# DICAS & TRUQUES

**Dicas & truques** é uma iniciativa **InnoDev** com o foco na divulgação e promoção do Talento Angolano e fundamentalmente a partilha do conhecimento com estudantes, profissionais e aos amantes da leitura.


9 DICAS
FRONT
END

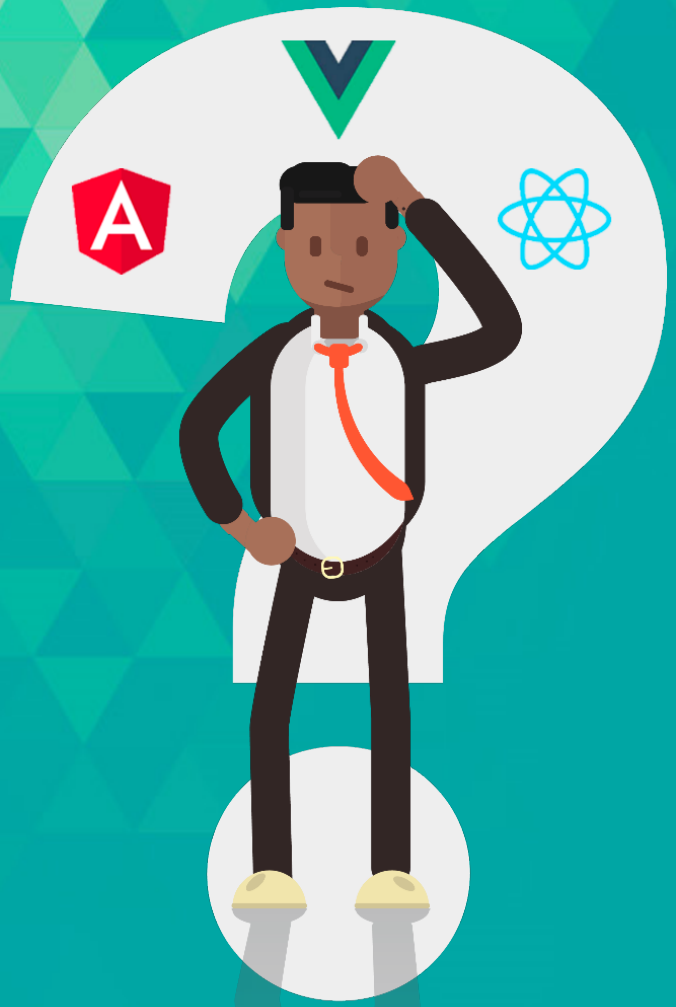
# PARTILHADA POR:


**EDILSON COELHO**

CONSULTOR SÊNIOR IT

**CARLOS SPRANGER**

CONSULTOR SÊNIOR IT




Ainda existem muitas discussões sobre qual **framework front-end** adotar entre os mais populares da actualidade nomeadamente: **Angular**, **Vue** e **React**. Não queremos influenciar ninguém, mas **nossa visão** é que todos eles são muito bons para **front-end**, é uma questão de amor a primeira vista. Já realizamos alguns Casos de Uso “cases” e não notamos diferenças, é claro que existem particularidades.


02

Você sabia que **JavaScript é a linguagem mais popular da actualidade**? Disponibilize Tempo para aprender e adotar em seus projectos. Garantimos que a curva de aprendizado é bem menor em relação as linguagens tradicionais como **Java**, **C#**, **Ruby**, etc. Porque eu adotaria **js**? Pergunta difícil resposta fácil: **js** é interpretado, atualmente pode ser utilizado no cliente e no servidor “Graças ao **Node.js**”, sintaxe simples e quando se fala de ganhos de performance ele é o cara. “JavaScript é uma formiga que pode entrar em qualquer residência e adptar-se ao meio”.

03

A comunidade **js** “JavaScript” tem levado a linguagem para caminhos que ninguém esperava, vamos recuar uns 13 anos e a imagem que nos vem do **js** são **animações** bem básicas, *validações de formulários*, o famoso **alert(‘Hello js’)**, uns nem achavam interessante nem piada na **sintaxe**, outros nem conseguiam perceber a real utilidade.


Para os devs que muito reclamavam da sintaxe JavaScript e eu fui um deles, mas sempre apaixonado e seguidor activo, agora já podem sentir-se realmente em casa porque a partir do ES6 ou ECMAScript 6 as coisas começam a ficar mais interessantes para comunidade **js** e principalmente os devs que veem de linguagens orientadas a objectos como Java, C#, etc. Mas ES6 o que é? É uma nova especificação do **js** que nos trás conceitos e recursos muito interessantes como *setters* e *getters*, *class*, *herança*, novos tipos: Set, Map, novas kewords: *let* e *const*, *async*/*await*, etc.

```
class Pessoa {
    constructor(nome) {
        this._nome = nome;
    }
    set nome(nome) {
        this._nome = nome;
    }
    get nome() {
        return this._nome;
    }
}
```


```
class Funcionario extends Pessoa {
    constructor(nome, salario) {
        super(nome);
        this._salario = salario;
    }
    set salario(salario) {
        this._salario = salario;
    }
    get salario() {
        return this._salario;
    }
}
```


Fala-se bastante em performance nas aplicações, e é um tema que temos acompanhado bastante porque afecta directamente o consumidor ou utilizador final na sua experiência com as aplicações. Mais como ter ganhos de performance? Reflita sobre pontos enumerados abaixo:


Você tem adoptado **SPA "Single Page Application"** que é uma aplicação com apenas uma única pagina, que consiste em trazer a mesma experiência das aplicações desktop? Comece a adoptar;

Já pensou em separar seu **front-end (UI) do seu backend**, ter total desacoplamento;

Já pensou adoptar para o front-end frameworks como **Angular, Vue e React?** Pense nisso;

Sua equipa de projeto tem conhecimento de **JavaScript**? Tempo para aprender;

Você tem minificado e concatenado seus arquivos **css**, **js**? Comece a fazer e verá que no primeiro carregamento da aplicação será carregado apenas dois arquivos *main.css* e *main.js*;

05

Fala-se bastante em performance nas aplicações, e é um tema que temos acompanhado bastante porque afecta directamente o consumidor ou utilizador final na sua experiência com as aplicações. Mais como ter ganhos de performance? Reflita sobre pontos enumerados abaixo:

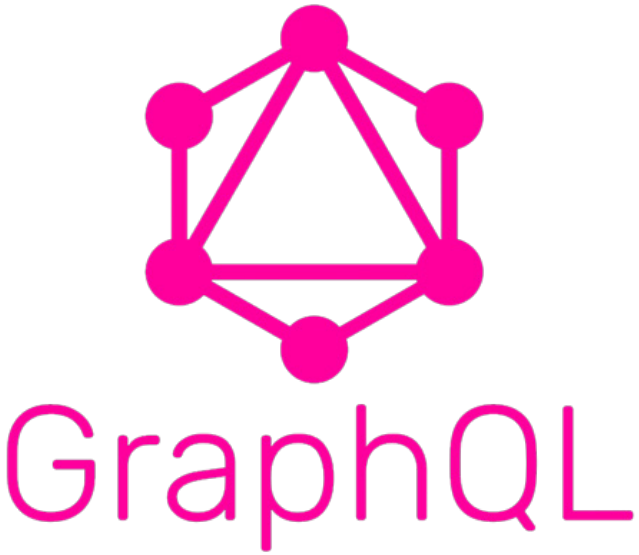
Seus *scripts* são carregados sobre demanda ou a cada *renderização* todos os scripts são entregues mesmo que o utilizador não tenha acesso? Pense nisso **Lazyload**;

Você separa o teu front-end do backend em *web servers* ou servidores distintivos? Comece já a separar as responsabilidades e dividir a carga.

Você já pensou em realizar mais acções no cliente e evitar ligações ao servidor? Medite sobre isso.

Você separa sua aplicação em módulos? Comece a separar e implementa **Lazyload**, carregue seus scripts sobre demanda.

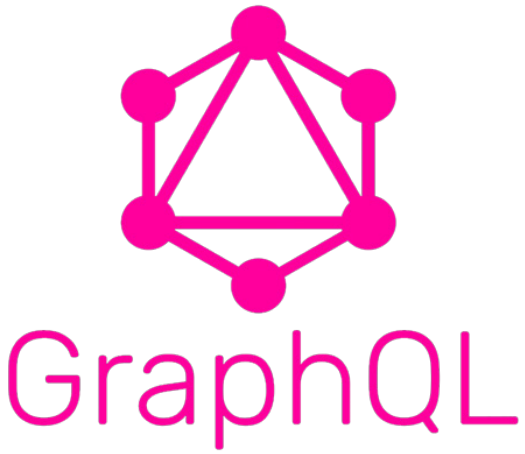

Para aqueles que têm acompanhado o **Graphql do Facebook**, mas o que é **Graphql**? É uma linguagem de consulta a API. Está a ser considerada tendência para 2019. Quem não conhece recomendo a investigar. A quem diz que veio substituir a *arquitectura Rest*, mas nós achamos que é um complemento porque trás apenas ingredientes novos.




## Graphql porquê adoptar:

- **Flexibilidade** que os clientes têm em realizar suas querys no processo de consulta de informação numa fonte de dados;
- Redução da quantidade de *requisições* no servidor;
- **Single endpoint**, existe apenas um único endpoint para atender todos os pedidos;


Para aqueles que têm acompanhado o **Graphql do Facebook**, mas o que é **Graphql**? É uma linguagem de consulta a API. Está a ser considerada tendência para 2019. Quem não conhece recomendo a investigar. A quem diz que veio substituir a *arquitectura Rest*, mas nós achamos que é um complemento porque trás apenas ingredientes novos.




## Graphql porquê adoptar:

- Delimitação dos dados de acordo a plataforma (*web*, *mobile*, *etc*.), isto é, como são enviadas **query** podemos delimitar os campos a serem entregues como resposta do servidor *Graphql*;
- Existem várias implementações: *Java*, *C#*, *Ruby*, *PHP*, *Go*, *Scala*, *Erland*, *Python*, *JavaScript*, *etc*;
- A estrutura da resposta do servidor para uma *query* é praticamente a mesma estrutura da própria query, evitando assim o comprometimento de performance ao processar dados desnecessários.


Como você trata suas operações assíncronas? Já sei **then** e **catch**!!! Deixe de usar ou restrinja o seu uso e procure adoptar o async e await novo recurso do *JavaScript ECMAScript 6*. Só existem vantagens, uma delas é esperar ou aguardar pelo resultado da operação e tratar da mesma forma que as linguagens tradicionais.

| Sem async/await | Com async/await |
| --- | --- |

```
var promise = new Promise(function (resolve, reject) {
  resolve('success');
});

promise.then(function (resultado) {
  console.log(resultado);
});
```

```
var promise = new Promise(function (resolve, reject) {
  resolve('success');
});

async function aguardaResultado() {
  const resultado = await promise;
  console.log(resultado);
}
```


# 08

Já ouviu falar de **Arrow functions**? A nova maneira ou forma de trabalhar com funções em JavaScript, isso a partir da especificação *ES6* ou *ECMAScript 6*.

| Forma antiga | Forma nova |
| --- | --- |
| `var soma = function(a, b) {`<br>`  return a + b;`<br>`}` | `const soma = (a, b) => a + b;` |


Para declaração de variáveis acredito que tem usado o “*var*”, não use mais, passe a usar “***let***”, sabe por quê? O *let* tem escopo de bloco e o *var* tem apenas escopo de função.

| Declaração com Var | Declaração sem Var |
| --- | --- |

```
for(var i = 0; i< 4; i++)
{
  console.log(i); // 0, 1, 2, 3
}
console.log(i) // resultado fora do escopo 4
```

```
for(let i = 0; i< 4; i++)
{
  console.log(i); // 0, 1, 2, 3
}
console.log(i) // ReferenceError: i is not defined
```

“Se puderes, ajuda os outros; se não o puderes fazer, ao menos não lhes faças mal”.

DALAI LAMA