DIO
CEZAR
GO

Ecossistema JavaScript

AGENDA

- web components
- virtual dom
- qual devo escolher?
- ReactJS
- Angular2
- Aurelia
- VUE.js

- gerenciadores de pacotes
- bundlers
- transpilers
- plugins para JQuery
- escrevendo seu plugin para JQuery
- exemplo ao vivo
- escrevendo seus próprios componentes

WEB COMPONENTS

- é a capacidade de criar custom tags html que encapsulam estrutura (html), estilo (css) e comportamento (javascript).
- entenda como trechos de html reaproveitáveis;
- é algo oficial e padronizado pela W3C;
  - não é algo simples;
- algumas ferramentas facilitam a sua utilização, outras usam o conceito mas sem se preocupar com a especificação;

VIRTUAL DOM

- uma constatação → manipular elementos do DOM tem muito custo para o navegador;
- o que é virtual dom? → é apenas uma representação em javascript puro (memória) do DOM “real”;
- chamada de v-dom;
- com v-dom você passa a manipular objetos JS ao invés do DOM original;
- quando o objeto v-dom é atualizado um algoritmo calcula a diferença entre o v-dom e o DOM real, alterando então pedaços de DOM;

VIRTUAL DOM

- mas o por que manipular DOM é lento?
- sempre foi de conhecimento comum que é mais produtivo você criar os elementos DOM no JavaScript, processar eles e “aplicar eles de uma vez” na árvore DOM do navegador.
    - reactjs veio facilitar isso.

QUAL DEVO ESCOLHER?

- tudo depende da sua aplicação!
  - mas na verdade... não importa!
- problema? interação com o DOM:
  - mostrar os resultados de uma busca no banco de dados, feita com AJAX;
    - precisa-se de uma forma organizada e otimizada de apresentação;
    - cabe a você analisar o seu projeto e entender qual *framework* é o melhor para a sua aplicação;

QUAL DEVO ESCOLHER?

- situação 1: criação de um *HotSite* no estilo *one page* com apenas um formulário de contato.
  - O que escolher?
  - JQuery resolve;
- situação 2: aplicativo com login e senha e várias interfaces que vão representar as views da sua aplicação.
  - O que escolher?
  - Vue? React? Angular? Aurelia? ou... JQuery mesmo?
- https://goo.gl/XXnafg → Entenda de uma vez por todas o que é React.JS, Angular 2, Aurelia e Vue.JS

REACT

- foi o primeiro a popularizar-se;
- react é uma ferramenta somente para: criar componentes;
- criada pela equipe do Instagram;
- é uma biblioteca para criar interfaces;
- JSX é o herói e o vilão da história → **JSX** é uma especificação de sintaxe para escrever JavaScript como se estivéssemos escrevendo XML;

EXEMPLO REACT JSX

EXEMPLO DE UTILIZAÇÃO DE JSX

```
var Hello = React.createClass({
  render: function() {
    return <div>Hello {this.props.name}</div>;
  }
});

ReactDOM.render(
  <Hello name="World" />,
  document.getElementById('container')
);
```

REACT

- não é suficiente (sozinho) para compor todo um grande projeto;
- a palavra chave é: componentes;
- micro libs *JavaScript* → é a base do conceito do react; vários pequenos componentes que compõem um componente maior;

ANGULAR 2

- angular 1 → projetado para ser uma lib para validação de formulário;
- é um dos frameworks mais utilizados no mundo;
- antecessor ao React, que não estava pronto para os *webcomponentes*;
- por isso precisava ser refeito do zero!
- seus códigos, filosofia e linguagem, mudaram completamente;
- *typescript* é sua linguagem de desenvolvimento;
- possui foco na orientação a objetos clássica;
- adotado por desenvolvedores back-end pela grande similaridade do ambiente que estão acostumados a trabalhar;

ANGULAR 2

- projetado para ser uma lib para validação de formulário;
- é um dos frameworks mais utilizados no mundo;
- antecessor ao React, que não estava pronto para os *webcomponentes*;
- por isso precisava ser refeito do zero!
- seus códigos, filosofia e linguagem, mudaram completamente;
- *typescript* é sua linguagem de desenvolvimento;
- possui foco na orientação a objetos clássica;

ANGULAR 2

- adotado por desenvolvedores back-end pela grande similaridade do ambiente que estão acostumados a trabalhar;
- ainda em processo de aperfeiçoamento;
- full stack platform: indica que ele pretende atuar em todas as áreas; não (somente) no Front-End;

AURELIA

- objetivo: tornar sua aplicação o mais “pura” possível com o passar do tempo;
- quem trabalha com JavaScript já está acostumado com uma baita sopa de palavras: Babel, webpack, browserify, npm, ES6, ES7, ES-NEXT...
- tudo de mais novo e moderno → transpiladores de código;
  - babel → transforma um código ES6 compatível com ES5;
  - https://goo.gl/r9edJp
- ele possui fortes influências do Angular;

VUE.JS <3

- utiliza o conceito de virtual dom;
- é bastante semelhante ao ReactJS;
- permite a criação e reutilização de componentes;
- utiliza a linguagem HTML que já se está acostumado com a inserção de variáveis;
- vários plugins e customizações;
- sendo uma lib, não faz todo o trabalho sozinho e assim como react precisa de ajuda!
    - outros *plugins* que incrementam suas funcionalidades;

VUE.JS <3

- possui uma baixa curva de aprendizagem;
- pode ser escrito 100% em JavaScript Vanilla;
- possui um recurso (interessante?) single file components:
  - você escreve HTML + JS + CSS em um único arquivo;
- curiosidade: seu desenvolvedor vive de patrocínio para dedicação exclusiva ao projeto: https://www.patreon.com/evanyou

VUE.JS <3

HELLO WORLD!

```
<script src="https://unpkg.com/vue"></script>

<div id="app">
  <p>{{ message }}</p>
</div>

<script>
new Vue({
  el: '#app',
  data: {
    message: 'Hello Vue.js!'
  }
})
</script>
```

VUE.JS <3

- guia de introdução ao vue.js → https://vuejs.org/v2/guide/
- vimos que possui declaração reativa;
- como sabemos? vamos abrir o console e alterar o message;
- todas diretivas começam com o prefixo v-
- v-bind → é uma diretiva para atribuição do valor do HTML;
- v-if é uma diretiva condicional para manipular a estrutura DOM (show/hide);
- v-for é uma diretiva que permite exibir uma lista de ítens, um laço;
- v-on é uma diretiva para atribuir eventos;
- v-model é uma diretiva de mão dupla entre um input e o app;

VUE.JS <3

altera o title do span

ATRIBUTOS ESPECÍFICOS

```
<div id="app-2">
  <span v-bind:title="message">
    Hover your mouse over me for a few seconds
    to see my dynamically bound title!
  </span>
</div>

var app2 = new Vue({
  el: '#app-2',
  data: {
    message: 'You loaded this page on ' + new
Date().toLocaleString()
  }
})
```

VUE.JS <3

DIRETIVA V-IF

```
<div id="app-3">
  <p v-if="seen">Now you see me</p>
</div>

var app3 = new Vue({
  el: '#app-3',
  data: {
    seen: true
  }
})
```

app3.seen = false

VUE.JS <3

DIRETIVA V-FOR

```
<div id="app-4">
  <ol>
    <li v-for="todo in todos">
      {{ todo.text }}
    </li>
  </ol>
</div>

var app4 = new Vue({
  el: '#app-4',
  data: {
    todos: [
      { text: 'Learn JavaScript' },
      { text: 'Learn Vue' },
      { text: 'Build something awesome' }
    ]
  }
})
```

app4.todos.push({ text: 'New' })

VUE.JS <3

DIRETIVA V-ON

```
<div id="app-5">
  <p>{{ message }}</p>
  <button v-on:click="reverseMessage">Reverse Message</button>
</div>

var app5 = new Vue({
  el: '#app-5',
  data: {
    message: 'Hello Vue.js!'
  },
  methods: {
    reverseMessage: function () {
      this.message = this.message.split('').reverse().join('')
    }
  }
})
```

VUE.JS <3

DIRETIVA V-MODEL

```
<div id="app-6">
  <p>{{ message }}</p>
  <input v-model="message">
</div>

var app6 = new Vue({
  el: '#app-6',
  data: {
    message: 'Hello Vue!'
  }
})
```

VUE.JS <3

- como dissemos, precisamos de componentes!
- é um importante conceito do VUE.js pois permite a construção de aplicações em larga escala compostas por componentes pequenos, auto-contidos, e reusáveis;

VUE.JS <3

COMPONENTES NO VUE.JS

```
// Define a new component called todo-item
Vue.component('todo-item', {
  template: '<li>This is a todo</li>'
})

<ol>
  <!-- Create an instance of the todo-item component -->
  <todo-item></todo-item>
</ol>
```

mas... isso vai renderizar o mesmo texto para cada todo

VUE.JS <3

o elemento todo agora aceita “prop” que é como um atributo customizado essa propriedade foi chamada de todo

COMPONENTES NO VUE.JS

```
<div id="app-7">
  <ol>
    <todo-item
      v-for="item in groceryList"
      v-bind:todo="item"
      v-bind:key="item.id">
    </todo-item>
  </ol>
</div>
```

```
Vue.component('todo-item', {
  props: ['todo'],
  template: '<li>{{ todo.text }}</li>'
})
var app7 = new Vue({
  el: '#app-7',
  data: {
    groceryList: [
      { id: 0, text: 'Vegetables' },
      { id: 1, text: 'Cheese' },
      { id: 2, text: 'Whatever else humans are
supposed to eat' }
    ]
  }
})
```

precisamos passar dados do escopo pai para os nossos componentes

GERENCIADORES DE PACOTES

- é comum você inserir códigos de terceiros em seus projetos;
- esses projetos estão em constante evolução, certo?
- problema → como manter tudo atualizado e organizado?
- solução → utilizando empacotadores de código;
- os empacotadores utilizam estruturas pré-definidas para disponibilizar componentes e plugins para sua aplicação;
- podem ser conhecidos também como: gerenciadores de dependências;

GERENCIADORES DE PACOTES

- um dos mais conhecidos é o **npm**;
- é o empacotador oficial da linguagem *NodeJS*;
- com ele é possível instalar milhares de projetos, não somente em *NodeJS*;
- onde encontrar os projetos?
  - https://www.npmjs.com/
- como instalar o **npm**? ele vem junto com o *NodeJS*, ou... `apt-get install npm`

GERENCIADORES DE PACOTES

- o **npm** gerencia os seus projetos a partir de um arquivo JSON chamado `package.json`
- é no `package.json` onde estão todas as informações do seu projeto, como:
  - nome;
  - versão;
  - descrição;
  - autor;
  - licença;
  - dependências;
  - outros.

GERENCIADORES DE PACOTES

EXEMPLO DE UM PACKAGE.JSON

```
{
  "name": "minha-api",
  "version": "1.0.0",
  "description": "Api de testes",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [
    "testes",
  ],
  "author": "Diogo Cezar",
  "license": "WTFPL"
}
```

GERENCIADORES DE PACOTES

- o objeto de dependências é o mais importante em seu projeto;
- fica armazenado em uma pasta chamada *node_modules*;
- essa pasta nunca deverá ser enviada para seu repositório git;
    - por que? *para economizar espaço*, visto que, sempre que alguém utilizar a estrutura, poderá baixar novamente as dependências utilizando o comando npm install que buscará as informações em *package.json*
    - por isso, não se esqueça de adicioná-la ao .gitignore

GERENCIADORES DE PACOTES

- o comando `npm init` inicia um novo projeto;
- o comando `npm install nome_modulo` serve para instalar uma nova dependência;
- o parâmetro `npm install nome_modulo -g` ou `--global` diz qual o modo de instalação:
  - se uma instalação é local ela será utilizada somente em seu projeto;
  - se uma instalação é global ela será instalada e estará disponível em todo seu sistema, criando um comando para seu sistema;
- deve-se utilizar o parâmetro `--save` para adicionar esta dependência na lista de dependências do arquivo `package.json`

GERENCIADORES DE PACOTES

- caso precise instalar uma versão específica de um pacote, pode-se utilizar: `npm install --save modulo@versão;`
- então possuímos algumas formas diferentes de especificar a versão do nosso módulo, que são:
  - ~versão → equivalente à versão;
  - ^versão → compatível com a versão;
  - versão → precisa ser a versão exata;
  - >versão → precisa ser maior que a versão;
  - >=versão → precisa ser maior ou igual a versão;
  - < versão → precisa ser menor que a versão;
  - <=versão → precisa ser menor ou igual a versão;

GERENCIADORES DE PACOTES

- às vezes será preciso de algumas dependências apenas em modo de desenvolvimento, e não no modo de produção;
- para isso você pode salvar uma dependência específica como dependência de desenvolvimento;
- para isso basta utiliza ao invés do `--save` o `--save-dev`;

GERENCIADORES DE PACOTES

- ou ainda, você poderá instalar uma dependência opcional:
- para isso basta utiliza ao invés do `--save` o `--optional;`

GERENCIADORES DE PACOTES

- o `npm run` é o que executa seu projeto;
- você pode executar *scripts* para automatizar suas tarefas;
- para isso, você precisará configurar a seção scripts do seu package.json:

```
"scripts": {
    "roda": "node script.js"
},
...
```

- e depois, basta executar o comando: `npm run roda` para executar o script em questão;

GERENCIADORES DE PACOTES

- **Bower** é um gerenciador de pacotes para a web;
- como se fosse um **npm**, mas para o desenvolvimento *web*, ao invés de desenvolvimento para o node;
- o propósito de Bower é para gerenciar os pacotes de um front-end, que podem incluir não apenas os arquivos javascript, mas também HTML, CSS, imagens e arquivos de fontes;
- onde encontrar os pacotes?
    - https://bower.io/search/

GERENCIADORES DE PACOTES

- e para instalar o bower, bom... **npm**:
- `npm install bower -global`
- o bower criará toda a estrutura em um objeto separado do *node_modules*;
- seu objeto de dependência é: *bower_components*

BUNDLERS

- os *bundlers* são ferramentas que compilam determinadas funcionalidades não existente nativamente nos navegadores;

BUNDLERS

- **Browserify** é um bundler que nos permite usar o padrão de módulos do NodeJS no navegador;
- nós definimos as dependências e depois o *Browserify* empacota tudo isso em apenas um arquivo JS, limpo e estruturado;
- mais? → https://goo.gl/avnAXQ
- site oficial → http://browserify.org/

BUNDLERS

- **webpack** é um bundler (empacotador) de módulos JavaScript;
- trabalhar com módulos não é possível nas implementações atuais dos navegadores;
- quando você executa o *webpack*, ele lê a árvore de dependência do projeto e faz todos os cálculos dos assets necessários para o seu projeto;
- o que ele nos retorna? um único arquivo que representa todo o projeto e o torna compatível a ser executado pelo navegador;
- o *webpack* é focado em front-end;
- também atua como um automatizador;
- mais? → https://goo.gl/E8vrV8

TRANSPILERS

- de maneira geral, a função de um *transpiler* é traduzir um código escrito em linguagem para um código de outra;
- isso tem a capacidade de facilitar muito determinadas tarefas, pois no lugar de um código extremamente verboso e burocrático você pode usar uma sintaxe mais direta e agradável;
- transpilers também podem trazer novas funcionalidades para uma linguagem, como novos tipos primitivos, novas funções, fluxos... Não há um limite definido para isso.

TRANSPILERS

- podemos citar alguns transpilers:
- CoffeeScript → http://coffeescript.org/
- TypeScript → https://www.typescriptlang.org/
- Flow → https://flow.org/
- Babel → http://babeljs.io/
- JSX → https://facebook.github.io/jsx/

PLUGINS PARA O JQUERY

- mas por que ainda JQuery?
  - a questão não é somente estrutura;
  - efeitos, estética e componentes visuais ainda são bastante familiares com JQuery;
- alguns plugins legais que eu recomendo:
  - NiceScroll - https://goo.gl/2FPcgu
  - ScrollReveal - https://goo.gl/QVGvdm
  - OwlCarousel - https://goo.gl/aBvu6f
  - Modernizer - https://goo.gl/zWVf3a
  - Easing - https://goo.gl/R2ewHd
  - Howler - https://goo.gl/j9mkfD
  - Mousewheel - https://goo.gl/fTYmSW
  - Mais? - https://goo.gl/Zb1GAQ

ESCREVENDO SEU PLUGIN PARA JQUERY

- eventualmente você pode precisar escrever seus próprios Plugins;
- como?
  - https://github.com/diogocezar/dctb-jqplugin
- praticando: vamos criar um plugin?
  - plugin para alterar a cor os elementos;
  - a cor deve ser informada na configuração do plugin;

EXEMPLO AO VIVO

- praticando: vamos criar um plugin?
  - plugin para alterar a cor os elementos;
  - a cor deve ser informada na configuração do plugin;
  - tornar possível: `$(“.my-class”).myPlugin(‘red’);`

ESCREVENDO SEUS PRÓPRIOS COMPONENTES

- ou criar seus próprios componentes
- como?
    - https://github.com/diogocezar/dctb-bootstrap-model

MATERIAIS COMPLEMENTARES

- https://goo.gl/YcYTnz
- https://goo.gl/rY6pWa
- https://goo.gl/jm3agd
- https://goo.gl/KiMvV2
- https://goo.gl/SZWE37
- https://goo.gl/c91kpg
- https://goo.gl/e3R8Mb
- https://goo.gl/q726Tz