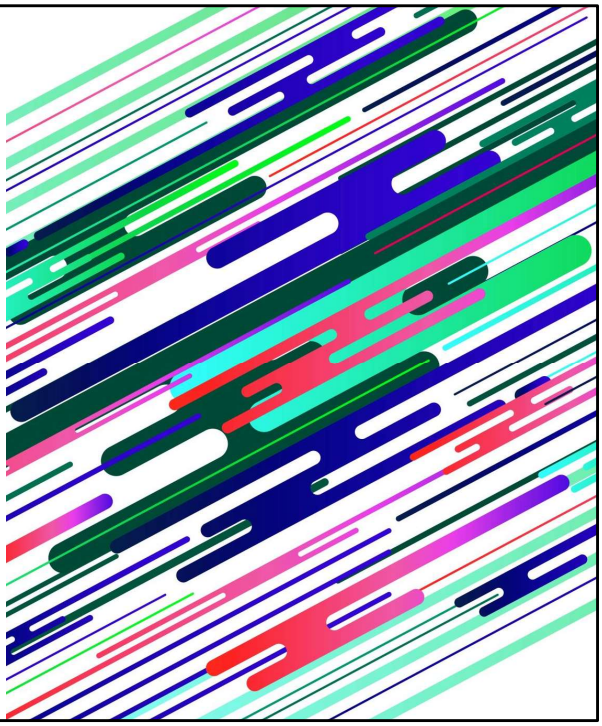
NODEJS



O QUE É O NODEJS

- O Node.js, ou Node para abreviar, é um ambiente de runtime JavaScript do lado do servidor open source
  - Servidores Web HTTP
  - Microsserviços ou back-ends de API sem servidor
  - Controladores para consulta e acesso a bases de dados
  - Interfaces de linha de comandos interativas
  - Aplicações de computador
  - Bibliotecas de servidor e cliente Internet das Coisas (IoT) em tempo real
  - Plug-ins para aplicações de computador
  - Scripts da shell para manipulação de ficheiros ou acesso à rede
  - Modelos e bibliotecas de machine learning

COMO FUNCIONA O NODEJS

- O Node.js é baseado num ciclo de eventos de um só thread. Este modelo de arquitetura processa operações simultâneas de forma eficiente. A *simultaneidade* refere-se à capacidade de o ciclo de eventos executar funções de chamada de retorno de JavaScript após a conclusão de outros trabalhos.
- Neste modelo de arquitetura:
  - *Um só thread* significa que o JavaScript tem apenas uma pilha de chamadas e só pode realizar uma ação de cada vez.
  - O *ciclo de eventos* executa o código, recolhe e processa os eventos e executa as próximas subtarefas na fila de eventos.

Uma *thread* é uma sequência única de instruções programadas que o sistema operativo pode gerir de forma independente.

CRIAR UM NOVO PROJETO NODE.JS E TRABALHAR COM DEPENDÊNCIAS

Configurar o package.json

PACKAGE.JSON

- é um ficheiro de manifesto para o seu projeto Node.js.
- Contém informações de metadados sobre o seu projeto.
- Controla como as suas dependências são geridas, que ficheiros entram num ficheiro destinado ao npm, entre outros.

DEMONSTRAÇÃO

•npm init: este comando inicia um assistente que lhe pede informações sobre o nome, versão, descrição, ponto de entrada, comando de teste, repositório Git, palavras-chave, autor e licença de um projeto.
•npm init -y: este comando utiliza o -y sinalizador e é uma versão mais rápida do npm init comando porque *não* é interativo. Em vez disso, este comando atribui automaticamente valores predefinidos para todos os valores que lhe é pedido para introduzir com o npm init.

EXEMPLO DE FICHEIRO

```
{
  "name": "my project",
  "version": "1.0.0",
  "description": "",
  "main": "script.js",
  "dependencies": {},
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

ode considerar que todas as propriedades possíveis no ficheiro package.json pertencem aos seguintes grupos:

•**Metadados**: as propriedades neste grupo definem as meta-informações sobre o projeto. As propriedades incluem o nome do projeto, descrição, autor, palavras-chave, etc.

•**Dependências**: existem duas propriedades que descrevem as bibliotecas que estão a ser utilizadas: dependencies e devDependencies. Mais adiante no módulo, irá aprender a utilizar estas propriedades para instalar, atualizar e separar dependências.

•**Scripts**: nesta secção, pode listar scripts para tarefas de projeto, como iniciar, compilar, testar e lint.

EXEMPLOS DE SCRIPT

- **start**: invoca o node comando com o ficheiro de entrada como um argumento. Um exemplo pode ser node ./src/index.js. Esta ação invoca o node comando e utiliza o ficheiro de index.jsentrada .
- **build**: descreve como criar o seu projeto. O processo de compilação deve produzir algo que possa enviar. Por exemplo, um comando de compilação pode executar um compilador TypeScript para produzir a versão JavaScript do projeto que pretende enviar.
- **test**: executa os testes do seu projeto. Se estiver a utilizar uma biblioteca de testes de terceiros, o comando deve invocar o ficheiro executável da biblioteca.
- **lint**: invoca um programa linter como o ESLint.

O linting encontra inconsistências no código. Um linter geralmente oferece uma forma de também corrigir as inconsistências. Ter código consistente pode aumentar significativamente a legibilidade, o que acelera o desenvolvimento de funcionalidades e adições ao código.

```
"scripts" : {
  "start" : "node ./dist/index.js",
  "test": "jest",
  "build": "tsc",
  "lint": "eslint"
}
```

PACOTES: PORQUE DE SEREM NECESSÁRIOS?

- **Obter um código melhor**
- **Poupar tempo**
- **Manutenção**

•**Obter um código melhor**: O seu programa precisa de suportar uma tarefa específica, como a segurança, onde precisa de implementar a autenticação e a autorização? A segurança é um tipo de tarefa que precisa de *ter razão* para proteger os seus dados e os dados do cliente. Existem padrões de segurança padrão e bibliotecas utilizadas por muitos programadores. Estas bibliotecas implementam funcionalidades de que provavelmente sempre precisará e os problemas são corrigidos nas bibliotecas à medida que surgem. Deve utilizar estas bibliotecas em vez de reinventar a roda. A principal razão é que não é provável que faça um trabalho tão bom de escrever o código por si mesmo, porque há tantos casos edge a considerar.
•**Poupar tempo**: provavelmente, pode codificar a maior parte do seu programa e criar bibliotecas de componentes utilitários ou de IU. Mas a programação leva tempo. Mesmo que o resultado final seja comparável ao que está lá fora, não é uma boa utilização do seu tempo para replicar o trabalho de escrever o código se não for necessário.
•**Manutenção**: todas as bibliotecas e aplicações precisam de manutenção mais cedo ou mais tarde. A manutenção implica a adição de novas funcionalidades e a correção de erros. A manutenção de uma biblioteca é uma boa utilização do seu tempo ou do tempo da sua equipa? Ou é melhor permitir que uma

equipa de software open source lide com o mesmo?

PACOTES: AVALIAR UM PACOTE

- **Tamanho**: o número de dependências pode criar uma grande pegada. Se existir limitações de hardware, o tamanho pode ser um fator significativo.
- **Licenciamento**: o licenciamento pode ser um fator se estiver a produzir software que pretende vender. Se tiver uma licença numa biblioteca de terceiros e o criador da biblioteca não permitir que seja incluída no software que está à venda, poderá deparar-se com uma situação legal.
- **Manutenção ativa**: se o seu pacote depender de uma dependência preterida ou não tiver sido atualizada durante muito tempo, manter o programa poderá ser difícil.

TRABALHAR COM SISTEMA DE FICHEIROS

- Módulo fs
  - O módulo *fs* está incluído por predefinição no Node.js, por isso não precisa de o instalar a partir do npm
- A utilização do espaço de nomes de OnPremisse é a forma preferencial de trabalhar com o módulo fs , uma vez que lhe permite utilizar async.
  - Evita a confusão de chamadas de retorno ou o bloqueio de métodos síncronos.

- `const fs = require("fs").promises;`

Os grandes retalhistas escrevem frequentemente dados em ficheiros para que possam ser processados mais tarde em lotes.
A Tailwind Traders pede a cada uma das respetivas lojas que escrevam os seus totais de vendas e que enviem esse ficheiro para uma localização central. Para utilizar esses ficheiros, a empresa precisa de criar um processamento em lotes que possa funcionar com o sistema de ficheiros.
Aqui, vai aprender a utilizar Node.js para ler o sistema de ficheiros para detetar ficheiros e diretórios.

MODULO FS

- Para ler os conteúdos da pasta, pode utilizar o método **readdir**.


```
const items = await fs.readdir("stores");
console.log(items);
OUTPUT: [ 201, 202, sales.json, totals.txt ]
```

```
const items = await fs.readdir("stores", { withFileTypes: true });
for (let item of items)
{
        const type = item.isDirectory() ? "folder" : "file";
        console.log(`${item.name}: ${type}`);
}
```

node ./nome.js

// 201: folder, 202: folder, sales.json: file, totals.txt: file

RECURSÃO

```
function findFiles(folderName) {
  const items = await fs.readdir(folderName, { withFileTypes: true });
  items.forEach((item) => {
    if (item.isDirectory()) {
      // this is a folder, so call this method again and pass in
      // the path to the folder
      findFiles(`${folderName}/${item.name}`);
    } else {
      console.log(`Found file: ${item.name} in folder: ${folderName}`);
    }
  });
}

findFiles("stores");
```

Um requisito comum consiste em ter pastas com subpastas, que também têm subpastas. Algures nesta árvore de pastas aninhadas, estão os ficheiros de que precisa. Precisa de um programa que consiga encontrar os ficheiros na árvore de pastas. Para tal, pode determinar se um item é uma pasta e, em seguida, procurar ficheiros nessa pasta. Repita esta operação para todas as pastas que encontrar.

Pode procurar estruturas de diretório aninhadas com um método que encontra pastas e, em seguida, chama-se a si mesmo para encontrar pastas dentro dessas pastas. Desta forma, o programa "orienta" a árvore do diretório até ler todas as pastas no interior. Quando um método se chama a si mesmo, denomina-se por *recursão*.

A recursão é uma funcionalidade avançada de muitas linguagens de programação. É provável que o utilizes muito no mundo real.

EXERCICIO EM CONJUNTO

Exercício – Trabalhar com o sistema de ficheiros - Training | Microsoft Learn

LER UM FICHEIRO

- `await fs.readFile("stores/201/sales.json");`

- `const bufferData = await fs.readFile("stores/201/sales.json");`
- `console.log(String(bufferData));`

- `const data = JSON.parse(await fs.readFile("stores/201/sales.json"));`
- `console.log(data.total);`

Os ficheiros existem em vários formatos. Os ficheiros JSON são os mais aconselháveis para trabalhar, devido ao suporte incorporado na linguagem. No entanto, poderá encontrar ficheiros que sejam .csv, de largura fixa ou algum outro formato. Nesse caso, é melhor procurar um analisador para esse tipo de ficheiro em npmjs.org.

ESCREVER NUM FICHEIRO

- const data = JSON.parse(await fs.readFile("stores/201/sales.json"));

// write the total to the "totals.json" file

- await fs.writeFile("salesTotals/totals.txt"), data.total);
- // totals.txt // 22385.32

ACRESCENTAR DADOS A UM FICHEIRO

```
const data = JSON.parse(await fs.readFile("stores/201/sales.json"));
// write the total to the "totals.json" file
await fs.writeFile(path.join("salesTotals/totals.txt"), `${data.total}\r\n`, { flag: "a" });
```

No exemplo anterior, o ficheiro é substituído sempre que escreve no mesmo. Por vezes, não quer isso. Por vezes, quer acrescentar dados ao ficheiro e não substituí-lo totalmente. Pode acrescentar dados ao transmitir um sinalizador para o writeFile método. Por predefinição, o sinalizador está definido como w, o que significa "substituir o ficheiro". Em vez disso, para acrescentar ao ficheiro, passe o sinalizador, o a que significa "acrescentar".

EXERCICIO EM CONJUNTO

- Exercício – Ler e escrever em ficheiros - Training | Microsoft Learn