*bptiago*

**Conceitos importantes:**

Callback: é uma função que passamos como parâmetro de outra função. A ideia é que a função callback seja executada em algum momento durante a execução da função principal.

Buffer: é um local finito (tamanho limitado) e temporário para armazenamento de dados. Geralmente, os dados serão recebidos no buffer e rapidamente transferidos para outro lugar, como um BufferedReader do Java, que armazena temporariamente os dados/bytes de um arquivo, por exemplo.

Stream: é uma sequência de dados que fica disponível dentro de um período de tempo. Por exemplo, streaming de vídeos; não se baixa o filme por completo, mas sim pequenas porções de dados do filme ao longo do tempo.

Stream & Buffer: é comum um buffer e uma stream atuarem juntos. Uma porção de dados "desce" pela stream e fica armazenada temporariamente no buffer. A informação então saíra do buffer e será processada finalmente.

- Chunk: é uma porção de dados de uma stream.

Pipe: é um meio de conectar duas streams, de modo que você é capaz de escrever numa stream o que está sendo lida numa outra stream.

Character set: tabela de conversão onde cada caracter possui um código numérico especifico. A partir disso, é possível converter números em caracteres e vice-versa.

Encoding: são formas como caracteres são armazenados em valores binários. h → 104 (char. set) → 01101000 (char. encoding).

Error-first callback: ao passar uma função como callback, o primeiro parâmetro disponível no callback é um objeto de erro.

- Isso é padrão para a funções core do node. Por exemplo: fs.readFile(filename, (err, data) = > {...})

## Node

**ECMASCRIPT:** é um padrão/design no qual o JavaScript é baseado. Esse padrão define regras de função, código, operações de lógica e aritmética etc.

**JavaScript V8:** é a engine na qual o node foi desenvolvido em cima. A engine é capaz de processar e executar código JavaScript por fora do navegador (isso acontece pois o código em JavaScript no final será compilado e interpretado em linguagem de máquina, não necessitando mais de um navegador para rodar).

- Projeto open source desenvolvido pela Google
- Escrita em C++

## Modulação de arquivos e componentes

**Módulos:** são códigos/arquivos separados que podem ser unidos e utilizados para executar um código principal e formar um arquivo central. Resumindo, módulos podem ser entendidos como subdivisões de um script.

**require():** comando para adicionar um  módulo num arquivo.

- A importação de módulos customizados pode ser feita em arquivos e em diretórios. Para arquivos, é preciso passar o path do script .js no require(). Para diretórios, após passar o path da pasta, o node irá procurar por um arquivo index.js dentro da pasta.
- O require() mantém um cache dos módulos importados, dessa forma, ao importar o mesmo módulo pela segunda vez, é retornado na verdade o cache do módulo, não sua versão “original”. Isso pode gerar confusão dependendo do jeito como o módulo é exportado.
    - Por exemplo, se você exporta uma instância de um objeto no módulo, todos os require() desse módulo irão retornar objetos compartilhando o mesmo endereço de memória, podendo gerar cenários inesperados devido a referência de memória. Uma alternativa a isso será exportar a função construtora do objeto, não a instância em si.

**module.exports:** expõe os trechos de um módulo que devem ser exportados quando require() for utilizado.

- module.exports é na verdade um atributo dentro de um objeto Module/module que é criado pelo nodejs quando um módulo é requirido via require().
- Nesse sentido, é possível atribuir um ou mais dados ao module.exports, passando listas, objetos com múltiplos atributos, funções como atributos secundários (ex: module.exports.minhaFn = () => {}), etc.

Módulos personalizados também podem ser importados e exportados com recursos nativos do JavaScript ES6. Por exemplo, export default function name() {}, import name from ‘./functions.js’;, etc.

## Eventos

São ações e coisas que acontecem no nosso aplicativo que podem ser respondidas. No Node, existe eventos do sistema (providos a partir do código C++) e eventos personalizados (providos pelo JavaScript).

- Conceitos importantes: listener e emitter. De forma resumida, quando um listener identificar que um evento específico ocorreu, o emitter irá transmitir ao código a resposta adequada àquele evento ouvido/percebido.

## Leitura e escrita de arquivos

**Buffers:** armazenará dados de forma binária, em bits. Ao ser instanciado - new Buffer(args) - receberá um dado e armazenará ele sob forma de bits. Posteriormente, poderá ser feita a conversão dos bits a partir das tabelas de encoding e charset.

- Na maioria dos casos, buffers não serão utilizados diretamente, mas aparecerão como etapas secundárias de processos ou funções comuns no node, como manipulação de arquivos.

- É preciso tomar cuidado com buffers, no sentido que se muitas pessoas utilizarem buffers grandes (que contem arquivos grandes) simultaneamente no seu aplicativo pode haver um problema de memória e performance. Um modo de contornar isso é usando streams.

**Streams:** no node, possuem ligação com os event emitters, sendo assim, ao ler uma stream de dados é possível ouvir por eventos específicos. Uma stream pode ser readable (somente lê) ou writeable (somente escreve) sob a perspectiva do node.

- Quando um dado completo é menor que o tamanho de um buffer, a stream será composta pelo dado completo. Agora, quando o dado é maior que o tamanho de um buffer, serão produzidas streams suficientes para ler o arquivo inteiro.

## HTTP / Server side

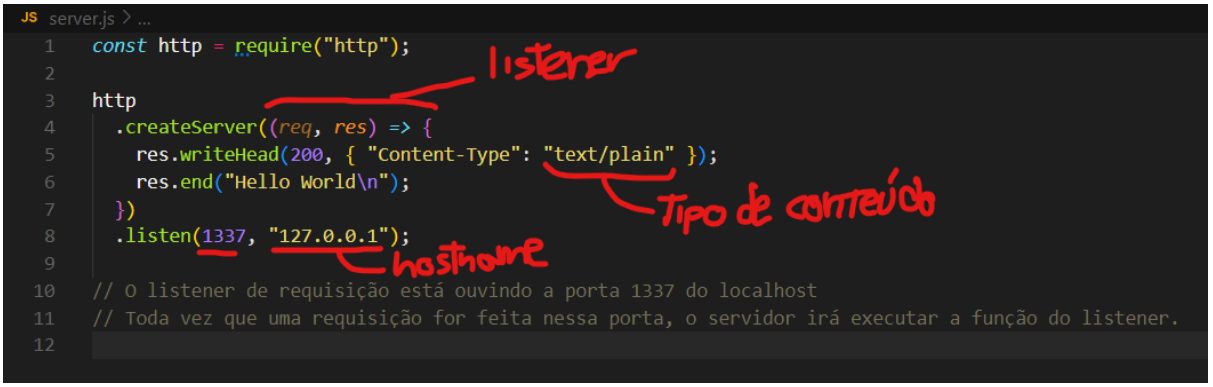
Em função do protocolo TCP, a informação trocada entre duas máquinas é transmitida sob a forma de pacotes de informações. O node trata esses pacotes como se fossem streams.

Dentro de um servidor web, podem existir inúmeros programas que recebem dados via servidor, dessa forma, para que o pacote de informação recebido seja encaminhado para o programa correto é preciso definir uma "porta" única para cada ferramenta, incluindo o node.

Um servidor node está atrelado diretamente a eventos, no sentido que o servidor, quando criado, deve receber um listener para requisições.

- **http_parser:** módulo core do node; faz o "parse" das requisições e respostas http, quebrando as informações da requisição e transformando todos os dados em um objeto JavaScript (que poderá ser acessado e manipulado posteriormente).

Criando um servidor com módulos core:




- No header content-type, deve ser definido a forma como você quer o seu conteúdo seja enviado. Por exemplo: application/json, text/html, text/plain etc.
- Para um JSON (basicamente um objeto JavaScript) ser enviado, é preciso "transformá-lo" numa string, através do método JSON.stringify(obj).

**API (Application Programming Interface):** uma API é um conjunto de definições e protocolos usado no desenvolvimento e na integração de aplicações. Uma API funciona como um mediador, ou comunicador, entre o usuário e o sistema. Na web, as APIs

geralmente ficam disponíveis por meio de URLs/endpoints e transmitem informações via requisições HTTP/HTTPS.

**Routing / Roteamento:** é o mapeamento de diferentes URLs e requisições, de modo que cada local seja responsável pela transmissão ou recebimento de um conteúdo específico.

## NPM (Node Package Manager)

- Package: porção de código, um script.

Resumidamente, NPM é um sistema de gerenciamento que facilita o download e instalação de pacotes e dependências.

A versão de um pacote é especificado por um número no seguinte modelo 0.0.0 (Major-Minor-Patch).

- Alteração de patch: ocorre quando se arruma bugs, por exemplo.
- Alteração de minor: ocorre quando algumas funcionalidades são adicionadas, porém elas não alteram o funcionamento geral do código.
- Alteração de major: ocorre quando há mudanças intensas no código, alterando significativamente o funcionamento geral. Quando há um major update, a versão antiga de um código pode não funcionar ao utilizar a nova versão major de um pacote.

Alguns comandos:

- npm init
- npm install <identificador do pacote>
- npm run <nome de um script>

## Express

Utilização do express:

```
const express = require("express");
app = express();

app.get(url, middleware, callback fn);
            (opcional)

app.post(url, middleware, callback fn);
            (opcional)

app.listen(port); // Invoca o servidor.
```

**Middlewares:** é um código que fica entre duas camadas de software. Por exemplo, em requisições para a web, um middleware seria um código executado entre um request e uma response.

- Isso pode ser utilizado para carregar um arquivo estático (imagem, css etc.) quando há uma requisição numa rota específica, por exemplo.
- Sintaxe: app.use(endpoint, (req, res, next) => {...});

API Rest: REST é um tipo específico de características e restrições de arquitetura para construir uma API. Segue algumas das especificações REST:

- As aplicações do cliente e do servidor devem ser completamente independentes uma da outra. Sendo assim, por exemplo, uma atualização no cliente não deve impactar o servidor de nenhum modo.
- Utilização dos métodos HTTP, como GET, POST, PUT e DELETE, para realizar operações em recursos.
- Transferência de dados entre cliente e servidor em um formato padrão, geralmente JSON ou XML.
- A API deve fornecer uma interface consistente e padronizada para acessar e manipular recursos.

## Node e databases

Ao recuperar informações de um banco de dados, é comum a tabela do select resultante ser automaticamente transformada em um lista de objetos, de modo que cada linha da tabela vire um objeto dentro da lista. Dentro desses objetos, as colunas e os valores das células da tabela são convertidas em atributos de objeto chave-valor, respectivamente.