RAZOES PARA REASONML

Olar!

Meu nome é Ana Bastos

Sou engenheira de software e cientista da computação.

anabastos
@naluhh
@anapbastos

Talvez você esteja aqui...

1. Talvez você esteja aqui porque já ouviu falar de linguagens que transpilam pra JS(Typescript, ELM, Clojurescript) mas não entendeu nada do que o Reason faz ou como aplicar.
2. Porque vc ouviu falar sobre ReasonReact.
3. Por saber que é uma tecnologia facebook assim como GraphQL, Flow, Jest etc e portanto tem muito hype.

Talvez você esteja aqui...

A ideia hoje é tentar tornar essa curva de mistério mais tênue

O que esperar

- O que é
- Como funciona
- Razões

MOTIVOS

1.

“JAVASCRIPT FADIGUE”

Javascript Fatigue

Eric Clemmons Follow
Dec 27, 2015 · 4 min read

A few days ago, I met up with a friend & peer over coffee.

> *Saul: "How's it going?"*
> *Me: "Fatigued."*
> *Saul: "Family?"*
> *Me: "No, Javascript."*

More accurately, I meant *React* and the Javascript ecosystem that comes with it.

COMO E COMEÇAR UM PROJETO JAVASCRIPT DO 0?

Independente de node, browser, mobile, electron.....

- Criar Package.json
- Configurar ESLINT
- Usar TS ou Flow
- Utilizar libs de imutabilidade
- Configurar testes
- Instalar e configurar o babel
- Escolher uma lib utilitária (Lodash, Underscore, Ramda)
- Muitas funções utilitárias


flow
REQUIRE.JS
BABEL
mocha
IMMUTABLE
browserify
Modernizr

1. Curva de aprendizagem

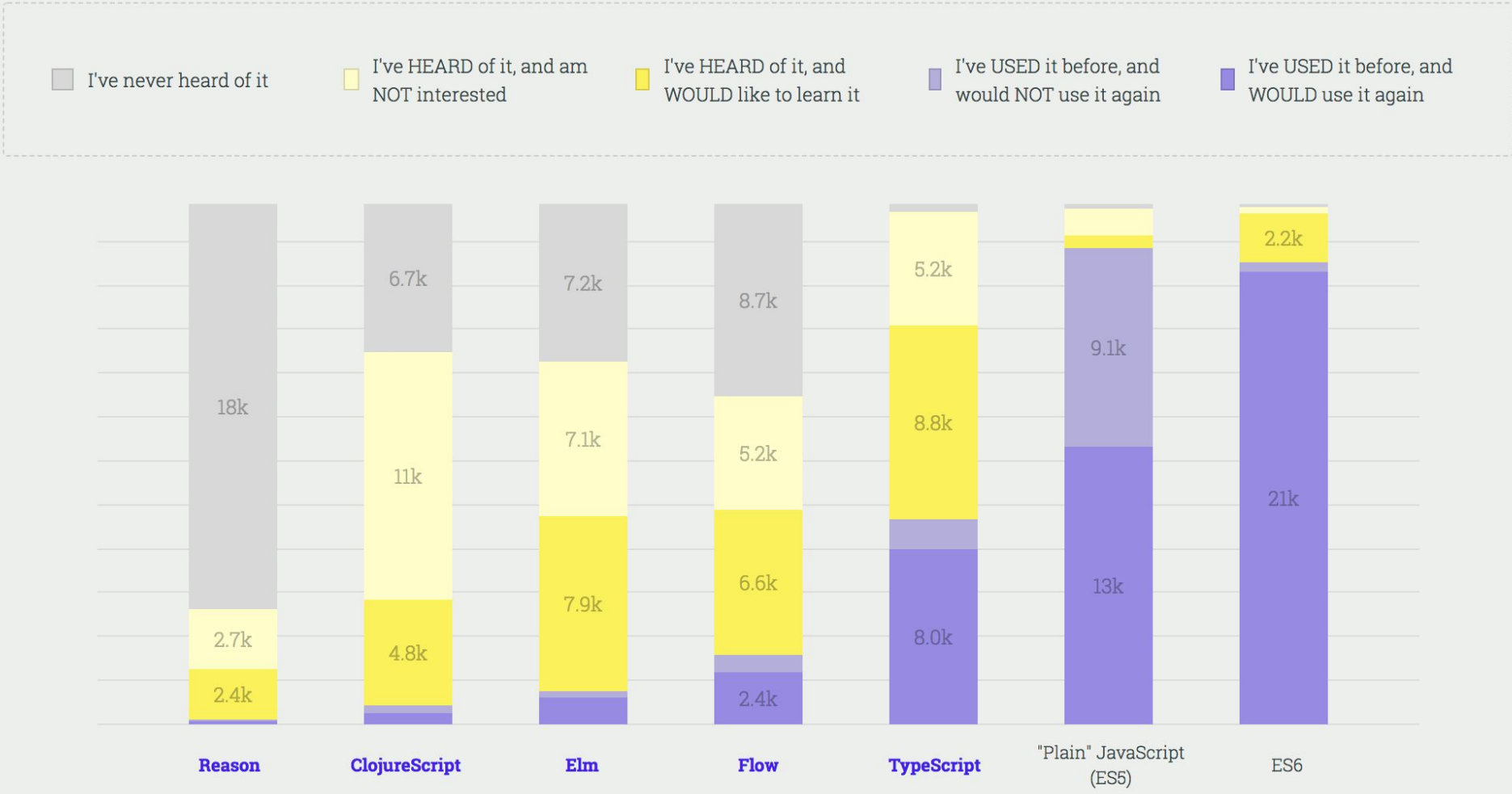
2. Ferramental suprindo faltas da propria linguagem

A complexidade de projetos grandes JS acaba sendo muito mais lidar com tooling do que efetivamente codar e implementar novas features.

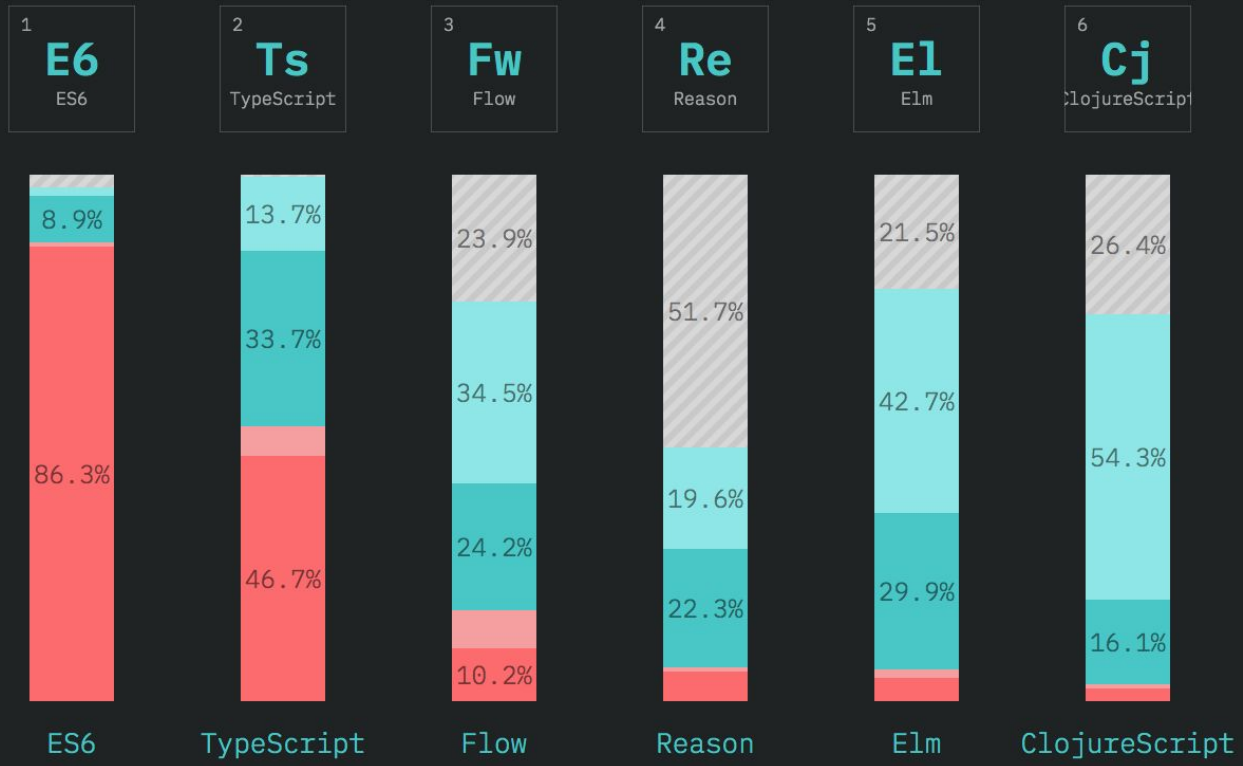
Não precisamos de mais tooling
Precisamos de uma linguagem melhor

Linguagens que compilam pra JS

State of JS 2018


I've never heard of it
I've HEARD of it, and am NOT interested
I've HEARD of it, and WOULD like to learn it
I've USED it before, and would NOT use it again
I've USED it before, and WOULD use it again
18k
2.7k
2.4k
6.7k
11k
4.8k
7.2k
7.1k
7.9k
8.7k
5.2k
6.6k
2.4k
5.2k
8.8k
8.0k
9.1k
13k
2.2k
21k
Reason
ClojureScript
Elm
Flow
TypeScript
"Plain" JavaScript (ES5)
ES6

State of JS 2019


1
E6
ES6
2
Ts
TypeScript
3
Fw
Flow
4
Re
Reason
5
El
Elm
6
Cj
ClojureScript
Percents
Counts
8.9%
86.3%
13.7%
33.7%
46.7%
23.9%
34.5%
24.2%
10.2%
51.7%
19.6%
22.3%
21.5%
42.7%
29.9%
26.4%
54.3%
16.1%
Never heard of it
Heard of it, not interested
Heard of it, would like to learn
Used it, would not use again
Used it, would use again
ES6
TypeScript
Flow
Reason
Elm
ClojureScript

O que é o ReasonML?

“It’s a new syntax and toolchain powered by the battle-tested language.”

O que é ser uma nova sintaxe?

Ao invés de criar uma linguagem completamente nova, os criadores do ReasonML o construíram em cima da linguagem Ocaml.

O que é OCaml?

- Linguagem functional da familia ML(Sml ou OCaml)
- Linguagem High Lvl com uma fundação fortíssima de décadas de pesquisa de sistemas de tipos e engenharia de compiladores topissimos.
- Conhecido por sua mantenabilidade
- Usado em ferramentas internas do Facebook (A linguagem Hack e o Flow do Javascript tem várias ferramentas internas em OCaml)
- A biblioteca React a princípio foi feita em SML

O que é ReasonML?

É o meio termo perfeito entre um ecossistema bom e uma sintaxe familiar para a adoção de programadores Javascript.

O que é ReasonML?

- Sintaxe de OCaml inspirada em JS e Rust(Ou qualquer linguagem similar a C)
- Mantida pelo Facebook
- Feita pelo mesmo criador do React(Jordan Walke)
- Se tornou open source em 2016
- Com intenção de fazer interop com Javascript
- Suporta JSX e apresentou recentemente o ReasonReact

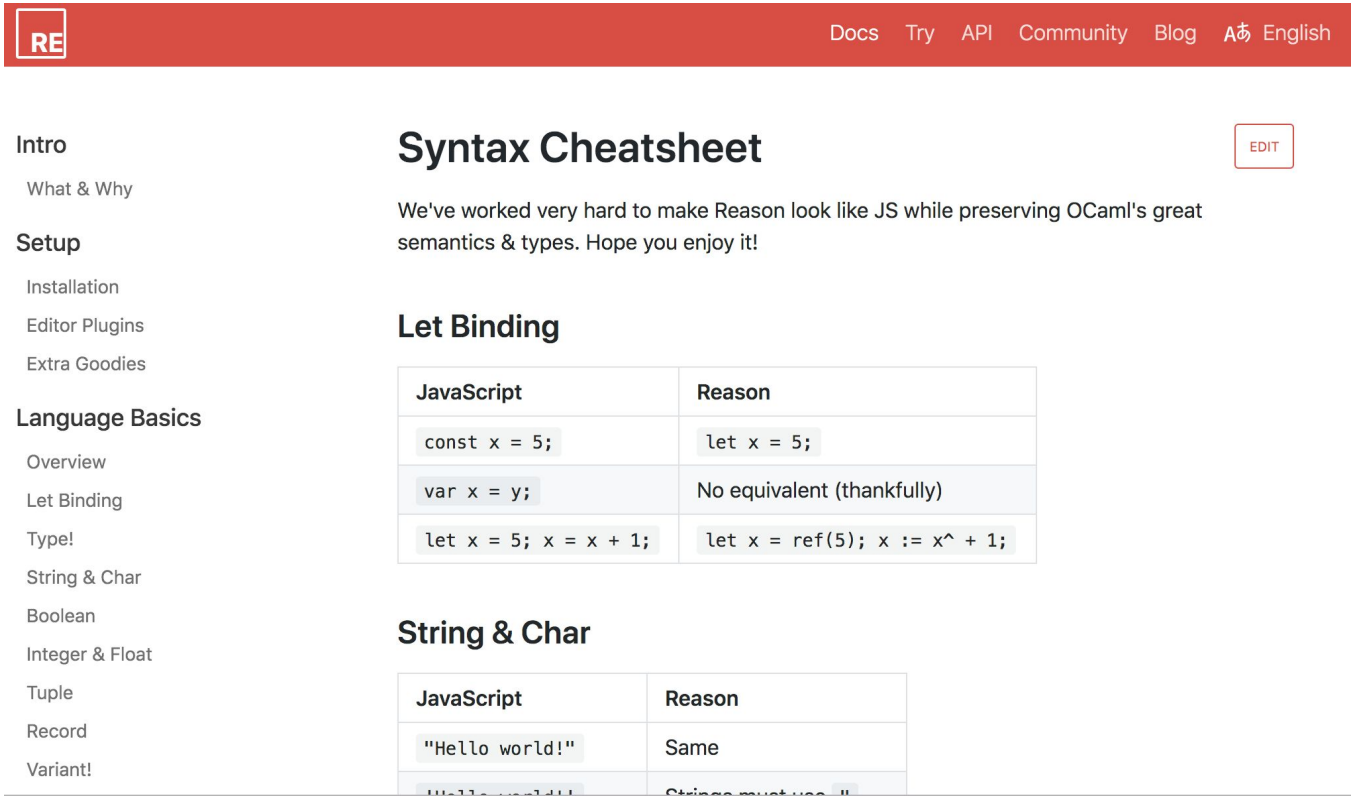
ReasonML anda entrando no “hype” por basicamente ser um JS com coisas novas e legais, e sem coisas como classes, e com melhoras no seu ecossistema.

Tem certeza que aprender uma linguagem nova pode ser melhor do que o overhead de tooling?

2.

JAVASCRIPT FRIENDLY + EASY INTEROP

O time do Reason tomou bastante cuidado pra que a linguagem tenha uma sintaxe bem similar.



https://reasonml.github.io/docs/en/syntax-cheatsheet

INTEROP

```
Js.log("React Meetup");
Js.Promise(asyncStuff);
Js.Number.Date.getMinutes(date);
```

INTEROP

```
[%bs.raw {| console.log('TDCBH') |}];
```

HELLO WORLD

```
let msg = ”hello world!”;
Js.log(msg); // hello world!
```

FUNCTION

math.re

```
let add = (a, b) => a + b;
```

Sem imports

Sem export defaults

Módulos são referenciados pelo próprio nome do arquivo.

```
Math.add(1, 2)
// Compilador já vai atrás de um arquivo
math.re ou math.ml
```

math.re

```
module Operations = {
    module Basic = {
        let add = (a, b) => a + b;
    }
}
```

```
Math.Operations.Basic.add(1, 2)
```

test1.js

```
import { add } from “test2.js”;
```

test2.re

```
[@bs.module] external add: int => int = “./test1”;
```

3.

SISTEMA DE TIPOS ESTÁTICO E FORTE + INFERENCIA DE TIPOS

Sistema de tipos, inferencia, declaração de tipos, option

O que é um type system?

É um mecanismo de definir, detectar estados ilegais definindo e aplicando limitações.

“Ei, você não pode misturar leite com manga”

```
> {} + ""
0
> "" + {}
"[object Object]"
```

Javascript consistency

```
> '1' + '1'
"11"
> '1' + '1' - 1
10
```

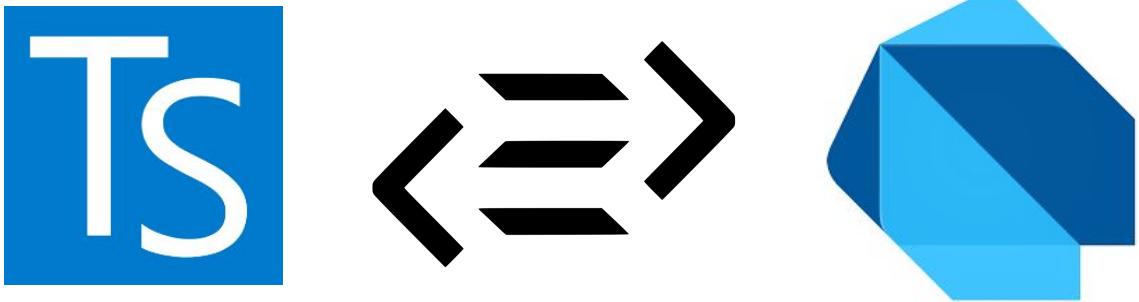
```
> '5' - 3
2       // weak typing + implicit conversions * headaches
> '5' + 3
'53'    // Because we all love consistency
> '5' - '4'
1       // string - string * integer. What?
> '5' + + '5'
'55'
> 'foo' + + 'foo'
'fooNaN' // Marvelous.
> '5' + - '2'
'5-2'
> '5' + - + - - + - - + + - + - + - + - - - '-2'
'52'    // Apparently it's ok

> var x * 3;
> '5' + x - x
50
> '5' - x + x
5     // Because fuck math
```

http://bit.ly/wtfjavascript


ECMA4

Bibliotecas que programadores JS usam p/ suprir isso:

Linguagens que programadores JS usam p/ suprir isso:



Apenas adicionam capacidades da linguagem atual forçando a verificação de tipos.

JavaScript(Dinâmico)

```
const helloMessage = 'Hello world';
```

Em tempo de compilação: Só a declaração da variável é considerada sem nenhum tipo associado

Em runtime a variavel passa a ser uma string

ReasonML(Estático)

```
let helloWorld = "Hello world";
```

Checa os tipos toda vez que o programa é compilado recebendo um feedback se está tudo correto antes de rodar.

A tipagem estática também ajuda a detectar alguns tipos de erros. E muitas vezes ajuda a documentar como funciona o código (de forma que seja automaticamente verificada quanto à consistência).

JavaScript(Dinâmico)

```
const helloMessage = 'Hello world';

console.log(helloMessage + 2);
// Hello world2
```

Inferencia de tipos

```
let helloWorld = "Hello world";
Js.log(helloWorld + 2);
```

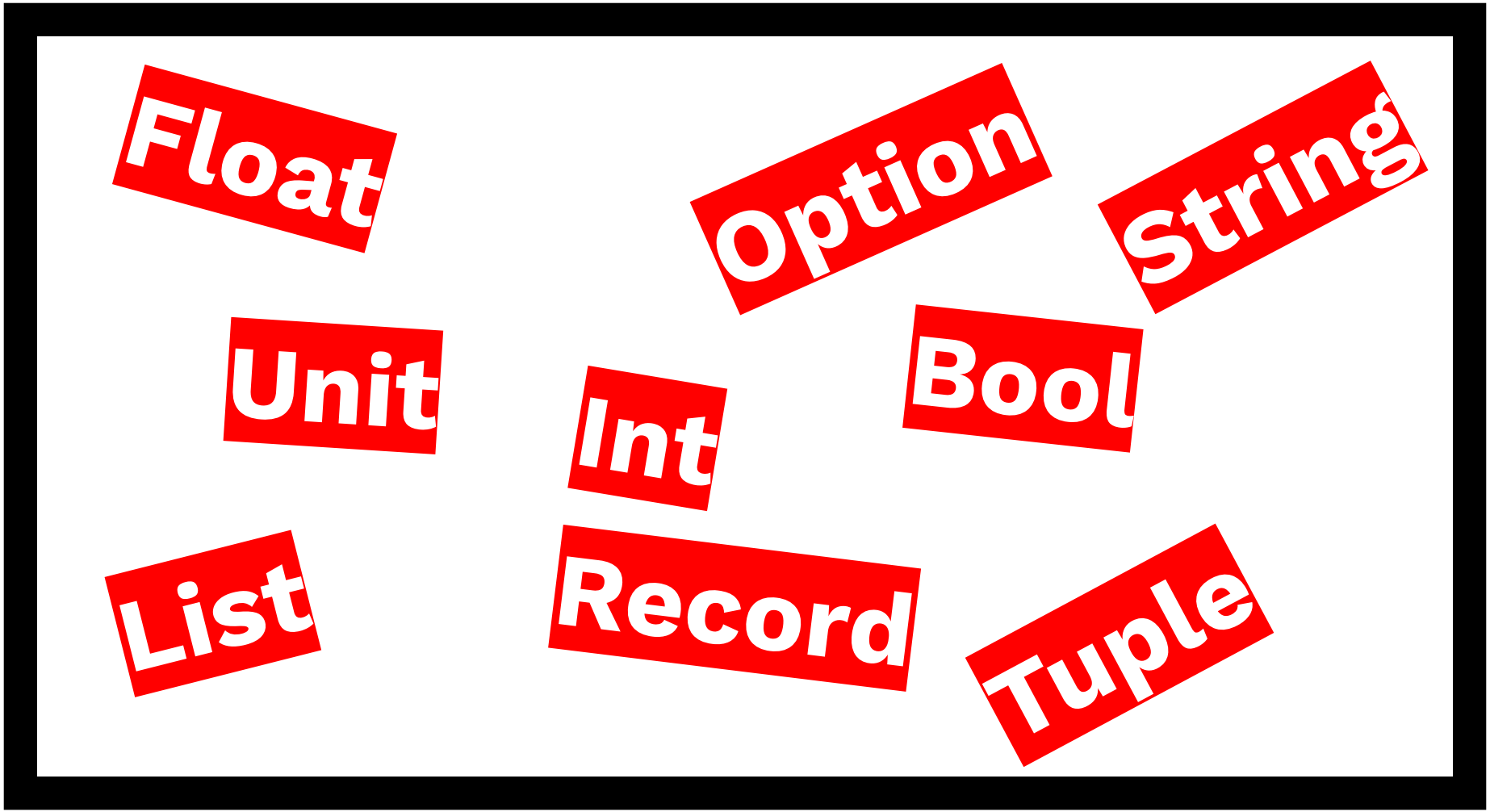
```
We've found a bug for you! OCaml preview 2:17-26 This has
type: string But somewhere wanted: int
```

Se quero tipos então porque não TypeScript?

```
/* REASON */

let diff = (a, b) => a - b;

/* JS */

const diff = (a, b) => a - b;

/* JS + FLOW / TYPESCRIPT */

const diff = (a, b) => a - b;
```

TypeScript foi feito para ser um superSet em Javascript propondo uma verificação de tipo em tempo de compilação enquanto Reason pra ser a extensão de uma linguagem funcional

Em Reason dados imutáveis estão na linguagem. Temos validadores em runtime e uma compilação bem mais rápida.


Float
Option
String
Unit
Bool
Int
List
Record
Tuple

PRIMITIVE	EXAMPLE
Strings	`"Hello"`
Characters	`'x'`
Integers	`23`, `-23`
Floats	`23.0`, `-23.0`
Integer Addition	`23 + 1`
Float Addition	`23.0 +. 1.0`
Integer Division/Multiplication	`2 / 23 * 1`
Float Division/Multiplication	`2.0 /. 23.0 *. 1.0`
Float Exponentiation	`2.0 ** 2.0`
String Concatenation	`"Hello " ++ "World"`
Comparison	`>`, `<`, `>=`, `=<`
Boolean operations	`!`, `&&`, `||`
Reference, Physical (deep) Equality	`===`, `==`
Immutable Lists	`[1, 2, 3]`
Immutable Prepend	`[item1, item2, ...theRest]`
Arrays	`[|1, 2, 3|]`
Records	`type player = {score: int}; {score: 100}`
Comments	`/* Comment here */`

Numeros

```
/* JS */
let a = 1; // number (64 bits)

/* REASON */
let a = 1; // integer(32 bits)
let a = 1.0; // float(64 bits)
```

Inferencia de tipos

```
let add1 = (a: int, b: int) => a + b;

let add2 = (a: float, b: float) => a +. b;
```

Inferencia de tipos

```
let add1 = (a, b) => a + b;
// Assume que “a” e “b” são inteiros

let add2 = (a, b) => a +. b;
// Assume que “a” e “b” são float
```

Inferencia de tipos

```
let add2 = (a, b) => a +. b;
// Assume que “a” e “b” são float

add2(1, 2)
Error: this expression has type int but an
expression was expected of type float
```

Declarar tipos é necessário com tipos mais complicados mas redundante com tipos simples.


String -> String -> String -> String
Types are a lie

TypeScript

```
interface Person {  name: string;  age: number;  sign: string;}


let printPerson = (p: Person):void => {
    console.log(p.name, p.age, p.sign);
}

printCar({ name: "Ana", age: 24, sign: "Gemini" });
```

Reason

```
type person = {  name: string,  age: int,   sign: string}

let printPerson = (p: person):unit => {
    print_endline(c.name ++ string_of_int(c.age) ++ c.sign);
}

printPerson({ name: "Ana", age: 24, sign: “Gemini” });
```

Nominal TypeChecking

Você tem a garantia que o dado que você está passando não está absurdamente nada a ver

Nominal TypeChecking

Temos um feedback instantâneo do Reason de algo errado que só seria descoberto em fase em testes.

Nominal TypeChecking

```
let ana: person = {
   name: "Ana",age: 25,sign: “Gemini”
}
```

Nominal TypeChecking

```
let ana = {
   name: "Ana",age: 25,sign: “Gemini”
}
// O tipo user é inferido pelos seus
items.
```

Nominal TypeChecking

Com esse sistema de tipos passivo tu passa a ter memória muscular na hora do desenvolvimento para evitar merdas

Variant

```
type animal = | Person(string) | Dog | Cat;
let euzinha = Person("Ana");
```

null

NaN

-Infinity

undefined

Não existem bugs com mensagens tipo “undefined is not a function”

Em TS temos “nulls” mas em linguagens funcionais não devem existir bugs por receber esses valores sem querer.

Option

“Maybe”

Lidar com a nullabilidade de forma pura.

Option

```
type option('a) = None | Some('a);
```

Option

```
let possiblyNull: option(string) = Some("Ana");
switch (possiblyNull) {
  | None => print_endline("Olá pessoa")
  | Some(message) => print_endline("Olá " ++ message)
};
```

Refatorar é mais fácil em uma linguagem tipada.

Just do it

Você pode só mudar a função
Mudar o tipo do objeto
Mudar o nome da variável

O compilador é seu amigo e vai te avisar quando algo for dar problema!

Com um sistema de tipos bom assim que o programa compila você se sente bem confiante de que está tudo certo, coisa que você só teria com muuuuuuitos testes no Javascript normal.

4.

ESTRUTURA DE DADOS IMUTÁVEIS E OTIMIZADAS

$x + y$

$$x + y$$

$$x = 1 \text{ e } y = 2$$

Eu sei que o resultado da operação é 3

$x + y$

$x = 1$ e $y = ???$

Y pode ser diferente dependendo de condições de funções e da posição das estrelas

Mutações escondem mudanças

Se você quer ter uma aplicação que te deixe são você precisa ter previsibilidade do que cada variável é.

Ter objetos(records) ou variáveis congelados por default evitam diversos possíveis erros acidentais.

CONST

```
/* JS */
let a = 10;
const b = 15;


a += 1; // 11
b += 1; // TypeError: Assignment
to constant variable
```

```
/* JS */

const c = {};
const d = [];

c.foo = 10; // 10
d.push(10) // 1
```

```
/* JS */

const c = {};
const d = [];

c.foo = 10; // 10
d.push(10) // 1
```

Criar novos valores ao invés de mutar-los

Object.assign({})

Object.freeze()

[...foo]

{ ...bar }

Const é uma mentira e até objetos e arrays guardam referência de seus filhos

Bibliotecas que programadores JS usam p/ suprir isso:

IMMUTABLE

Todas as estruturas de dados e tipos em Reason são imutáveis.

“Records” não podem ser mudados.

```
/* REASON */
type test ={ a: int, b: int };
let x ={ a: 2, b: 3 };
x.a = 3 // Error: The record field is not
mutable
```

A não ser que deixe explícito na tipagem.

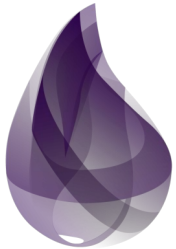
```
/* REASON */
type test ={ a: int, b: int };
let x ={ mutable a: 2, b: 3 };
x.a = 3
```

5.

FEATURES FUNCIONAIS

Currying, partial application, pipeline operator, pattern matching etc.

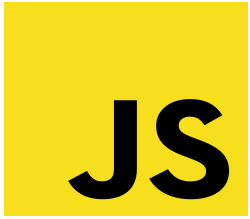
LINGUAGEM FUNCIONAL

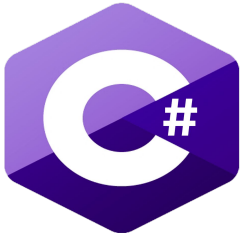


Funções anônimas

Closures

Map / Filter / Reduce





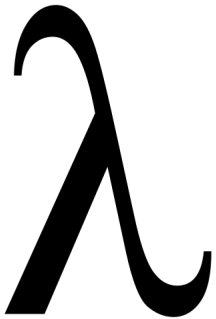
Pipeline

Currying

Pattern Matching

Recursão

Etc etc



PROPOSALS AINDA PRA ENTRAR NO JAVASCRIPT

Elementos comuns de linguagens funcionais que ainda não existem no JS:

- Currying / Partial application: A ideia de passar apenas alguns dos argumentos de uma função
- Pipeline: Compor funções de forma sequencial
- Pattern Matching: Outra forma lidar com fluxo condicional do programa(if else)

CURRYING E REUSO DE CODIGO

```
int ⇒ int ⇒ int
let multiply = (a, b) ⇒ a * b;
```

CURRYING E REUSO DE CODIGO

```
int ⇒ int ⇒ int
let multiply = (a, b) ⇒ a * b;
let multiplyBy10 = multiply(10)
```

Em programação muitas coisas podem dar errado sempre. Muitos edge-cases e ReasonML te dá poderes pra lidar com isso te força estruturar o código lidando com condições triviais.

PATTERN MATCHING

Lidar com cada um dos casos de input recebidos.

```
type animal = Human(string) | Cat | Dog
let action = switch(value) {
  /* `++` é concatenação de string */
  | Human(name) => "Olá meu nome é " ++ name
  | Cat => "Miau!"
  | Dog => "Au!"
}
```

PATTERN MATCHING

Lidar com cada um dos casos de input recebidos.

```
let animal = Human(string) | Cat | Dog | Fish
```

```
Warning: this pattern-matching is not exhaustive. Here
is an example of a case that is not matched: Fish
```

Option

```
let possiblyNull: option(string) = Some("Ana");
switch (possiblyNull) {
  | None => print_endline("Olá pessoa")
  | Some(message) => print_endline("Olá " ++ message)
};
```

Te força estruturar o código lidando com condições triviais. Pattern Matching ajuda a descobrir states e edge cases que passam batido.
ReasonML te força a ver esses casos.

Exemplo: Se o user não está logado mas envia um form que deveria estar logado.

PROPOSALS AINDA PRA ENTRAR NO JAVASCRIPT

Bibliotecas que programadores JS usam p/ suprir isso:

Não se assuste com programação funcional

Você não precisa saber magia negra em jargões pra fazer algo funcionar(*exemplo: monadas*)

TRANSPARENCIA REFERENCIAL

Toda parte do programa sempre tem o mesmo resultado para o mesmo input.
Similar a uma função matemática(f(x) = x + 1)

“Só funções”

TRANSPARENCIA REFERENCIAL

Sempre recebo algo
Sempre retorno algo
Uma função não depende de coisas fora do escopo dela
E pro mesmo input sempre tenho o mesmo output

```
let tax = ?;
let getPrice = x => x + tax;

getPrice(10) // ???
```

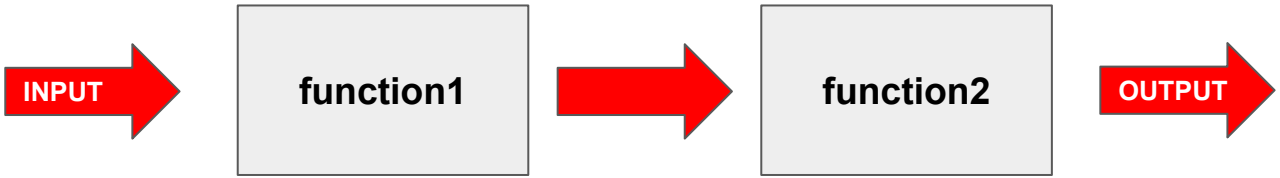
```
let getPrice = (x, tax) => x +
tax;


getPrice(10, 1) // 11
```

- São mais fáceis de testar
- São mais fáceis de debuggar
- Funções sem efeitos colaterais são mais fáceis de “bater o olho e entender” pois o input é claro com o output e portanto, mais fáceis de “reason about”

OO pattern/principle	FP pattern/principle
• Single Responsibility Principle	• Functions
• Open/Closed principle	• Functions
• Dependency Inversion Principle	• Functions, also
• Interface Segregation Principle	• Functions
• Factory pattern	• Yes, functions
• Strategy pattern	• Oh my, functions again!
• Decorator pattern	• Functions
• Visitor pattern	• Functions □

TRANSPARENCIA REFERENCIAL

Para se resolver um problema deve-se compor funções uma dentro da outra até chegar em um resultado esperado.



Dividir a conta do bar entre 3 amigos contando com a taxa de serviço.

PIPELINE OPERATOR

```
let conta = [14,50, 14,50, 4,0];
conta
   |> sum
   |> multiply(1.1)
   |> divide(3)
```

6.

“EASY TO REASON ABOUT”

“Fácil de executar na sua cabeça”

“Fácil de compreender o que o código espera e faz”

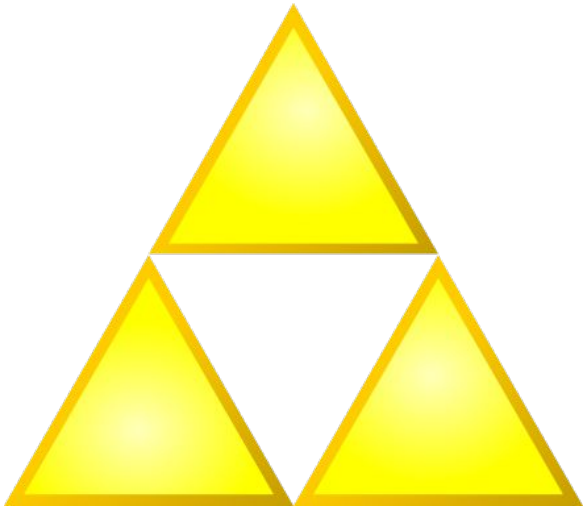
“Deterministico”

Entendendo o comportamento do seu programa

- Nomes que fazem sentido
- Código com uma lógica clara
- Código curto
- Não depende de um estado
- Pouca mudança de valores
- Tipagem

```
val areaOfCircle: Float => Float

let areaOfCircle = (r) => 2.0 *. 3.14 *. r;
```

- Imutabilidade
- Tipagem
- Transparencia referencial

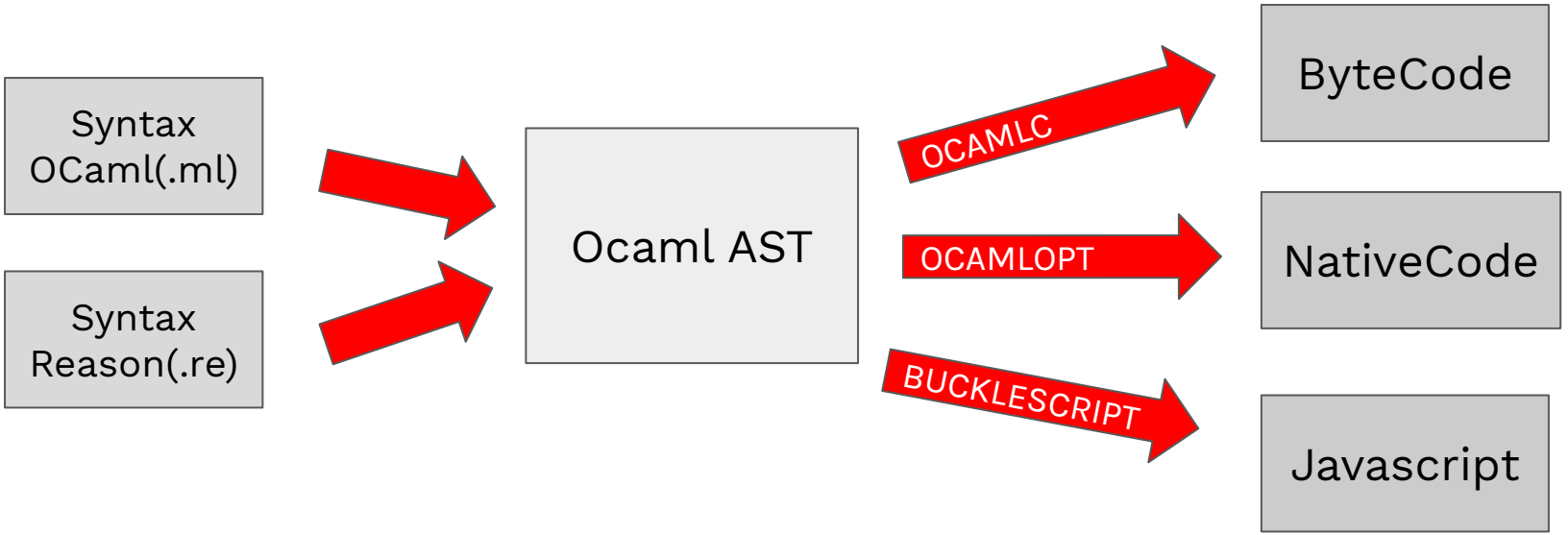


7.

MULTIPLATAFORMA

Uma das 7 maravilhas que literalmente todo framework / lang tenta fazer.

“written once, run anywhere”.


Syntax
OCaml(.ml)
Syntax
Reason(.re)
Ocaml AST
OCAMLC
OCAMLOPT
BUCKLESCRIPT
ByteCode
NativeCode
Javascript

Escrevendo um programa em ReasonML você pode facilmente pra Bytecode, Nativo(iOS, Android, Windows ou Linux) ou Javascript.

BuckleScript

Write safer and simpler code in OCaml & Reason, compile to JavaScript.

```
let result = Js.(
  [| 1; 2; 3; 4 |]
  |> Array.filter (fun x -> x > 2)
  |> Array.mapi (fun x i -> x + i)
  |> Array.reduce (fun x y -> x + y) 0
)
```

GET STARTED TUTORIAL

Lean Developer Experience

Simple, small and blazing fast build workflow. No more configuration debugging!

The Whole JavaScript Ecosystem

Readable JS output + comprehensive support for communicating with existing code.

Solid, Stable & Cross-platform

BuckleScript is backed by OCaml. Decades of **type system** research and compiler engineering.

O que é o BS?

- Criado pelo Bob Zhang da Bloomberg.
- É um backend pro compilador de OCaml que transforma Ocaml/Reason para Javascript legível para ser usado em qualquer navegador.
- Extremamente otimizado gerando melhoras em milisegundos.
- Extremamente bem documentado e com interop direto com o ecossistema Javascript e NPM/Yarn

O Buckescript é um compilador incrível extremamente rápido que não deixou a desejar.

Transforma em código JS completamente legível.


demo.re
index.js
{} package
my-first-app ▸ src ▸ demo.re
1 Js.log("Olá TDC!");
2
3 let add = (a, b) => a + b;
4


.re
index.js
package.json my-first-app
demo.bs.js
my-first-app ▸ src ▸ demo.bs.js ▸ ...
1 // Generated by BUCKLESCRIPT VERSION 2.1.0, PLEASE EDIT WITH CARE
2 'use strict';
3
4
5 console.log("Olá TDC!");
6
7 function add(a, b) {
8 return a + b | 0;
9 }
10
11 exports.add = add;
12 /* Not a pure module */
13

8.

OPTIMIZED

Messenger.com Now 50% Converted to Reason

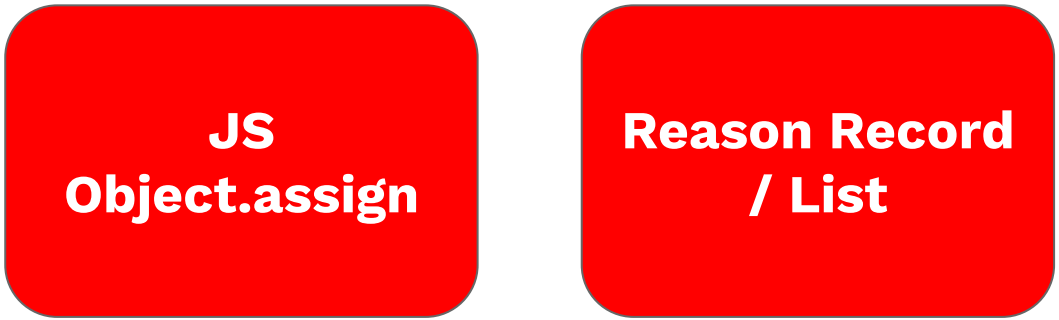
September 8, 2017

Boom!

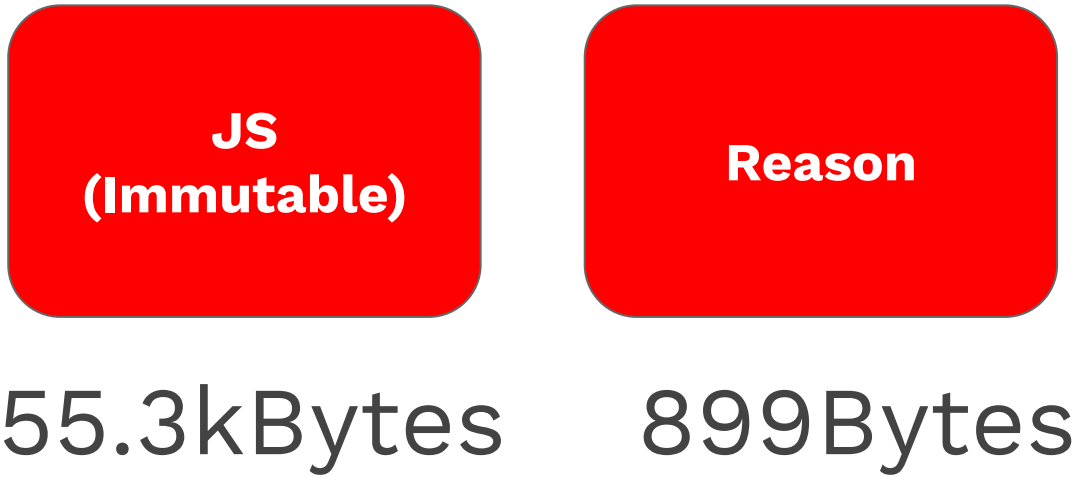
Messenger.com is the web version of Facebook Messenger; we also share code with facebook.com's inbox view and chat tabs. For over a year, the Reason team has been working directly on Messenger in order to integrate Reason + BuckleScript into the codebases. As of a while ago, we've reached 50% Reason code coverage!

Some Statistics

- Full rebuild of the Reason part of the codebase is ~2s (a few hundreds of files), incremental build (the norm) is **<100ms** on average. The BuckleScript author estimates that the build system should scale to a few hundred thousands files in the current condition.
- Messenger used to receive bugs reports on a daily basis; since the introduction of Reason, there have been a total of **10 bugs** (that's during the whole year, not per week)! *



https://github.com/neonsquare/bucklescript-benchmark



https://github.com/BuckleScript/bucklescript#immutable-data-structures

9.

MESMO ECOSSISTEMA JAVASCRIPT

Pq trocaria minha stack em uma linguagem nova?

Spoiler alert: Não precisa trocar

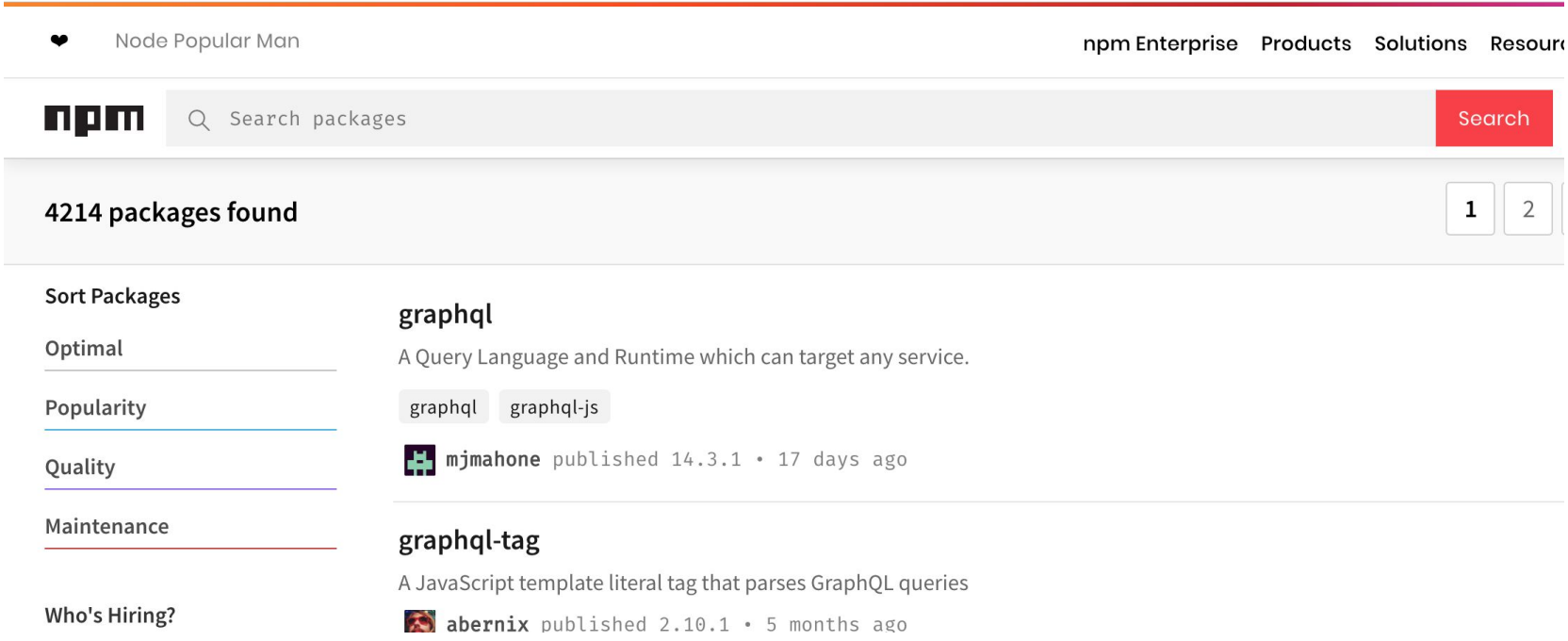
Foreign Function Interface

Bucklescript tem uma ponte chamada FFI que permite que você interaja com o ecossistema NPM apenas com um package de binding.

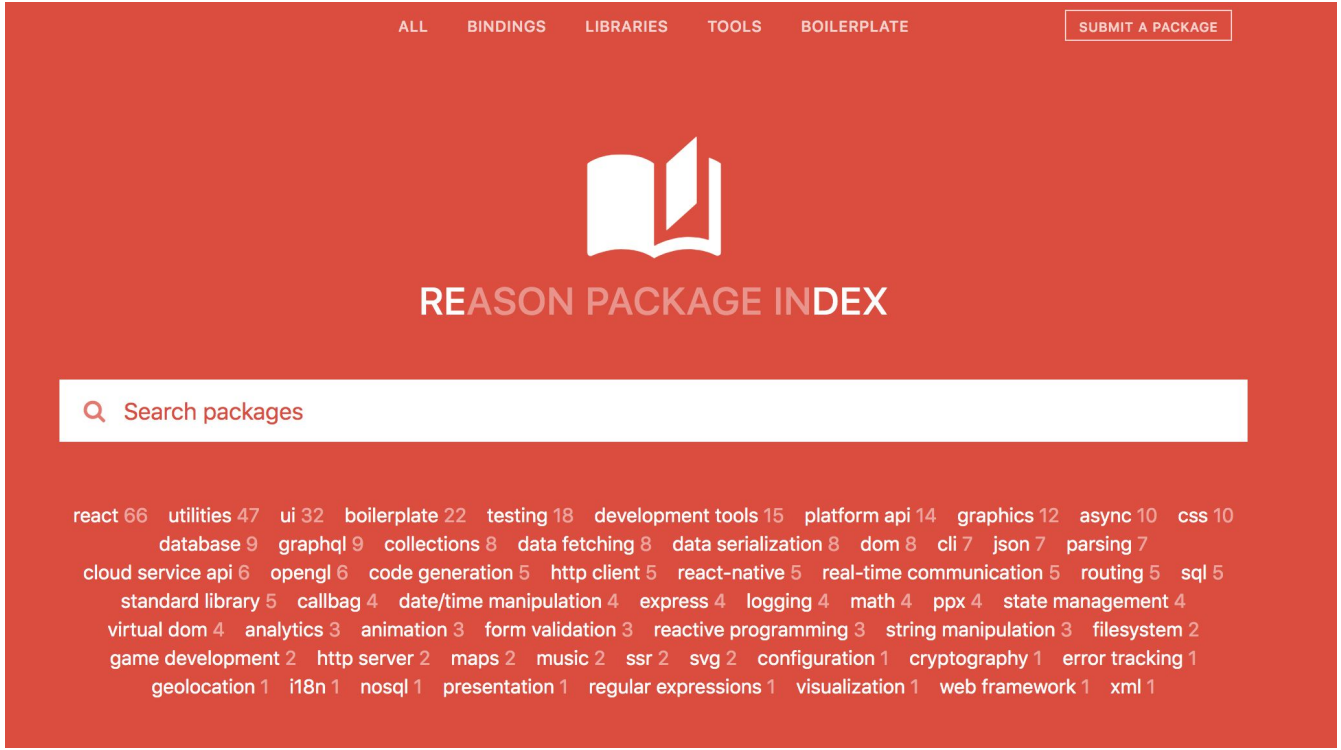
Foreign Function Interface

Isso garante que você tenha vantagens de tipos e outras coisas enquanto você usa bibliotecas que você já conhece e gosta do Javascript(Ramda <3)

Você tem bibliotecas do node...



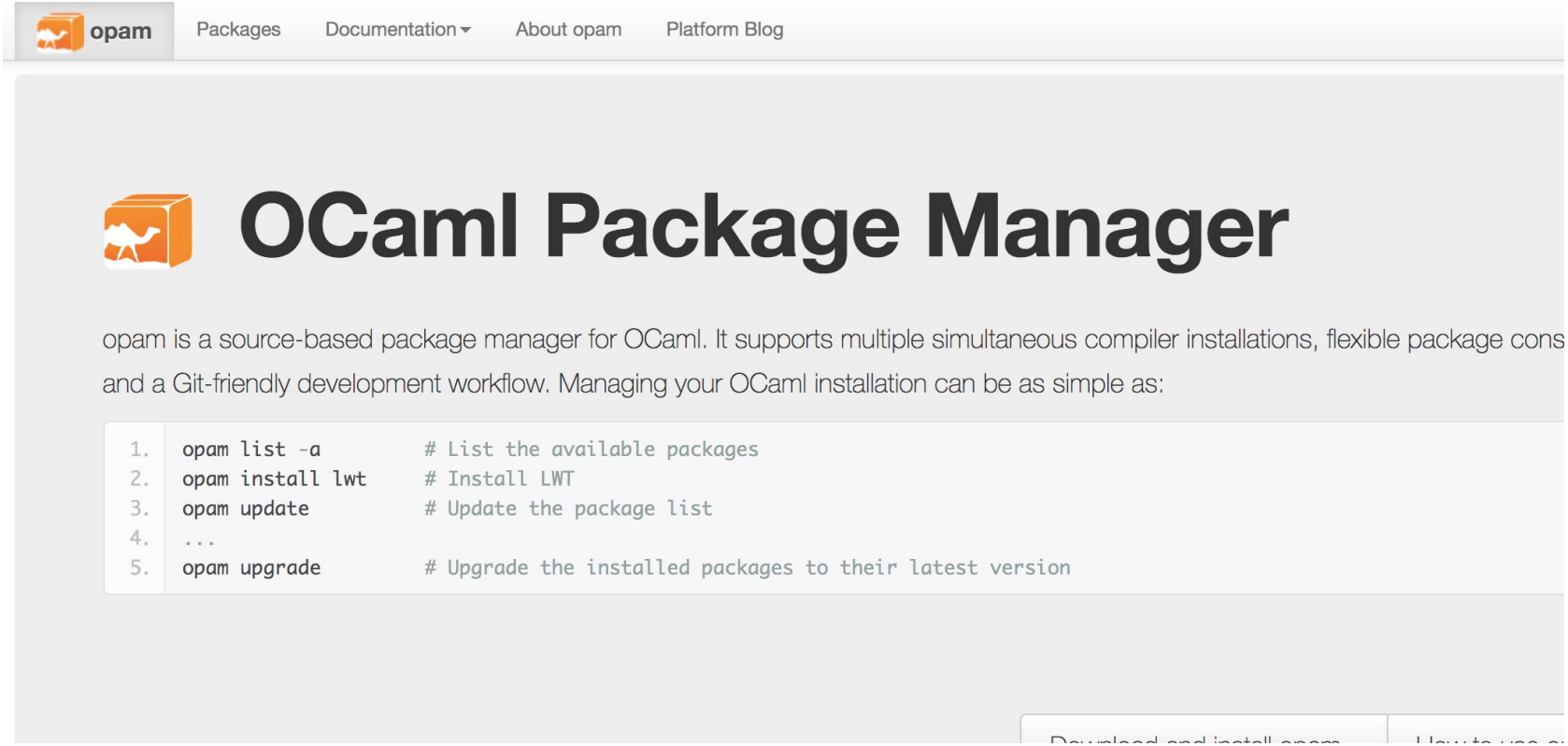
Você tem bibliotecas do reason...



Tudo junto no buckescript!

E caso você esteja usando o ambiente nativo do Ocaml...

Você tem bibliotecas do OCaml!



Se quero começar a usar ReasonML em um projeto javascript já existente.....

ADICIONAR UM BS-CONFIG

```
{
  "name": <app-name>,
  "sources": [
    "src"
  ],
  "bs-dependencies": [
    "reason-react",
    "bs-jest",
  ],
  "reason": {
    "react-jsx": 2
  },
  "bsc-flags": [
    "-bs-super-errors"
  ],
  "refmt": 3
}
```

“Package.json do reason”

ADICIONAR NO PACKAGE.JSON SCRIPTS DO BS

```
“clean”: “bsb -clean-world”,
“build”: “bsb -make-world”,
“watch”: “bsb -make-world -w”,
```

DEMO

10.

REACT BINDINGS

Que massa que ReasonML é puro, imutavel etc.

Mas como eu que eu faço pra lidar com impurezas e caos como lidar com a DOM por exemplo?

ReasonReact

All your ReactJS knowledge, codified.

```
let component = ReasonReact.statelessComponent("Greeting");

let make = (~name, _children) => {
  ...component,
  render: _self =>
    <button>
      {ReasonReact.stringToElement("Hello!")}
    </button>
};
```

GET STARTED TUTORIAL

Safe and Sound

It's Just Reason. We leverage the existing type system to create a library that types just right.

Playground for Future React

Lightweight, first-class support for the ReactJS community idioms you've been using.

Drop In

Easily integrate ReasonReact into your existing app. Convert a file, quickly see benefits.

REACT

A graça do Reason é que a partir do momento que começou a fazer binding com React tudo ficou mais mágico então você programa coisas boas no seu workflow comum Npm/Yarn pegando então o melhor dos dois ecossistemas.

SUPORTE A JSX

Actions Variant

```
type actions =
    | Click
    | Toggle
```

Typed State

```
type visibility = Visible | Invisible


type state = {
  count: int,
  show: visibility
}
```

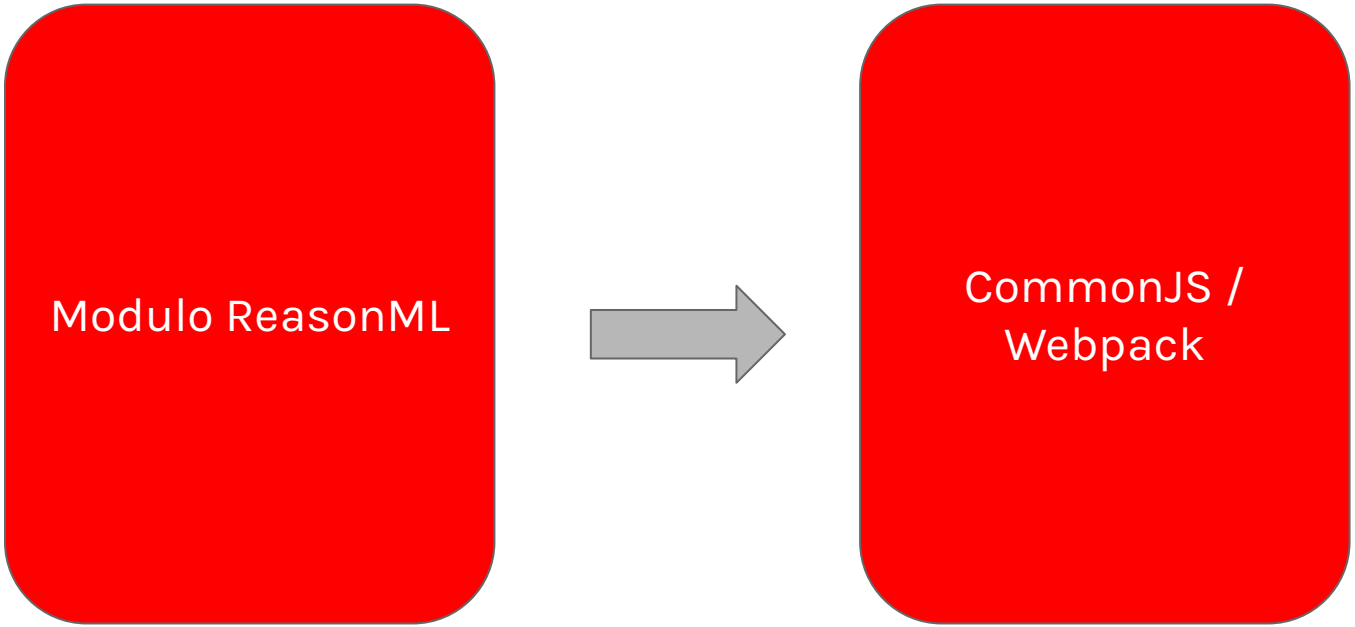
Typed State

```
switch (action) {
| Click => {...state, count: state.count + 1}
| Toggle => {...state, show: Invisible}
}, {count: 0, show: Visible});
```

REACT

```
[%bs.raw {|require('./app.css')|}];

[@bs.module] external logo : string = "./logo.svg";

let component = ReasonReact.statelessComponent("App");

let make = (~message, _children) => {
  ...component,
  render: (_self) => {
    <div className="App">
      <div className="App-header">
        <img src=logo className="App-logo" alt="logo" />
        <h2> (ReasonReact.stringToElement(message)) </h2>
      </div>
      <p className="App-intro">
        (ReasonReact.stringToElement("To get started, edit"))
        <code> (ReasonReact.stringToElement(" src/app.re ")) </code>
        (ReasonReact.stringToElement("and save to reload."))
      </p>
    </div>
  }
};
```

```
[@react.component]
let make = (~greeting) => {
 let (state, dispatch) = React.useReducer((state, action) =>
 switch (action) {
 | Click => {...state, count: state.count + 1}
 | Toggle => {...state, show: ! state.show}
 }, {count: 0, show: true});


 let message =
   "You've clicked this " ++ string_of_int(state.count) ++ " times(s)";
 <div>
   <button onClick={_event => dispatch(Click)}>
     …..
```


Modulo ReasonML
CommonJS /
Webpack

11.

EXPERIENCIA DO DESENVOLVEDOR / TIME

_

Uma oportunidade de aprender algo que vai mudar o que vc pensa em código.

Abrir a cabeça pra outros paradigmas te ajuda a ter decisões diferentes de arquitetura.

Programação é sobre
“comunicar pensamentos”
então experimentar novas
implementações e
bibliotecas te ajudam a
crescer.

Nem sempre o mais prático ou o mais produtivo é o que vai trazer um produto de melhor qualidade e mantenabilidade.

Demonstrar valor não é só entrega mas também é sobre diminuir bugs e tratar todos os problemas.

COMO COMEÇAR?

Temas do BS

```
npm install -g bs-platform
```

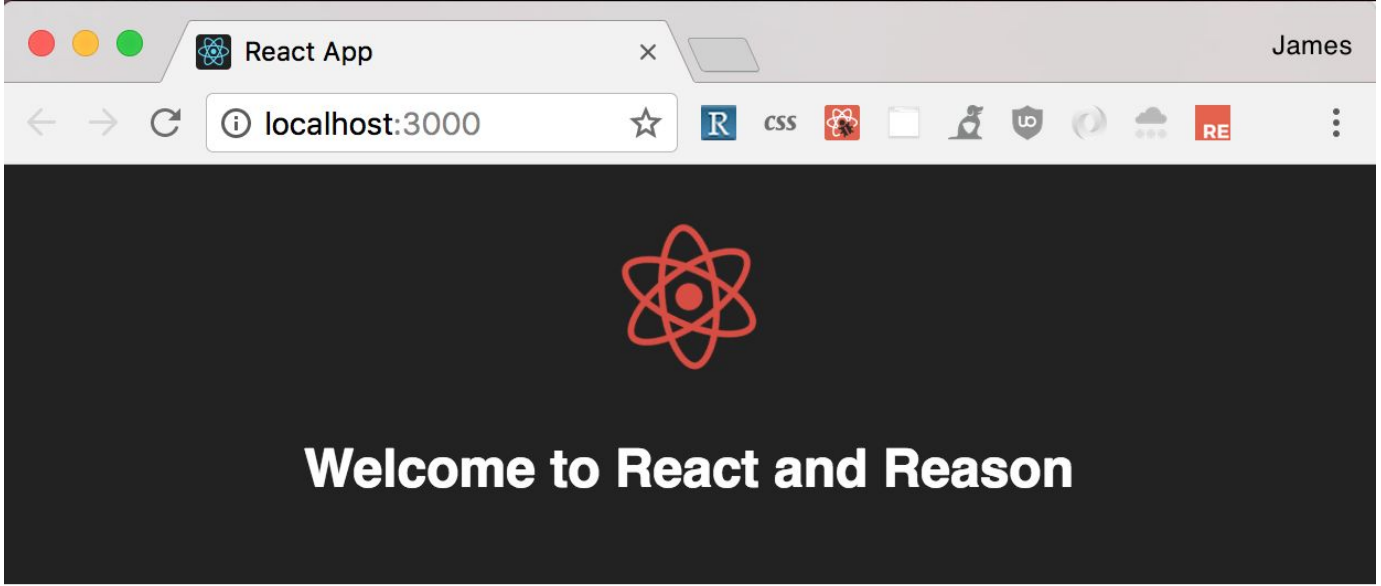
```
bsb -init nome-do-app -theme basic-reason
```

Pelo react-app

```
yarn create react-app nome-do-app
```

Pelo react-app

```
yarn create react-app nome-do-app
   -scripts-version reason-scripts
```



To get started, edit `src/App.re` and save to reload.

Pelo gatsby

```
npm install -g gatsby-cli

gatsby theme new nome-do-app gatsby-starter-reasonml
```

Gatsby ❤ ReasonML

Use this starter to create static sites with Gatsby using ReasonML components.

Features

- reason-react for type-safe React components in ReasonML
- bs-css for css-in-reason styling

Reference

- see re/Header.re for example component implementation
- see re/types/Gatsby.re for example BuckleScript bindings to Gatsby module

“Using @reasonml feels like gaining super powers. Learn one familiar, high level language built on rock solid language foundation (it's OCaml!). Integrate w/ existing JS/React/JSX. But then use that same language to compile native programs that start and run incredibly fast!”

Jordan Walke

2ality – JavaScript and more

2019-05

Unpacking hoisting

[2019-05-30] dev, javascript

2019-04

The new ECMAScript module support in Node.js 12

[2019-04-23] dev, javascript, nodejs, jsmodules

2019-01

ES proposal: Well-formed `JSON.stringify`

Dr. Axel Rauschmayer
Homepage | Twitter

Merlin doesn't recognize new `.re` file

all categories Latest Top Categories

Topic	Category	Users	Replies	Views	Activity
Do we need another ReasonReact boilerplate? reasonreact			1	19	1h
User-provided type inside a module			3	18	2h
Record vs. Js.t Object			2	14	3h
Snapshot testing a ReasonReact component reasonreact			4	59	4h
Iterate over Reason Object properties and get their types reasonreact			3	63	6h
Preferable form validators API			17	97	12h
How to create an object in ReasonML?			1	96	18h
Dead code elimination with Uglify and Bucklescript interop			6	93	1d
Merlin doesn't recognize new `.re` file			4	44	2d

https://sketch.sh/new/reason

Sketch.sh ▷ Run Save RE ML + New

untitled sketch

anonymous

```
1 let add2 = (a, b) ⇒ a +. b;
  let add2: (float, float) ⇒ float = <fun>;
2
3 add2(1, 2)
        ^
  Error: This expression has type int but an expression was expected of type
           float
4
```

Documentação BuckleScript

bucklescript.github.io/

Documentação Reason

reasonml.github.io/docs/en/quickstart-javascript.html

Documentação ReasonReact

reasonml.github.io/reason-react/docs/en/installation.html

Reason Package Index

redex.github.io/

Comunidade Discord
discord.gg/reasonml

Twitter
twitter.com/reasonml

Blog
reasonml.github.io/blog/

“Stackoverflow”
reasonml.chat

Obrigada!


anabastos


@naluhh


@anapbastos

O ReasonML também suporta JSX (a sintaxe para template HTML dentro do JavaScript usado pelo framework React do Facebook). Devido a ReasonML estar baseado no OCaml, muitas pessoas usam os dois nomes de forma intercambiável.

Comentário blablablabal