**NODE**

**NPM – Node Package Manager**

**Criar npm**

npm init -y

**Installar pacotes:**

npm install {nome\_do\_pacote}

npm install express

**Atualizar pacotes:**

npm update

**Alterar a dependencies de um package (dev, prod):**

npm install express --save-dev // muda para devDependencies

npm install express --save-prod // muda para dependencies

**Instalar versão específica de package:**

npm install express@2.1.0 -E // -E para remover o ‘^’

**Desinstalar um pacakge:**

npm uninstall express

**Listar packages:**

npm ls // mostra os packages e suas dependencias

npm ls -depth=0 // mostra somente os packages instalados

npm outdated // mostra os packages desatualizados

**MANIPULANDO ARQUIVOS – FS**

**Leitura de pastas e arquivos:**

Importando o módulo:

const *fs* = require('fs').*promises*; // importa o módulo fs como promises

Criamos uma função para ler os arquivos e pastas de um caminho:

async function readdir(*rootDir*) {

*rootDir* = *rootDir* || path.resolve(\_\_dirname);

    const files = await *fs*.readdir(*rootDir*);

    walk(files, *rootDir*);

}

**\*IMPORTANTE:** fs.readdir retorna os arquivos e pastas do caminho informado

Na função walk, criamos uma lógica com fs.stat (que retorna status do arquivo lido) para podermos criar uma leitura recursiva dos itens.

async function walk(*files*, *rootDir*) {

    for (let file of *files*) {

// cria um caminho completo do arquivo

        const fileFullPath = path.resolve(*rootDir*, file);

        const stats = await *fs*.stat(fileFullPath);

        if (/\.git/g.test(fileFullPath)) continue; // re

        if (/node\_modules/g.test(fileFullPath)) continue; // re

// se o arquivo for uma pasta, ele recomeça a leitura

        if (stats.isDirectory()) {

            readdir(fileFullPath);

            continue;

        }

        if (!/\.html$/g.test(fileFullPath)) continue; // re

        console.log(fileFullPath);

    }

}

**Leitura e escrita de arquivos:**

***Escrita:***

Por meio de modulos, criamos um arquivo para a escrita de novos arquivos, que exporta um função que recebe o caminho onde o arquivo será escrito e os dados que irão conter nele:

const *fs* = require('fs').*promises*;

*module*.*exports* = (*caminho*, *dados*) => {

*fs*.writeFile(*caminho*, *dados*, { flag: 'w' });

}

**\*IMPORTANTE**: a flag é opcional (por padrão ‘w’)

No arquivo princial (app.js), criamos o caminho do arquivo a ser escrito e um array com objetos para servir de conteúdo. Nesse caso foi criado um arquivo .json, então utilizamos o método JSON.stringify() para converter o array de objetos para a formatação JSON:

const path = require('path');

const caminhoArquivo = path.resolve(\_\_dirname, 'teste.json');

const escreve = require('./modules/escreve');

const pessoas = [

    { nome: 'João' },

    { nome: 'Maria' },

    { nome: 'Eduardo' },

    { nome: 'Luiza' },

];

const json = JSON.stringify(pessoas, '', 2);

escreve(caminhoArquivo, json);

***Leitura:***

Para a leitura, foi criado um arquivo que exporta uma função com o fs.readFile, que serve para ler o conteúdo de um arquivo:

const *fs* = require('fs').*promises*;

*module*.*exports* = (*caminho*) => *fs*.readFile(*caminho*, 'utf-8');

\*IMPORTANTE: o fs.readFile() recebe o caminho do arquivo que será lido e a codificação que será utilizada (padrão: utf8)

Novamente no arquivo principal (app.js), importamos a função de leitura do modulo ler e criamos uma async function, pois a função de leitura retorna uma promise. Dentro dessa async function, enviamos o resultado da promise para uma outra função que irá transformar os dados JSON em formato JS utilizando o método JSON.parse(), onde podemos fazer o que quisermos com os dados lidos (no exemplo eles foram apenas logados na tela):

const path = require('path');

const caminhoArquivo = path.resolve(\_\_dirname, 'teste.json');

const ler = require('./modules/ler');

async function lerArquivo(*caminho*) {

    const arqLido = await ler(*caminho*);

    renderizaDados(arqLido);

}

function renderizaDados(*dados*) {

*dados* = JSON.parse(*dados*);

*dados*.forEach(*val* => console.log(*val*));

}

lerArquivo(caminhoArquivo);

**EXPRESS:**

O express é um módulo que é utilizado para gerenciar de forma prática as rotas do servidor de um site. Ele precisa ser instalado no node utilizando o comando npm i express.

Por padrão, para importar o empress utilizamos as seguintes nomenclaturas:

const express = require('express');

const app = express();

Para entender melhor o express, é importante conhecer o CRUD, que representa as operações que podem ser feitas no servidor:

CREATE, READ, UPDATE, DELETE -> significado literal de CRUD

POST    GET   PUT     DELETE -> elementos que representam CRUD no express

GET -> é uma solicitação de leitura

POST -> solicitação criação/envio de alguma informação

PUT -> solicitação de atualização de informação

DELETE -> solitação para deletar informação

Com o express, podemos definir o que cada solicitação pode fazer nas rotas que criamos para o site. Exemplo:

app.get('/', (*req*, *res*) => {

*res*.send(`

    <form action="/" method="POST">

    Nome: <input type="text" name="nome">

    <button>Enviar</button>

    `);

});

\* req e res significam, respectivamente, requisição e resposta do servidor

Aqui temos a definição de como a solitação GET irá se comportar na rota ‘/’, que seria a home do site. Aqui, utilizando o res.send(), estamos enviando como resposta um HTML que contém um formulário.

Para que esse formulário possa ser enviado, precisamos utilizar o método POST, como no exemplo abaixo:

app.post('/', (*req*, *res*) => {

*res*.send('Recebi o formulario!');

})

\*nesse caso o form irá carregar esse POST pois é para onde o action do form está apontando

Aqui configuramos o POST da rota ‘/’, que será exebido ao enviarmos com sucesso o formulário.