awari

Unidade 3: Node.js

O coração do Javascript no backend

O que é Node.js? 

Foi criado a partir do **V8**, motor responsável por ler e executar instruções **JS** no Chrome. Costumamos dizer que o **V8** interpreta o código **JS**, e por isso é chamado de **interpretador**, **engine** ou, ainda, **runtime**. Por isso, é comum ouvir que o **Node.js** é um runtime para rodar JS no backend.

Apesar de ser baseado no **V8**, o **Node.js** possui algumas diferenças em relação ao original. As principais são:

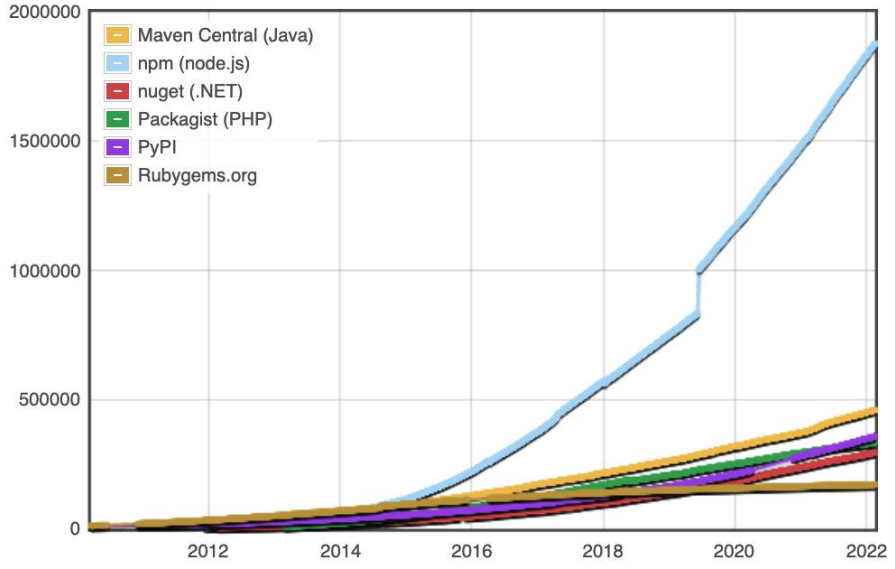
- ausência de métodos para manipulação de páginas web (DOM)
- métodos que permitem acessar o sistema de arquivos e rede diretamente

Por que usar Node.js? 💻

Se você fizer uma rápida pesquisa no Google, irá chegar aos resultados que o **Node.js** é uma das linguagens mais procuradas nas **vagas de emprego** que encontramos no mercado

Além disso, o **npm** (gerenciador de pacotes do **JS**) é, de forma disparada, o repositório de pacotes das linguagens que tem mais contribuição da comunidade. É possível constatar isso na imagem no próximo slide:

Por que usar Node.js? 💻



Mesmo em comparação com outras linguagens muito populares, como o **PHP** e o **Java**, o **Javascript**, com o **npm**, é único que tem mais de 500 mil pacotes (ou módulos) disponíveis, chegando quase em 2 milhões.

Fonte: modulecounts.com

Instalação e primeiros passos 💻

Fonte: modulecounts.com

NPM 

O **NPM** (Node Package Manager) é o repositório oficial para publicação de pacotes **Node.js**. Atualmente, uma média de 1104 pacotes são publicados no **NPM** todos os dias, segundo o site modulecounts.com.

Mas afinal, o que é um **pacote**?

Um **pacote** é um conjunto de arquivos que exportam um ou mais módulos do **Node.js**.

NPM

Javascript Assíncrono 💻

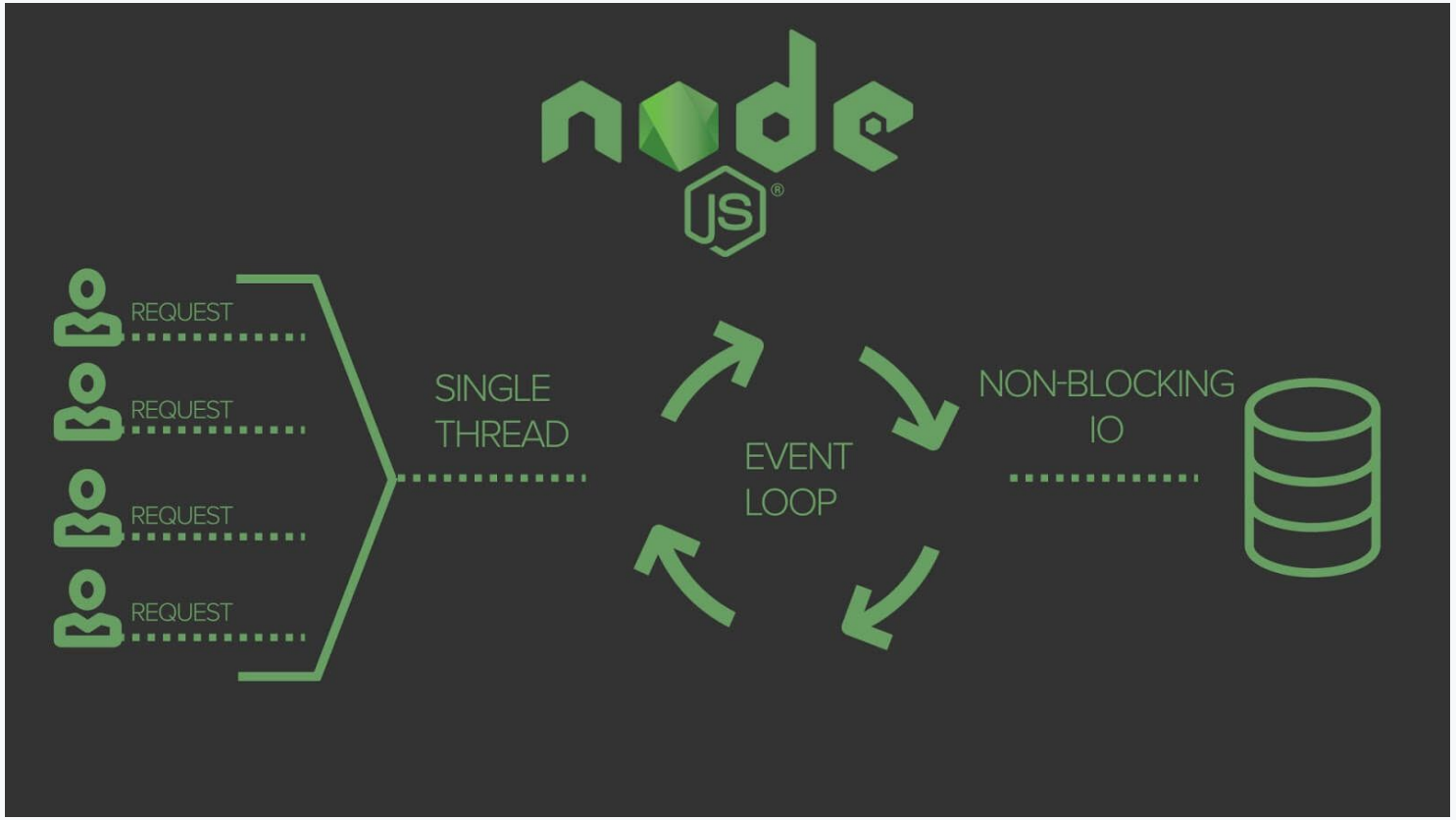
O Node.js é uma linguagem **single thread**, ou seja, só consegue executar uma instrução por vez. Mas, suas instruções podem ser executadas de forma **não-bloqueante**. Mas, o que isso quer dizer?

Por exemplo: se o servidor receber diversas instruções em uma requisição, e a primeira delas é acessar o banco de dados, as outras não precisam esperar essa primeira instrução terminar para serem executadas. (a menos que tenha alguma dependência)

Em outras palavras, o Node.js trabalha com requisições **assíncronas**.

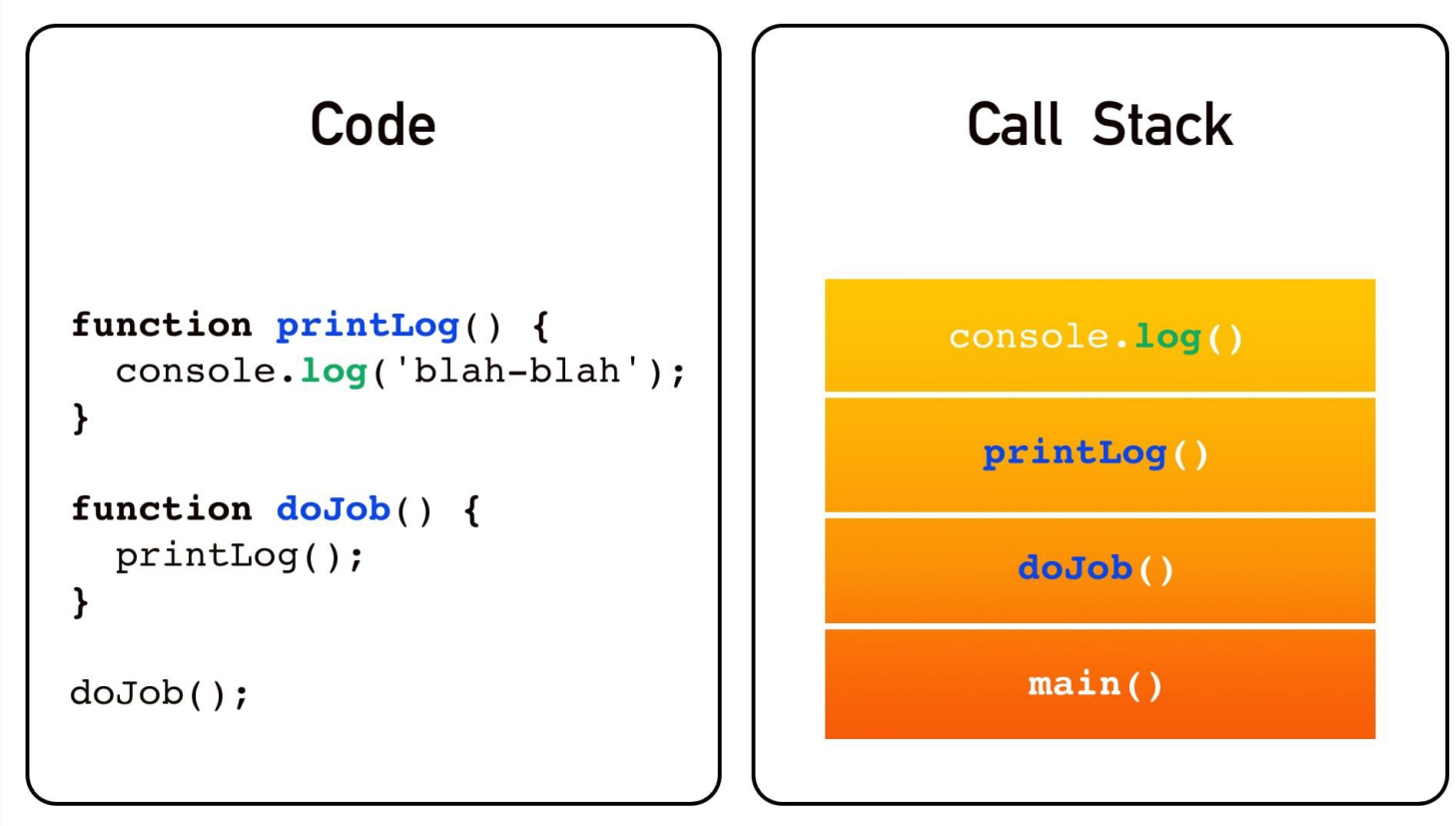
Fonte: modulecounts.com

Javascript Assíncrono 💻



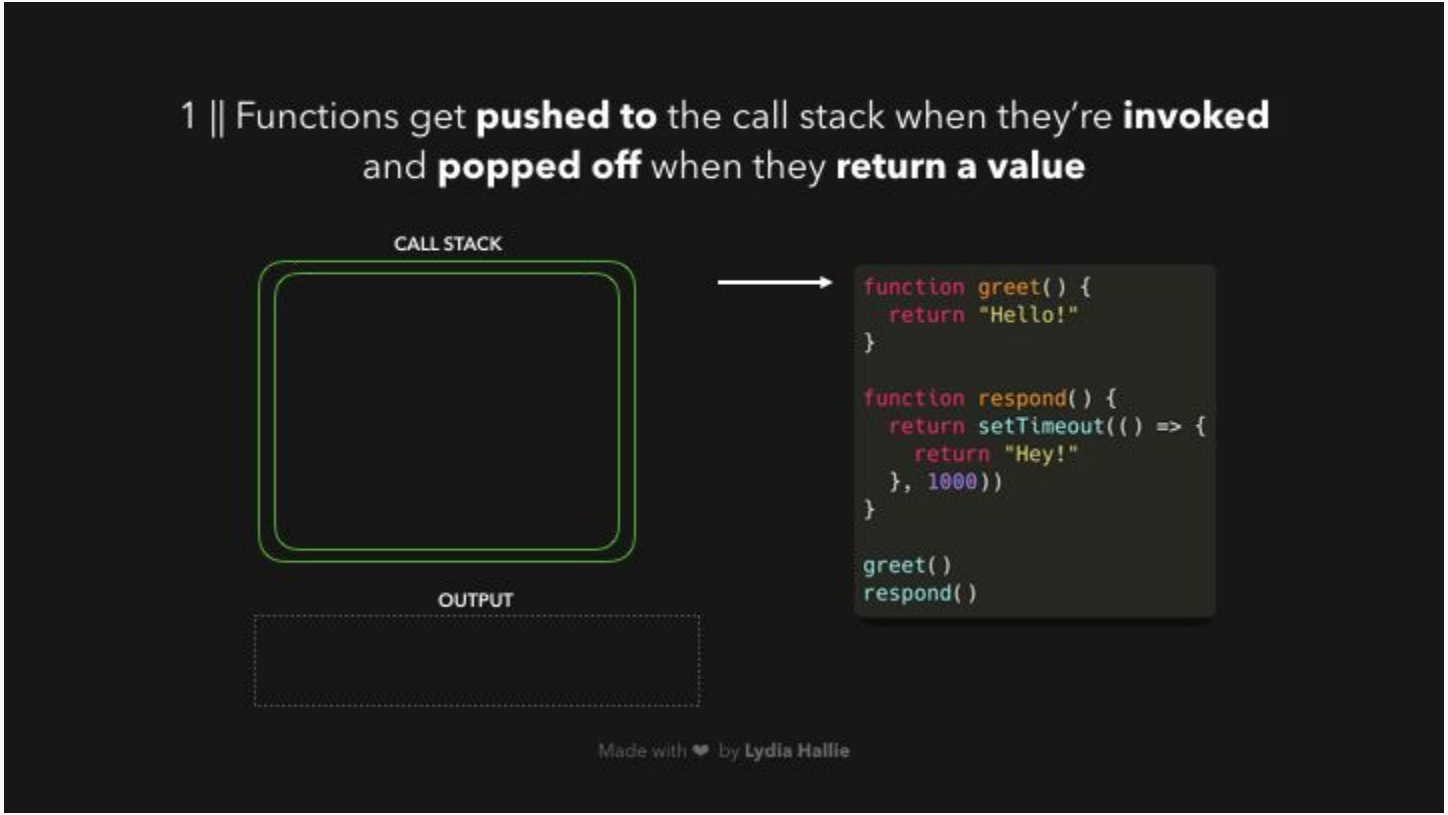
Console 

Call Stack

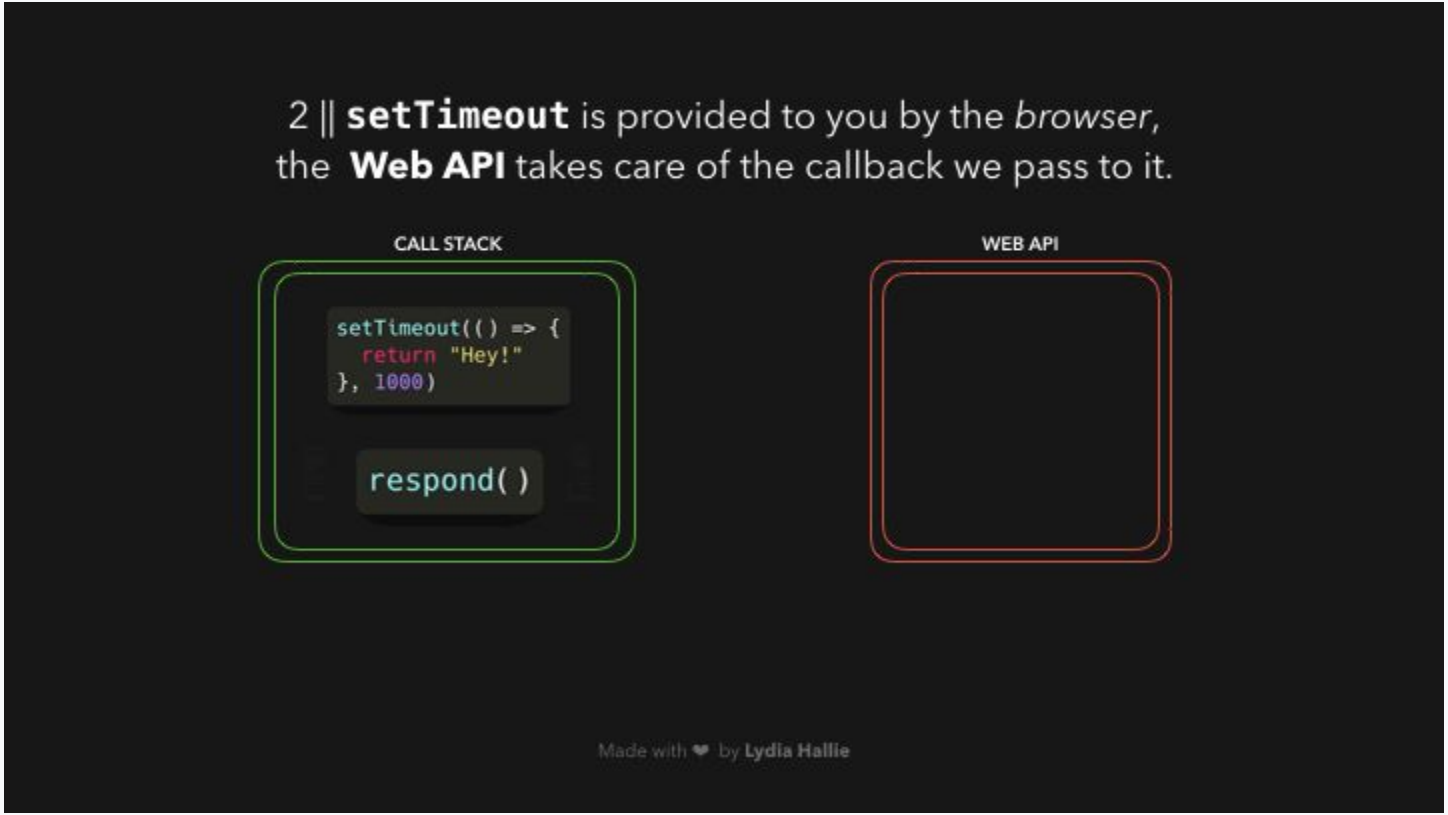

Code
function printLog() {
console.log('blah-blah');
}
function doJob() {
printLog();
}
doJob();
Call Stack
console.log()
printLog()
doJob()
main()

Input 

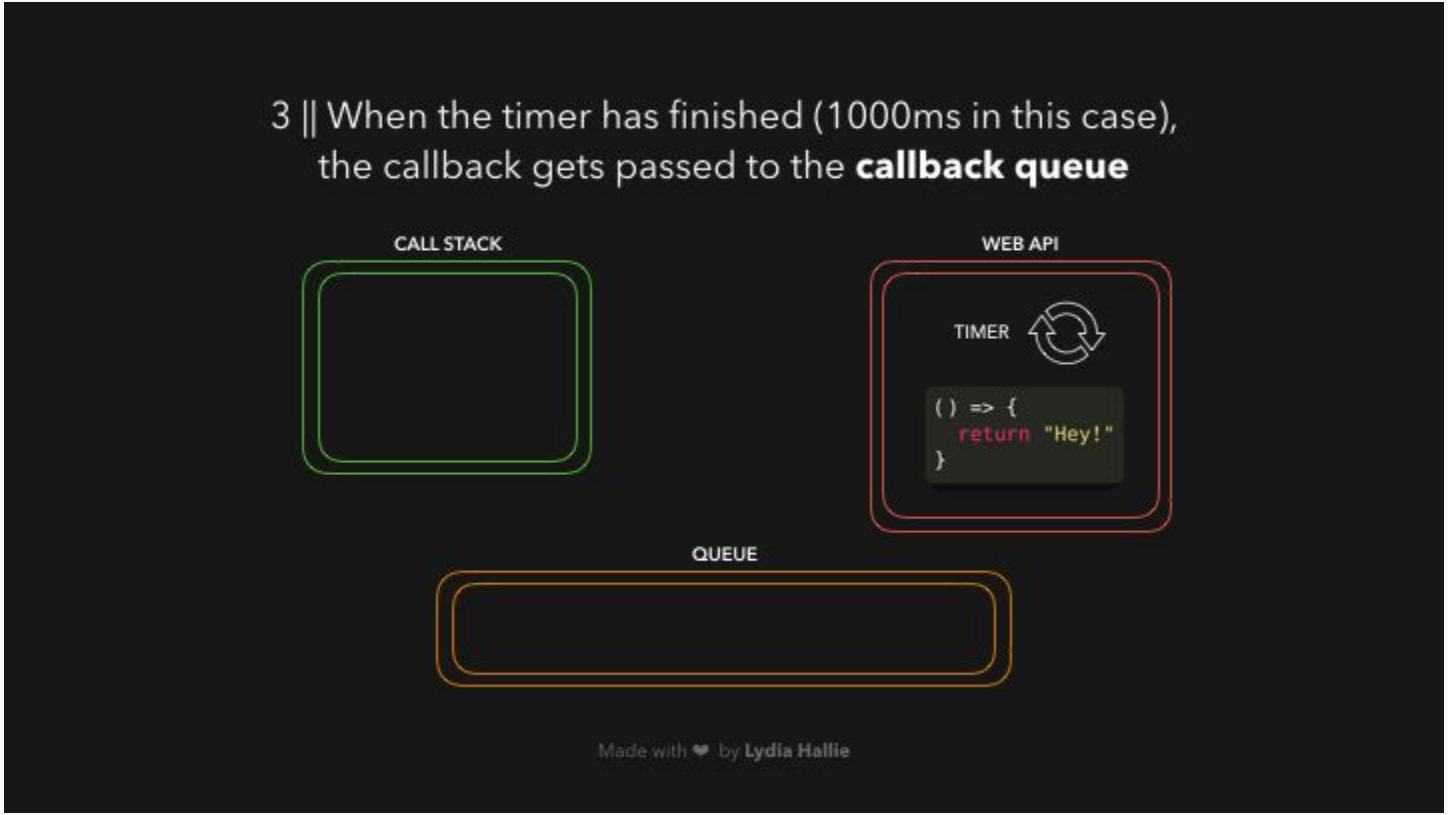
Event Loop 💻


1 || Functions get **pushed to** the call stack when they're **invoked** and **popped off** when they **return a value**
CALL STACK
OUTPUT
function greet() {
return "Hello!"
}
function respond() {
return setTimeout(() => {
return "Hey!"
}, 1000))
}
greet()
respond()
Made with ❤ by Lydia Hallie

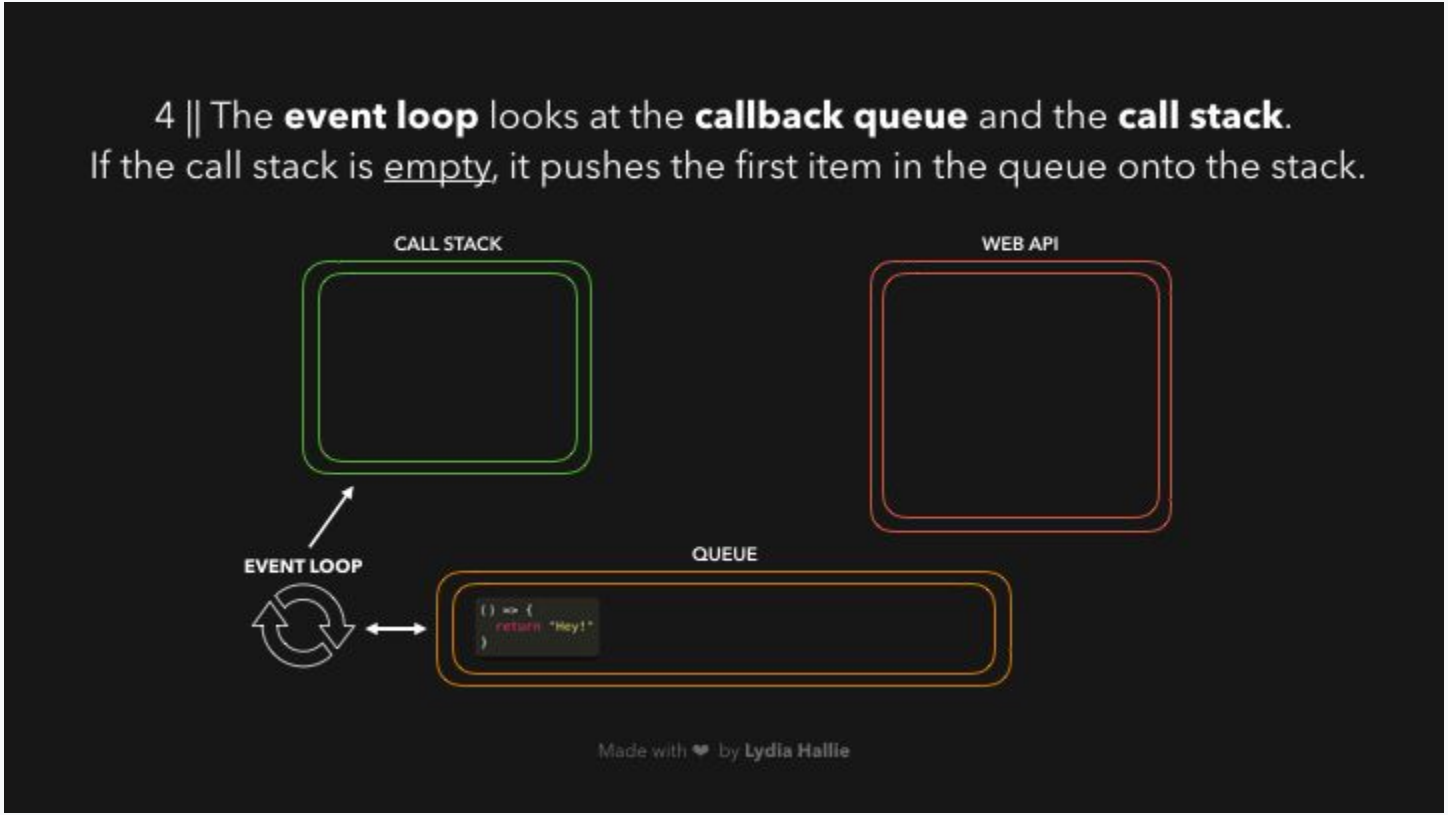
Event Loop 💻


2 || setTimeout is provided to you by the browser,
the Web API takes care of the callback we pass to it.
CALL STACK
setTimeout(() => {
return "Hey!"
}, 1000)
respond()
WEB API
Made with ❤ by Lydia Hallie

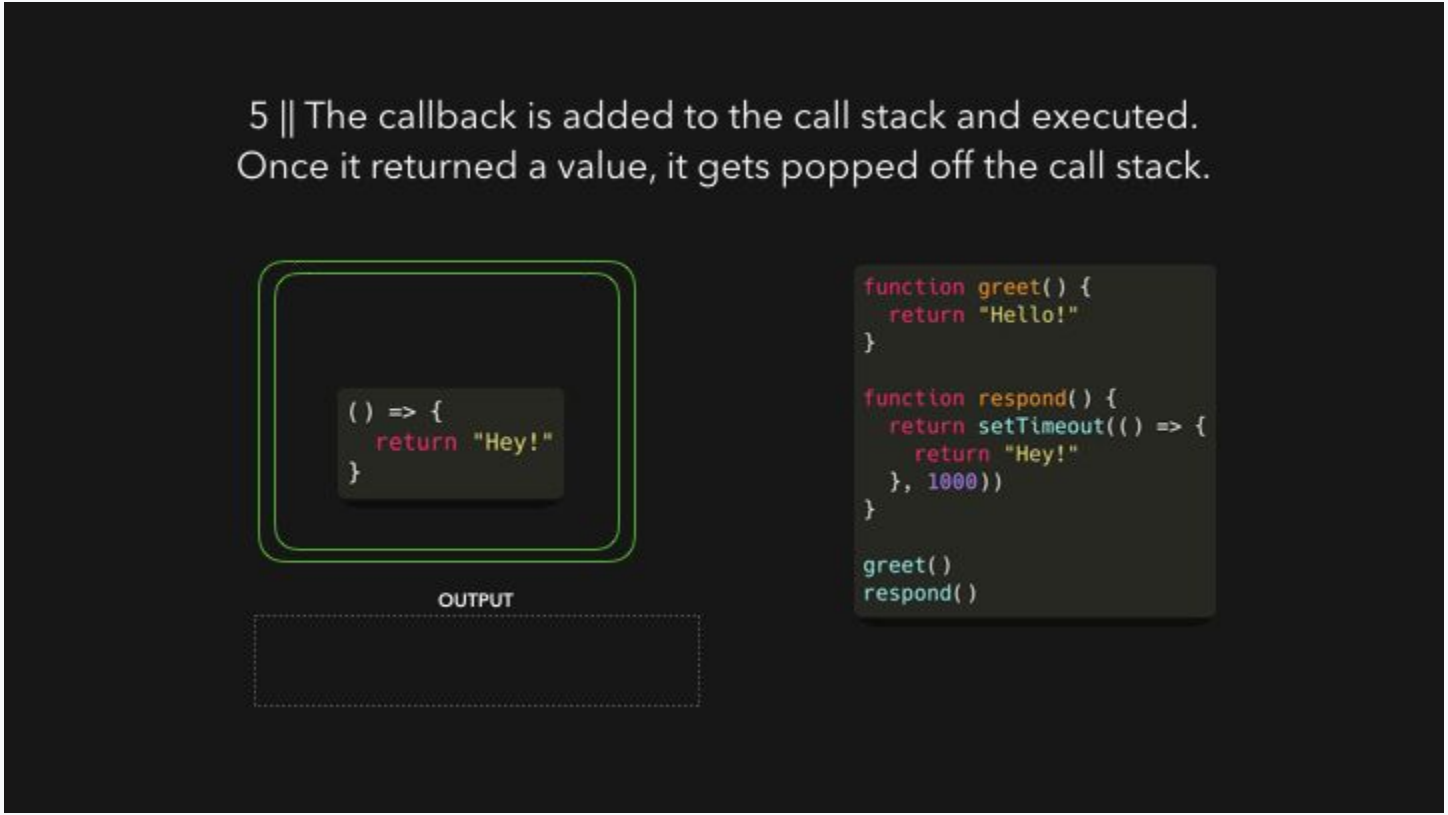
Event Loop 💻


3 || When the timer has finished (1000ms in this case), the callback gets passed to the **callback queue**
CALL STACK
WEB API
TIMER
() => {
return "Hey!"
}
QUEUE
Made with ❤ by Lydia Hallie

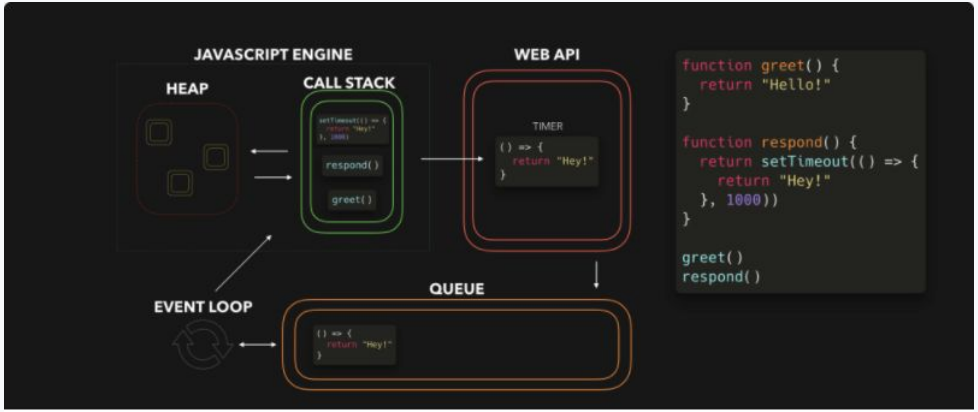
Event Loop 💻


4 || The event loop looks at the callback queue and the call stack.
If the call stack is empty, it pushes the first item in the queue onto the stack.
CALL STACK
WEB API
EVENT LOOP
QUEUE
() => {
return "Hey!"
}
Made with ❤ by Lydia Hallie

Event Loop 💻


5 || The callback is added to the call stack and executed.
Once it returned a value, it gets popped off the call stack.
() => {
return "Hey!"
}
OUTPUT
function greet() {
return "Hello!"
}
function respond() {
return setTimeout(() => {
return "Hey!"
}, 1000))
}
greet()
respond()

Event Loop 💻



Lydia Hallie
Posted on 20 de nov. de 2019 • Updated on 16 de jan. de 2020

✨♻️ JavaScript Visualized: Event Loop

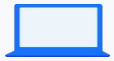
#javascript

Fonte: https://dev.to/lydiahallie/javascript-visualized-event-loop-3dif

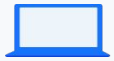
awari

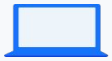
Unidade 4
Node. JS: Core Modules

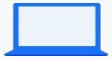
Visão geral sobre core modules 💻

Módulo http 

Módulo fs



Módulo url 

Adicionando funcionalidades 

awari

Node.js com Express

Transformando nosso backend em API

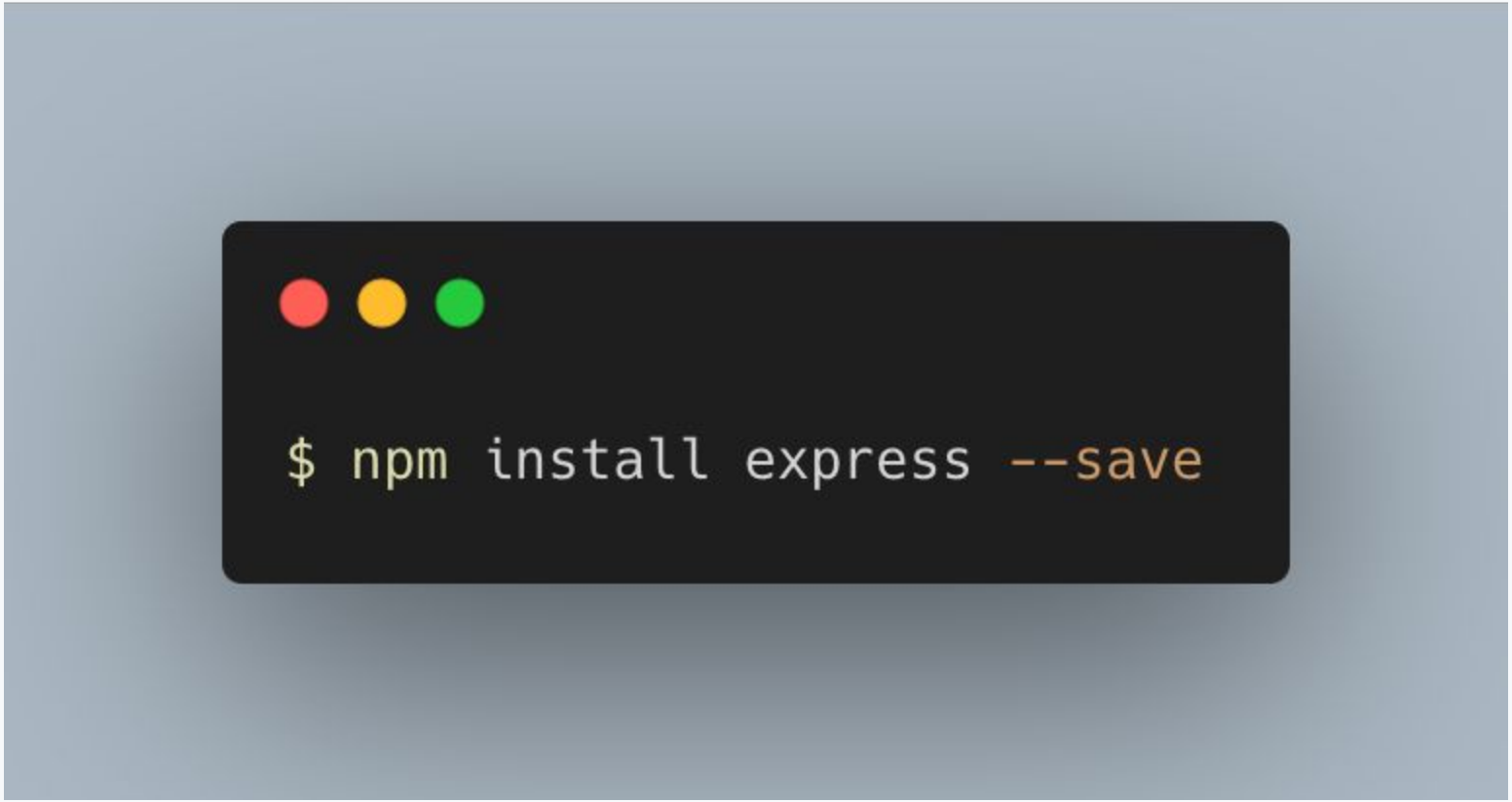
O que é Express? 

O **Express** é um framework para o **Node.js** desenvolvido para facilitar a criação de **API**'s. Ele traz uma série de recursos e abstrações que facilitam a vida na hora de construir seus códigos, como tratar as requisições, definição de regras de negócio, etc.

O **framework** foi construído pensado para o padrão de construção de **API**'s chamado **REST**, foi visto no capítulo anterior. Existem outros frameworks semelhantes no mercado, mas o Express é o mais utilizado pela ampla maioria, e os principais motivos são:

- Ele foi lançado no final de 2010. Ou seja, é um framework maduro e com grande adoção da comunidade
- Ele é um *"Un-Opinionated framework"* (framework não opinativo). Ou seja, ele não impõe um padrão de desenvolvimento na hora de escrever sua aplicação

Como começar com o Express? 💻



Como começar com o Express? 💻

Como começar com o Express? 💻

index.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Aplicação ouvindo na porta ${port}`)
})
```

Como começar com o Express? 💻

```
const express = require('express')
const app = express()

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.post('/', (req, res) => {
  res.send('Got a POST request')
})

app.put('/user', (req, res) => {
  res.send('Got a PUT request at /user')
})

app.delete('/user', (req, res) => {
  res.send('Got a DELETE request at /user')
})
```

Documentação 💻

http://expressjs.com/pt-br/

awari

Typescript

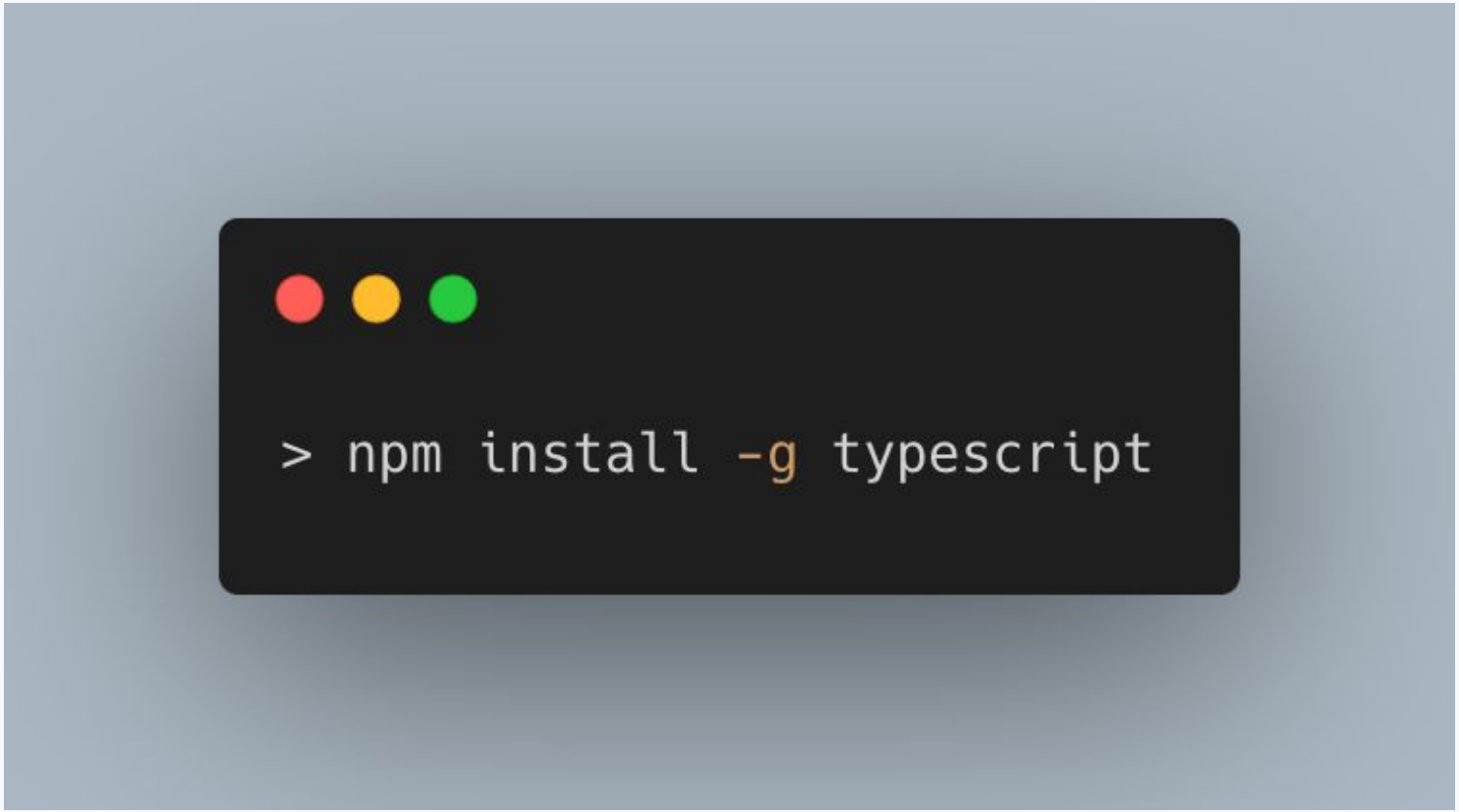
Mais poder ao Node e ao Javascript

O que é Typescript? 💻

O **TypeScript** mantém uma relação incomum com o JavaScript. Ou seja, oferece todos os recursos do JavaScript e uma camada adicional sobre eles: o sistema de tipagem.

Typescript é uma linguagem de código aberto desenvolvida pela **Microsoft** em 2012**,** e foi construída em cima do Javascript. Então esse *“superset”* foi criado para adicionar recursos de **tipagem estáticas** e **orientação a objetos** à linguagem original.

O que é Typescript? 💻



O que é Typescript? 💻

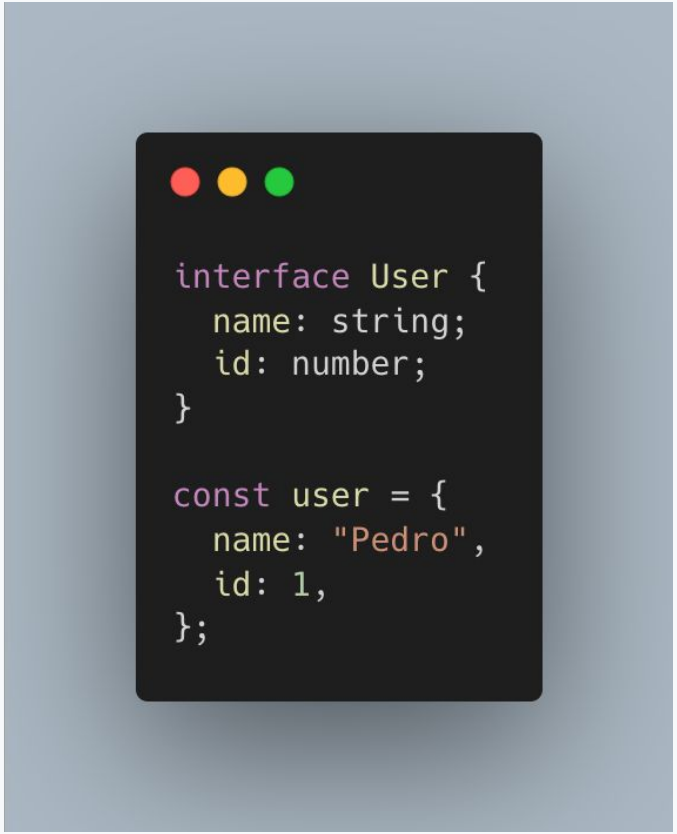
O que é Typescript? 💻

O que é Typescript? 💻

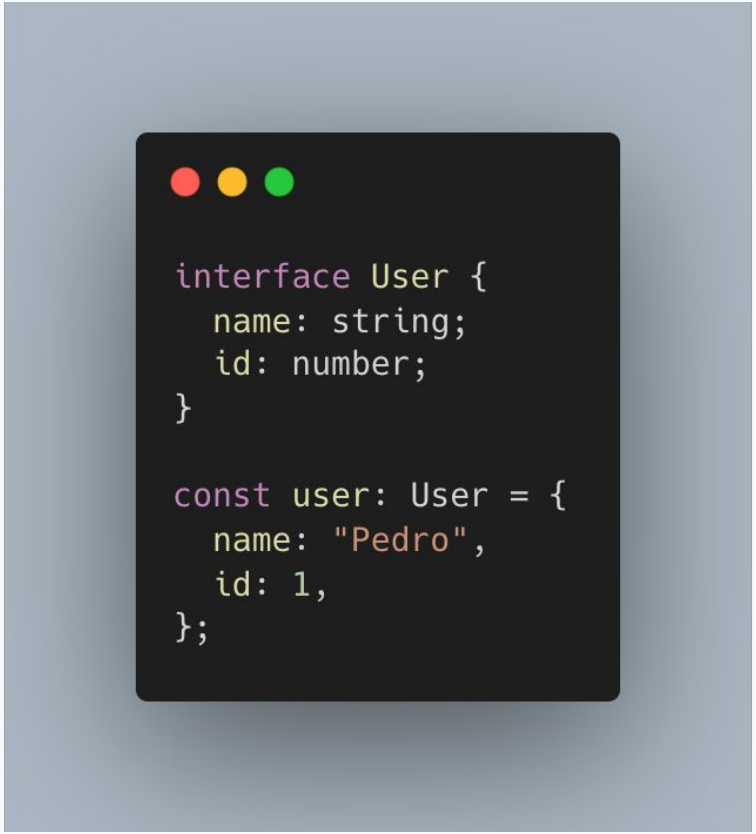
```typescript
function greet(person: string) {
  return "Hello, " + person;
}

let user = [0, 1, 2];

const hello = greet(user);

// Argument of type 'number[]' is not assignable to parameter of type 'string'.
```

O que é Typescript? 💻

O que é Typescript? 💻



O que é Typescript? 💻



O que é Typescript? 💻

```typescript
interface User {
  name: string;
  id: number;
}

class UserAccount {
  name: string;
  id: number;

  constructor(name: string, id: number) {
    this.name = name;
    this.id = id;
  }
}

const user: User = new UserAccount("Pedro", 1);
```