `require()` e o caminho especificado em que o módulo de registro está armazenado. O módulo de *log* está contido no arquivo `Log.js` na pasta raiz. Portanto, especificamos o caminho `./Log.js` na função `require()`. O `./` indica uma pasta raiz.

A função `require()` retorna um objeto de *log* porque o módulo de *log* expõe um objeto no `Log.js` usando `module.exports`. Então agora você pode usar o módulo de *log* como um objeto e chamar qualquer uma de suas funções usando a notação de ponto, por exemplo, `myLogModule.info()` ou `myLogModule.warning()` ou `myLogModule.error()`.

Execute o exemplo acima usando o *prompt* de comando (no Windows), como mostrado abaixo.

```
C: \> node app.js  
Informações: Node.js iniciado
```

Assim, você pode criar um módulo local usando `module.exports` e usá-lo em seu aplicativo.

2.2.5. Módulo de exportação no Node.js

O `module.exports` ou exportações é um objeto especial que é incluído em todos os arquivos JS no aplicativo Node.js. Por padrão `.module` é uma variável que representa o módulo atual e as **exportações** são um objeto que será exposto como um módulo. Portanto, o que você atribuir ao `module.exports` ou `export` será exposto como um módulo.

As **exportações** são um objeto. Portanto, ele expõe o que você atribuiu a ele como um módulo. Por exemplo, se você atribuir um literal de sequência, ele a exporá como um módulo.

O exemplo abaixo expõe uma mensagem de sequência simples como um módulo no `Message.js`.

Arquivo = `Message.js`

```
module.exports = 'Hello world';
//or
exports = 'Hello world';
```

Agora, importe este módulo de mensagem e use-o como mostrado abaixo.

Arquivo = `app.js`

```
var msg = require('./Messages.js');
console.log(msg);
```

Execute o exemplo acima no *prompt* de comando e veja o resultado como mostrado abaixo.

```
C:\> node app.js
Hello World
```

Nota: Você deve especificar `./` como um caminho da pasta raiz para importar um módulo local. No entanto, você não precisa especificar o caminho para importar o módulo principal do Node.js ou o módulo NPM na função `require()`.

2.3. Exportando objetos

`module.exports` é um objeto. Portanto, você pode anexar propriedades ou métodos a ele. O exemplo a seguir expõe um objeto com uma propriedade *string* no arquivo `Message.js`.

Arquivo = `Message.js`

```
exports.SimpleMessage = 'Hello world';
//or
module.exports.SimpleMessage = 'Hello world';
```

No exemplo acima, anexamos uma propriedade `"SimpleMessage"` ao objeto de exportação. Agora, importe e use este módulo como mostrado abaixo.

Arquivo = app.js

```
var msg = require('./Messages.js');  
console.log(msg.SimpleMessage);
```

No exemplo acima, a função `require()` retornará um objeto `{ SimpleMessage : 'Hello World' }` e o atribuirá à variável `msg`. Portanto, agora você pode usar o `msg.SimpleMessage`.

Execute o exemplo acima, escrevendo `node app.js` no *prompt* de comando e veja a saída como mostrado abaixo.

```
C:\> node app.js  
Hello World
```

Da mesma maneira que acima, você pode expor um objeto com função. O exemplo a seguir expõe um objeto com função de *log* como um módulo.

Arquivo = Log.js

```
module.exports.log = function (msg) {  
  console.log(msg);  
};
```

O módulo acima irá expor um objeto `{ log : function(msg) { console.log(msg); } }`. Use o módulo acima, como mostrado abaixo.

Arquivo = app.js

```
var msg = require('./Log.js');  
msg.log('Hello World');
```

Execute e veja a saída no *prompt* de comando, como mostrado abaixo.

```
C:\> node app.js  
Hello World
```

Você também pode anexar um objeto ao `module.exports`, como mostrado abaixo.

Arquivo = data.js

```
module.exports = {  
  firstName: 'James',  
  lastName: 'Bond'  
}
```

Arquivo = app.js

```
var person = require('./data.js');  
console.log(person.firstName + ' ' + person.lastName);
```

Execute o exemplo acima e veja o resultado, como mostrado abaixo.

```
C:\> node app.js  
James Bond
```

2.4. Função Exportar

Você pode anexar uma função anônima ao objeto de exportação, como mostrado abaixo.

Arquivo = Log.js

```
module.exports = function (msg) {  
  console.log(msg);  
};
```

Agora, você pode usar o módulo acima como abaixo.

Arquivo = app.js

```
var msg = require('./Log.js');  
msg('Hello World');
```

A variável msg torna-se expressão de função no exemplo acima. Portanto, você pode invocar a função usando parênteses (). Execute o exemplo acima e veja a saída como mostrado abaixo.

```
C:\> node app.js  
Hello World
```

2.5. Exportar função como uma classe

No JavaScript, uma função pode ser tratada como uma classe. O exemplo a seguir expõe uma função que pode ser usada como uma classe.

Arquivo = Person.js

```
module.exports = function (firstName, lastName) {  
  this.firstName = firstName;  
  this.lastName = lastName;  
  this.fullName = function () {  
    return this.firstName + ' ' + this.lastName;  
  }  
}
```

O módulo acima pode ser usado como mostrado abaixo.

Arquivo = app.js

```
var person = require('./Person.js');  
var person1 = new person('James', 'Bond');  
console.log(person1.fullName());
```

Como você pode ver, criamos um objeto de pessoa usando a nova palavra-chave. Execute o exemplo acima como abaixo.

```
C:\> node app.js  
James Bond
```

Dessa forma, você pode exportar e importar um módulo local criado em um arquivo separado na pasta raiz.

O Node.js também permite criar módulos em subpastas. Vamos ver como carregar o módulo de subpastas.

2.6. Carregar módulo da pasta separada

Use o caminho completo de um arquivo de módulo para o qual você o exportou usando `module.exports`. Por exemplo, se o módulo de *log* no *log.js* estiver armazenado na pasta “utilitário” na pasta raiz do seu aplicativo, importe-o, conforme mostrado abaixo.

Arquivo = app.js

```
var log = require('./utility/log.js');
```

No exemplo acima, o *log* é setado para a pasta raiz e especifica o caminho exato do seu arquivo de módulo. O Node.js também nos permite especificar o caminho para a pasta sem especificar o nome do arquivo. Por exemplo, você pode especificar apenas a pasta do utilitário sem especificar *log.js*, como mostrado abaixo.

Arquivo = app.js

```
var log = require('./utility');
```

No exemplo acima, o Node procurará um arquivo de definição de pacote chamado *package.json* dentro da pasta do utilitário. Isso ocorre porque o Node assume que esta pasta é um pacote e tentará procurar uma definição de pacote. O arquivo *package.json* deve estar em um diretório de módulo. O *package.json* na pasta utilitário especifica o nome do arquivo usando a tecla “main” como abaixo.

./utility/package.json

```
{
  "name" : "log",
  "main" : "./log.js"
}
```

Agora, o Node.js encontrará o arquivo *log.js* usando a entrada principal em *package.json* e importará.

Importante: Se o arquivo *package.json* não existir, ele procurará o arquivo *index.js* como um arquivo de módulo por padrão.

2.7. NPM

O NPM *Package Manager* (NPM) é uma ferramenta de linha de comando que instala, atualiza ou desinstala os pacotes Node.js no seu aplicativo. Também é um repositório *on-line* para pacotes Node.js de código aberto. A comunidade de *Node* em todo o mundo cria módulos úteis e os publica como pacotes neste repositório.

O NPM está incluído na instalação do Node.js. Depois de instalar o Node.js, verifique a instalação do NPM escrevendo o seguinte comando no terminal ou no *prompt* de comando.

```
C: \> npm -v  
5.1.0
```

Se você possui uma versão mais antiga do NPM, pode atualizá-la para a versão mais recente usando o seguinte comando:

```
C: \> npm install npm -g
```

Para acessar a ajuda do NPM, escreva `npm help` no *prompt* de comando ou na janela do terminal:

```
C: \> npm help
```

O NPM executa a operação em dois modos: **global** e **local**. No modo global, o NPM executa operações que afetam todos os aplicativos Node.js no computador, enquanto, no modo local, o NPM executa operações para o diretório local específico que afeta apenas um aplicativo nesse diretório.

2.8. Instalar pacote localmente

Use o comando a seguir para instalar qualquer módulo de terceiros na pasta local do projeto Node.js.

```
C:\>npm install <package name>
```

Por exemplo, o comando a seguir instalará o ExpressJS na pasta MyNodeProj.

```
C:\MyNodeProj> npm install express
```

Todos os módulos instalados usando o NPM são instalados na pasta `node_modules`. O comando acima criará a pasta ExpressJS na pasta `node_modules` na pasta raiz do seu projeto e instalará o Express.js lá.

2.8.1. Adicionar Dependência ao package.json

Use `-save` no final do comando *install* para adicionar entrada de dependência no `package.json` do seu aplicativo.

Por exemplo, o comando a seguir instalará o ExpressJS no seu aplicativo e também adicionará entrada de dependência no `package.json`.

```
C:\MyNodeProj> npm install express -save
```

O `package.json` do projeto NodejsConsoleApp será semelhante ao abaixo.

Arquivo = package.json

```
{
  "name": "NodejsConsoleApp",
  "version": "0.0.0",
  "description": "NodejsConsoleApp",
  "main": "app.js",
  "author": {
    "name": "Dev",
    "email": ""
  },
  "dependencies": {
    "express": "^4.17.1"
  }
}
```

2.8.2. Instalar pacote globalmente

O NPM também pode instalar pacotes globalmente, para que todo o aplicativo node.js do computador possa importar e usar os pacotes instalados. O NPM instala pacotes globais na pasta `/ <User> / local / lib / node_modules`.

Aplique `-g` no comando *install* para instalar o pacote globalmente. Por exemplo, o comando a seguir instalará o ExpressJS globalmente.

```
C:\MyNodeProj> npm install -g express
```

2.8.3. Pacote de atualização

Para atualizar o pacote instalado localmente no seu projeto Node.js, navegue no *prompt* de comando ou no caminho da janela do terminal para a pasta do projeto e escreva o seguinte comando de atualização.

```
C:\MyNodeProj> npm update <package name>
```

O comando a seguir atualizará o módulo ExpressJS existente para a versão mais recente.

```
C:\MyNodeProj> npm update express
```

2.8.4. Desinstalar pacotes

Use o comando a seguir para remover um pacote local do seu projeto.

```
C:\>npm uninstall <package name>
```

O comando a seguir desinstalará o ExpressJS do aplicativo.

```
C:\MyNodeProj> npm uninstall express
```


CAPÍTULO 3

CRIANDO UM APLICATIVO

Antes de dar início ao desenvolvimento de um aplicativo *web*, precisamos entender como criar um servidor *web* Node.js simples, como manipular solicitações HTTP e como funciona o Express.js. Vamos lá!

3.1. Servidor Web Node.js

Para acessar páginas da *web* de qualquer aplicativo da *web*, você precisa de um servidor da *web*. O servidor da Web tratará de todas as solicitações HTTP para o aplicativo da *web*, por exemplo, o IIS é um servidor da *web* para aplicativos da Web ASP.NET e o Apache é um servidor da *web* para aplicativos da Web PHP ou Java.

O Node.js fornece recursos para criar seu próprio servidor *web*, que manipulará solicitações HTTP de forma assíncrona. Você pode usar o IIS ou o Apache para executar o aplicativo da *web* Node.js, mas é recomendável usar o servidor da web Node.js.

3.1.1. Criar servidor da Web Node.js

O Node.js facilita a criação de um servidor *web* simples que processa solicitações recebidas de forma assíncrona.

O exemplo a seguir é um servidor *web* Node.js simples contido no arquivo *server.js*.

Arquivo = server.js

```
var http = require('http'); // 1 - Import Node.js core module
var server = http.createServer(function (req, res) { // 2 - creating server
    //handle incoming requests here..
});
server.listen(5000); // 3 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')
```

No exemplo acima, importamos o módulo *http* usando a função `require()`. O módulo *http* é um módulo principal do Node.js, portanto, não é necessário instalá-lo usando o NPM. A próxima etapa é chamar o método `createServer()` do *http* e especificar a função de retorno de chamada com o parâmetro *request* e *response*. Por fim, chame o método `listen()` do objeto de servidor retornado do método `createServer()` com o número da porta, para começar a ouvir solicitações recebidas na porta 5000. Você pode especificar qualquer porta não utilizada aqui.

Execute o servidor da *web* acima, escrevendo o `node server.js` comando no *prompt* de comando ou na janela do terminal e ele exibirá a mensagem como mostrado abaixo.

```
C:\> node server.js
Node.js web server at port 5000 is running.
```

É assim que você cria um servidor da web Node.js usando etapas simples. Agora, vamos ver como lidar com a solicitação HTTP e enviar resposta no servidor da web Node.js.

3.1.2. Manipular solicitação HTTP

O método `http.createServer()` inclui parâmetros de solicitação e resposta fornecidos pelo Node.js. O objeto de solicitação pode ser usado para obter informações sobre a solicitação HTTP atual, por exemplo, url, cabeçalho da solicitação e dados. O objeto de resposta pode ser usado para enviar uma resposta para uma solicitação HTTP atual.

O exemplo a seguir demonstra como manipular solicitação e resposta HTTP no Node.js.

Arquivo = `server.js`

```
var http = require('http'); // Import Node.js core module
var server = http.createServer(function (req, res) { //create web server
    if (req.url == '/') { //check the URL of the current request
        // set response header
        res.writeHead(200, { 'Content-Type': 'text/html' });

        // set response content
        res.write('<html><body><p>This is home Page.</p></body></html>');
        res.end();
    }
    else if (req.url == "/student") {
        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is student Page.</p></body></html>');
        res.end();
    }
    else if (req.url == "/admin") {
```

```

        res.writeHead(200, { 'Content-Type': 'text/html' });
        res.write('<html><body><p>This is admin Page.</p></body></html>');
        res.end();
    }
    else
        res.end('Invalid Request!');
});
server.listen(5000); //6 - listen for any incoming requests
console.log('Node.js web server at port 5000 is running..')

```

No exemplo acima, `req.url` é usado para verificar o URL da solicitação atual e, com base nisso, envia a resposta. Para enviar uma resposta, primeiro ele define o cabeçalho da resposta usando o método `writeHead()` e depois grava uma *string* como um corpo de resposta usando o método `write()`. Por fim, o servidor da *web* Node.js envia a resposta usando o método `end()`.

Agora, execute o servidor da *web* acima, como mostrado abaixo.

```

C:\> node server.js
Node.js web server at port 5000 is running..

```

Para testá-lo, você pode usar o programa de linha de comando *curl*, que a maioria das máquinas Mac e Linux possui pré-instalado.

```
curl -i http: // localhost: 5000
```

Você deve ver a seguinte resposta.

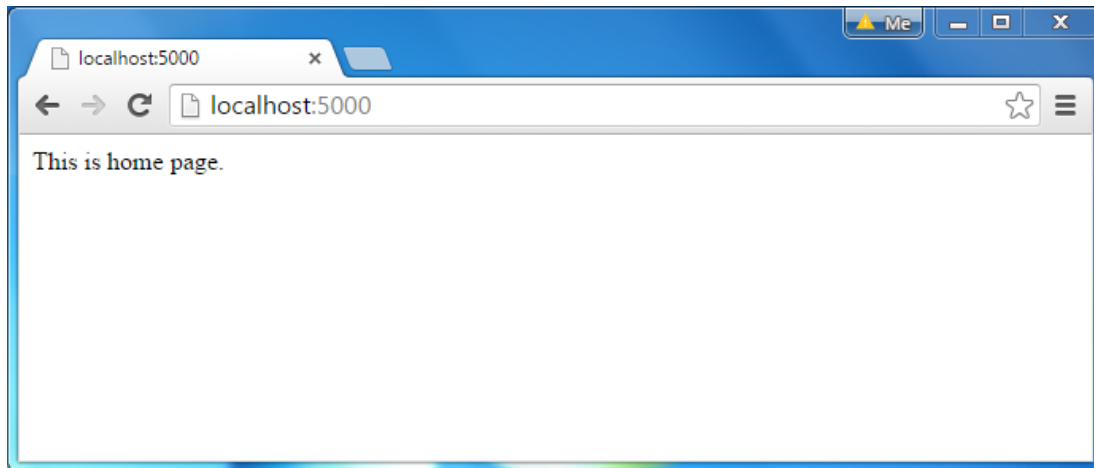
```

HTTP/1.1 200 OK
Content-Type: text/plain
Date: Tue, 28 Mar 2020 03:05:08 GMT
Connection: keep-alive
This is home page.

```

Para usuários do Windows, aponte seu navegador para `http://localhost:5000` e veja o resultado apresentado na Figura 14.

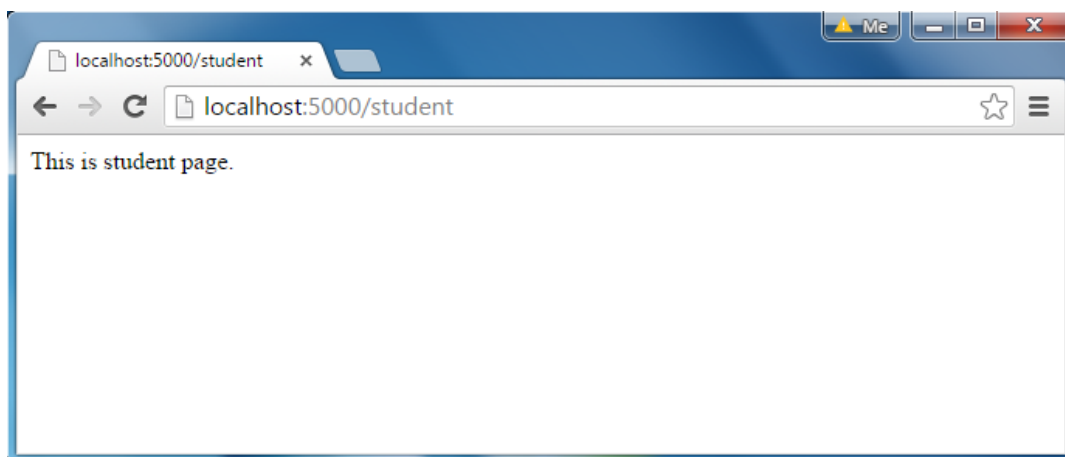
Figura 14. Resposta do servidor da web Node.js



Fonte: Elaborado autora.

Da mesma maneira, aponte seu navegador para `http://localhost:5000/student` e veja o seguinte resultado, ilustrado na Figura 15.

Figura 15. Resposta do servidor da web Node.js (student)



Fonte: Elaborado pela autora.

Ele exibirá “Solicitação inválida” para todas as solicitações além das URLs acima.

3.1.3. Enviando resposta JSON

O exemplo a seguir demonstra como atender à resposta JSON do servidor da *web* Node.js.

Arquivo = server.js

```
var http = require('http');  
var server = http.createServer(function (req, res) {  
  if (req.url == '/data') { //check the URL of the current request
```

```
        res.writeHead(200, { 'Content-Type': 'application/json' });
        res.write(JSON.stringify({ message: "Hello World" }));
        res.end();
    }
});
server.listen(5000);
console.log('Node.js web server at port 5000 is running..')
```

Dessa forma, você pode criar um servidor *web* simples que atenda a diferentes respostas.

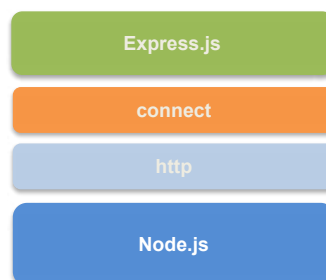
3.2. Express.js

“O Express é uma estrutura da *web* minimalista, rápida e sem opinião para o Node.js”
– site oficial: [Expressjs.com](https://expressjs.com).

Express.js é uma estrutura de aplicativo da *web* para Node.js. Ele fornece vários recursos que tornam o desenvolvimento de aplicativos da *web* rápido e fácil, o que leva mais tempo usando apenas o Node.js.

O Express.js é baseado no módulo de *middleware* do Node.js chamado *connect*, que, por sua vez, usa o módulo *http*. Portanto, qualquer *middleware* baseado em *connect* também funcionará com o Express.js. A Figura 16 apresenta a estrutura do Express.js.

Figura 16. Estrutura do Express.js



Fonte: Elaborado pela autora.

3.2.1. Vantagens do Express.js

- » Torna o desenvolvimento de aplicativos da Web Node.js rápido e fácil.
- » Fácil de configurar e personalizar.
- » Permite definir rotas do seu aplicativo com base em métodos HTTP e URLs.
- » Inclui vários módulos de *middleware* que você pode usar para executar tarefas adicionais mediante solicitação e resposta.

- » Fácil de integrar com diferentes mecanismos de modelos, como Jade, Vash, EJS, etc.
- » Permite definir um erro ao manipular o *middleware*.
- » Fácil de servir arquivos e recursos estáticos do seu aplicativo.
- » Permite criar um servidor de API REST.
- » Fácil de conectar com bancos de dados como MongoDB, Redis, MySQL.

3.2.2. Instale o Express.js

Você pode instalar o `express.js` usando o `npm`. O comando a seguir instalará a versão mais recente do `express.js` globalmente em sua máquina, para que todos os aplicativos Node.js em sua máquina possam usá-lo.

```
npm install -g express
```

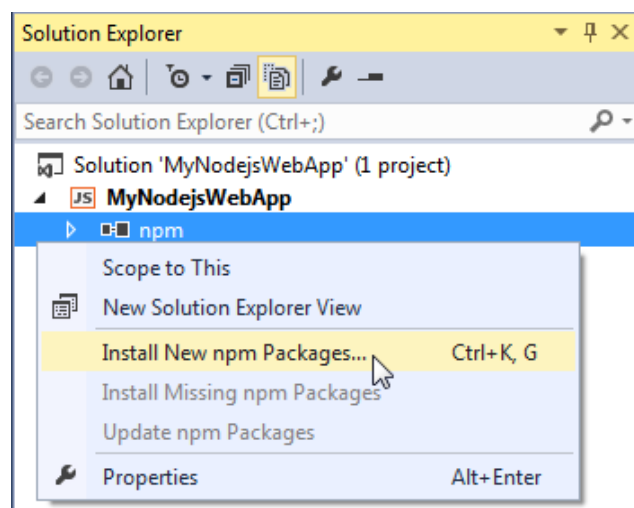
O comando a seguir instalará a versão mais recente do `express.js` local na sua pasta do projeto.

```
C:\MyNodeJSApp> npm install express --save
```

3.2.3. Instale o Express.js no Visual Studio

No aplicativo da *web* node.js, criaremos um aplicativo da *web* node.js simples no Visual Studio. Agora, para instalar o Express.js, clique com o botão direito do *mouse* no projeto *MyNodejsWebApp* -> selecione *Instalar Novos Pacotes npm*. A Figura 17 ilustra esse procedimento.

Figura 17. Instalando o Express.js no Visual Studio



Fonte: Elaborado pela autora.

Ele abrirá uma caixa de diálogo, digite “express”. Ele exibirá todos os pacotes começando com express. Selecione a versão mais recente do *express.js*, marque a caixa de seleção *Adicionar ao package.json* e clique no botão *Instalar pacote*.

Portanto, isso instalará o pacote *express.js* no seu projeto na pasta `node_modules`. Ele também irá adicionar uma entrada de dependências para *express.js*.

Agora, você está pronto para usar o Express.js no seu aplicativo.

3.3.3.3 O Aplicativo

Nesta seção, você aprenderá como criar um aplicativo *web* usando o Express.js.

O Express.js fornece uma maneira fácil de criar servidor da *web* e renderizar páginas HTML para diferentes solicitações HTTP, configurando rotas para seu aplicativo.

3.3.1. Servidor web

Antes de tudo, importe o módulo Express.js e crie o servidor da *web*, como mostrado abaixo.

Arquivo = `app.js`: Express.js Web Server

```
var express = require('express');
var app = express();
// define routes here..
var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```

No exemplo acima, importamos o módulo Express.js usando a função `require()`. O módulo *expresso* retorna uma função. Esta função retorna um objeto que pode ser usado para configurar o aplicativo Express (aplicativo no exemplo acima).

O objeto do aplicativo inclui métodos para rotear solicitações HTTP, configurar *middleware*, renderizar visualizações HTML e registrar um mecanismo de modelo.

A função `app.listen()` cria o servidor da *web* Node.js no *host* e na porta especificados. É idêntico ao método `http.Server.listen()` do Node.

Execute o exemplo acima usando o node `app.js` comando e aponte seu navegador para `http://localhost:5000`. Ele exibirá “Não é possível obter porque ainda não configuramos nenhuma rota”.

3.3.2. Configurando rotas

Use o objeto `app` para definir rotas diferentes do seu aplicativo. O objeto do aplicativo inclui métodos `get()`, `post()`, `put()` e `delete()` para definir rotas para solicitações HTTP GET, POST, PUT e DELETE, respectivamente.

O exemplo a seguir demonstra a configuração de rotas para solicitações HTTP.

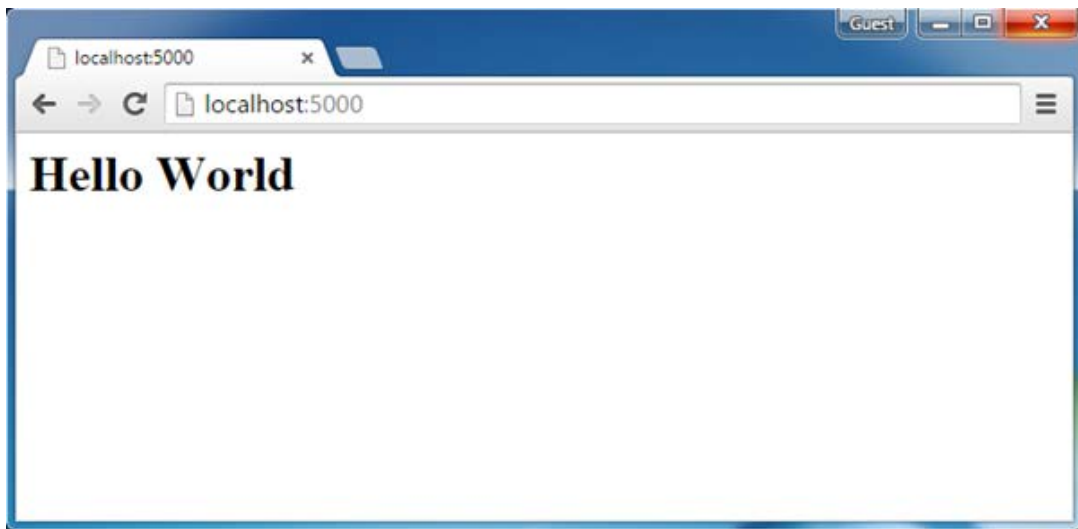
Exemplo: Configurando rotas no Express.js

```
var express = require('express');
var app = express();
app.get('/', function (req, res) {
  res.send('<html><body><h1>Hello World</h1></body></html>');
});
app.post('/submit-data', function (req, res) {
  res.send('POST Request');
});
app.put('/update-data', function (req, res) {
  res.send('PUT Request');
});
app.delete('/delete-data', function (req, res) {
  res.send('DELETE Request');
});
var server = app.listen(5000, function () {
  console.log('Node server is running..');
});
```

No exemplo acima, os métodos `app.get()`, `app.post()`, `app.put()` e `app.delete()` definem rotas para HTTP GET, POST, PUT, DELETE, respectivamente. O primeiro parâmetro é o caminho de uma rota que será iniciada após o URL base. A função de retorno de chamada inclui objeto de solicitação e resposta que será executado em cada solicitação.

Execute o exemplo acima usando o node `server.js` comando e aponte seu navegador para `http://localhost:5000` e você verá o resultado como apresentado na Figura 18.

Figura 18. Aplicativo Web Express.js



Fonte: Elaborado pela autora.

3.3.3. Manipulando solicitação POST

Aqui, você aprenderá como lidar com a solicitação HTTP POST e obter dados do formulário enviado.

Primeiro, crie o arquivo *Index.html* na pasta raiz do seu aplicativo e escreva o seguinte código HTML.

Exemplo: Configurando rotas no Express.js

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <meta charset="utf-8" />
  <title></title>
</head>
<body>
  <form action="/submit-student-data" method="post">
    First Name: <input name="firstName" type="text" /> <br />
    Last Name: <input name="lastName" type="text" /> <br />
    <input type="submit" />
  </form>
</body>
</html>
```

3.3.4. Body-parser

Para lidar com a solicitação HTTP POST no Express.js versão 4 e superior, é necessário instalar o módulo de *middleware* chamado *body-parser*. O *middleware* fazia parte do Express.js anteriormente, mas agora você deve instalá-lo separadamente.

Este módulo analisador de corpo analisa os dados codificados em JSON, *buffer*, *string* e URL enviados usando a solicitação HTTP POST. Instale o *body-parser* usando o NPM, como mostrado abaixo.

```
npm install body-parser -save
```

Agora, importe o *body-parser* e obtenha os dados da solicitação POST, conforme mostrado abaixo.

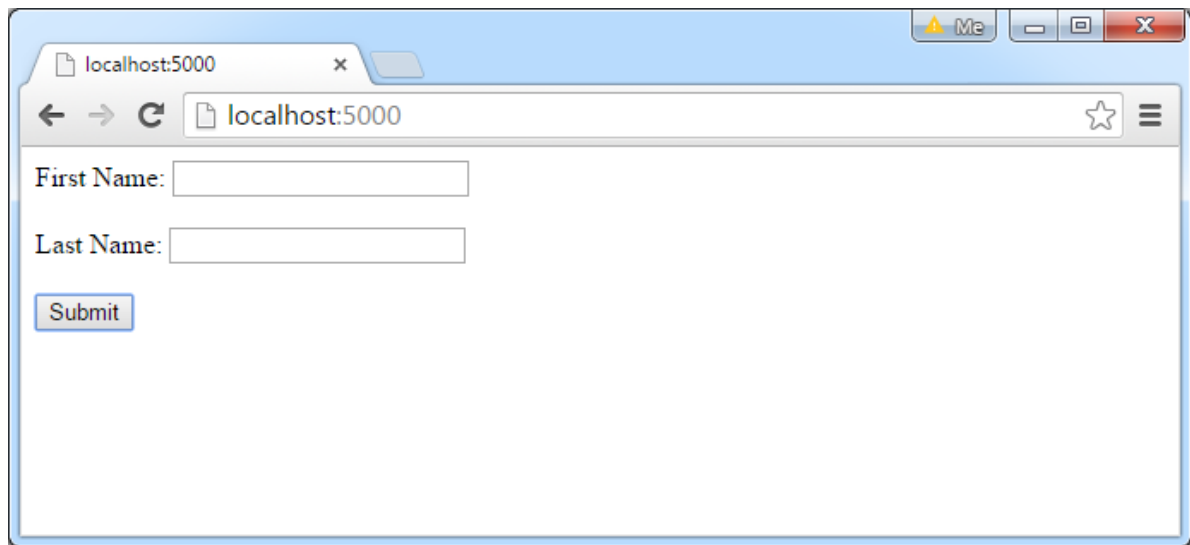
app.js: manipular rota POST no Express.js

```
var express = require('express');
var app = express();
var bodyParser = require("body-parser");
app.use(bodyParser.urlencoded({ extended: false }));
app.get('/', function (req, res) {
    res.sendFile('index.html');
});
app.post('/submit-student-data', function (req, res) {
    var name = req.body.firstName + ' ' + req.body.lastName;
    res.send(name + ' Submitted Successfully!');
});
var server = app.listen(5000, function () {
    console.log('Node server is running..');
});
```

No exemplo acima, os dados POST podem ser acessados usando `req.body`. O `req.body` é um objeto que inclui propriedades para cada formulário enviado. `Index.html` contém os tipos de entrada *firstName* e *lastName*, para que você possa acessá-lo usando `req.body.firstName` e `req.body.lastName`.

Agora, execute o exemplo acima usando o `node server.js` comando, aponte seu navegador para `http://localhost:5000` e veja o seguinte resultado, como ilustrado na Figura 19.

Figura 19. Formulário HTML para enviar solicitação POST



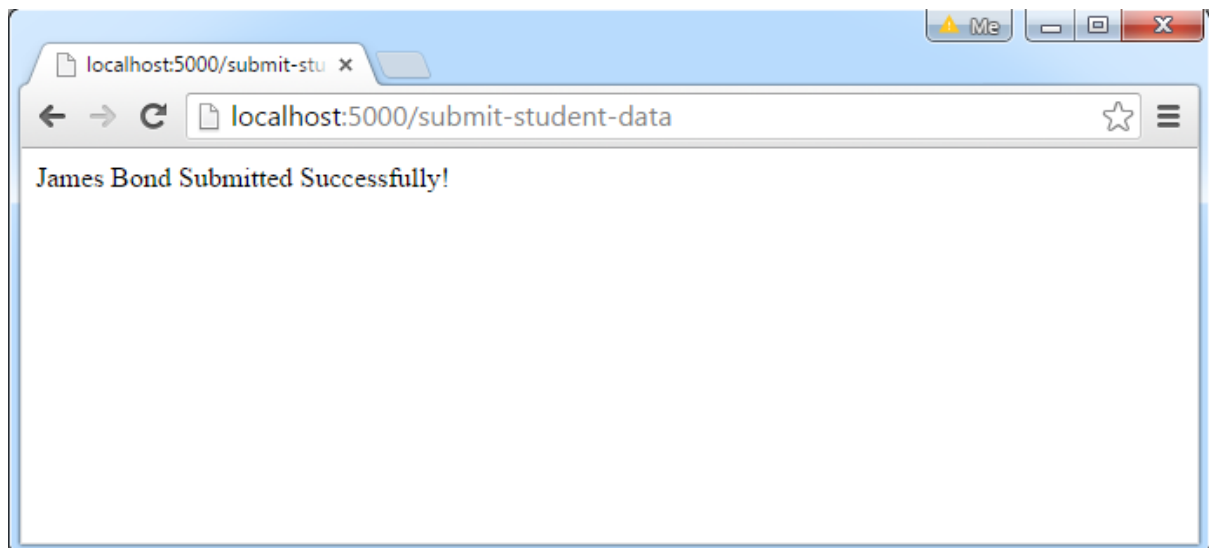
The screenshot shows a web browser window with the address bar set to `localhost:5000`. The page content includes a form with the following elements:

- A text input field labeled "First Name:".
- A text input field labeled "Last Name:".
- A button labeled "Submit" located below the "Last Name:" field.

Fonte: Elaborado pela autora.

Preencha o nome e o sobrenome no exemplo acima e clique em *enviar*. Por exemplo, digite “James” na caixa de texto Nome e “Bond” na caixa de texto Sobrenome e clique no botão Enviar. O seguinte resultado é exibido, conforme ilustrado na Figura 20.

Figura 20. Resposta da solicitação POST



Fonte: Elaborado pela autora.

É assim que você pode lidar com solicitações HTTP usando o Express.js.

REFERÊNCIAS

ALLIED. **7 Surprising Facts About Cloud Computing**. 2015. Disponível em: <https://www.alliedtelecom.net/facts-about-cloud-computing/>. Acesso em 23 mar. 2020.

BELSHE, M.; PEON, R. **Hypertext Transfer Protocol Version 2**. 2015. Disponível em: <https://httpwg.org/specs/rfc7540.html>. Acesso em 19 mar. 2020.

CONFLUENT. **Featuring Apache Kafka in the Netflix Studio and Finance World**. 2020. Disponível em: <https://www.confluent.io/blog/how-kafka-is-used-by-netflix/>. Acesso em 28 mar. 2020.

CONVENTIONAL COMMITS. **Conventional Commits 1.0.0-beta.2**. 2020. Disponível em: <https://www.conventionalcommits.org/en/v1.0.0-beta.2/>. Acesso em 29 out. 2020.

CROCKFORD, D. **Introducing JSON**. 2020. Disponível em: <https://www.json.org/json-en.html>. Acesso em 19 mar. 2020.

CRUZ, G. **Como é trabalhar como Desenvolvedor(a) Back-End, por Giovanni Cruz**. 2017. Disponível em: <https://medium.com/trainingcenter/como-%C3%A9-trabalhar-como-desenvolvedor-backend-b40255d75626>. Acesso em 23 mar. 2020.

DAVIS, D; VENNAM, B. **Knative 101: Exercises designed to help you achieve an understanding of Knative**. 2019. Disponível em: <https://developer.ibm.com/tutorials/knative-101-labs/>. Acesso em 24 mar. 2020.

DEV MEDIA. **O que é HTML5**. 2012. Disponível em: <https://www.devmedia.com.br/o-que-e-o-html5/25820>. Acesso em 20 mar. 2020.

ECMA. **ECMAScript Language Specification**. 2018. Disponível em: <http://ecma-international.org/ecma-262/9.0/index.html#Title>. Acesso em 19 mar. 2020.

FREECODCAMP. **Introduction to NPM Scripts**. 2018. Disponível em: <https://www.freecodecamp.org/news/introduction-to-npm-scripts-1dbb2ae01633/>. Acesso em 29 out. 2020.

FORSUM, G. **Backend infrastructure at Spotify**. 2013. Disponível em: <https://labs.spotify.com/2013/03/15/backend-infrastructure-at-spotify/>. Acesso em 21 mar. 2020.

FOWLER, C. O programador apaixonado. Casa do Código, 1º ed, 2014.

GITHUB. 2020. Disponível em: <https://github.com/nodejs/node>. Acesso em 29 out. 2020.

GRUNT. 2020. Disponível em: <https://gruntjs.com/>. Acesso em 29 out. 2020.

GULP. 2020. Disponível em: <https://gulpjs.com/>. Acesso em 29 out. 2020.

HOSTGATOR BRASIL. **Tudo o que você precisa para ser um programador front-end**. 2018. Disponível em: <https://www.hostgator.com.br/blog/tudo-o-que-voce-precisa-para-ser-um-programador-front-end/>. Acesso em 23 mar. 2020.

IBM. **Cloud Storage**. 2019. Disponível em: <https://www.ibm.com/cloud/learn/cloud-storage>. Acesso em 25 mar. 2020.

IBM. **Hypervisors**. 2019. Disponível em: <https://www.ibm.com/cloud/learn/hypervisors>. Acesso em 25 mar. 2020.

- IBM. **Istio**. 2019. Disponível em: <https://www.ibm.com/cloud/learn/istio>. Acesso em 24 mar. 2020.
- IBM. **Microservices**. Disponível em: <https://www.ibm.com/cloud/learn/microservices>. Acesso em 24 mar. 2020.
- IBM. **Virtual Machines (VMs)**. 2019. Disponível em: <https://www.ibm.com/cloud/learn/virtual-machines>. Acesso em 25 mar. 2020.
- IBM. **Desktop-as-a-Service (DaaS)**. 2018. Disponível em: <https://www.ibm.com/cloud/learn/desktop-as-a-service>. Acesso em 25 mar. 2020.
- IBM. **The state of container-based application development**. 2017. Disponível em: <https://www.ibm.com/cloud-computing/info/container-development/>. Acesso em 24 mar. 2020.
- IBM Cloud. Containerization Explained. 2019. Disponível em: https://www.youtube.com/watch?v=oqotVMX-J5s&feature=emb_logo. Acesso em 29 out. 2020.
- IBM Cloud. Kubernetes Explained. 2019. Disponível em: https://www.youtube.com/watch?v=aSrQRSk43lY&feature=emb_logo. Acesso em 29 out. 2020.
- IBM Cloud. Implantações Kubernetes: comece rapidamente. 2019. Disponível em: https://www.youtube.com/watch?v=Sulw5ndbE88&feature=emb_logo. Acesso em 29 out. 2020.
- IBM Cloud. Virtualization Explained. 2019. Disponível em: https://www.youtube.com/watch?v=FZRorG3HKIk&feature=emb_logo. Acesso em 29 out. 2020.
- IBM Cloud. GPUs: Explained. 2019. Disponível em: https://www.youtube.com/watch?v=LfdK-voSbGI&feature=emb_logo. Acesso em 29 out. 2020.
- IGTI BLOG. **O que faz um Desenvolvedor Full Stack?** 2018. Disponível em: <https://www.igti.com.br/blog/o-que-faz-um-desenvolvedor-full-stack/>. Acesso em 23 mar. 2020.
- IPERIUS BACKUP BRASIL. **Computação em nuvem: Inovações em TI para impulsionar os negócios**. 2019. Disponível em: <https://www.iperiusbackup.net/pt-br/computacao-em-nuvem-inovacoes-em-ti-para-impulsionar-os-negocios/>. Acesso em: 24 mar. 2020.
- KATZ, B. **How Much Does It Cost to Fight in the Streaming Wars?** 2019. Disponível em: <https://observer.com/2019/10/netflix-disney-apple-amazon-hbo-max-peacock-content-budgets/>. Acesso em 28 mar. 2020.
- KIM, J. **Kubernetes Networking: a lab on basic networking concepts**. 2019. Disponível em: <https://developer.ibm.com/tutorials/kubernetes-networking-101-lab/>. Acesso em 24 mar. 2020.
- LINDLEY, C. **Front-End Developer Handbook 2018**. 2018. Disponível em: <https://frontendmasters.com/books/front-end-handbook/2018/>. Acesso em 18 mar. 2020.
- LINTHICUM, D. **The essential guide to software containers for application development**. 2020. Disponível em: <https://techbeacon.com/enterprise-it/essential-guide-software-containers-application-development>. Acesso em 24 mar. 2020.
- MAMP. 2020. Disponível em: <https://www.mamp.info/en/windows/>. Acesso em 29 out. 2020.
- MATTOKA, M. **Are you sure you're using microservices?** 2019. Disponível em: <https://blog.softwaremill.com/are-you-sure-youre-using-microservices-f8d4e912d014>. Acesso em 28 mar. 2020.

REFERÊNCIAS

- MDN. **CSS Reference**. 2020. Disponível em: <https://developer.mozilla.org/en-US/docs/Web/CSS/Reference>. Acesso em 19 mar. 2020.
- MONTEIRO, C. **Desenvolvimento full stack: o que significa ser um profissional completo**. 2019. Disponível em: <https://mobile.blog/desenvolvimento-full-stack-o-que-significa-ser-um-profissional-completo/>. Acesso em 23 mar. 2020.
- NARKNEDE, N; SHAPIRA, G; PALINO, T. **Kafka: The definitive guide: real-time data and streams Processing at scale**. 2017. O'Reilly Media, Inc.
- NEEMAN, T. **Debug and log your Kubernetes applications**. 2019. Disponível em: <https://developer.ibm.com/tutorials/debug-and-log-your-kubernetes-app/>. Acesso em 24 mar. 2020.
- NETFLIX. **Evolution of the Netflix Data Pipeline**. 2016. Disponível em: <https://netflixtechblog.com/evolution-of-the-netflix-data-pipeline-da246ca36905>. Acesso em 28 mar. 2020.
- NETFLIX. **Spark and Spark Streaming at Netflix-(Kedar Sedekar and Monal Daxini, Netflix)**. 2015. Disponível em: <https://www.slideshare.net/SparkSummit/spark-and-spark-streaming-at-netflix-sedekar-daxini>. Acesso em 28 mar. 2020.
- NODEJS. 2020. Disponível em: <https://nodejs.org>. Acesso em 29 out. 2020.
- OLIVEIRA, W. **Como é trabalhar como Desenvolvedor Front-End, por Felipe Fialho**. 2017. Disponível em: <https://medium.com/trainingcenter/como-%C3%A9-trabalhar-como-desenvolvedor-front-end-por-felipe-fialho-1e3efbadef90>. Acesso em 23 mar. 2020.
- PAGANI, T. **Documentos acessíveis com WAI-ARIA em HTML5**. 2011. Disponível em: <https://tableless.com.br/documentos-acessiveis-com-aria-em-html5/>. Acesso em 20 mar. 2020.
- PARIS-WHITE, D. **Scale security while innovating microservices fast**. 2018. Disponível em: <https://www.ibm.com/cloud/blog/scale-security-innovating-microservices-fast>. Acesso em 24 mar. 2020.
- PINTEREST. 2020. Disponível em: <https://i.pinimg.com/564x/57/b8/05/57b805a0f624c0837b10e0c4cc5b1c55.jpg>. Acesso em 29 out 2020.
- REDMONK. **The Continued Rise of Apache Kafka**. 2017. Disponível em: <https://redmonk.com/fryan/2017/05/07/the-continued-rise-of-apache-kafka/>. Acesso em 28 mar. 2020.
- SHAPLAND, R; COLE, B. **What are cloud containers and how do they work?** 2019. Disponível em: <https://searchcloudsecurity.techtarget.com/feature/Cloud-containers-what-they-are-and-how-they-work>. Acesso em 24 mar. 2020.
- SOMAN, C. *et al.* **uReplicator: Uber Engineering's Robust Apache Kafka Replicator**. 2016. Disponível em: <https://eng.uber.com/ureplicator-apache-kafka-replicator/>. Acesso em 29 mar. 2020.
- STORYBOOK. **Build bulletproof UI components faster**. 2020. Disponível em: <https://storybook.js.org/>. Acesso em 29 out. 2020.
- TAKE BLOG. **Como se tornar um desenvolvedor back-end? 4 dicas para começar a sua carreira**. 2019. Disponível em: <https://take.net/blog/devs/desenvolvedor-back-end/>. Acesso em 23 mar. 2020.
- TUTORIALSTEACHER.COM. **JavaScript Tutorial**. 2020. Disponível em: <https://www.tutorialsteacher.com/javascript/javascript-tutorials>. Acesso em 29 out. 2020.

- TUTORIALSTEACHER.COM. **Node.js Process Model**. 2020. Disponível em: <https://www.tutorialsteacher.com/nodejs/nodejs-process-model>. Acesso em 26 mar. 2020.
- VENNAM, S. **Kubernetes Clusters: Architecture for Rapid, Controlled Cloud App Delivery**. 2019. Disponível em: <https://www.ibm.com/cloud/blog/new-builders/kubernetes-clusters-architecture-for-rapid-controlled-cloud-app-delivery>. Acesso em 24 mar. 2020.
- WACHAL, M. **Digital transformation with streaming application development**. 2019. Disponível em: <https://blog.softwaremill.com/digital-transformation-with-streaming-application-development-100fb4189149>. Acesso em 28 mar. 2020.
- WAMP SERVER. 2020. Disponível em: <https://www.wampserver.com/en/>. Acesso em 29 out. 2020.
- WARSKI, A. **Evaluating persistent, replicated message queues**. 2017. Disponível em: <https://softwaremill.com/mqperf/>. Acesso em 28 mar. 2020.
- WARSKI, A. **Event sourcing using Kafka**. 2018. Disponível em: <https://blog.softwaremill.com/event-sourcing-using-kafka-53dfd72ad45d>. Acesso em 28 mar. 2020.
- WEBPACK. 2020. Disponível em: <https://webpack.js.org/>. Acesso em 29 out. 2020.
- WHATWG. **DOM Living Standard**. 2020. Disponível em: <https://dom.spec.whatwg.org/>. Acesso em 19 mar. 2020.
- WHATWG. **HTTP Living Standard**. 2020. Disponível em: <https://html.spec.whatwg.org/multipage/introduction.html#introduction>. Acesso em 19 mar. 2020.
- WHATWG. **URL Living Standard**. 2020. Disponível em: <https://url.spec.whatwg.org/>. Acesso em 19 mar. 2020.
- WODEHOUSE, C. **The role of a front-end web developer: creating user experience & interactivity**. 2015. Disponível em: <https://www.upwork.com/hiring/development/front-end-developer/>. Acesso em 18 mar. 2020.
- W3C BRASIL. **Cartilha de Acessibilidade na Web**. 2020. Disponível em: <https://ceweb.br/cartilhas/cartilha-w3cbr-acessibilidade-web-fasciculo-I.html>. Acesso em 20 mar. 2020.
- W3 SCHOOLS. **Cloud Computing Architecture**. 2020. Disponível em: <https://www.w3schools.in/cloud-computing/cloud-computing-architecture/>. Acesso em 23 mar. 2020.
- W3TECHS. **Usage statistics of PHP for websites**. 2020. Disponível em: <https://w3techs.com/technologies/details/pl-php>. Acesso em 21 mar. 2020.
- WOODIE, A. **The Real-Time Rise of Apache Kafka**. 2016. Disponível em: <https://www.datanami.com/2016/04/06/real-time-rise-apache-kafka/>. Acesso em 28 mar. 2020.