HTTP

O módulo **http** é o principal módulo da nossas aplicações pois é com ele que criamos um servidor web para fornecer nossos sistemas.

Ele trabalha com diversas funcionalidades do protocolo HTTP, porém não iremos abranger todas.

Esse é um módulo nativo, não necessitando que seja instalado anteriormente.

Como usar

Para utilizar esse módulo basta importá-lo para seu código:

```
require('http')
```

Cada requisição enviada possui cabeçalhos que dizem o que essa requisição faz, vamos ver um exemplo de uma requisição **GET.**

```
GET / HTTP/1.1
Host: localhost
Connection: close
User-Agent: Chrome/46.0.2490.86
Accept-Encoding: gzip
Accept-Charset: ISO-8859-1,UTF-8;q=0.7,*;q=0.7
Cache-Control: no
Accept-Language: de,en;q=0.7,en-us;q=0.3
Referer:
```

HTTP/1.1 200 OK
Connection: keep-alive
Content-Encoding: gzip
Content-Type: text/html
Date: Sun, 06 Dec 2015 01:07:17 GMT
ETag: W/"55f9df1c-23f"
Last-Modified: Wed, 16 Sep 2015 21:29:00 GMT
Server: nginx
Transfer-Encoding: chunked

Note na primeira linha onde recebemos: **HTTP/1.1 200 OK**.

Esse **200** é o código de *status* da nossa resposta.

Então perceba que o **cliente** envia uma requisição com um **verbo HTTP** e seus cabeçalhos, depois de recebida e tratada o servidor responde com um **STATUS CODE** e seus cabeçalhos.

Methods

O protocolo **HTTP** possui um conjunto de métodos/verbos que o cliente pode invocar, veja abaixo a lista dos verbos mais usados:

GET

Requisita um representação do recurso especificado (O mesmo recurso pode ter várias representações, ao exemplo de serviços que retornam XML e JSON).

HEAD

Retorna os cabeçalhos de uma resposta (sem o corpo contendo o recurso)

POST

Envia uma entidade e requisita que o servidor aceita-a como subordinada do recurso identificado pela URI.

PUT

Requisita que uma entidade seja armazenada embaixo da URI fornecida. Se a URI se refere a um recurso que já existe, ele é modificado; se a URI não aponta para um recurso existente, então o servidor pode criar o recurso com essa URI.

DELETE

Apaga o recurso especificado.

TRACE

Ecoa de volta a requisição recebida para que o cliente veja se houveram mudanças e adições feitas por servidores intermediários.

OPTIONS

Retorna os métodos HTTP que o servidor suporta para a URL especificada.

CONNECT

Converte a requisição de conexão para um túnel TCP/IP transparente, usualmente para facilitar comunicação criptografada com SSL (HTTPS) através de um proxy HTTP não criptografado.

PATCH

Usado para aplicar modificações parciais a um recurso.

E são com 4 verbos diferentes que criamos um CRUD, que é **essencial em qualquer sistema**.

No CRUD precisamos ter 4 ações:

- Create
- Retrieve/Read
- Update
- Delete

Status Codes

Os códigos de retorno HTTP são compostos por 3 dígitos que seguem um formato padrão dando melhor direcionamento para a identificação correta do retorno.

Os códigos de *status* são divididos em:

1XX Informacional

Não há necessidade de se preocupar com este, serve apenas para informar que a informação foi recebida e que o processo continua.

2XX Sucesso

Significa que o pedido foi recebido com sucesso. É o que sempre acontece quando suas páginas são carregadas

200 – OK

O pedido ao servidor foi atendido com sucesso. A página web existe e será enviada ao user-agent (navegador, robô de busca…).

3XX Redirecionamento

Serve para avisar direto no cabeçalho HTTP uma mudança de página. Diferente de um Meta Refresh ou usar javascript, ele permite um redirecionamento, e é importante para SEO.

302 – Movido Temporariamente

Serve também para mover, mas com função temporária. A vantagem é que você pode reverter isto. Funciona bem para manutenções ou alteração não definitiva. O robô de busca continua visitando o endereço original.

4XX Erro do Cliente

Deve ser tratado com atenção pois o conteúdo não estará acessível para o visitante nem para o site de busca.
Problema para indexar.

401 – Não autorizado

O acesso a página não esta autorizado pois possivelmente a pessoa não está logada. Isto impede de uma página ser indexada por exemplo.

403 – Proibido

Neste caso o robô de busca também não terá como indexar o conteúdo.

404 – Não encontrado

É o código de retorno pode ser uma página ou arquivo que não existe no servidor, como um arquivo apagado. Pode ser usado para apresentar uma página com conteúdos relacionados à URL procurada.

5XX Erro do Servidor

O servidor não consegui atender o pedido por algum erro.

Também não permitirá a indexação da página.

500 – Erro interno do servidor

503 – Serviço indisponível

Pode ser um erro temporário. Uma manutenção ou uma grande quantidade de acessos pode derrubar o servidor.

Lista dos códigos de *status*.

createServer

Para iniciarmos um servidor HTTP utilizaremos a função **createServer** que recebe uma função com 2 parâmetros:

- request;
- response.

```
// hello-world.js
var http = require('http');

http.createServer(function(request, response){
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Oi Mundo");
  response.end();
}).listen(3000, function(){
  console.log('Servidor rodando em localhost:3000');
});
```

OU

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

var server = http.createServer(function(request, response){
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Oi Mundo");
  response.end();
});

server.listen(3000, function(){
  console.log('Executando Servidor HTTP');
});
```

A única diferença que no primeiro código não atribuimos o servidor em uma variável por isso encadeamos a função **listen** que faz o servidor *subir* na porta passada para ela e executando um *callback* após.

Percebeu que uma função sem nome foi passada para o createServer?

```
function(request, response){
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Oi Mundo");
  response.end();
});
```

Isso se chama **função anônima** e é uma característica **muito importante** do JavaScript, nessa função respondemos para o cliente que fez a requisição.

Mas como estamos trabalhando com o Navegador para acessar nosso servidor vamos retornar um HTML então.

```
// hello-http.js
var http = require('http');

http.createServer(function(request, response){
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("<h1>Oi Mundo</h1>");
  response.end();
}).listen(3000, function(){
  console.log('Servidor rodando em localhost:3000');
});
```

Então como nós retornamos nossa resposta corretamente?

Corrigindo o cabeçalho da resposta.

```
// hello-http.js
var http = require('http');

http.createServer(function(request, response){
  response.writeHead(200, {"Content-Type": "text/html"});
  response.write("<h1>Be MEAN</h1>");
  response.end();
}).listen(3000, function(){
  console.log('Servidor rodando em localhost:3000');
});
```

Aprendemos a enviar um HTML escrevendo ele na resposta, agora é a hora de respondermos com um HTML já criado, então primeiramente crie um **index.html**, na mesma pasta dos seus códigos, com o seguinte conteúdo:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <title>Aula web II - html</title>
</head>

<body>
    <h1>Aula web II - html</h1>
</body>

</html>
```

Depois crie o seguinte script **hello-html.js**.

```
// hello-html.js
var http = require('http')
  , fs = require('fs')
  , index = fs.readFileSync('index.html');

http.createServer(function(request, response){
  response.writeHead(200, {"Content-Type": "text/html"});
  response.end(index);
}).listen(3000, function(){
  console.log('Servidor rodando em localhost:3000');
});
```

Com isso aprendemos como a criar um simples servidor HTTP para nossas futuras aplicações.

Rotas

```
var http = require('http');
var server = http.createServer(function(request, response) {
    response.writeHead(200, {
        "Content-Type": "text/html"
    });
    if (request.url == "/") {
        response.write("<h1>Páginaprincipal</h1>");
    } else if (request.url == "/bemvindo") {
        response.write("<h1>Bem-vindo</h1>");
    } else {
        response.write("<h1>Páginanãoencontrada</h1>");
    }
    response.end();
});
server.listen(3000, function() {
    console.log('Servidorrodando!');
});
```

Exercícios

1 Descreva os tipos de requisição e qual e a função de cada 1

2 Crie uma aplicação node que mostre uma mensagem no navegador acessando localhost:3000

3 Crie uma aplicação node que tenha 2 paginas uma bem vindo na rota ‘/’ e outra que mostra os dados de um contato na rota ‘/user’, e deve mostrar uma mensagem Página não encontrada