

componentes web funcionais com **REACT & RAMDAJS**

(ou: todos a bordo no trem do hype)

2017, @olarclara



FRONT IN
SALVADOR

paradigma

pa·ra·dig·ma (sm)

- 1** Algo que serve de exemplo ou modelo; padrão.
- 2** GRAM Modelo de conjugação ou de declinação de uma palavra.
- 3** LING Conjunto de termos comutáveis entre si, em uma mesma posição, numa estrutura.
- 4** FILOS Segundo o filósofo americano Thomas Kuhn (1922-1996), qualquer campo de investigação e de experiência que está na origem da evolução científica.

Dicionário Michaelis

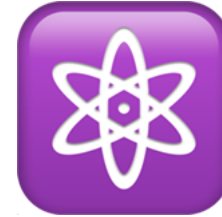
PROGRAMAÇÃO FUNCIONAL

- ~~Funções de alta-ordem, Monads, Applicatives, Functors...~~
- Funções que não têm efeitos colaterais;
- Funções que não dependem de dados de fora;
- Funções que não alteram dados de fora;

LIVE CODING!



REACT



- Baseado em componentes;
- Paradigma declarativo;

**“Todos os componentes no React devem agir
como funções puras em respeito as suas
props.”**

–Algun engenheiro do React

ChromeFileEditViewHistoryBookmarksPeopleWindowHelp

Ramda Documentationx

ramdajs.com

Sat 12:58 PM

Maria Clara

Ramda v0.23.0HomeDocumentationTry RamdaGitHubDiscuss

Ramda

A practical functional library for JavaScript programmers.

build passingnpm package 0.24.0dependencies nonegitterjoin chat

Why Ramda?

There are already several excellent libraries with a functional flavor. Typically, they are meant to be general-purpose toolkits, suitable for working in multiple paradigms. Ramda has a more focused goal. We wanted a library designed specifically for a functional programming style, one that makes it easy to create functional pipelines, one that never mutates user data.

What's Different?

The primary distinguishing features of Ramda are:

- Ramda emphasizes a purer functional style. Immutability and side-effect free functions are at the heart of its design philosophy. This can help you get the job done with simple, elegant code.
- Ramda functions are automatically curried. This allows you to easily build up new functions from old ones simply by not supplying the final parameters.
- The parameters to Ramda functions are arranged to make it convenient for currying. The data to be operated on is generally supplied last.


The last two points together make it very easy to build functions as sequences of simpler functions, each of which transforms the data and passes it along to the next. Ramda is designed to support this style of coding.

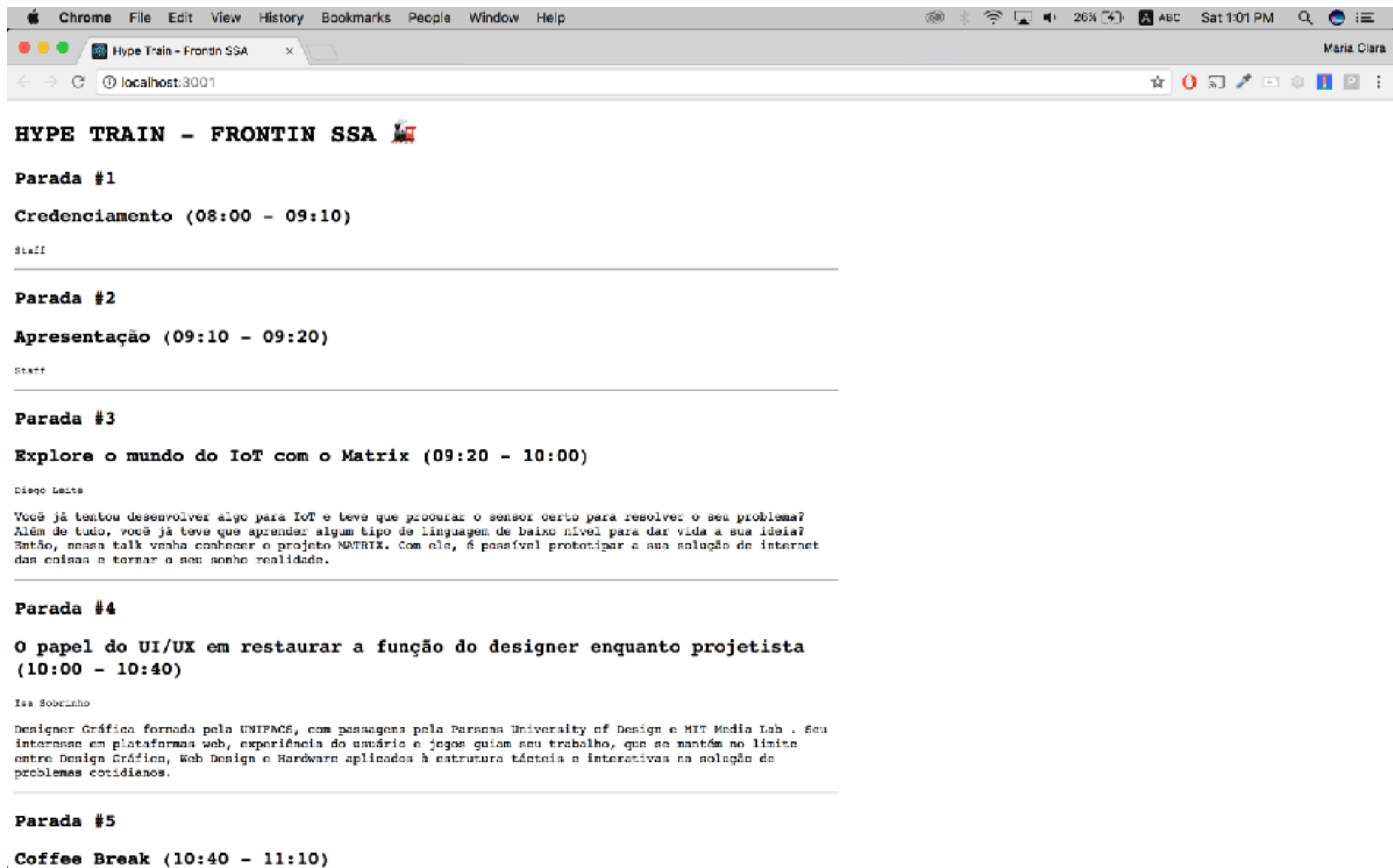
Introductions

- [Introducing Ramda](#) by Buzz de Cafe
- [Why Ramda?](#) by Scott Sauyet
- [Favoring Curry](#) by Scott Sauyet
- [Why Curry Helps](#) by Hugh Jackson
- [Hey Underscore, You're Doing It Wrong!](#) by Brian Lonsdorf
- [Thinking in Ramda](#) by Randy Coulman

Philosophy

OPEN CHAT






```
1  import React, { Component } from 'react'
2  import R from 'ramda'
3
4  class Schedule extends Component {
5    render() {
6      return (
7        <div className="App-schedule">
8          <ul>
9            {this.props.schedule.map((x) =>
10              <li key={x.id}>
11                <h2>Parada #{x.id}</h2>
12                <h2>{x.title} ({x.start} - {x.end})</h2>
13                <small>{ x.speaker ? x.speaker : 'Staff'}</small>
14                <p>{x.description}</p>
15                <hr />
16              </li>
17            )}
18          </ul>
19        </div>
20      )
21    }
22  }
23
24  export default Schedule
```

Schedule.js

TalkItem.js



```
1  import React, { Component } from 'react'
2
3  class TalkItem extends Component {
4    render() {
5      return (
6        <div>
7          <h2>Parada #{this.props.talk.id}</h2>
8          <h2>{this.props.talk.title} ({this.props.talk.start} - {this.props.talk.end})</h2>
9          <small>{ this.props.talk.speaker ? this.props.talk.speaker : 'Staff'}</small>
10         <p>{this.props.talk.description}</p>
11         <hr />
12       </div>
13     )
14   }
15 }
16
17 export default TalkItem
18
19
20 |
```

MakeList.js

map-filter-reduce.js



```
1  import React from 'react';
2  import R from 'ramda';
3  import TalkItem from './TalkItem'
4
5  const Container = children => (<div>{children}</div>);
6
7  const List = children => (<ul>
8    {children}
9  </ul>);
10
11 const ListItem = (x) => (<li key={x.id}>
12   <TalkItem talk={x}/>
13 </li>);
14
15 const MakeList = R.compose(Container, List, R.map(ListItem), R.prop('items'));
16
17 export default MakeList;
18
19
20
```

Schedule.js x



```
1  import React, { Component } from 'react'
2  import MakeList from './MakeList'
3
4  class Schedule extends Component {
5      render() {
6          return (
7              <MakeList items={this.props.schedule} />
8          )
9      }
10 }
11
12 export default Schedule
13
14
15
16
```

R.PROP()

`s → {s: a} → a | Undefined`

Retorna uma função que, quando fornecido um objeto, retorna a propriedade indicada desse objeto, se existir.

R.MAP()

`Functor f \Rightarrow (a \rightarrow b) \rightarrow f a \rightarrow f b`

Recebe uma função e um functor, aplica a função a cada um dos valores do functor e retorna um functor da mesma forma.

R.COMPOSE()

```
((y → z), (x → y), ..., (o → p), ((a, b, ..., n) → o))  
→ ((a, b, ..., n) → z)
```

Executa a composição da função direita para a esquerda.

OBRIGADA! 

OLARCLARA.GITHUB.IO