

Arquitetura de Front Ends

Samuel Martins

Ementa

- Perfil do profissional - o que o mercado espera?
- Introdução à web: a tríade HTML, CSS e JavaScript + Javascript Moderno;
- Pré-processadores: SASS e LESS;
- Frameworks javascript: um panorama geral sobre React, Angular e Vue;
- Arquitetura Flux: gerenciamento de estados de forma eficiente.

Ementa

- Arquitetura de monorepos com Lerna;
- Arquitetura de Micro Frontends;
- Aplicações *server-side rendered* e estratégias de SEO;
- Progressive web applications (PWA);
- Web assembly;
- *Serverless computing*;
- Aspectos de segurança em front-end.

Objetivo da disciplina

- Adquirir conhecimento macro sobre as principais tecnologias front-end;
- Apontar caminhos para solucionar problemas conhecidos do mercado, sem apego a um único framework específico;
- Criar provas de conceito em cima dos problemas propostos.

Perfil profissional



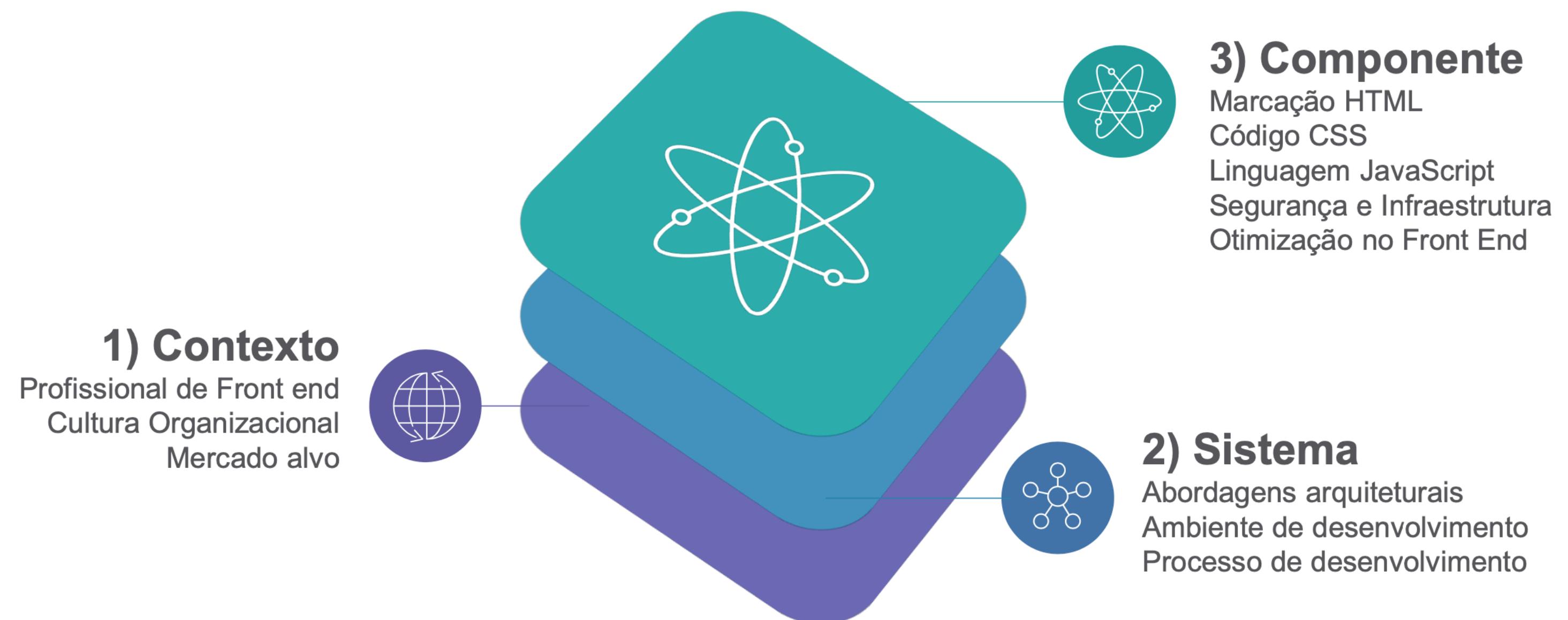
Samuel Martins

Arquitetura de front ends

“Arquitetura é a organização fundamental de um sistema, compreendido pelos seus componentes, o relacionamento entre eles e os princípios que direcionam seu projeto e sua evolução.”

Recommended Practice for Architecture Description of Software-Intensive Systems, IEEE (2000)

Arquitetura de front ends



Arquitetura de front ends - Contexto

O que forma um(a) profissional de arquitetura

- Profissional de front end:
 - Background, experiências e aptidão.
- Cultura organizacional:
 - Regras de negócio, ambiente, budget e etc.
- Mercado alvo:
 - Demandas de negócio e necessidades externas.

Arquitetura de front ends - Sistema

- Abordagens arquiteturais:
 - Multi page | Single Page Application | PWA | Serverless Computing | Micro frontends.
- Ambiente de desenvolvimento:
 - Frameworks | Gerenciadores de dependências | Bundlers.
- Processo de desenvolvimento:
 - Testes | CI/CD | Cultura DevOps.

Arquitetura de front ends - Componente

- **Marcação HTML:**
 - HTML semântico | Acessibilidade | Performance.
- **CSS:**
 - Style guides | Padrão de código (OOCSS, BEM, CSS Funcional) | Pré-processadores | Responsividade.
- **Linguagem JavaScript:**
 - Recursos da linguagem | Transpiladores | Especificações (ES7).
- **Segurança e Infraestrutura:**
 - Open Authorization (OAuth) | Cross-origin (CORS) | Cross-site Scripting (XSS).
- **Otimização no Front End:**
 - Avaliação de desempenho | Redução de requisições | Otimização de conteúdo | Controle de Cache.

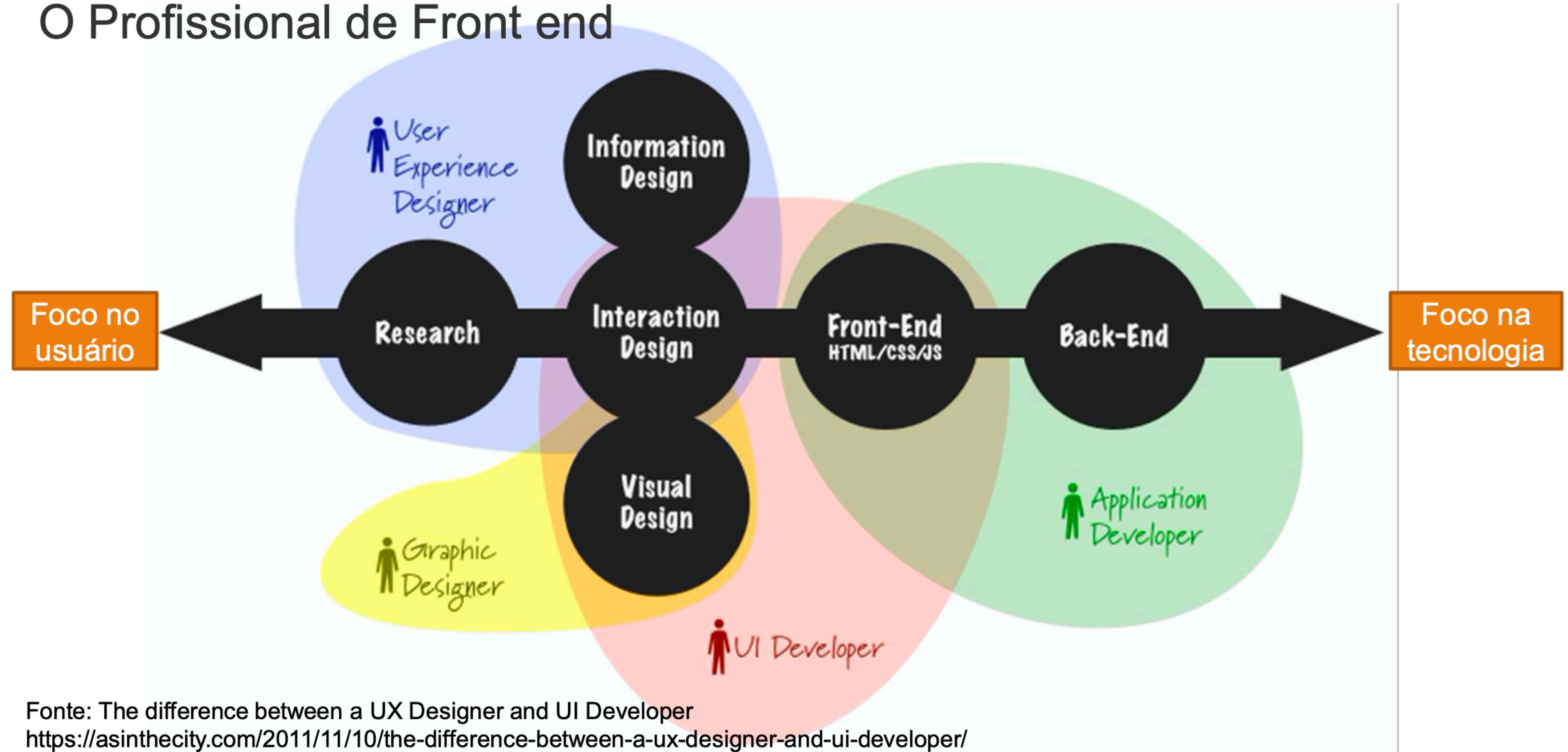
Perfil profissional e responsabilidades

- Desapego de frameworks
 - Toda semana, um frameworks javascript novo. Precisamos sempre manter a mente aberta para todas as tecnologias
- Capacidade de fazer provas de conceito (POCs)
 - Realizar experimentos e prototipação de funcionalidades que auxiliem a modelagem da visão do produto
- Direcionar a visão técnica dos produtos web
- Oferecer consultoria ao time de desenvolvimento
- Ser o(a) guardião(a) dos componentes compartilhados entre os times de desenvolvimento

Perfil profissional - Soft skills

- Disseminação de conhecimentos;
 - Pair programming;
 - Análise de pull requests.
- Liderança e referência técnica para a equipe e clientes.

O Profissional de Front end



Front end - Roadmap

<https://roadmap.sh/frontend>

Ambiente

Samuel Martins

Ambiente

World wide web (protocolo HTTP, websocket...)

Engine de interpretação (Chromium, Gecko, WebKit...)

Browser (Chrome, Firefox, Edge...)

Tecnologia (linguagem e frameworks)

Abordagens arquiteturais, paradigmas, padrões de projeto

Protocolo HTTP

Samuel Martins

Protocolo http

- Requisições e respostas:
 - Verbos HTTP (GET, POST, DELETE...);
 - Cabeçalhos de requisição resposta.

Client



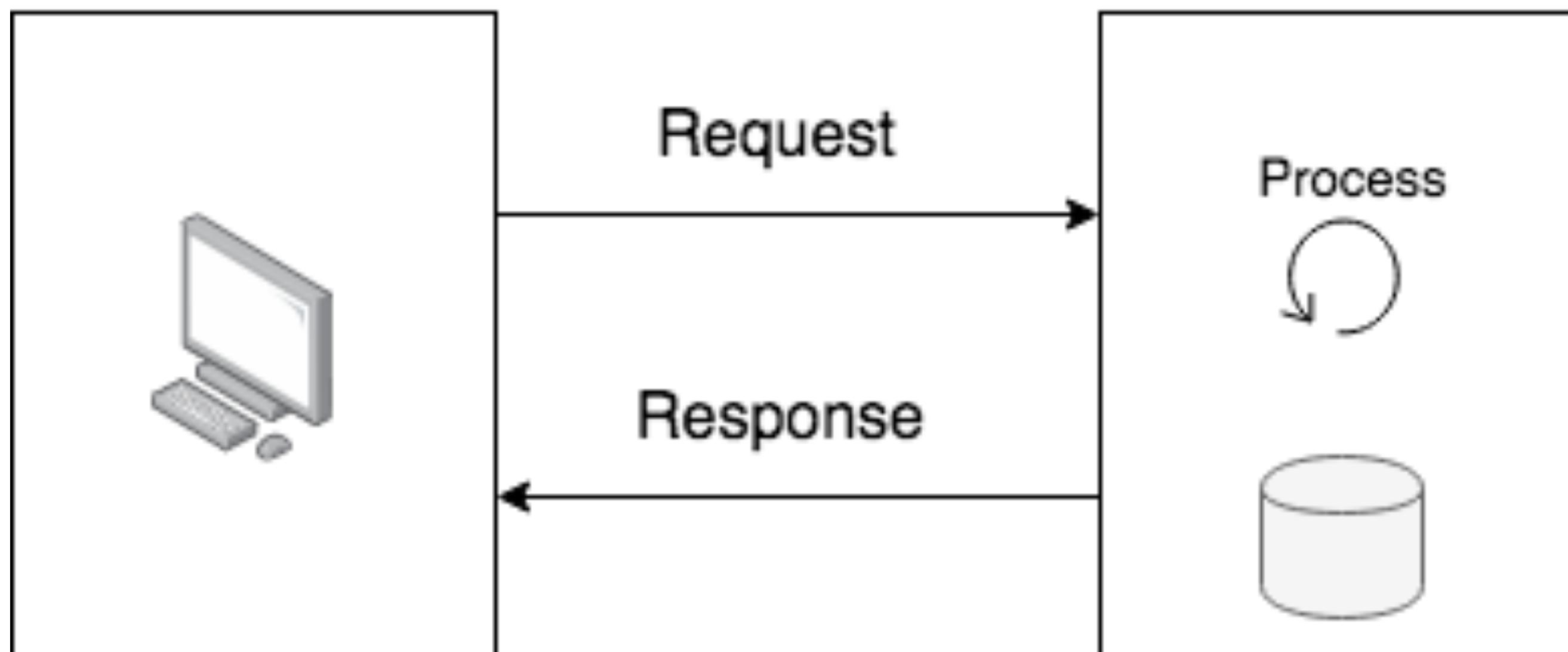
Server

Process



Request

Response



Cabeçalho de requisição

X Headers Preview Response Cookies Timing

▼ General

Request URL: <https://www.pucminas.br/destaques/Paginas/default.aspx>

Request Method: GET

Status Code: 200 OK

Remote Address: 200.229.32.27:443

Referrer Policy: no-referrer-when-downgrade

Cabeçalho de resposta

▼ Response Headers

[view source](#)

Cache-Control: private

Content-Encoding: gzip

Content-Length: 55711

Content-Type: text/html; charset=utf-8

Date: Fri, 26 Apr 2019 02:35:07 GMT

Expires: Fri, 26 Apr 2019 02:35:52 GMT

Cabeçalhos

- Autenticação cliente/servidor;
- Proteção de páginas por login e senha;
- Metadados a respeito da requisição/resposta enviada.

Principais verbos HTTP

GET

- ▶ Requisita dados de um determinado recurso

POST

- ▶ Envia informações para o servidor

DELETE

- ▶ Envia um comando para remover um recurso específico

UPDATE

- ▶ Envia um comando para atualizar um recurso específico

Engine de interpretação

Samuel Martins

Engine de interpretação (browser engines)

- Motor responsável por interpretar os códigos escritos no browser;
 - Páginas HTML, CSS e Javascript;
- Explica comportamentos de estilo diferentes para cada browser;
- Curiosidade: uma boa engine escrita pelo Google (V8 Engine) deu origem à plataforma Node, uma possibilidade de interpretar JavaScript no lado do servidor.

Engine de interpretação (browser engines)

- **Chrome e Edge:** Chromium;
- **Firefox:** Gecko;
- **Safari:** WebKit;
- **Internet Explorer:** descontinuado pela microsoft em Janeiro/2020;

HTML, CSS e Javascript

Uma breve história...

HTML - hypertext markup language

- Linguagem de marcação;
- Utilizada para desenvolvimento de páginas web (.html);
- Interpretadas pelos navegadores;
- Utilizada para identificar elementos em uma página.

```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4 </head>
5 <body>
6     <section>
7         Section 1
8         <article>
9             <h1>Article title</h1>
10            </article>
11        </section>
12    </body>
13 </html>
```

```
1 <html lang="en"> ← Indica o início de um documento HTML
2 <head>
3   <title>Document</title>
4 </head>
5 <body>
6   <section>
7     Section 1
8     <article>
9       <h1>Article title</h1>
10      </article>
11    </section>
12  </body>
13 </html>
```

```
1 <html lang="en">
2 <head> ← Cabeçalho: contém metadados sobre documento criado
3   <title>Document</title>
4 </head>
5 <body>
6   <section>
7     Section 1
8     <article>
9       <h1>Article title</h1>
10      </article>
11    </section>
12  </body>
13 </html>
```

```
1 <html lang="en">
2 <head>
3   <title>Document</title>
4 </head>
5 <body> ←———— Corpo: conteúdo visível ao usuário
6   <section>
7     Section 1
8     <article>
9       <h1>Article title</h1>
10      </article>
11    </section>
12  </body>
13 </html>
```

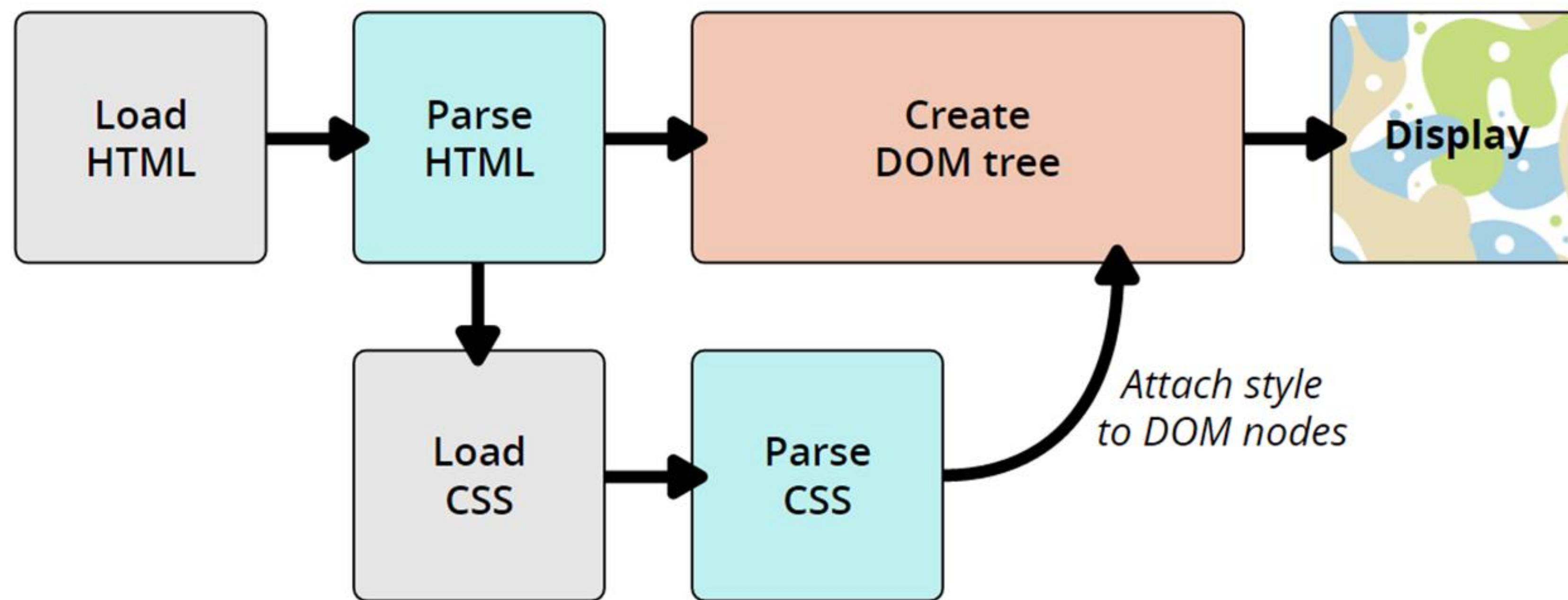
```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4 </head>
5 <body>
6     <section>
7         Section 1
8         <article>
9             <h1>Article title</h1>
10            </article>
11        </section>
12    </body>
13 </html>
```

Descreve o conteúdo da
página

CSS - Cascading style sheet

- Linguagem declarativa para definição de regras visuais de um documento HTML;
- Um arquivo HTML pode ter múltiplos arquivos CSS associados;

Dinâmica de processamento



Fonte: [Mozilla Developer Network](#)

CSS - Formas de utilização

- Arquivo externo:
 - Melhor modularização e manutenibilidade;
 - Separação de conceitos e responsabilidades;
 - Permite atualizar regras css de um único lugar

Page.html

```
<html>
  <head>
    <link rel="stylesheet"
          href="style.css">
  </head>
  <body>
    <h1>Page</h1>
    <p>Hello World!</p>
  </body>
</html>
```

Style.css

```
body{
  background-color:yellow;
  color: blue;
  font-family: Arial;
  font-size: 20px;
}
h1 {
  color: red;
}
```

Page

Hello World!

Page

Hello World!

CSS - Formas de utilização

- Bloco de estilos:
 - Estilo declarado dentro do arquivo HTML;
 - Regras restritas a um único documento;
 - Pouca reusabilidade;
 - Maior performance.

```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4
5     <style>
6         #main-article {
7             background: yellow;
8         }
9
10        .main-title {
11            color: white;
12        }
13    </style>
14 </head>
15 <body>
16     <article id="main-article">
17         <h1 class="main-title">Article title</h1>
18     </article>
19 </body>
20 </html>
```

CSS - Formas de utilização

- Estilos inline:
 - Menor legibilidade;
 - Pouquíssima reusabilidade;
 - Requer definições de regra para todos os elementos que utilizarem aquele estilo.

```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4 </head>
5 <body>
6     <article id="main-article" style="background:
yellow;">
7         <h1 class="main-title" style="color:
white;">Article title</h1>
8     </article>
9 </body>
10 </html>
```

Javascript

- Linguagem de programação interpretada (no caso, pela engine do browser);
- Linguagem fracamente tipada: possível declarar uma variável que pode receber qualquer tipo a qualquer momento;
- Adiciona interatividade nas aplicações web.

```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4
5     <script>
6         var x = "Message";
7         alert(x);
8     </script>
9 </head>
10 <body>
11     <article id="main-article">
12         <h1 class="main-title">Article title</h1>
13     </article>
14 </body>
15 </html>
```

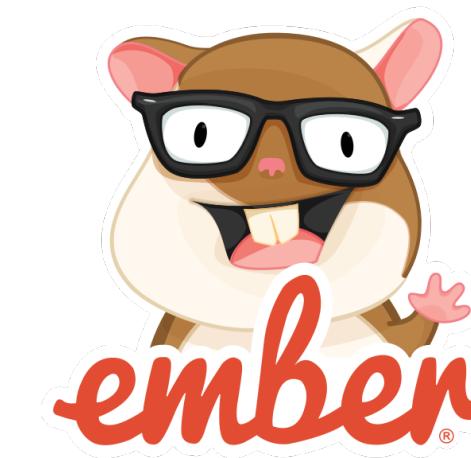
```
1 <html lang="en">
2 <head>
3     <title>Document</title>
4
5     <script src=".//script.js"></script>
6 </head>
7 <body>
8     <article id="main-article">
9         <h1 class="main-title">Article title</h1>
10    </article>
11 </body>
12 </html>
```

script.js

```
1     var x = "Message";  
2     alert(x);
```

Frameworks e ecossistema

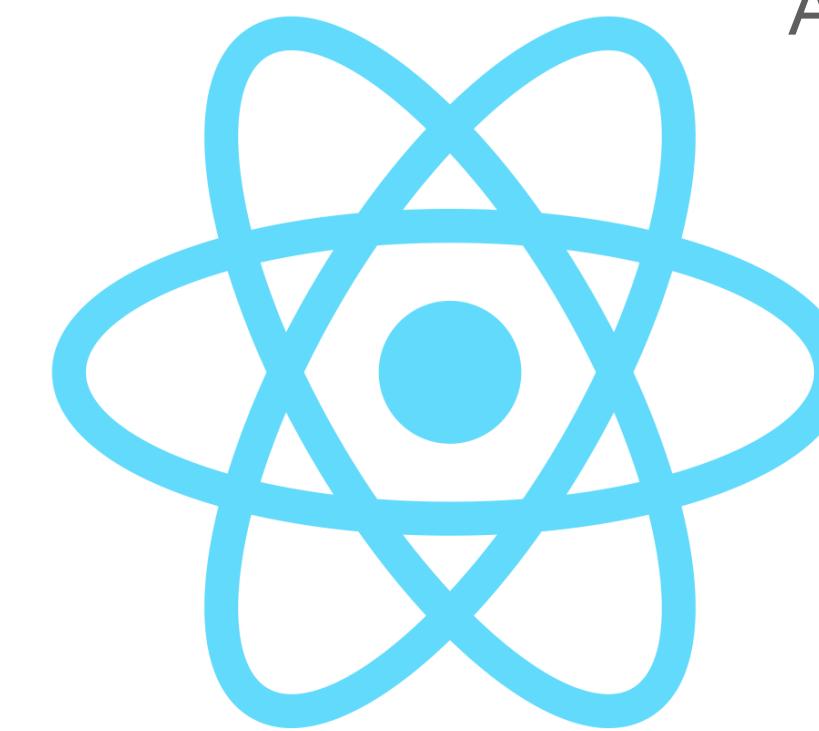
Samuel Martins



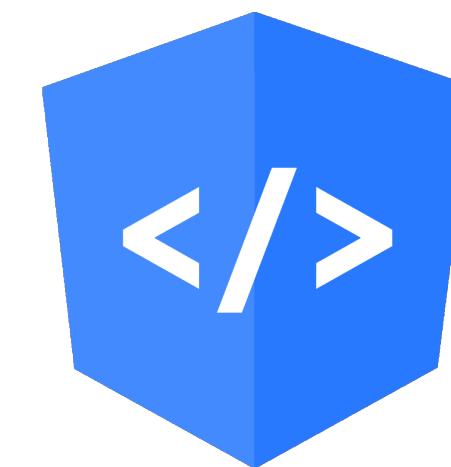
Angular



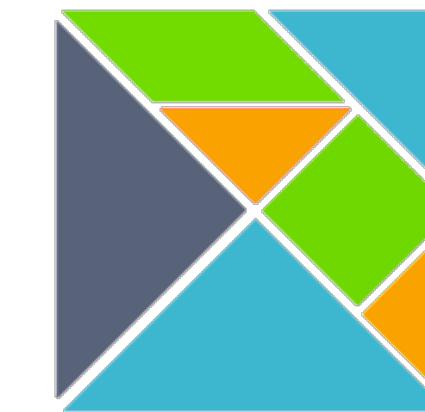
Vuejs



ReactJs



Angular Elements



Elm



Polymer



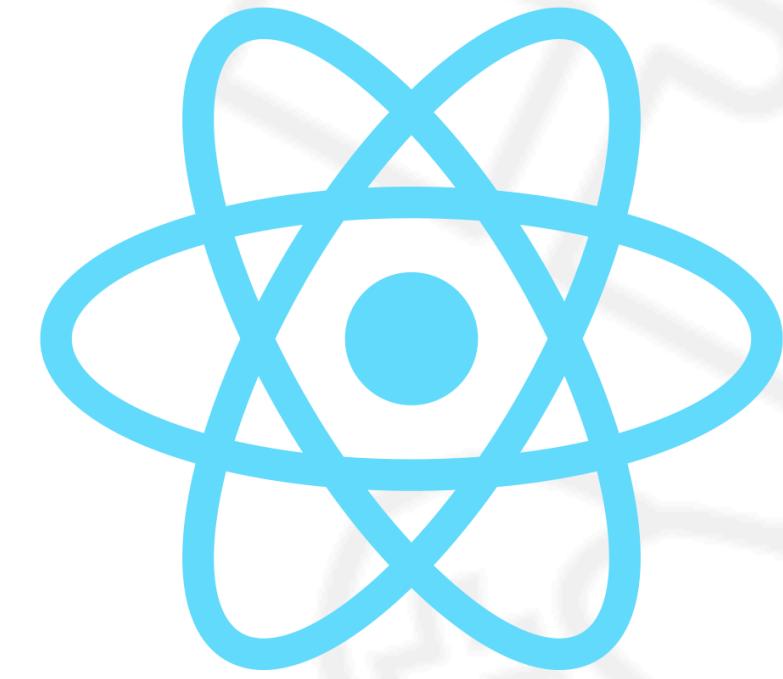
3 frameworks mais utilizados atualmente



Angular



Vuejs



ReactJs



Angular

Angularjs != Angular

Versão 1.x

Versão 2.x

Angular - Arquitetura

- Pensado para o desenvolvimento de aplicações front-end de **ponta-a-ponta**;
- Setup inicial com **tudo que é necessário** para desenvolver uma aplicação moderna:
 - Sistema de rotas;
 - Arquitetura e organização de pastas sugeridas;
 - Sugere o uso dos *Services* para consultas a *APIs* externas.

Angular - Arquitetura

- Utiliza, desde o início, uma linguagem tipada (typescript);
- Permite uma boa escalabilidade. Linguagem tipada tem se tornado essencial em grandes projetos;
- Desenvolvido para permitir o desenvolvedor focar na lógica de negócio.

Angular - Aspectos técnicos

- Proximidade à linguagens tipadas de back-end mais conhecidas (C#, Java...);
- Utilização do conceito conhecido de orientação a objetos. Ex.: um mesmo modelo definido na aplicação de back-end pode servir de base para um mesmo domínio no front-end.

Angular - Aspectos técnicos

- **angular-cli**: utilitário de linha de comando para criação de partes da aplicação;
- Permite criar componentes, serviços, módulos e diretivas.

```
ng generate component my-component  
ng g c my-component
```

Comunidade e evolução

- Conferência oficial anual para apresentação de novidades e evoluções do framework (NG-Conf);
- Possui bibliotecas de componentes para agilizar ainda mais o desenvolvimento das aplicações:
 - Angular-material: <https://material.angular.io/>;

Angular - Pontos de atenção

- Necessário mais linhas de código para escrita de uma aplicação;
- Curva de aprendizado pode ser alta para pessoas com backgrounds 100% front-end:
 - Necessário aprender conceitos de orientação a objetos (Tipos, Heranças, Interfaces, Generics, Annotations/Decorators);
 - Necessário aprender a utilizar o angular-cli.

Angular - Pontos de atenção

- Necessário o aprendizado de muitos conceitos core do framework:
 - ▶ Componentes;
 - ▶ Diretivas;
 - ▶ Módulos;
 - ▶ Serviços;
 - ▶ Programação reativa com RxJs (Observables, subscribers...);
 - ▶ Injeção de dependências;
 - ▶ HttpClient.

Angular - Casos de uso comuns

- Aplicações web corporativas (ERP, Dashboards...);
- Aplicações de grande porte em geral;
- Utilizada por times com mais conhecimento em linguagens server-side tipadas (Java, C#) e com pouco conhecimento em JavaScript.

Exemplo

<https://codesandbox.io/s/angular-demo-ygv74>



Vue

“Vue is designed from the ground up to be incrementally adoptable”

Vue - Arquitetura

- Focado 100% em componentes. Resumidamente, é um framework para criação de componentes web;
- Vue não é um framework para desenvolvimento de SPAs (*Single Page Applications*);
- É possível desenvolver SPAs instalando bibliotecas externas:
 - **Vue-router**: possibilita o gerenciamento de rotas;
 - **Vuex**: possibilita o gerenciamento de estados;

Vue - Arquitetura

- Similaridade com o AngularJs;
- Facilita a adoção incremental em um projeto já existente;
- TODO o código relacionado a um componente pode ser escrito no mesmo arquivo.

The screenshot shows a macOS application window titled "Hello.vue". The window has a dark-themed interface with red, yellow, and green window control buttons at the top left. The title bar also displays the file name "Hello.vue". The main content area is a code editor with the following code:

```
1 <template>
2   <p>{{ greeting }} World!</p>
3 </template>
4
5 <script>
6 module.exports = {
7   data: function () {
8     return {
9       greeting: 'Hello'
10    }
11  }
12}
13</script>
14
15<style scoped>
16 p {
17   font-size: 2em;
18   text-align: center;
19 }
20</style>
```

The code is color-coded: red for XML tags, orange for the export statement, purple for the data function, pink for the return keyword, green for the greeting variable, blue for the font-size and text-align CSS properties, and cyan for the style tag.

At the bottom of the code editor, there are status indicators: "Line 21, Column 1" on the left, "Spaces: 2" in the center, and "Vue Component" on the right.

Vue - Aspectos técnicos

- Possui um formato de arquivo próprio (.vue);
- Configuração inicial utiliza JavaScript “puro”;
- Permite a escrita de CSS com escopo de forma nativa;
- Permite a configuração da aplicação para aceitar a escrita dos componentes em TypeScript.

Vue - Comunidade e evolução

- Possui um bibliotecas de componentes para agilizar o desenvolvimento das aplicações (ex.: <https://vuematerial.io/>);
- Lista de projetos da comunidade para o Vue: <https://github.com/vuejs/awesome-vue>;
- Eventos conhecidos: <https://events.vuejs.org/conferences/>;

Vue - Pontos de atenção

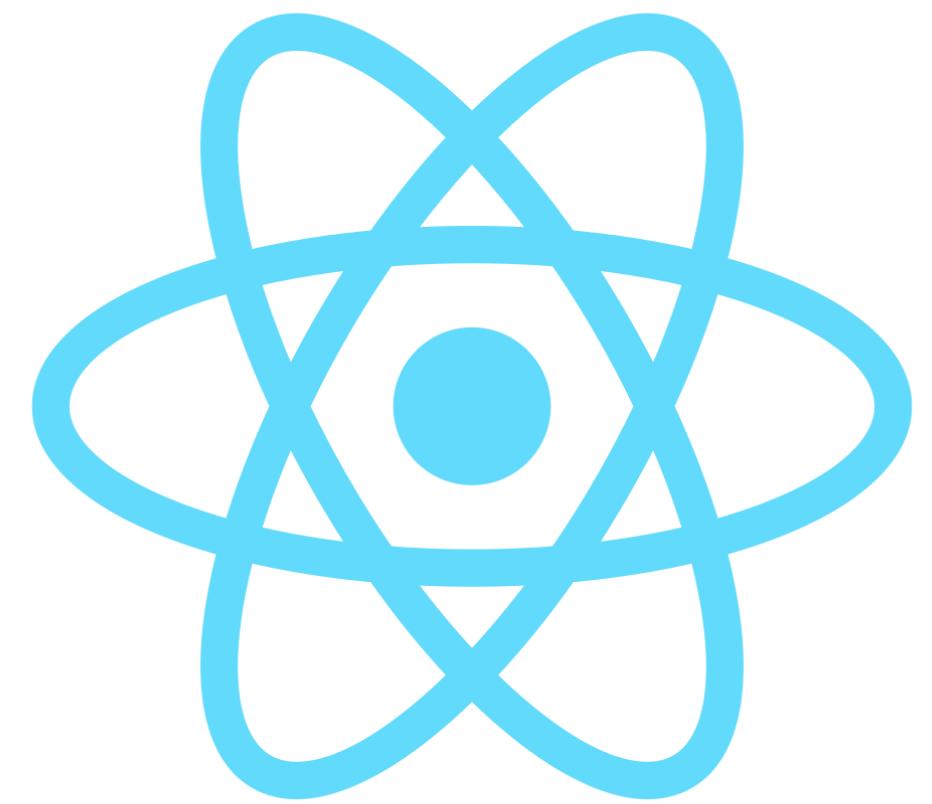
- Não possui nenhuma definição de arquitetura da aplicação (organização de pastas, por exemplo);
- Deixa sempre a cargo do desenvolvedor definir o seu próprio padrão de desenvolvimento;
- Arquivos dos componentes podem crescer exponencialmente. Por isso a necessidade de separar os componentes em partes mais pequenas o possível.

Vue - Casos de uso comuns

- Desenvolvimento de bibliotecas de componentes *cross-framework (Design systems, bibliotecas UI)*;
- Componentização de aplicações já existentes;
- Criação de aplicações de pequeno porte.

Exemplo

<https://codesandbox.io/s/vue-demo-s8d4m>



React

“A JavaScript library for building user interfaces”

React - Arquitetura

- Arquitetura 100% focada em componentes (no React, TUDO é um componente);
- Biblioteca concebida inicialmente para desenvolvimento de interfaces de usuário;
- Utiliza uma linguagem diferente, o JSX;
- Primeiro framework implementar o conceito de Virtual DOM;
- Nasceu focado em performance.

React - Arquitetura

- Permite a integração com o TypeScript;
- Permite a criação de aplicações SPA por meio de bibliotecas da comunidade (react-router, por exemplo);
- Permite a adoção incremental em projetos já existentes;
- Utiliza um paradigma de escrita de componentes diferente dos demais.

```
const MyComponent = () => <h1>Hello World</h1>
```

```
function App(props) {  
  return (  
    <div className="App">  
      <h1>{props.title}</h1>  
      <h2>Start editing to see some magic happen!</h2>  
    </div>  
  );  
}
```

React - Arquitetura

- Não possui mecanismo nativo para gerência de estados da aplicação;
- Possui como forte aliado o *Redux*, biblioteca da comunidade amplamente utilizada para gerenciamento de estados;
- Não possui conceitos além dos componentes. Não existe serviço, diretiva nem módulos.

React - Aspectos técnicos

- Utiliza um dos melhores algoritmos de “reconciliação” de elementos DOM;
- Boa integração com outra linguagem tipada, o *FlowTyped*;
- Possui diversos caminhos para resolver o mesmo problema;
- Um componente pode ser escrito:
 - Com funções;
 - Com classes;
 - Com funções + react hooks.

React - Aspectos técnicos

- Utiliza vários conceitos da programação funcional;
 - Componentes escritos como funções;
 - Componentes não alteram valores fora do seu escopo (funções puras);
 - Estados do componente são modificados atribuindo um novo valor, e nunca alterando diretamente (imutabilidade);
 - Efeitos colaterais (chamadas em APIs, por exemplo), podem ser tratados de forma separa do restante do ciclo de vida (tratamento adequado de efeitos colaterais).

Aspectos técnicos

- Possibilidade de utilizar o conceito de composição de funções nos componentes;
- Por ter os componentes escritos em funções, permite utilizar qualquer conceito conhecido relacionado a funções:
 - Funções que retornam outros componentes;
 - Funções que recebem parâmetros que e retornam componentes dinâmicos.

Comunidade e evolução

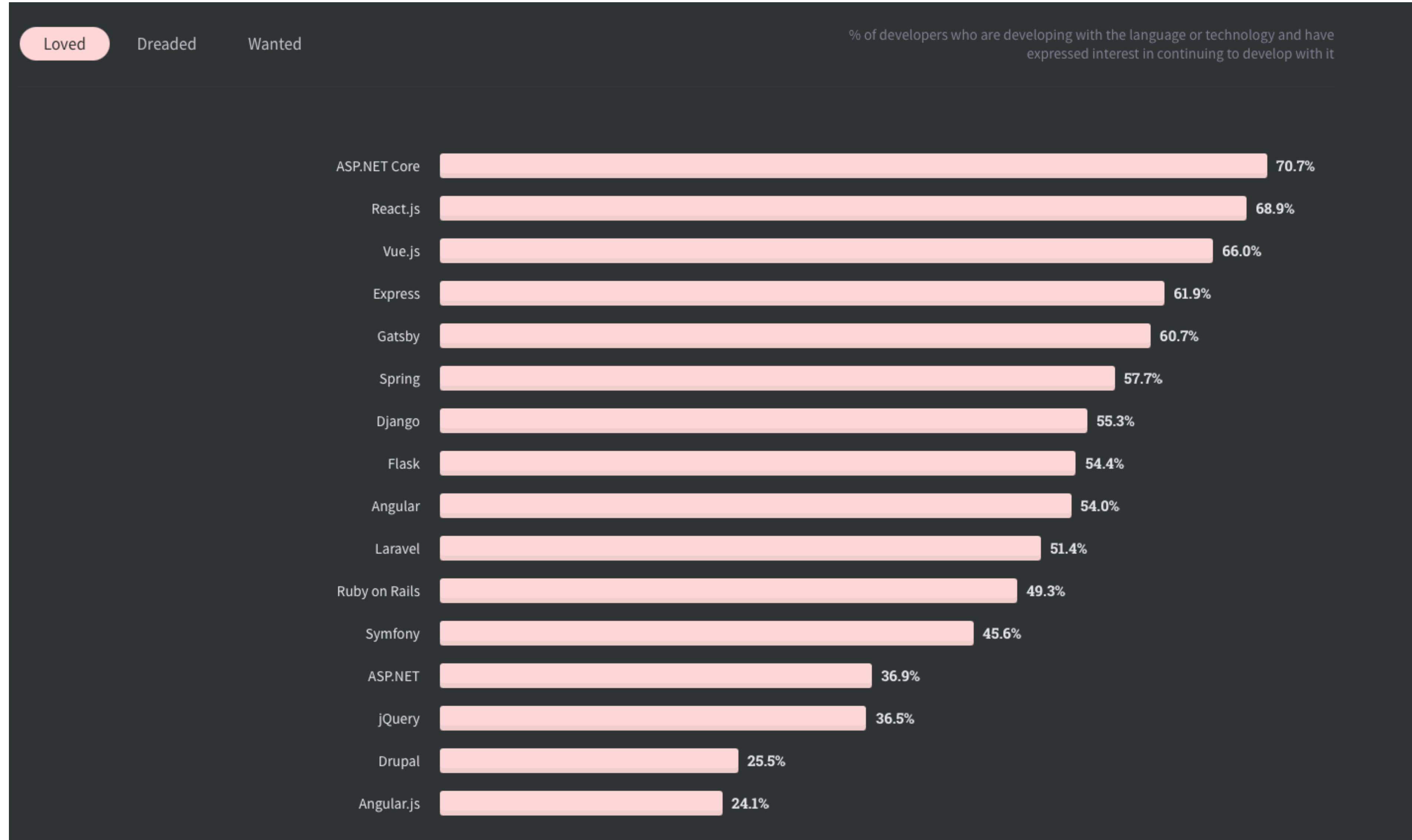
- Possui a maior e melhor comunidade;
- Constantemente em evolução;
- Evoluções, até então, foram tratadas de forma a manter uma maior retrocompatibilidade;
- Eventos conhecidos: <https://reactconf.com.br/>;
- TUDO já foi feito por alguém no StackOverflow.

Loved

Dreaded

Wanted

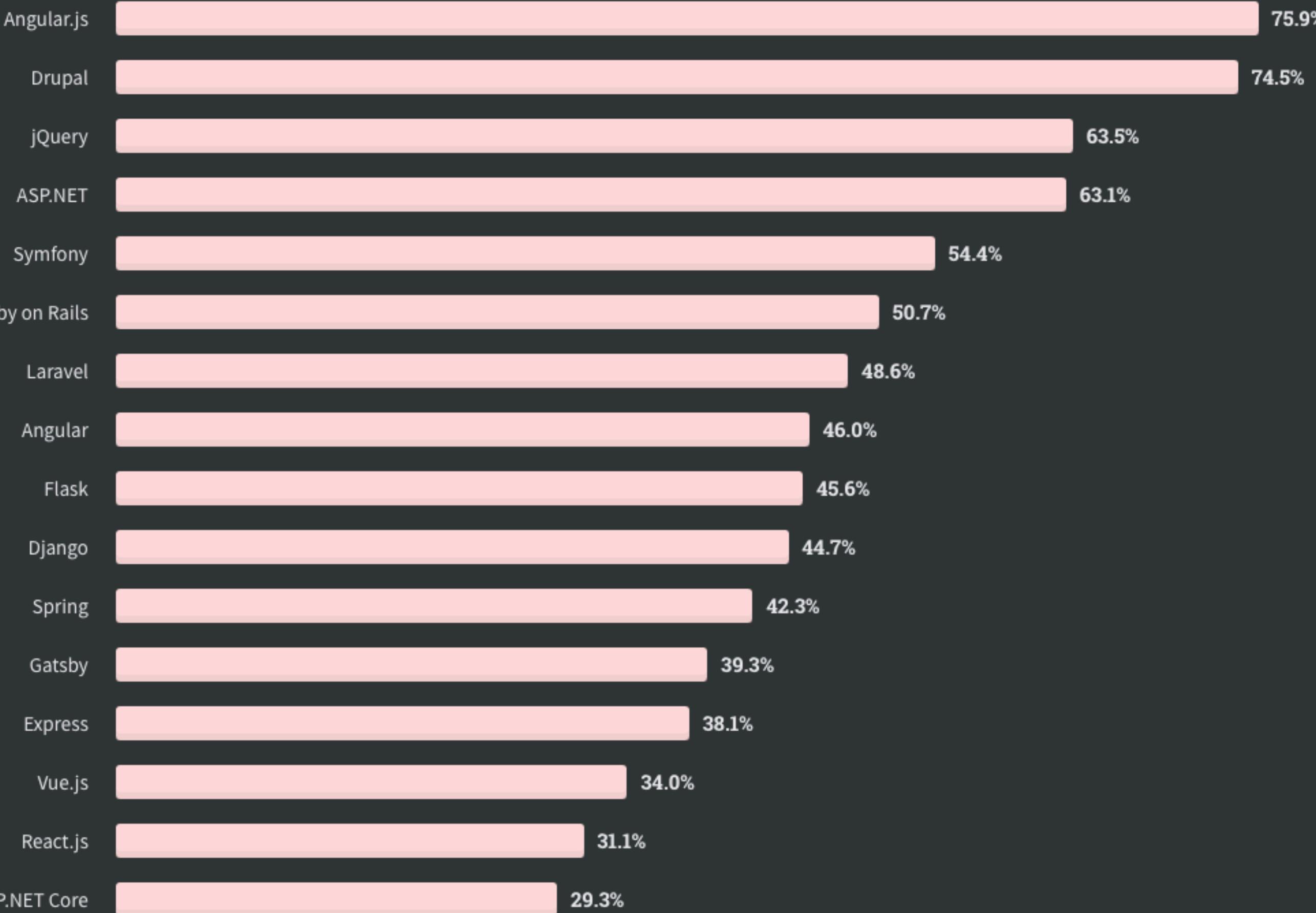
% of developers who are developing with the language or technology and have expressed interest in continuing to develop with it



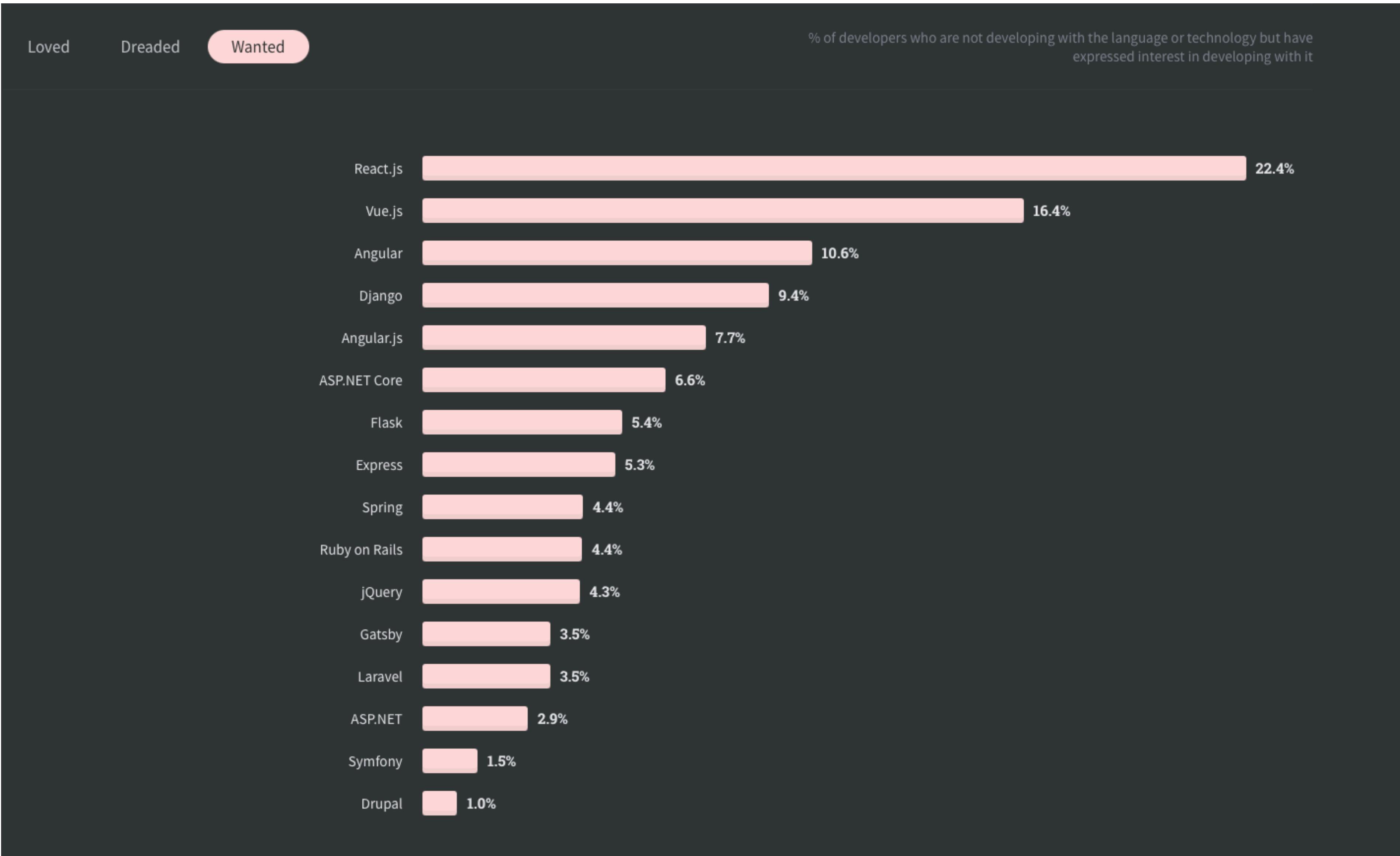
Fonte: <https://insights.stackoverflow.com/survey/2020#most-loved-dreaded-and-wanted>

Loved Dreaded Wanted

% of developers who are developing with the language or technology but have not expressed interest in continuing to do so



Fonte: <https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-dreaded2>



Fonte: <https://insights.stackoverflow.com/survey/2020#technology-most-loved-dreaded-and-wanted-web-frameworks-wanted2>

Pontos de atenção

- Não possui estrutura de pastas e arquivos definidos;
- Mudança de *mindset* na criação dos componentes por conta da linguagem (`jsx`);
- Necessidade de instalação de pacotes adicionais para funcionar como um SPA.

Casos de uso comuns

- Criação de bibliotecas de componentes (*Design Systems*, componentização de projetos existentes);
- Criação de aplicações de grande porte;
 - Boa forma de manutenção de estados de aplicação através do Redux;
 - Vários casos de sucesso para inspirar;
- Criação de aplicações que possuem uma versão Mobile;
 - React Native utiliza o mesmo conceito do React para desenvolvimento de aplicações mobile nativas.

Casos de uso comuns

- Desenvolvimento de aplicações que requerem foco maior em performance;
- Utilizado muito por times que já possuem um bom background em *JavaScript* e front-end de uma forma geral.

Exemplo

<https://codesandbox.io/s/react-demo-voc4m>

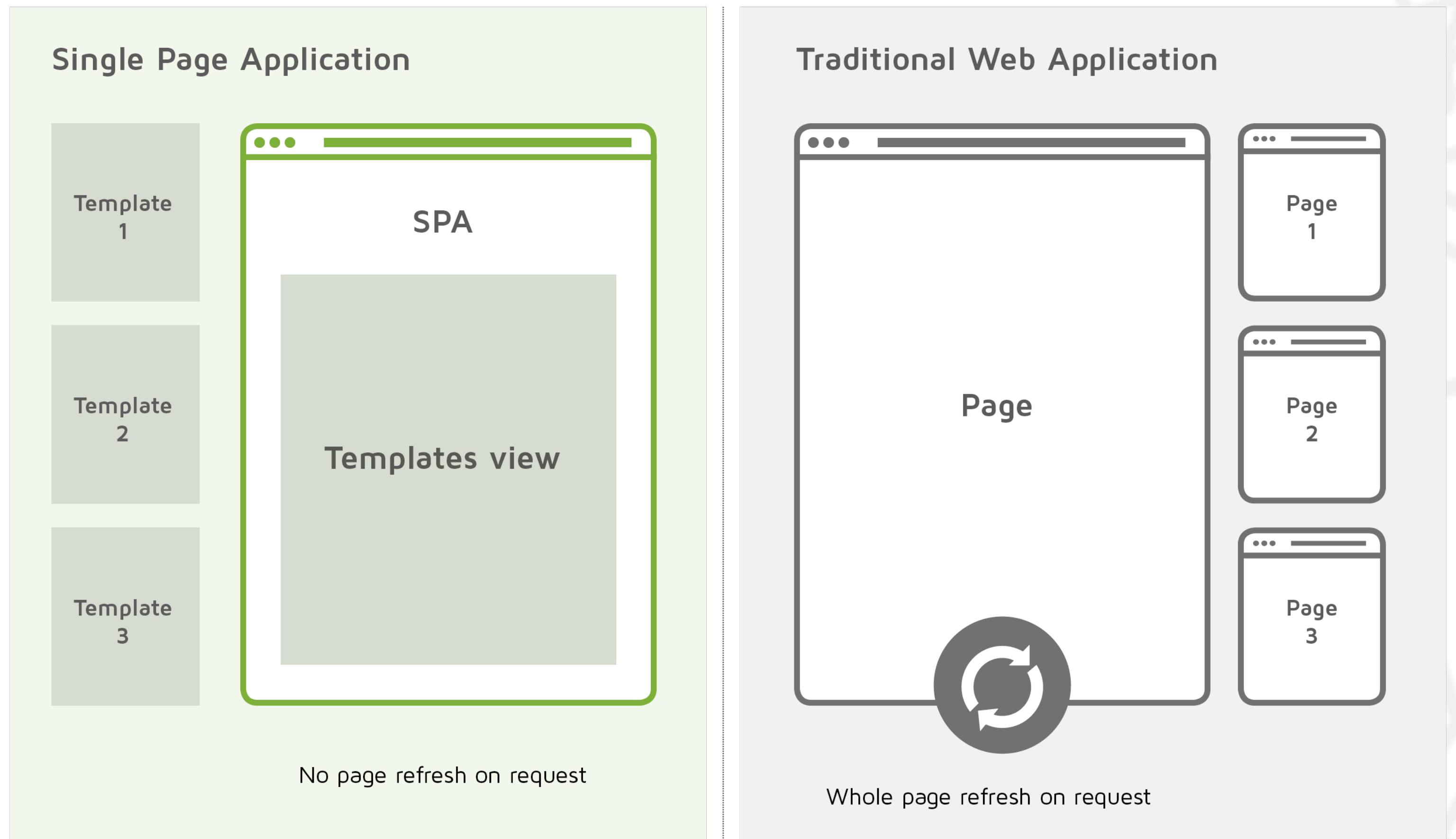
Arquitetura Single Page Application (SPA)

Samuel Martins

Arquitetura Single Page Application

- Em tradução livre: *aplicações de uma única página*;
- Aplicações que interagem com o usuário reescrevendo o conteúdo da página atual, sem a necessidade de recarregar a página;
- Sensação de fluidez, uma vez que apenas o conteúdo principal é trocado nas mudanças de páginas

Arquitetura Single Page Application



Fonte: <https://www.digitalclaritygroup.com/single-page-application-make-sense/>



PUC Minas
Virtual