UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web.

Alexandre Cechinel

Florianópolis – Santa Catarina
2017/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web.

Alexandre Cechinel

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis – Santa Catarina
2017/2

Alexandre Cechinel

Avaliação do framework Angular e das bibliotecas React e Knockout para o desenvolvimento do Frontend de aplicações Web.

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação

Banca examinadora:

_____

Prof. Dr. Frank Augusto Siqueira

Orientador

_____

Prof. Dr. Leandro José Komosinski

_____

Prof. Dr. Maurício Floriano Galimberti

À todos que direta ou indiretamente
fizeram parte da minha formação,
o meu muito obrigado.

Resumo

Originalmente pensados para auxiliar no desenvolvimento Web, frameworks JavaScript se tornaram muito populares entre desenvolvedores para construções de aplicações Web na medida em que as páginas Web cresceram e se tornaram aplicações completas ao lado do cliente. No entanto, o número de frameworks JavaScript aumentou rapidamente, o que torna difícil para os profissionais identificar o framework que mais se adequa às suas necessidades e desenvolver novas estruturas que atendam essas necessidades.

O presente trabalho se concentrou em avaliar um framework e duas bibliotecas JavaScript, e também compará-las com JavaScript puro, bem como, identificar as razões que impulsionam os desenvolvedores para a escolha de uma ferramenta *Frontend.* Assim, este estudo, se aprofundou em características importantes para o conhecimento dos programadores, como por exemplo, as diferenças arquiteturais, diferenças de custo, eficiência e implementação, além do desempenho e de identificar o melhor cenário de desenvolvimento em que se encaixam.

Entre os testes apresentados, há diferenças significativas nos resultados encontrados, visto que para uma mesma aplicação obteve-se resultados diferentes entre o framework e as bibliotecas testadas. Foi identificado como a manipulação do DOM influencia no desempenho final da mesma aplicação. A biblioteca React por fazer uso do Virtual DOM teve um desempenho melhor em relação ao AngularJS e Knockout.

Palavras-chave: Web, JavaScript, HTML, CSS, Framework, Biblioteca, Single page applications, Frontend, Angular, React, Knockout.

Abstract

Originally thought to aid Web development, JavaScript frameworks have become very popular among developers for Web application builds as Web pages have grown to become fully client-side applications. However, the number of JavaScript frameworks has increased rapidly, making it difficult for professionals to identify the framework that best fits their needs and to develop new frameworks that address those needs.

The present work focused on evaluating a framework and two JavaScript libraries, as well as comparing them with pure JavaScript, as well as identifying the reasons that motivate developers to choose a Frontend tool. Thus, this study has deepened in features important to programmers' knowledge, such as architectural differences, cost differences, efficiency and implementation, as well as performance and to identify the best development scenario in which they fit.

Among the tests presented, there are significant differences in the results found, since for the same application different results were obtained between the framework and the tested libraries. It was identified how DOM manipulation influences the final performance of the same application. The React library for making use of Virtual DOM performed better than AngularJS and Knockout.

Keywords: Web, JavaScript, HTML, CSS, Framework, Library, Single page applications, Frontend, Angular, React, Knockout.

Lista de figuras

Lista de Tabelas

Lista de Gráficos

Lista de Abreviações e Siglas

WWW – World Wide Web

HTML – HyperText Markup Language

CSS – Cascading Style Sheets

DOM – Document Object Model

JSON – Javascript Object Notation

CERN – Organização Europeia para a Pesquisa Nuclear

AJAX – Javascript e XML Assíncrono

XML – eXtensible Markup Language

UI - User Interface

KO - Knockout

SUMÁRIO

1. Introdução

A World Wide Web - em português, teia mundial, ou apenas Web, foi inventada com o objetivo de prover acesso a um amplo universo de documentos e informações produzidas no laboratório CERN (Organização Européia para a Pesquisa Nuclear) (BERNERS-LEE, 1989). A evolução desta tecnologia permitiu mais do que apenas acesso à informação. Pessoas e empresas usam diariamente a Web como forma de fazer negócios, seja como um simples blog ou uma plataforma completa como serviço. Esse contexto possibilitou o surgimento de novas modalidades de desenvolvimento de software voltadas ao desenvolvimento de aplicações Web (SAMY SILVA, 2010).

Desenvolvimento Web refere-se ao processo de construção e testes do software específico para a Web, com a finalidade de obter-se um conjunto de programas, que satisfazem as funções pretendidas, quer em termos de usabilidade dos usuários ou compatibilidade com outros programas existentes, podendo variar desde simples páginas estáticas a aplicações ricas, comércios eletrônicos ou redes sociais (SAMY SILVA, 2010). Neste contexto, este trabalho identifica alguns problemas existentes na área de desenvolvimento Web e as soluções mais utilizadas.

1.1 Problema

Muitos aspectos relativos ao desenvolvimento Web o tornam diferente do desenvolvimento de um software tradicional. Temos a escalabilidade e a necessidade das mudanças contínuas em seu conteúdo como os dois atributos-chave (GINIGE, 2001). De extrema relevância temos a multidisciplinaridade, e entre as características críticas temos a segurança, a usabilidade e também a manutenabilidade. O início da Web não exigia tal esforço e conhecimento. O hipertexto[1] consistia em HTML[2] codificado e publicado em servidores Web. As aplicações Web (WebApps) se caracterizam por possuir um amplo conjunto de

1Hipertexto é uma maneira de conectar e acessar informações de vários tipos, como uma teia de nós no qual o usuário pode navegar livremente.

2HTML (abreviação para a expressão inglesa HyperText Markup Language, que significa Linguagem de Marcação de Hipertexto) é uma linguagem de marcação utilizada na construção de páginas na Web

conceitos e terminologias associadas, além de possuírem características de aplicações de hipermídia[3] (PRESSMAN, 2006).

Esse tipo de sistema não exige que a sua execução seja realizada mediante sua prévia instalação local, assim como é feito com aplicações *desktop*. Muito pelo contrário, a execução da grande maioria de aplicações Web acontece somente com o uso de um navegador Web como, por exemplo, o Mozilla Firefox. Aplicações Web são sistemas que possuem um alto grau de interação (KERER e KIRDA, 2001), além de atender simultaneamente diversos usuários, distribuídos em locais distintos fisicamente com a necessidade de disponibilização contínua e rápida de tais aplicações (HENDRICKSON e FOWLER, 2002).

Hoje, no processo de produção para uma aplicação Web, é necessário o esforço integrado de profissionais de várias áreas do saber. À medida que aumentam a complexidade e a sofisticação dos novos sistemas, uma melhor abordagem sistemática, controle de qualidade e garantia são exigidos, e uma melhor metodologia contribui para um desenvolvimento harmônico, estruturado e bem integrado nos aspectos multidisciplinares (PRESSMAN, 2006).

Para (PRESSMAN, 2006), uma aplicação Web pode ser desde uma simples página a um Web Site completo. Esses sistemas são orientados a documentos que contenham páginas estáticas ou dinâmicas, e são centrados na aparência e nas sensações. Tais aplicações se aproveitam do meio e fazem uma interligação dos conteúdos existentes, dando ao usuário final acesso fácil para edição e criação do mesmo, sustentados por ferramentas amplamente difundidas.

Para o desenvolvimento do Front End[4] de aplicações Web existe uma série de ferramentas disponíveis para os desenvolvedores usarem e otimizarem seu tempo de trabalho. Para CSS existem pré-compiladores como SASS[5], LESS[6] e Stylus[7] que ajudam bastante na reutilização de CSS. Para HTML, uma série de

3Hipermídia é a reunião de várias mídias num ambiente computacional, suportada por sistemas eletrônicos de comunicação
4Front End é responsável por coletar a entrada do usuário em várias formas e processá-la para adequá-la a uma especificação. De forma mais genérica, é a interface visual de uma aplicação Web.
5Sass é uma extensão de CSS que adiciona novas possibilidades e elegância à linguagem base. O objectivo final de Sass é corrigir as falhas de CSS.
6LESS estende CSS com comportamento dinâmico como variáveis, mixins, operações e funções.
7Stylus é uma linguagem de estilo de estilos dinâmico compilada em folhas de estilo em cascata (CSS).

frameworks como Bootstrap[8] e Skeleton[9] ajudam desenvolvedores a criar aplicações Web responsivas sem a necessidade de escrever muitas linhas com CSS. Há ferramentas de automatização de tarefas como Gulp[10] e Grunt[11], gerenciamento de dependência como Browserify e RequireJS, gerenciamento de pacotes como o Bower, Yeoman, NPM, Yard, entre outros.

JavaScript foi introduzido em 1995, criado por Brendan Eich, como uma maneira de adicionar programas para páginas da web no navegador Netscape Navigator. A linguagem já foi adaptada por todos os outros principais navegadores. O JavaScript fez a atual geração de aplicativos baseados na web possível em navegadores de e-mail, mapas e redes sociais, e também é usado em locais mais tradicionais por fornecer diversas formas de interatividade (HAVERBEK, 2014).

Um framework JavaScript é uma estrutura de aplicativo Web escrito em JavaScript e que difere de uma biblioteca JavaScript. Bibliotecas JavaScript oferecem uma série de funções pré-definidas úteis que você pode “chamar” para melhorar e expandir a sua aplicação. Um framework descreve a estrutura do aplicativo e mostra uma maneira de organizar seu código para tornar seu aplicativo mais flexível e escalável. Frameworks JavaScript são geralmente baseados no padrão de arquitetura MVC, porque eles também são projetados para separar os diferentes aspectos de um aplicativo da Web para melhorar a qualidade do seu código e para tornar o desenvolvimento mais fácil (SESHADRI e GREEN, 2014).

1.2 Objetivo

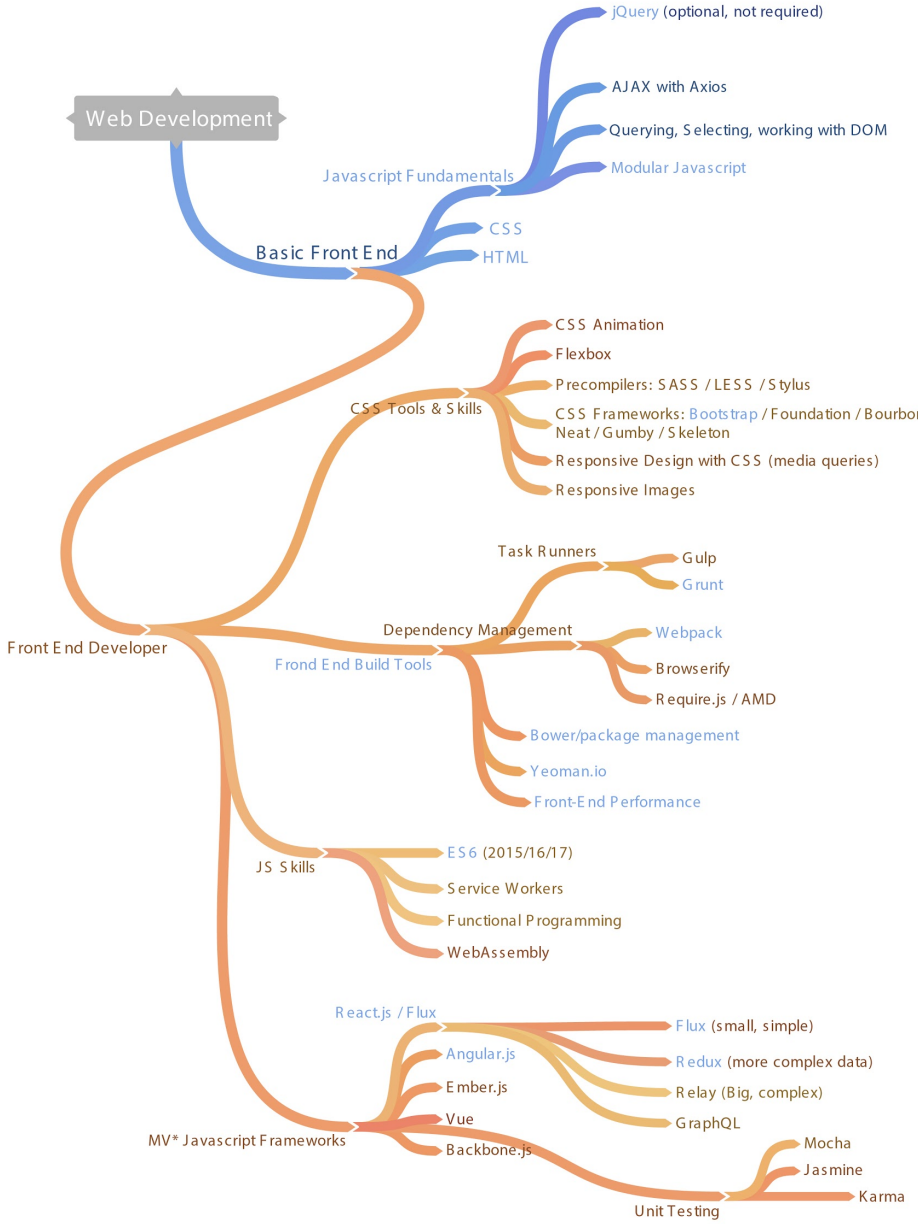
O objetivo deste trabalho consiste em apresentar e avaliar três frameworks ou bibliotecas utilizados atualmente para desenvolvimento de aplicações Web e também compará-los com JavaScript puro (sem a necessidade de um framework ou biblioteca), assim como esclarecer a arquitetura de cada um, custo, eficiência (desempenho, tamanho), diferenças de implementação entre os mesmos, funcionalidade (flexibilidade, isolamento, modularidade), e outros aspectos que cada framework ou biblioteca tem de exclusivo. Para isso será feito uma aplicação de exemplo em que se pode testar os requisitos necessários para avaliação.

8O Bootstrap é uma estrutura web de front-end livre e de código aberto para projetar sites e aplicativos da web.
9Skeleton é uma coleção de arquivos CSS que pode ser muito útil aos desenvolvedores web. Para desenvolver rapidamente sites que se adaptam a qualquer resolução de tela.
10Gulp: gerenciador de pacotes e automatizador de tarefas.
11Grunt: gerenciador de pacotes e automatizador de tarefas.

1.3 Justificativa

A área de desenvolvimento Web chamada de Front End possui muitas ferramentas e fatores influenciam a vida dos desenvolvedores. As diferenças de ferramentas e conhecimentos necessários entre uma aplicação básica como um Web site simples e uma mais completa como um software como serviço em ambiente Web, podem ser vistos na Figura 1.

Figura 1: Diferenças entre desenvolvimento web básico e complexo (editado).



Fonte: Disponível em: <https://coggle.it/diagram/52e97f8c5a143de239005d1b/56212c4e4c505e0045c0d3bda59b77e5977c2c9bd40f3fd0b451bdcf8da4aa52>

Pode-se notar na Figura 1 uma grande quantidade de ferramentas disponíveis para o desenvolvedor Web.

De forma sucinta, um framework é um conjunto de código que serve para facilitar o desenvolvimento de funcionalidades genéricas, possibilitando que o desenvolvedor mantenha o foco em deveres mais complexos e que realmente necessitam de mais atenção. Quem acompanha o desenvolvimento Web nos

últimos anos, deve ter observado uma grande quantidade de frameworks e bibliotecas surgindo no mercado como opção de desenvolvimento.

No artigo de PANO, Amantia , GRAZIOTIN, Daniel e ABRAHAMSSON, Pekka. “*What leads developers towards the choice of a JavaScript framework?”* Os autores exploram a crescente popularidade dos frameworks que visam ajudar no desenvolvimento Web e as dificuldades dos desenvolvedores em escolher qual framework é o mais propício para o seu trabalho.

O número de Frameworks JavaScript aumentou rapidamente de muitos para milhares. É difícil para os profissionais identificar os frameworks que melhor se encaixam às suas necessidades, e desenvolver novos aplicações que se ajustem a essas necessidades (PANO, GRAZIOTIN, e ABRAHAMSSON, 2016).

Este TCC tem como foco os frameworks e bibliotecas que os desenvolvedores de Front End utilizam diariamente como ferramenta de trabalho, trazendo como forma de auxílio os resultados encontrados sobre os mesmos avaliados.

1.4 Metodologia de trabalho

No primeiro momento foi realizado um estudo sobre as tecnologias que fundamentam as aplicações Web e auxiliam o desenvolvedor. Dentro deste contexto, fez-se um estudo sobre os padrões de design JavaScript usados pelos frameworks e bibliotecas.

Para atingir o objetivo do trabalho são exploradas as diferenças entre o framework e biblioteca aplicados a uma aplicação de exemplo de igual aparência, sendo que a estrutura adotada deve estar de acordo com o estilo arquitetural de cada framework e biblioteca aplicada. Então será esclarecido a usabilidade (atratividade, capacidade de aprendizado, compreensão), custo, eficiência (desempenho, tamanho), funcionalidade (automatização, extensibilidade, flexibilidade, isolamento, modularidade, adequação atualização), e outros aspectos que cada framework exclusivo.

1.5 Organização do texto

Este trabalho está organizado em mais 5 capítulos. O capítulo 2 visa apresentar os conceitos fundamentais explorados neste projeto, dando uma visão

geral das tecnologias utilizadas no desenvolvimento Web e que fundamentam a Web.

O capítulo 3 descreve o framework e as bibliotecas JavaScript utilizadas para o desenvolvimento da aplicação de exemplo. Também apresenta suas respectivas arquiteturas de forma mais aprofundada, além dos motivos da escolha deles para a aplicação. O capítulo 4 descreve o como é a aplicação de exemplo e as características de cada exemplo. O capítulo 5 apresenta os resultados encontrados. Finalmente, o capítulo 6 expõe as conclusões da avaliação e apresenta os possíveis trabalhos futuros derivados deste estudo.

2. Conceitos Fundamentais

Este capítulo descreve os conceitos básicos necessários para entendimento do presente trabalho.

2.1 Origem da Web

Em 1989, um pequeno grupo de pessoas liderado por Tim Berners-Lee no CERN (Organização Europeia para a Pesquisa Nuclear) propôs um novo protocolo para a internet, bem como um sistema de documentos de acesso para usá-lo. A intenção desse novo sistema, que o grupo chamou de *World Wide Web*, era permitir que cientistas do mundo todo usassem a internet para trocar documentos descrevendo seus trabalhos (SEBESTA, 2012).

O novo sistema proposto foi projetado para permitir que um usuário, em qualquer lugar, conectado à internet, possa pesquisar e recuperar documentos em qualquer número de diferentes computadores conectados à internet (SEBESTA, 2012).

A Web é um sistema de informação, baseado em documentos de hipertexto interligados, construído sobre a infraestrutura de comunicação provida pela Internet. Servidores gerenciam a maioria dos processos e armazenam dados. Um cliente solicita dados ou processos específicos. A saída do processo acontece do servidor para o cliente. Os clientes, por vezes, lidam com o processamento, mas exigem recursos de dados do servidor para a conclusão (SEBESTA, 2012).

2.2 Modelo Cliente-Servidor

Documentos fornecidos pelos servidores na web são solicitados pelos navegadores. O navegador é um cliente na Web porque ele inicia a comunicação com o servidor, que aguarda um pedido do cliente antes de fazer qualquer coisa. No caso mais simples, um navegador faz o pedido de um documento estático ao servidor. O servidor localiza o documento entre os seus documentos veiculáveis e envia para o navegador, que exibe para o usuário. No entanto situações mais complicadas são comuns. Por exemplo, o servidor pode fornecer um documento que solicita a entrada de algum dado do usuário no navegador. Depois que o usuário fornece o dado necessário, é transmitido a partir do navegador para o servidor, o qual pode usar a entrada para executar alguma computação e, em seguida, voltar um novo documento para o navegador (SEBESTA, 2012).

Embora a web possa suportar uma variedade de protocolos, o mais comum é o Protocolo de Transferência de Hipertexto (HTTP). O HTTP fornece uma forma padrão de comunicação entre o navegador e os servidores Web.

2.3 HTTP

HTTP consiste em duas fases: a requisição e a resposta. Cada comunicação HTTP (requisição e resposta) entre o navegador e o servidor consiste em duas partes: o cabeçalho e o corpo. O cabeçalho contém informações sobre a comunicação; o corpo contém os dados da comunicação, se houver algum.

2.3.1 Fase requisição

A forma geral de um pedido HTTP é como se segue:

1. Método HTTP; Parte do domínio da URL; Versão HTTP
2. Campos de cabeçalho
3. Linha em branco
4. Corpo da mensagem

Segue um exemplo da primeira linha de uma requisição HTTP:

GET /user HTTP/1.1

Apenas alguns métodos de solicitação são definidos por HTTP, e ainda um número menor destes são tipicamente usados. A Figura 2 lista os métodos mais comumente usados.

Figura 2: Métodos de requisição HTTP.

Método	Descrição
GET	Retorna o conteúdo do documento especificado
HEAD	Retorna a informação do cabeçalho para o documento específico
POST	Executa o documento especificado, usando os dados fechados
PUT	Substitui o documento especificado com os dados fechados
DELETE	Deleta o documento especificado

Fonte: Repositório do autor.

2.3.2 Fase resposta

A forma geral de uma resposta HTTP é como se segue:

1. Status
2. Campos resposta do cabeçalho

3. Linha em branco
4. Corpo da resposta

A linha de status inclui a versão HTTP usada, um código de status de três dígitos para a resposta, e uma breve explicação textual do código de status. Por exemplo, a maioria das respostas começa com:

HTTP/1.1 200 OK

Os códigos dos status começam com 1, 2, 3, 4 ou 5. Os significados gerais das cinco categorias especificadas por estes primeiros dígitos são apresentados na Figura 3.

Figura 3: Primeiros dígitos do código de status HTTP.

Primeiro Dígito	Categoria
1	Informacional
2	Sucesso
3	Redirecionamento
4	Erro de cliente
5	Erro de servidor

Fonte: Repositório do autor.

2.4 Web 2.0

A Web 2.0 geralmente refere-se a um conjunto de políticas sociais, arquiteturais e padrões de design, resultando em uma migração em massa de negócios para a internet como uma plataforma. Esses padrões focam na interação de modelos entre comunidades, pessoas, computadores e software. Interações Humanas são um importante aspecto da arquitetura do software e, mais especificamente, do conjunto de sites e aplicações baseadas na Web construídas em torno do núcleo de um conjunto de padrões de design que mistura a experiência humana com a tecnologia (GOVERNOR, HINCHCLIFFE, NICKULL, 2009).

O termo Web 2.0 é usado para denotar vários conceitos diferentes: sites com base em um determinado conjunto de tecnologias, como AJAX[12]; Web sites que incorporam um forte componente social, envolvendo os perfis de utilizador e

12Ajax é o uso metodológico de tecnologias como Javascript e XML, providas por navegadores, para tornar páginas Web mais interativas com o usuário, utilizando-se de solicitações assíncronas de informações.

ligações de amizade; Web sites que incentivam conteúdo gerado pelo usuário na forma de texto, vídeo e postagens de fotos, juntamente com comentários, tags e classificações; ou apenas sites que ganharam popularidade nos últimos anos e estão sujeitos a especulações febris sobre valorizações e oferta de ações em bolsas de valores (CORMODE, KRISHNAMURTHY, 2008).

A diferença essencial entre Web 1.0 e Web 2.0 é que os criadores de conteúdo eram poucos na Web 1.0, com a grande maioria dos usuários simplesmente agindo como consumidores de conteúdo, enquanto que qualquer participante pode ser um criador de conteúdo na Web 2.0. Numerosas ajudas tecnológicas têm sido criadas para maximizar o potencial de criação de conteúdo. A natureza democrática da Web 2.0 é exemplificada pela criação de um grande número de grupos de nicho (coleções de amigos) que podem trocar o conteúdo de qualquer tipo (texto, áudio, vídeo), *tags*, comentários e links. Uma inovação popular na web 2.0 são os "*mashups*[13]", que combinam ou tornam o conteúdo em novas formas. Por exemplo, endereços de rua presentes em uma propaganda de banco de dados confidenciais estão ligados com um mapa em um Web site para visualizar os locais. Tal ligação entre sites capta o conceito genérico de criar ligações adicionais entre registros de qualquer banco de dados semi-estruturado com outro banco de dados (CORMODE, KRISHNAMURTHY, 2008).

As aplicações “Web 2.0” têm algumas características importantes que as diferenciam :

- São auto-contidas e tem um objetivo principal (muitas vezes fazem uso de uma técnica chamada *Single Page Application*);
- Elas usam muito do HTML5 para ter a sensação de ser uma aplicação nativa (ou *desktop*);
- São desenvolvidas pensando-se que, ao ficar *offline*, o usuário possa continuar a operar a aplicação (como numa aplicativo *desktop*);
- Elas reconhecem o dispositivo em que estão sendo executadas (celular, *desktop*, *laptop*, etc);
- Apresentam uma performance bem maior que as aplicações Web antigas e seu uso é muito mais agradável;

13Um mashup é um site personalizado ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo

- O cliente (navegador) se comunica de forma assíncrona com o servidor e muitas vezes o servidor envia dados sem haver uma requisição (*push* de conteúdo) usando tecnologias como AJAX, WebSockets ou Comet;
- Apresentam um estilo de navegação sem *links* e elementos de navegação tradicionais da Web aparentes;

As *Single page applications* (SPAs) são aplicações completas, desenvolvidas em JavaScript, que funcionam quase como se estivessem sendo executadas nativamente no *desktop*. O Google foi pioneiro nesta tecnologia e o mundo o seguiu. Atualmente, a maior parte das aplicações Web 2.0 usa este modelo: o GMail, a busca do Google, o Google Drive, Facebook, o Twitter, o FourSquare, o Instagram, blogs, sites corporativos, dentre outros.

2.5 JavaScript

Quando o JavaScript apareceu pela primeira vez em 1995, seu objetivo principal era manipular algumas das validações de entrada que haviam sido deixadas anteriormente para linguagens do lado do servidor, como o Perl. Era necessária uma chamada ao servidor para determinar se um campo necessário havia sido deixado em branco ou se um valor inserido era inválido. O navegador Netscape procurou alterar isso com a introdução do JavaScript. A capacidade de lidar com alguma validação básica no cliente foi um novo recurso emocionante em um momento em que o uso de *modens* telefônicos era generalizado. As velocidades lentas associadas transformaram cada chamada ao servidor em um exercício de paciência (ZAKAS, 2012).

Desde aquela época, o JavaScript se tornou uma característica importantes de todos os principais navegadores do mercado. Hoje, o JavaScript já não está vinculado somente para validação de dados simples, agora interage com quase todos os aspectos da janela do navegador e seus conteúdos. O JavaScript é reconhecido como uma linguagem de programação completa, capaz de cálculos e interações complexas, incluindo *closures*[14], funções anônimas (lambda) e até mesmo *metaprogramação*[15]. O JavaScript tornou-se uma parte tão importante da

14Closures (fechamentos) são funções que se referem a variáveis livres (independentes). Em outras palavras, a função definida no closure "lembra" do ambiente em que ela foi criada.
15Metaprogramação é toda programação que atua sobre outro programa, seja em formato fonte, binário, ou numa representação abstrata em memória.

Web que até navegadores alternativos, incluindo aqueles em telefones celulares e aqueles projetados para usuários com deficiência o suportam. Mesmo a Microsoft, com sua linguagem de script do lado cliente chamada VBScript, acabou incluindo sua própria implementação de JavaScript no Internet Explorer desde sua versão mais antiga (ZAKAS, 2012).

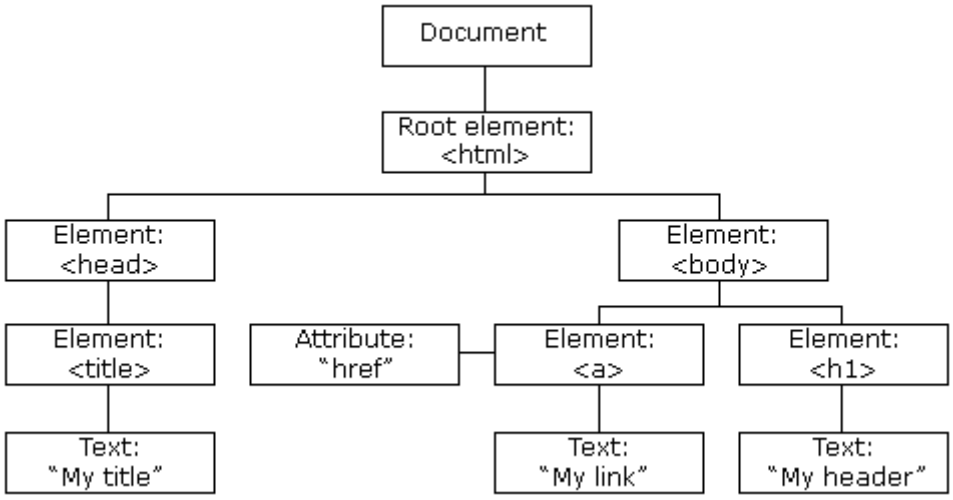
A ascenção do JavaScript de um validador de entrada simples para uma poderosa linguagem de programação não poderia ter sido prevista. JavaScript é ao mesmo tempo uma linguagem muito simples e muito complicada que leva minutos para aprender, porém anos para dominar (ZAKAS, 2012).

2.6 DOM

O DOM (*Document Object Model*) é uma convenção multiplataforma e independente de linguagem para representar e interagir com objetos em HTML, XHTML, e documentos XML. Os nós de todos os documentos são organizados em uma estrutura de árvore, chamada de árvore DOM, como representada na figura 4. Objetos na árvore DOM podem ser endereçados e manipulados utilizando métodos nos objetos. A interface pública de um DOM é especificada na sua API (*Application Programming Interface*) (SAMY SILVA, 2010).

A história do DOM está intimamente ligada com o início das linguagens de *script*. Após o lançamento do ECMAScript (explicado no capítulo sobre JavaScript) a W3C (*World Wide Web Consortium*) começou a trabalhar na padronização do DOM. O padrão inicial, conhecido como "DOM Nível 1", foi recomendado no final de 1998 (W3C, 1998). Na mesma época, o Internet Explorer 5.0 foi entregue com suporte limitado para DOM Nível 1. DOM Nível 1 proveu desde um modelo completo para um documento HTML como para um XML, incluindo meios para modificar qualquer parte deste. Navegadores não conformes, tais como Internet Explorer e Netscape 4.x, ainda foram amplamente utilizados até o ano 2000. DOM Nível 2, publicado no final de 2000 (W3C, 2000), introduziu a função "getElementById", bem como um modelo de evento e suporte para namespaces XML e CSS. O DOM Nível 3, a versão atual da especificação do DOM (W3C, 2004), adicionou suporte para XPath e manipulação de eventos de teclado, bem como uma interface para a serialização de documentos XML (SAMY SILVA, 2010).

Figura 4: Árvore de objetos DOM do HTML.



Fonte: Disponível em <https://developer.mozilla.org/pt-BR/docs/DOM/Referencia_do_DOM>

2.7 HTML

O HTML é uma linguagem de marcação de hipertexto. Hipertextos são conjuntos de elementos interligados através de *hyperlinks*[16], podendo esses elementos ser palavras, imagens, vídeos, áudio, documentos, etc., que conectados formam uma grande rede de informação. Propõe-se a ser uma linguagem que seja entendida universalmente no sentido de que a informação publicada por meio deste código possa ser acessível por dispositivos e outros meios com características diferentes. Fornece um meio para criar documentos estruturados, denotando uma semântica estrutural para o texto, como cabeçalhos, parágrafos, listas, links, citações e outros itens. É possível incorporar *scripts*, em linguagens como JavaScript, que podem modificar o comportamento das páginas Web. Uma página também pode se referir a estilos definidos em arquivos CSSs para definir a aparência e o *layout* de texto e outros elementos (SAMY SILVA, 2010).

A primeira descrição disponível publicamente do HTML foi um documento chamado de "*tags* HTML", mencionado na Internet por Berners-Lee, no final de 1991

16Nome que se dá às imagens ou palavras que dão acesso a outros conteúdos em um documento hipertexto. O hyperlink pode levar a outra parte do mesmo documento ou a outros documentos.

(BERNERS-LEE, 1991). A versão atual, HTML5 (W3C, 2014) teve como principal objetivo facilitar a manipulação dos elementos HTML, permitindo o desenvolvedor modificar as características dos objetos de forma não intrusiva e de maneira que seja transparente ao usuário final. Ao contrário das versões anteriores, o HTML5 fornece ferramentas para a CSS e o JavaScript fazerem seu trabalho da melhor maneira possível. O HTML5 também cria novas *tags* e modifica a função de outras. As versões antigas do HTML não continham um padrão universal para a criação de seções comuns e específicas como rodapé, cabeçalho, sidebar, menus. Não havia um padrão de nomenclatura de IDs, classes ou *tags*. Tão pouco um método para capturar de maneira automática as informações localizadas nos rodapés de sites. Há outros elementos e atributos que tiveram sua função e significado modificados e que agora podem ser reutilizados de forma mais eficaz. Propõe mais semântica com menos código, mais interatividade sem a necessidade de instalação de *plugins* e perda de desempenho (SAMY SILVA, 2010).

2.8 CSS

CSS é uma linguagem de folha de estilo utilizada para descrever a aparência e a formatação de um documento escrito em uma linguagem de marcação. Foi proposta por Hakon Wium Lie, em 10 de outubro de 1994 (HAKON, 1994). Na época, Lie trabalhava com Berners-Lee no CERN. Embora na maioria das vezes utilizada para alterar o estilo de páginas Web e interfaces de utilizador escritas em HTML e XHTML, a linguagem pode ser aplicada a qualquer tipo de documento XML. Junto com HTML e JavaScript, CSS é uma tecnologia fundamental, utilizada pela maioria dos sites para criar páginas Web visualmente atraentes e interfaces de usuário para muitas aplicações móveis (SAMY SILVA, 2010).

Os estilos podem ser incorporados no documento HTML de três maneiras diferentes: estilo no documento, declarado no cabeçalho, quer dizer, entre as *tags* <head> e </head>; em linha, como atributo das *tags*; e externo, declarado num arquivo à parte cuja extensão é .css (SAMY SILVA, 2010).

Com a popularidade do HTML, passou-se a abranger uma ampla variedade de recursos estilísticos para atender às demandas dos desenvolvedores Web. Esta evolução deu ao *designer* mais controle sobre a aparência do site, ao custo de um HTML mais complexo. Variações em implementações de navegadores tornaram

difícil a consistência na aparência de sites, fazendo códigos *cross-browser* proliferarem (SAMY SILVA, 2010).

2.9 Padrões de design Javascript MV*

No passado, os padrões de design - MVC (*Model-View-Controller*), MVP (*Model-View-Presenter*) e MVVM (*Model-View-ViewModel*) foram muito utilizados para estruturar aplicações *desktop* e do lado do servidor, mas só nos últimos anos que vêm sendo aplicadas para Javascript (OSMANI, 2015).

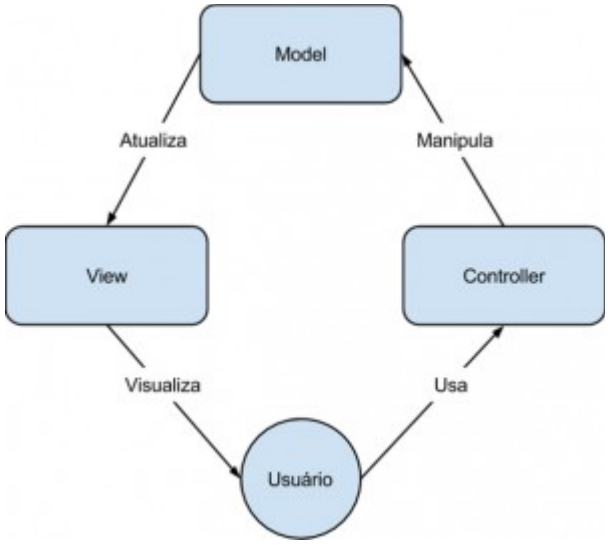
Algumas das vantagens de se usar um destes modelos de desenvolvimento são:

- Facilidade de manutenção;
- Possibilidade de se ter várias *Views* para um modelo de dados;
- Separação bem clara entre interface visual e persistência de dados;
- Isolamento das regras de negócio;
- Possibilidade de alterar a forma com que uma interface para o usuário se comporta apenas alterando o *controller*;

2.9.1 Model-View-Controller (MVC)

MVC é um padrão de projeto arquitetônico que incentiva a organização de uma melhor aplicação através de uma separação de interesses. Ele aplica o isolamento de dados de negócios (*model*) de interfaces de usuário (*view*), com um terceiro componente (*controller*) lógico tradicionalmente de gestão e de entrada do usuário. O padrão foi originalmente concebido por Trygve Reenskaug durante seu tempo trabalhando em Smalltalk-80 (1979), onde foi inicialmente chamado MVC-Editor. MVC passou a ser descrito em profundidade em 1995, no livro Padrões de Projetos - Soluções reutilizáveis de software orientado a objetos, que desempenhou um papel importante na popularização do seu uso (OSMANI, 2015).

Figura 5: Modelo MVC.



Fonte: Disponível em: <https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html>.

Modelos gerenciam os dados para um aplicativo. Eles representam formas únicas de dados que um aplicativo pode exigir. Quando um modelo é alterado (por exemplo, quando ele é atualizado), ele irá normalmente notificar os seus observadores que ocorreu uma mudança para que eles possam reagir em conformidade (OSMANI, 2015).

Uma visão tipicamente observa um modelo e é notificada quando o modelo muda, permitindo se atualizar em conformidade. A literatura padrão comumente refere-se a pontos de vista como "burro", dado que os seu conhecimento de modelos e controladores em uma aplicação é limitado.

Os controladores são um intermediário entre modelos e visão, que são responsáveis por atualizar o modelo quando o usuário manipula a visão.

Esta separação de interesses em MVC facilita a modularização mais simples da funcionalidade do aplicativo e permite:

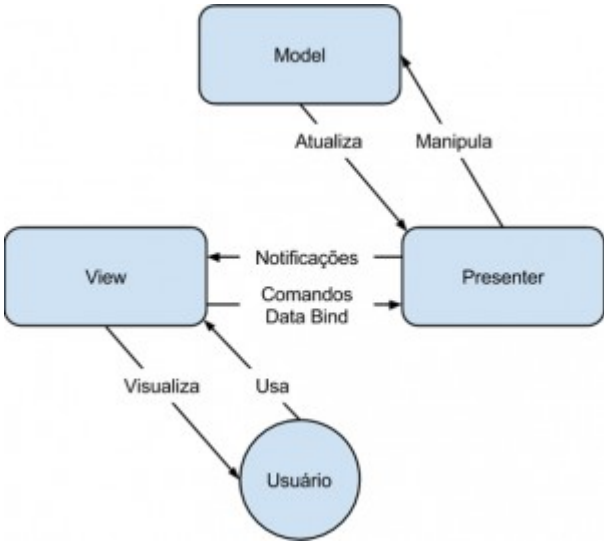
- Mais fácil manutenção geral. Quando as atualizações precisam ser feitas no aplicativo, é muito claro se as mudanças são centradas em dados, ou seja, alterações em modelos e, possivelmente, controladores ou meramente visual, ou seja, alterações nas visualizações;
- A dissociação entre modelos e a visão significa que é significativamente mais simples e direto escrever testes de unidade para lógica de negócios;
- Duplicação de modelo de baixo nível e código do controlador é eliminada da aplicação;
- Dependendo do tamanho do aplicativo e da separação de funções, esta modularidade permite que os desenvolvedores responsáveis pela lógica do núcleo e desenvolvedores que trabalham nas interfaces de usuário possam trabalhar simultaneamente.

2.9.2 Model-View-Presenter[17] (MVP)

Model-View-Presenter (MVP) é um derivado do padrão de projeto MVC, que se concentra em melhorar a lógica de apresentação. Originou-se em uma empresa chamada Taligent no início de 1990, enquanto eles estavam trabalhando em um modelo para um ambiente C++ *CommonPoint*. Embora ambos MVC e MVP concentrem-se na separação de interesses entre vários componentes, existem algumas diferenças fundamentais entre eles (OSMANI, 2015).

17 Para os fins deste trabalho, é apresentada a versão do MVP mais adequado para arquiteturas baseadas na web.

Figura 6: Modelo MVP.



Fonte: Disponível em:<https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html.>

O P em MVP significa apresentador. É um componente que contém a lógica de negócios de interface do usuário para a visão. Ao contrário de MVC, invocações da visualização são delegadas para o apresentador, que está dissociado da visão em vez falar com ele através de uma interface. Isto permite várias coisas úteis, tais como ser capaz de fazer *mock*[18] da visão em testes de unidade (OSMANI, 2015).

A implementação mais comum de MVP é aquele que usa uma exibição passiva (uma visão que é para todos os efeitos "burra"), que contém pouca ou nenhuma lógica. Se MVC e MVP são diferentes é porque o *Controller* e o *Presenter* fazem coisas diferentes. No MVP, o *Presenter* observa modelos e visualizações atualizadas quando os modelos mudam. O *Presenter* efetivamente liga modelos de

18O termo "Mock" é utilizado para descrever um caso especial de objetos que imitam objetos reais para teste. Esses Mock's atualmente podem ser criados através de frameworks que facilitam bastante a sua criação. Praticamente todas as principais linguagens possuem frameworks disponíveis para a criação de Mock. Os Mock's são mais uma forma de objeto de teste.

pontos de visão, uma responsabilidade que era anteriormente detida pelos controladores em MVC (OSMANI, 2015).

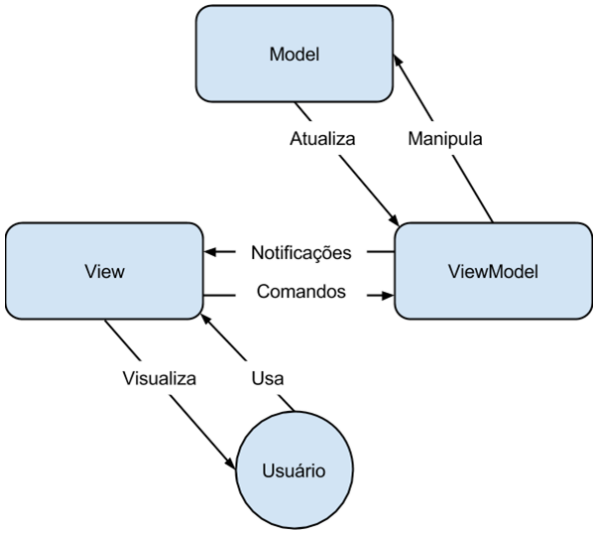
O benefício desta mudança do MVC para MVP é que ela aumenta a capacidade de teste da aplicação e fornece uma separação mais clara entre a visão e o modelo (OSMANI, 2015).

2.9.3 Model-View-ViewModel (MVVM)

MVVM (*Model View ViewModel*) é um padrão de arquitetura baseada em MVC e MVP, que tenta separar mais claramente o desenvolvimento de interfaces de usuário (UI) da lógica de negócios e comportamento em um aplicativo. Para este fim, muitas implementações deste padrão fazem uso de ligações de dados declarativa para permitir uma separação de trabalho em visões de outras camadas (OSMANI, 2015).

Isso facilita a interface do usuário e o trabalho de desenvolvimento que ocorre quase simultaneamente dentro da mesma base de código. Deste modo, desenvolvedores de interface escrevem ligações para o ViewModel dentro de sua marcação de documentos (HTML), enquanto o modelo e o ViewModel são mantidos pelos desenvolvedores que trabalham na lógica do aplicativo (OSMANI, 2015).

Figura 7: Modelo MVVP.



Fonte: Disponível em:<https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html.>

O *ViewModel* pode ser considerado um controlador especializado que atua como um conversor de dados. Ele muda as informações do modelo para

visualização de informações, passando comandos da visão para o modelo (OSMANI, 2015).

3. Frameworks para Desenvolvimento de Aplicações Web

Para efeitos deste trabalho, este capítulo cita um framework e duas bibliotecas escolhidas para o desenvolvimento da aplicação de exemplo e apresenta suas respectivas arquiteturas e a razão da escolha destes frameworks.

São os Seguintes:

Framework AngularJS: O AngularJS trouxe um novo conceito de desenvolvimento frontend e deu início a nova era dos web apps.

Tem uma comunidade sólida, o repositório do Angular no GitHub tem 57 mil estrelas e mais de mil contribuintes. No Stack Overflow, a maior comunidade de perguntas e respostas do mundo, são quase 180 mil perguntas referentes ao AngularJS. No *Youtube* há 470 mil vídeos sobre este framework.

Biblioteca React: React é uma biblioteca javascript para criar as interfaces do Facebook e Instagram, foi criado para resolver o problema: criar grandes aplicações com dados que mudam a cada segundo.

Assim como o Angular, o React também tem uma comunidade grande, com números muito parecidos e expressivos, o que os torna os principais framework e biblioteca para desenvolvimento Web.

Biblioteca Knockout: O Knockout é uma biblioteca JavaScript menor e menos conhecida, porém, por ter uma arquitetura e estrutura de dados bem diferente (que serão mencionados logo mais neste capítulo), foi feita a escolha desta biblioteca para a avaliação neste trabalho.

3.1 Evolução do desenvolvimento de aplicações web

O JavaScript percorreu um longo caminho, partindo de uma simples linguagem de scripting, usada somente para realizar validações menos significativas, até se tornar uma linguagem de programação completa. A JQuery

realizou boa parte do trabalho de base para garantir a compatibilidade com os navegadores, oferecendo uma API sólida e estável para trabalhar com todos eles e interagir com o DOM. À medida que a complexidade e o tamanho das aplicações aumentaram, a JQuery, que é uma camada de manipulação de DOM, tornou-se insuficiente para oferecer, por si só, um framework sólido, modular, testável e de fácil compreensão para o desenvolvimento de aplicações. Cada projeto JQuery era totalmente diferente um do outro (SESHADRI, GREEN, 2014).

Os frameworks MVC, MVP, MVVM atacam exatamente essa questão, que é o problema de disponibilizar uma camada sobre a JQuery e, portanto, sobre o DOM, para poder pensar em termos de estrutura de aplicação e manutenção, ao mesmo tempo que a quantidade de código *boilerplate* (repetido) a ser escrito é reduzida. A melhor parte de usar um framework de maneira consistente é que um novo desenvolvedor terá, de imediato, uma noção da estrutura e do layout do sistema e saberá desenvolver de forma adequada desde o início (SESHADRI, GREEN, 2014).

Alguns dos conceitos que atualmente estão no centro do desenvolvimento de aplicações web são:

- programação orientada a dados, em que o objetivo é manipular o modelo e deixar que o framework faça o trabalho pesado e a renderização da UI;
- programação declarativa, que possibilita declarar a sua intenção ao realizar uma ação, em vez de usar programação imperativa, em que o verdadeiro trabalho é realizado em um arquivo/função separada, e não no local em que o efeito é necessário;
- modularidade e separação de responsabilidades, que é a capacidade de dividir sua aplicação em partes funcionais menores e reutilizáveis, cada qual responsável por uma e somente uma tarefa;
- possibilidade de o sistema ser testado para que possamos garantir que aquilo que nós, como desenvolvedores, implementamos faz o que deveria fazer.

Com a ajuda de frameworks, o desenvolvedor de aplicações Web pode focar em aplicações de alta complexidade, que possam ser bem administradas e mantidas (SESHADRI, GREEN, 2014).

3.2 Angular

O Angular nasceu de uma frustração de engenheiros do Google que passaram meses desenvolvendo o Google Feedback de modo eficiente e que

permitisse fácil manutenção. Depois de 18 mil linhas de códigos, muitas das quais não testadas e com sentimento de incapacidade de continuar adicionando funcionalidades rapidamente, o engenheiro Misko Hevery fez uma afirmação ousada a sua equipe dizendo que era capaz de reproduzir tudo o que haviam desenvolvido nos últimos seis meses em duas semanas (SESHADRI, GREEN, 2014).

Depois de duas semanas assistindo e observando Misko lutar, escorregar e cair, o sistema não estava pronto. Porém, uma semana depois, Misko havia criado uma réplica daquilo que a equipe havia levado seis meses para desenvolver. O que era composto de 18 mil linhas de código havia se reduzido a meras mil e quinhentas linhas, e, praticamente todas as funcionalidades eram modulares, reutilizáveis e testáveis. Misko havia descoberto algo (SESHADRI, GREEN, 2014).

Assim, os engenheiros em conjunto resolveram criar uma equipe em torno da idéia principal de simplificar o desenvolvimento de aplicações Web. O Google Feedback tornou-se o primeiro projeto a ser lançado com o Angular, e esse projeto ajudou os engenheiros a entenderem o que era importante do ponto de vista de um desenvolvedor Web em um framework JavaScript (SESHADRI, GREEN, 2014).

O que havia começado em um projeto secundário, rapidamente se transformou em um dos principais frameworks JavaScript na Web.

3.2.1 Arquitetura MVC

O Angular adota o padrão semelhante ao MVC (Model-View-Controller) para estruturar qualquer aplicação. Os dados para apresentar ao usuário corresponde ao modelo em um projeto Angular, que, em sua maior parte, é composto de dados puros e é representado por meio de objetos JSON.

A interface do usuário ou o HTML final renderizado, que o usuário vê e com o qual interage, e que exibe os dados ao usuário, corresponde à visão.

Por fim, a lógica de negócios e o código que acessa os dados, decide que parte do modelo deve ser apresentada ao usuário, lida com validação e assim por diante, ou seja, é a lógica centrar específica da aplicação, corresponde ao controlador.

A abordagem MVC ou semelhante a ela é adequada por alguns motivos consistentes:

- Há uma clara separação de responsabilidades entre as diversas partes de sua aplicação. Com estrutura e padrões bem definidos.

- O Angular é totalmente compatível com o MVC; o controlador jamais fará uma referência direta a visão, assim, o controlador fica independente da visão, além de permitir testar facilmente o controlador sem a necessidade de instanciar um DOM.

Há quatro princípios básicos que fundamentam o Angular e que permitem aos desenvolvedores criar aplicações complexas e de grande porte de forma rápida e fácil (SESHADRI, GREEN, 2014).

Orientação a dados:

A principal funcionalidade do Angular - que pode fazer com que milhares de linhas de código *boilerplate* sejam evitadas - é o seu *data-binding*. Basta efetuar o *binding* dos dados no HTML e o Angular cuidará de fazer esses valores chegarem até a UI. O Angular oferece o *data-binding* bidirecional, garantindo que o controlador e a UI compartilhem o mesmo modelo, de modo que a atualização de um deles, fará o outro ser atualizado automaticamente.

Declarativo:

Uma aplicação Web *single-page* é composta de vários trechos de HTML e de dados associados por meio de JavaScript. Com muita frequência, acaba-se com *templates* HTML que não oferecem nenhum indício daquilo em que se transformaram. Essencialmente, esse é o paradigma imperativo, em que dizemos exatamente a aplicação como realizar toda e qualquer ação. Isso é totalmente feito por meio de código JavaScript, e não no local em que o HTML realmente deve ser alterado. O HTML não reflete nenhuma parte desta lógica.

O Angular, por sua vez, promove um paradigma declarativo, em que declara-se diretamente no HTML o que estamos tentando realizar. Isso é feito por meio de um recurso que o Angular chama de diretiva. Basicamente as diretivas estendem o vocabulário do HTML para ensinar-lhe novos truques.

Injeção de dependência:

O Angular é um dos poucos frameworks JavaScript com um sistema completo de injeção de dependência incluído. A injeção de dependência corresponde ao conceito de solicitar as dependências de um controlador ou de um serviço em particular, em vez de instanciá-las online por meio de um operador *new* ou chamar explicitamente uma função.

Isso é útil porque:

- O controlador, o serviço ou a função que solicitar a dependência não precisa saber como construir suas dependências e pode percorrer toda a cadeia, por mais longa que seja.
- As dependências são explícitas, de modo que podemos saber de imediato o que é necessário antes de começarmos a trabalhar com nossa porção de código.
- É mais fácil de testar porque pode-se substituir dependências por mocks mais adequados aos testes.

A injeção de dependência do Angular é usada em todas as partes, que incluem desde os controladores e serviços até os módulos e testes. Ela permite implementar um código modular e reutilizável facilmente e pode ser usada de modo simples e fácil conforme for necessário.

Extensível:

As diretivas representam a maneira de o Angular ensinar novos truques ao navegador e ao HTML, que variam desde lidar com cliques e condicionais até criar novas estruturas e novos estilos. O Angular expõe uma API para que qualquer desenvolvedor possa estender as diretivas existentes ou criar suas próprias.

3.3 React

O React é uma biblioteca JavaScript criada pelo Facebook para construir interfaces de usuário. Diferentemente do Angular e de outros *frameworks* ou bibliotecas que fornecem todos os componentes necessários para uma aplicação *front-end*, o React se preocupa apenas com a parte de *view*.

O React trabalha com o conceito de componente que recebe propriedades, computa o estado e retorna uma representação virtual do DOM. A forma como ele

lida com isso é estruturando a representação do HTML em objetos.

Suas principais características são:

- Declarativo: projeta diferentes *views* para cada estado da aplicação, que serão renderizadas e atualizadas de forma eficiente;
- Baseado em componentes: componentes de compilação, que gerenciam seu próprio estado e os estruturam juntos em *views* mais complexas;
- Mantém uma representação interna da *view* representada ("virtual DOM[19]"), que processa apenas os elementos alterados.

Considerando que o React é muitas vezes referido como a visão em uma estrutura MVC, o Facebook apresentou sua própria arquitetura chamada Flux. O problema com uma estrutura MVC é a comunicação bidirecional, que provou ser muito difícil de depurar e entender quando uma mudança em uma entidade causa efeito em cascata em toda a base do código. Isso ocorre especialmente quando a aplicação Web está evoluindo constantemente e ganhando uma complexidade muito maior, como o Facebook, por exemplo. O fluxo de dados não foi suficientemente completo ou suficientemente fácil para grandes aplicações.

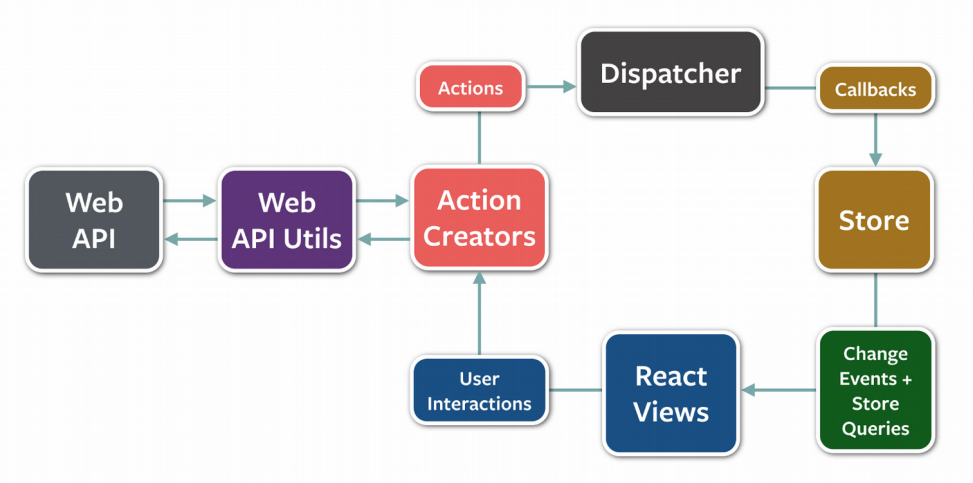
3.3.1 Arquitetura Flux

Flux é um padrão de arquitetura que o Facebook usa para construir aplicações web para o lado do cliente. O conceito mais importante é de que os dados fluem em uma única direção.

As aplicações com Flux têm 4 partes principais: *dispatcher*, *stores, actions* e as *views* (*components*), como mostra a figura 17.

Figura 17: Arquitetura Flux.

19Virtual DOM: diferentemente do DOM que é a representação de diferentes páginas, o virtual DOM é um framework para manipulação do DOM.



Fonte: Disponível em: <https://facebook.github.io/flux/docs/in-depth-overview.html#content>

O *Dispatcher* recebe e registra *actions* e *callbacks*[20]. Deve haver apenas uma instância do *dispatcher* em cada aplicação (ou seja, ele é um *singleton*[21]). O *dispatcher* é o *hub* central que gerencia todo o fluxo de dados em um aplicativo Flux. É essencialmente um registro de *callbacks* nas *stores* e não tem nenhuma inteligência real própria - é um mecanismo simples para distribuir as *actions* às *stores*. Cada *store* se registra e fornece um retorno de chamada. Quando um criador de *action* fornece ao *dispatcher* uma nova *action*, todas as *stores* no aplicativo recebem a *action* através dos *callbacks.*

À medida que um aplicativo cresce, o *dispatcher* torna-se mais vital, pois pode ser usado para gerenciar dependências entre as *stores*, invocando as devoluções de retorno registradas em uma ordem específica. As *stores* podem declarativamente esperar que outras *stores* finalizem a atualização.

As *stores* contêm o estado e a lógica da aplicação. Seu papel é algo semelhante a um modelo em um MVC tradicional, mas eles gerenciam o estado de muitos objetos - eles não representam um único registro de dados, como fazem os modelos ORM[22]. Mais do que simplesmente gerenciar uma coleção de objetos de

20Callback: é um pedaço de código executável que é passado como parâmetro para algum método, é esperado que o método execute o código do argumento em algum momento. A invocação do trecho pode ser imediato como em um callback síncrono, ou em outro momento, como em um callback assíncrono.
21Singleton: é um padrão de projeto que tem como definição garantir que uma classe tenha apenas uma instância de si mesma e que forneça um ponto global de acesso a ela. Ou seja, uma classe gerencia a própria instância dela além de evitar que qualquer outra classe crie uma instância dela.
22ORM (Object-relational mapping): é uma técnica de desenvolvimento utilizada para reduzir a impedância da programação orientada aos objetos utilizando bancos de dados relacionais. As tabelas do banco de dados são representadas através de classes e os registros de cada tabela são representados como instâncias das classes correspondentes.

estilo ORM, as *stores* gerenciam o estado da aplicação para um determinado domínio dentro do aplicativo.

Os dados em uma *store* só devem ser alterados respondendo a uma ação. Toda vez que os dados de uma *store* são alterados, ela deve emitir um evento de "mudança".

As *Actions* definem a API interna de uma aplicação. Elas determinam as maneiras pelas quais qualquer coisa pode interagir com a aplicação. São objetos simples que têm um campo "tipo" e alguns dados. As *actions* devem ser semânticas e descritivas da ação que está ocorrendo. Elas não devem descrever detalhes de implementação da *action* correspondente.

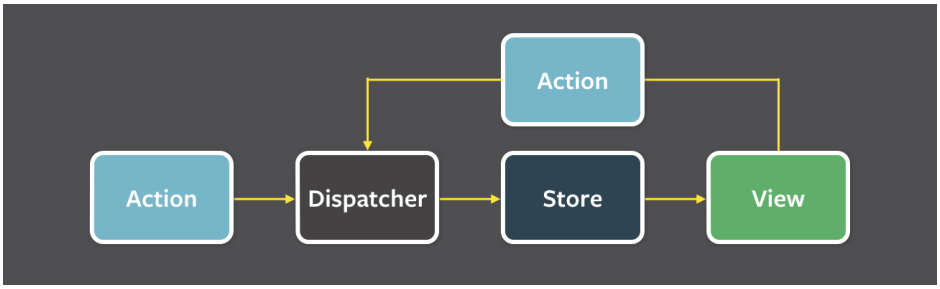
As *Views* fornecidas pelo React são livremente renderizadas e são usadas para a camada de exibição. Perto do topo da hierarquia de dados, um tipo especial de *view* escuta eventos que são transmitidos pelas *stores*. A isso dá-se o nome de *controller-view*, pois fornece o código para obter os dados das *stores* e passa esses dados pela cadeia de seus descendentes.

Os dados das *stores* são exibidos nas *views*, de modo que quando a *store* emite um aviso de alteração, a *view* obtém os novos dados e é novamente renderizada. As *actions* geralmente são enviadas de *views* à medida que o usuário interage com partes da interface da aplicação.

Todos os dados fluem através do *dispatcher* como um *hub* central. As *actions* são fornecidas ao *dispatcher* em um método criador de *actions* e, na maioria das vezes, são originadas pelas interações do usuário com as *views*. O *dispatcher* então invoca as devoluções de chamada que as *stores* registraram, enviando *actions* para todas as *stores*. Dentro de suas devoluções de retorno registradas, as *stores* respondem a qualquer *action* relevante para o estado que eles mantêm. As *stores* emitem um evento de alteração para alertar as visualizações do controlador de que ocorreu uma alteração na camada de dados.

A Figura 18 descreve como os dados fluem através do sistema. O fluxo de dados ocorre da seguinte maneira:

Figura 18: Diagrama do fluxo de dados da arquitetura Flux.



Fonte: Disponível em: <https://facebook.github.io/flux/docs/in-depth-overview.html#content.>

1. As *views* enviam actions para o *dispatcher*.
2. O *dispatcher* envia *actions* para todas as *stores*.
3. Os *stores* enviam dados para as *views*.

Esta estrutura permite raciocinar facilmente sobre a aplicação de uma forma que é uma reminiscência de programação funcional reativa ou, mais especificamente, programação baseada em fluxo (*stream*), onde os dados fluem através da aplicação em uma direção única. O estado da aplicação é mantido somente nas *stores*, permitindo que as diferentes partes do aplicativo permaneçam altamente desacopladas. Quando ocorrem dependências entre as *stores*, elas são mantidas em uma hierarquia rígida, com atualizações síncronas gerenciadas pelo *dispatcher*.

Durante o processo de criação do React e da arquitetura Flux, os programadores do Facebook descobriram que as ligações de dados bidirecionais levam a atualizações em cascata, onde a mudança de um objeto leva a outra alteração de objeto, o que também poderia desencadear mais atualizações. À medida que as aplicações cresciam, essas atualizações em cascata tornaram muito difícil prever o que mudaria como resultado de uma interação com um usuário. Quando as atualizações só podem alterar dados dentro de uma única rodada, o sistema como um todo se torna mais previsível.

3.4 Knockout

Knockout (KO) é uma biblioteca JavaScript desenvolvida e mantida como código aberto por Steve Sanderson, funcionário da Microsoft, por isso é bastante

usado na comunidade Microsoft .NET[23]. A arquitetura do Knockout segue o padrão *Model-View-ViewModel* (MVVM). Os princípios subjacentes são, portanto:

- uma clara separação entre os dados do domínio;
- a presença de uma camada claramente definida de código especializado para gerenciar as relações entre os componentes da *view.*

Estes componentes aproveitam os recursos de gerenciamento de eventos nativos do linguagem JavaScript. Esses recursos simplificam a especificação de relacionamentos complexos entre componentes de exibição (MUNRO, 2015).

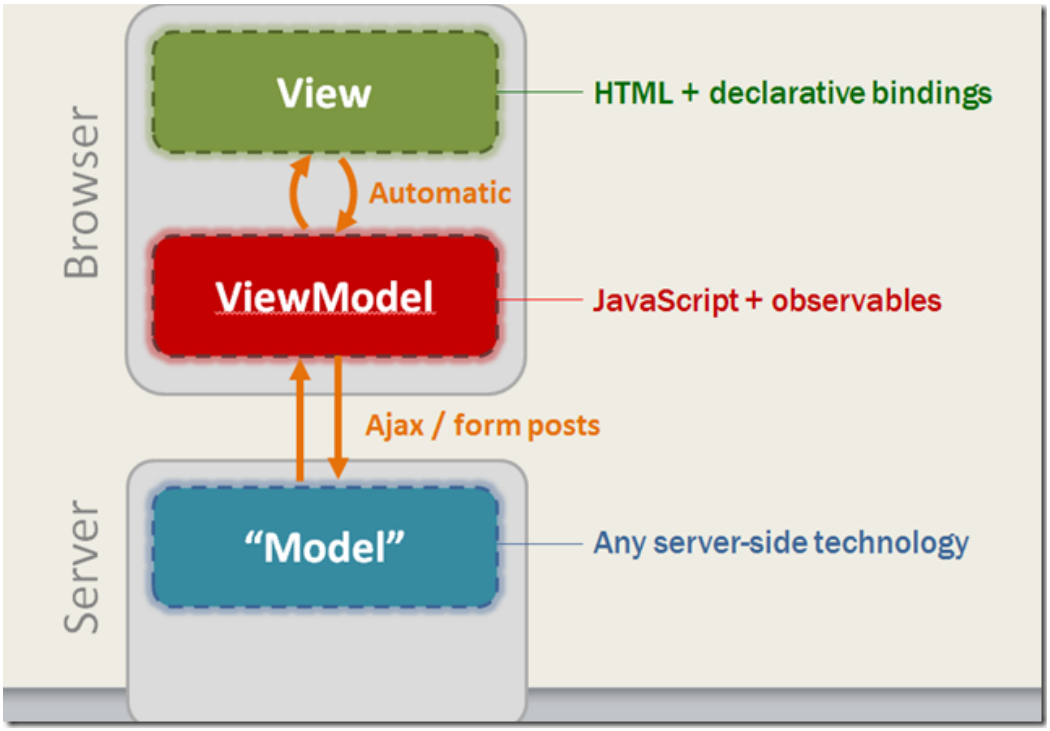
Knockout é construído em torno de três recursos principais:

- *Observables* e rastreamento de dependência
- Ligações declarativas
- *Templating*

Para que o KO saiba quando alguma parte da *viewmodel* teve mudanças é necessário declarar as propriedades do *model* como *observables.* A figura 19 essa ligação entre o *model* e a *viewmodel* com *observables.*

Figura 19: Arquitetura Knockout.

23O .NET Framework: é uma iniciativa da empresa Microsoft, que visa uma plataforma única para desenvolvimento e execução de sistemas e aplicações. Todo e qualquer código gerado para .NET pode ser executado em qualquer dispositivo que possua um framework de tal plataforma.



Fonte:Disponível em: < http://www.programering.com/a/MTN5MDMwATI.html.>

3.4.1 Arquitetura MVVP

A arquitetura *Model-View-ViewModel* (MVVM), como especificada no capítulo 2, é um padrão de design para a construção de interfaces de usuário, dividindo-a em três partes:

Model: os dados armazenados da aplicação. Estes dados representam objetos e operações no domínio de negócios (por exemplo, contas bancárias que podem realizar transferências de dinheiro) e é independente de UI. No KO (Knockout), os *models* fazem as chamadas Ajax para ler e escrever estes dados do modelo armazenado.

ViewModel: uma representação de código puro dos dados e operações em uma IU. Por exemplo, se você estiver implementando um editor de lista, seu modelo de exibição seria um objeto que contenha uma lista de itens e expondo métodos para adicionar e remover itens.

Esta não é a UI em si: não tem nenhum conceito de botões ou estilos de exibição. Também não é o modelo de dados persistente - ele contém os dados não salvos com os quais o usuário está trabalhando. Ao usar o KO, seus modelos de visão são objetos JavaScript simples que não possuem conhecimento de HTML. Manter o modelo de visão abstrato desta forma permite que ele fique simples, para que você possa gerenciar comportamentos mais sofisticados sem se perder.

View: uma UI visível e interativa que representa o estado do modelo de visão. Ele exibe informações do modelo de exibição, envia comandos para o modelo de exibição (por exemplo, quando o usuário clica nos botões) e atualiza sempre que o estado do modelo de exibição muda.

Ao usar o KO, a visão é simplesmente um documento HTML com ligações declarativas para vinculá-lo à visualização do modelo. Alternativamente, pode-se usar modelos que geram HTML usando dados do modelo de visão.

3.5. Considerações Finais do Capítulo

O Angular adota uma estrutura semelhante ao MVC, obtendo uma clara separação de responsabilidades. Também é conhecido pelo padrão MVW (Model View Whatever), podendo ser utilizado como MVVM por sua enorme facilidade de ligar a *view* aos dados do *model.*

O padrão MVC foi criado na década de 1970, com a proposta de separação da interface do usuário, com as regras de negócio do sistema. Nesta época, não existia tanta interação com a interface como temos hoje, e ainda assim, já existia este importante conceito de separação das responsabilidades.

Essa grande quantidade de interação com o usuário fez o Facebook criar a arquitetura Flux para solucionar os problemas de atualização de dados. A forma bidirecional de atualização de dados criou um efeito cascata na aplicação do Facebook, que provou ser difícil de depurar.

Em uma aplicação de grande porte, como o Facebook, a forma unidirecional de dados do Flux é garantida por algumas razões: O *Dispatcher* é único para toda a aplicação, e existe um único elemento responsável por receber as *actions* realizadas nas *views* e dispará-las para as *stores* da aplicação. Toda e qualquer mudança no estado da aplicação somente pode ser realizada através de uma *action*.

O Knockout segue o padrão MVVM com uma clara separação entre os dados do domínio e com uma camada de código especializado para gerenciar as relações entre os componentes da *view*.

Estas três arquiteturas bem diferentes justificam a realização de um estudo sobre as mesmas, de forma a entender melhor suas características e possíveis dificuldades.

4. Desenvolvimento

Ao longo deste capítulo será desenvolvida uma aplicação de exemplo para conferir os estilos e diferenças de implementação entre o framework Angular e das bibliotecas React e Knockout. Para fins de comparação, JavaScript puro também é utilizado. Será especificado o que cada um tem de exclusivo e suas principais funcionalidades que auxiliam o desenvolvedor.

As versões utilizadas foram: AngularJS - 1.5.8, React - 15.16.1 e Knockout - 3.2.0.

4.1 Aplicação de teste

Para a aplicação de teste criou-se um algoritmo que gera um array de objetos (que pode ser alterado de tamanho), cujos valores são determinados aleatoriamente com base em valores contidos em três arrays diferentes, para verificar o desempenho em pequenos e grandes arrays. Os itens serão renderizados em uma *tag* não ordenada (
) e, ao mesmo tempo, será verificada a hora local exata com o objeto *Date()* do JavaScript. Ao final da renderização nos respectivos framework e bibliotecas, será chamado novamente o objeto *Date()* para calcular a diferença de tempo entre o início da renderização e o seu término.

Figura 20: Código de construção do array de objetos.

```
6
7       // função que constroí o array que será renderizado
8       function run(size) {
9           // quantidade de elementos criados
10          size = size || 200;
11
12          // arrays de elementos
13          var citys = ['Florianópolis', 'São Paulo', 'Porto Alegre', 'Palhoça', 'Florença', 'Roma', 'Mendoza', 'Córdoba',
14                       'Blumenau', 'Criciúma', 'Barcelona', 'Londres', 'Paris', 'Xangai', 'Osaka', 'Pequin',
15                       'Salvador', 'Los Angeles', 'Las Vegas', 'Pato Branco', 'Curitiba', 'Buenos Aires', 'Manaus',
16                       'Nápoles', 'Porto'];
17          var states = ['Santa Catarina', 'Paraná', 'Bahia', 'São Paulo', 'Amazonas', 'Mato Grosso', 'Tocantins',
18                        'Pernambuco', 'Paraíba', 'Roraima', 'Acre'];
19          var countrys = ['Brasil', 'Argentina', 'Chile', 'Itália', 'Peru', 'Bolívia', 'França', 'Espanha', 'China',
20                      'Escócia', 'Irlanda', 'Grécia', 'Austrália'];
21
22          var data = [];
23
24          // o elementos são randomicamente adicionados no array data
25          for (var i = 0; i < size; i++)
26              data.push({ id: i + 1,
27                  label: citys[random(citys.length)] + " | " +
28                         states[random(states.length)] + " | " +
29                         countrys[random(countrys.length)] });
30
31          return data;
32      }
33
34      // função recebe a quantidade de elementos que irá criar
35      function random(max) {
36          return Math.round(Math.random() * 1000) % max;
37      }
```

Fonte: Repositório do autor

4.2 Desenvolvimento com Angular

Toda a página do *html* de teste está marcada com a diretiva de declaração de módulo *ng-app.* Esta diretiva inicia o Angular, avisando que aquele pedaço de HTML é a aplicação e que ele apenas funciona naquele escopo.

Quando o Angular é iniciado, ele usa a configuração do módulo com o nome definido pela diretiva *ng-app*, incluindo a configuração de todos os módulos dos quais ele depende. No exemplo da figura 21, o módulo contém a diretiva *ng-app="app"*. Isso diz ao Angular que use o módulo *app* como o módulo principal para o aplicativo.

Figura 21: Inicializando uma aplicação angular com a diretiva *ng-app.*

```
2   <!DOCTYPE html>
3
4   <html ng-app="app">
5       <head>
```

Fonte: Repositório do autor.

Agora que o Angular conhece as partes da aplicação, para interagir com a aplicação é necessário declarar o controlador da página, que representa o escopo de uma página. Nesta aplicação o controlador é declarado na tag *<body>,* como na figura 22, e então tudo o que está entre esta tag é o escopo da aplicação. Este é um aspecto chave de como o angular suporta os princípios por trás do padrão de design Model-View-Controller.

Figura 22: Informa que o escopo do controlador começa a partir da tag <body> .

```
21
22      <body ng-controller="controller">
23
```

Fonte: Repositório do autor.

O controlador da página, representado na figura 23, possui a função *initialize()*, que é invocada pelo usuário na camada de apresentação. Esta função chama a função *count()*, onde é criado o array de objetos e renderizado na página através da diretiva *ng-repeat,* como representado na figura 24.

No controlador da figura 23 o escopo global é representado por *$scope*, que envolve todo o contexto da página. O escopo é organizado em uma estrutura hierárquica que imita a estrutura DOM da aplicação. Assim, para toda a manipulação na *view* deve haver uma função, objeto ou array no controlador com o *$scope* declarado antes.

Digest cycle é o processo por trás do Angular para realizar o *data-binding*, quando o Angular termina um ciclo, a função na linha 45 com *$$postDigest* é chamada. Esta função atualiza o DOM e o tempo para renderizar é calculado.

Figura 23: Declaração do módulo app e controlador da página.

```
36      // Angular
37      angular.module('app', []).controller('controller', function($scope) {
38
39          $scope.initialize = function() {
40              var data = count(),
41                  date = new Date();
42
43              $scope.selected = null;
44              // é disparado quando um ciclo do Angular é completado
45              $scope.$$postDigest(function() {
46                  document.getElementById('run-angular').innerHTML =
47                      (new Date() - date) + ' ms';
48              });
49
50              $scope.data = data;
51          };
52
53          // função para selecionar um item
54          $scope.select = function(item) {
55              $scope.selected = item.id;
56          };
57
```

Fonte: Repositório do autor.

No HTML a diretiva *ng-click* captura o click do usuário para chamar a função *initialize().* A diretiva *ng-ŕepeat* instância um item por vez do array de objetos gerado e a diretiva *ng-class* adiciona a classe de css *selected*, definida no arquivo de css da página.

Figura 24: Html para a renderização dos objetos com Angular.

```
43      <!-- ANGULAR -->
44      <div class="col-md-3 band2">
45          <div class="row">
46              <div class="col-md-7">
47                  <h3>Angular</h3>
48              </div>
49              <div class="col-md-5 text-right time" id="run-angular" ng-click="initialize()">
50                  Iniciar
51              </div>
52          </div>
53          <div>
54              <div class="row" ng-repeat="item in data">
55                  <div class="col-md-12 test-data">
56                      <span ng-class="{ selected: item.id === $parent.selected }"
57                          ng-click="select(item)"> {{item.label}}
58                      </span>
59                  </div>
60              </div>
61          </div>
62      </div>
```

Fonte: Repositório do autor.

Alguns aspectos interessantes para notar aqui. O Angular não exige tanto código do desenvolvedor, porém, exige mais conhecimento sobre o framework em si. O desenvolvedor tem uma curva de aprendizado maior com esse framework do que com as bibliotecas React e Knockout.

4.3 Desenvolvimento com React

Quando o componente é inicializado, o método *render()* é chamado, gerando uma representação da *view*. Dessa representação, um *markup* é produzido e injetado no documento. Quando algum dado muda, o método *render* é chamado novamente. Para que a atualização seja o mais eficiente possível, o React verifica as diferenças (diff) entre o valor novo e o valor velho e aplica no DOM somente a mudança ocorrida.

A função *render* faz todo o trabalho que o *ng-repeat* faz no Angular, e então é necessário manipular o DOM através das *tags* e classes que estão definidas na *view*. Essa é a principal característica do React.

Para atualizar o valor de algum campo, pode-se usar com JavaScript *document.getElementById('idDoElemento').value = 'valor'*, ou com jQuery *$ ('#idDoElemento').val('valor'),* como mostra a figura 25. Nos dois casos, será acessada a árvore do DOM e manipulado (consultado ou alterado) o valor do elemento identificado, o que é algo muito custoso para o *browser.*

Para resolver esse problema, o React utiliza o Virtual DOM, que basicamente é a representação do DOM real em memória. Sempre após a primeira renderização do seu componente no React, é gerada essa representação e somente após isso seu HTML é renderizado no DOM real. As próximas renderizações desse mesmo

componente não irão ser feitas no DOM real diretamente, mas vão passar pelo algoritmo de *diff* do React.

Essa característica faz o React ser um pouco mais lento na primeira interação, e mais rápido nas próximas interações.

Figura 25: Componente React

```
39      // React
40      function _react() {
41          // definição de classe
42          var Class = React.createClass({
43              // função para selecionar um item
44              select: function(data) {
45                  this.props.selected = data.id;
46                  this.forceUpdate();
47              },
48
49              // função para o react renderizar o objeto e manipular diretamente o html
50              render: function() {
51                  var items = [];
52
53                  // react interage diretamente com os elementos e classes html
54                  for (var i = 0; i < this.props.data.length; i++) {
55                      items.push(React.createElement("div", { className: "row" },
56                          React.createElement("div", { className: "col-md-12 test-data" },
57                              React.createElement("span", {
58                                  className: this.props.selected ===
59                                      this.props.data[i].id ? "selected" : "",
60                                  onClick: this.select.bind(null, this.props.data[i]) },
61                                      this.props.data[i].label)
62                          )
63                      ));
64                  }
65
66                  return React.createElement("div", null, items);
67              }
68          });
69
70          var runReact = document.getElementById("run-react");
71          // evento para chamar a criação do objeto e renderizar o objeto
72          runReact.addEventListener("click", function() {
73              var data = run(),
74                  date = new Date();
75
76              React.render(new Class({ data: data, selected: null }),
77                  document.getElementById("react"));
78
79              runReact.innerHTML = (new Date() - date) + " ms";
80          });
81      }
```

Fonte: Repositório do autor.

No HTML, não há a necessidade de incluir marcadores como no Angular, apenas o id's para que o React possa interagir com a página.

Figura 26: Html para a renderização dos objetos com React.

```
32      <!-- REACT -->
33      <div class="col-md-3 band">
34          <div class="row">
35              <div class="col-md-7">
36                  <h3>React</h3>
37              </div>
38              <div class="col-md-5 text-right time" id="run-react">Iniciar</div>
39          </div>
40          <div id="react"></div>
41      </div>
```

Fonte: Repositório do autor.

4.4 Desenvolvimento com Knockout

Para rodar o Knockout (KO) é necessário executar a função *ko.applyBindings()*, representado na figura 27, linha 42. Dentro desta função se manipula toda a *viewModel* do KO. A *view* está ligada a propriedades que estão dentro desta função, e assim o KO consegue manipular o template através do atributo *data-bind* no HTML. Há um segundo parâmetro passado em *ko.applyBindings()* que define qual parte do documento o KO deve procurar por atributos *data-bind*.

KO atualiza a UI automaticamente quando a *viewModel* é alterada. Para isso é necessário declarar as propriedades do *model* como *observables*, que são objetos JavaScript especiais que podem notificar o KO sobre alterações, e pode automaticamente detectar dependências.

Figura 27: Model da aplicação com Knockout.

```
39      // Knockout
40      function _knockout() {
41          // função de ativação do KO
42          ko.applyBindings({
43              // select recebe o item selecionado pelo usuário
44              selected: ko.observable(),
45              // data é o array observable
46              data: ko.observableArray(),
47              // função para selecionar um item
48              select: function(item) {
49                  this.selected(item.id);
50              },
51
52              // run chama a função count para gerar o array de objetos
53              run: function() {
54                  var data = count(),
55                      date = new Date();
56
57                  this.selected(null);
58                  // data: ko.observableArray() é chamado para se tornar o observable
59                  this.data(data);
60                  // quando terminar a renderização é calculado a diferença de tempo
61                  document.getElementById("run-knockout").innerHTML =
62                      (new Date() - date) + " ms";
63              }
64          }, document.getElementById("knockout"));
65      }
66
67      // função para detectar e responder a mudanças ao array
68      ko.observableArray.fn.reset = function(values) {
69          var array = this();
70          this.valueWillMutate();
71          // adiciona os itens ao array
72          ko.utils.arrayPushAll(array, values);
73          // notifica o array que houve mudanças
74          this.valueHasMutated();
75      };
76
```

Fonte: Repositório do autor.

Na *view* o *id=”knockout”* informa ao KO onde procurar pelo atributo *data-bind.* O *data-bind=”foreach: data”* (linha 72 da Figura 28) repete cada item presente no array *data*, e associa cada elemento da marcação ao item correspondente.

Figura 28: Html com Knockout.

```
62      <!-- KNOCKOUT -->
63      <div id="knockout" class="col-md-3 band">
64          <div class="row">
65              <div class="col-md-7">
66                  <h3>Knockout</h3>
67              </div>
68              <div class="col-md-5 text-right time" id="run-knockout" data-bind="click: run">
69                  Iniciar
70              </div>
71          </div>
72          <div data-bind="foreach: data">
73              <div class="row">
74                  <div class="col-md-12 test-data">
75                      <span data-bind="click: $root.select.bind($root, $data),
76                          text: $data.label, css: { selected: $data.id === $root.selected() }">
77                      </span>
78                  </div>
79              </div>
80          </div>
81      </div>
```

Fonte: Repositório do autor.

4.5 Desenvolvimento com JavaScript puro

Uma das características mais importantes da execução de JavaScript em uma página Web é que o navegador disponibiliza acesso a todo e qualquer elemento declarado no HTML através do DOM. Isso significa que podemos, em nosso código JavaScript, criar objetos que fazem referência direta a *tags* do HTML. A mesma abordagem foi utilizada com o React.

Em JavaScript puro, temos a seguinte abordagem:

Figura 29: Capturar a referência de um id.

```
41      // retorna a referência do elemento atráves do seu id
42      var container = document.getElementById("raw");
```

Fonte: Repositório do autor.

Figura 30: Adicionar evento em um id.

```
46      // adiciona o evento "click" ao elemento de id "run-raw",
47      // quando o usuário clicar no botão a função é disparada
48      document.getElementById("run-raw").addEventListener("click", function() {
```

Fonte: Repositório do autor.

Esse padrão chama-se JavaScript não-obstrutivo (não intrusivo). Adiciona-se interatividade à página através do JavaScript, sem a adição de atributos e informações desnecessárias na marcação.

Toda a manipulação de página com JavaScript puro envolve identificar as referências e adicionar eventos. Na figura 31, linha 48, é adicionado um evento de clique a classe “run-raw”, a partir do clique, o algoritmo que gera o array de objetos é chamado. O loop na linha 56, é onde o objeto será renderizado e passado ao atributo html (linha 60), este, passa ao *container* que é a referência ao id “raw”, elemento em que o usuário verá o objeto final renderizado.

Figura 31: Aplicação com JavaScript puro.

```
39      // JavaScript puro
40      function _pureJs() {
41          // retorna a referência do elemento atráves do seu id
42          var container = document.getElementById("raw");
43          // innerHTML obtém a sintaxe HTML descrevendo os elementos descendentes.
44          var template = document.getElementById("raw-template").innerHTML;
45
46          // adiciona o evento "click" ao elemento de id "run-raw",
47          // quando o usuário clicar no botão a função é disparada
48          document.getElementById("run-raw").addEventListener("click", function() {
49              // data recebe os valores que retornarem da função run()
50              var data = run();
51              // date recebe o horário e data no momento que o evento "click" é disparado
52              var date = new Date(),
53                  html = "";
54
55              // loop para renderizar o objeto data na classe {{lab el}}
56              for (var i = 0; i < data.length; i++) {
57                  var render = template;
58                  render = render.replace("{{className}}", "");
59                  render = render.replace("{{label}}", data[i].label);
60                  html += render;
61              }
62
63              container.innerHTML = html;
64
65              var spans = container.querySelectorAll(".test-data span");
66
67              // loop para poder selecionar um item do objeto gerado na tag span
68              for (var i = 0; i < spans.length; i++)
69                  spans[i].addEventListener("click", function() {
70                      var selected = container.querySelector(".selected");
71                      if (selected)
72                          selected.className = "";
73                      this.className = "selected";
74                  });
75
76              // calculo para diferença de tempo entre o início e o final da renderização
77              document.getElementById("run-raw").innerHTML = (new Date() - date) + " ms";
78          });
79      }
```

Fonte: Repositório do autor.

Para ver as mudanças no HTML, é necessário utilizar a *tag* <script>, como na figura 31, linha 97.

Figura 32: HTML com JavaScript puro.

```
83      <!-- JAVASCRIPT PURO -->
84      <div class="col-md-3 band2">
85          <div class="row">
86              <div class="col-md-7">
87                  <h3>JS puro</h3>
88              </div>
89              <div class="col-md-5 text-right time" id="run-raw">
90                  Iniciar
91              </div>
92          </div>
93          <div id="raw"></div>
94      </div>
95  </div>
96
97  <script type="text/html" id="raw-template">
98      <div class="row">
99          <div class="col-md-12 test-data">
100             <span class="{{className}}">{{label}}</span>
101         </div>
102     </div>
103 </script>
```

Fonte: Repositório do autor.

4.6 Considerações finais do capítulo

Embora as diferenças entre Angular, React e Knockout sejam grandes, ambos podem realizar o mesmo. Angular possui uma estrutura muito mais completa do que o React e Knockout, mas isso não significa muito quando não há a necessidade da maioria dos recursos que o Angular fornece.

Tempo de renderização em diferentes *browsers*, bem como o tamanho, quantidade de código necessário para a aplicação de teste e outras características de desempenho, serão detalhados no capítulo 5.

5. Avaliação e análise de desempenho

No artigo *“What leads developers towards the choice of a JavaScript framework?*”, baseado em entrevistas com 18 desenvolvedores capazes de motivar o processo de tomada de decisão na escolha de um framework JavaScript em suas empresas, Pano, Graziotin e Abrahamsson (2016) relatam que os desenvolvedores esperam que algumas atividades básicas, como a vinculação de dados, a manipulação de DOM e a atualização em tempo real da aplicação estejam presentes. As bibliotecas e funcionalidades existentes da estrutura devem permitir modificações através de operações simples. A estrutura deve ser modular, no sentido de que, se uma parte do código for alterada, ela não comprometa a aplicação geral.

Aos participantes da entrevista foi ofertado o seguinte modelo de fatores importantes para o desenvolvimento web: usabilidade (atratividade, aprendizado, compreensão), custo, eficiência (desempenho, tamanho) e funcionalidade (automatização, extensibilidade, flexibilidade, isolamento, modularidade, atualização). Destes fatores, todos os participantes mencionaram eficiência (desempenho, tamanho) como o principal fator de influência na escolha de um framework JavaScript.

A eficiência (desempenho, tamanho) envolve tempo de desenvolvimento (quantidade de código necessário para a aplicação), tempo de renderização e carregamento da página para o usuário em diferentes *browsers* e tamanho do framework ou biblioteca, ou seja, tudo que afeta diretamente o desempenho da aplicação para o usuário final (Pano, Graziotin e Abrahamsson, 2016).

5.1 Desempenho

O tempo de execução foi apurado em três diferentes *browsers*: Chrome, Firefox e Safari. O algoritmo utilizado para teste gera um array de 1000 objetos em que é calculado o tempo desde o início da geração do array até a visualização do usuário no *browser.* Para cada execução é anotado o tempo e calculada a média geral, representada na Tabela 1. Foram feitas 10 execuções do algoritmo.

Tabela 1: Cálculo desvio padrão e média.

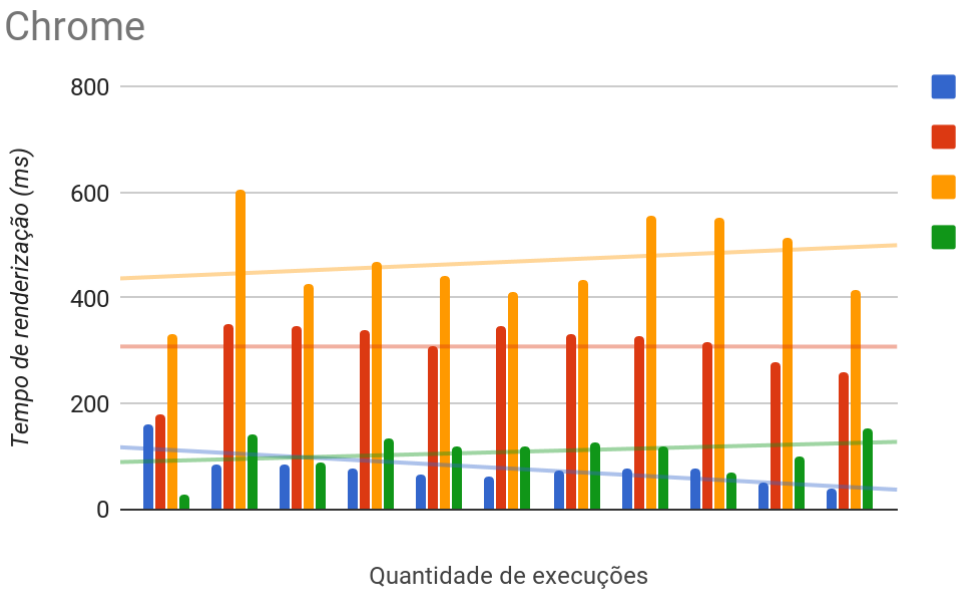
		Angular	React	Knockout	JS
Chrome - Versão 60.0.3112.113	**Desvio padrão**	51,81400302	31,70173497	79,99295424	35,76895556
	Média	307 ms	82 ms	468 ms	102 ms
		Angular	**React**	**Knockout**	**JS**
Firefox - Versão 55.0.2	**Desvio padrão**	75,40822236	23,50009671	41,46170193	27,94410004
	Média	440 ms	105 ms	801 ms	111 ms
		Angular	**React**	**Knockout**	**JS**
Safari - Versão 11.0	**Desvio padrão**	10,56753175	7,462634192	15,22557657	14,86117578
	Média	107 ms	29 ms	168 ms	57 ms

Fonte: Repositório do autor.

Com a medição do desvio padrão é possível observar que o React tem uma menor variação de acordo com a média nos três *browsers* testados, além de ter uma menor média de tempo de execução.

Para melhor visualização dos dados, três gráficos foram gerados. As linhas de tendência mostram o sentido dominante que o Angular (cor vermelha), React (cor azul), Knockout (cor laranja) e JavaScript puro (cor verde) seguiram.

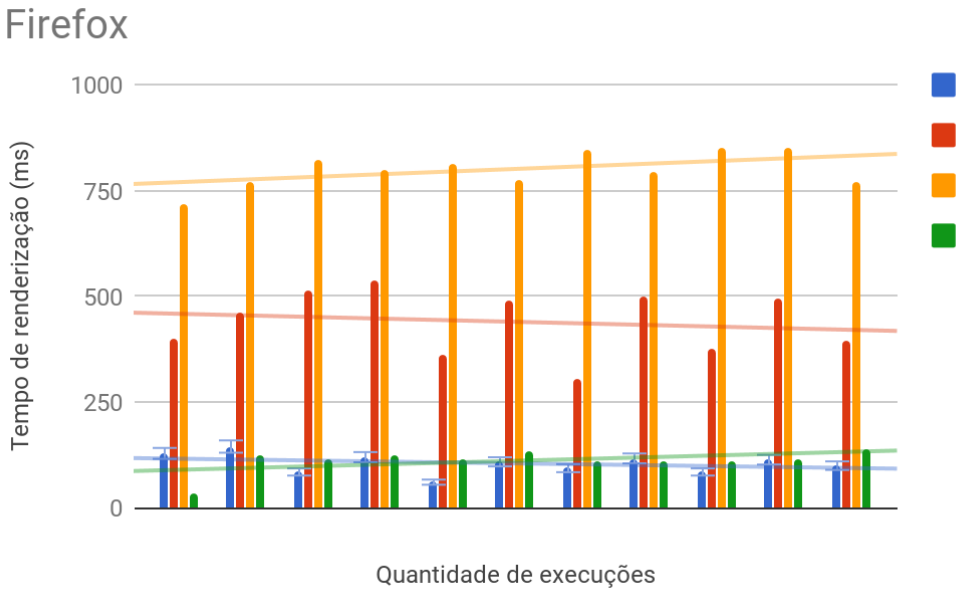
Gráfico 1: Browser Chrome.



Fonte: Repositório do autor.

React	Angular	Knockout	JavaScript

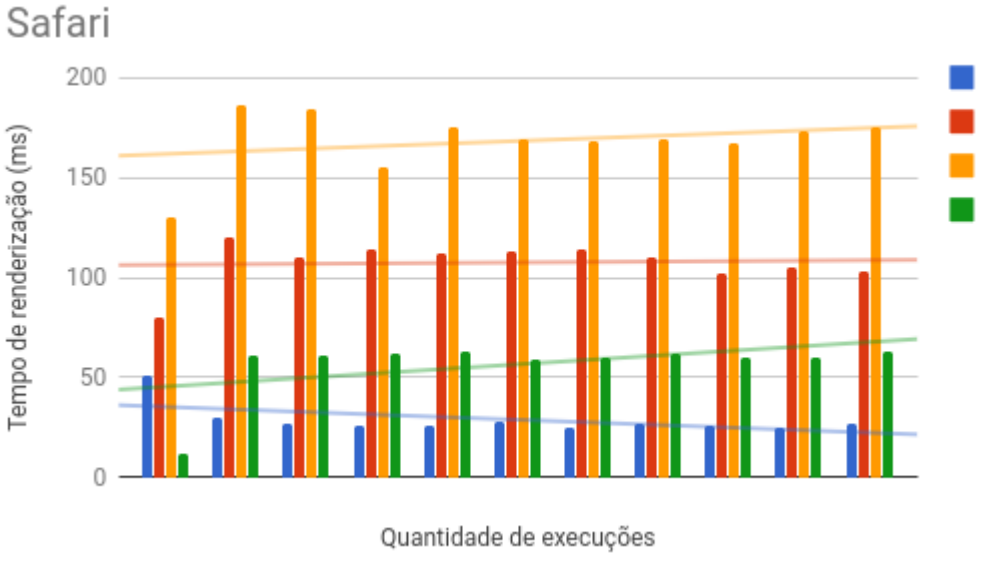
Gráfico 2: Browser Firefox.



Fonte: Repositório do autor.

React	Angular	Knockout	JavaScript

Gráfico 3: Browser Safari.



Fonte: Repositório do autor.

Cada *browser* interpreta JavaScript de uma maneira diferente e isso influencia muito os tempos de execução. A mesma aplicação tem diferentes tempos de execução para cada *browser. O* mecanismo de renderização que o Firefox utiliza é o Gecko, criado pelo Mozilla. Safari e Chrome utilizam o Webkit, não é foco deste trabalho os mecanismos de renderização que cada *browser* utiliza.

A forma como cada framework ou biblioteca lida com a manipulação do DOM influencia muito no desempenho final da aplicação. O Angular e Knockout manipulam o DOM real, enquanto que o React cria e manipula o DOM virtual. O DOM virtual evita mudanças no DOM, que são caras em termos de desempenho, porque causam uma nova renderização da página e por conseguinte do DOM. O DOM virtual também permite coletar várias alterações a serem aplicadas de uma só vez. Portanto, nem todas as alterações causam uma nova renderização, mas a renderização só acontece uma vez após o conjunto de alterações serem aplicadas ao DOM.

A quantidade de código que é carregado pelo *browser* também influencia no desempenho de uma aplicação. O Angular por ser um framework, possui uma estrutura que determina a arquitetura que o projeto seguirá. Depois de escolher uma estrutura para trabalhar, deve-se seguir o código da estrutura e as metodologias de design. A estrutura irá fornecer funções e *callbacks* ao desenvolvedor. Assim, o Angular possui o maior tamanho entre os produtos testados.

React e o Knockout, por serem uma biblioteca, são coleções de classes e métodos que podem ser estendidos e reutilizados. Uma biblioteca se concentra em diversas funcionalidades, que o desenvolvedor pode utilizar.

Para a aplicação de teste, o tamanho e a quantidade de código necessário para a mesma aplicação são mostrados na Tabela 2.

Tabela 2: Tamanho e quantidade de código necessário.

	Angular	**React**	**Knockout**	**JavaScript puro**
Tamanho do framework e bibliotecas	160 KB	23 KB	54,1 KB	-
Quantidade de código JS	538 caracteres	1321 caracteres	787 caracteres	1160 caracteres
Quantidade de código HTML	786 caracteres	350 caracteres	849 caracteres	668 caracteres
Total de	1324 caracteres	1671 caracteres	1636 caracteres	1828 caracteres

caracteres				

Fonte: Repositório do autor.

Com JavaScript puro não há a necessidade de adicionar código extra para a aplicação, já que os *browsers* interpretam JavaScript nativamente. A desvantagem de usar somente JavaScript para o desenvolvimento web está na menor reutilização de código, por isso, a quantidade de código necessário para uma aplicação tende a ser maior.

5.2 Comparação de características

A Tabela 3 apresenta uma comparação das diferentes características do Angular, React e Knockout, permitindo que se tenha uma idéia geral das suas principais diferenças. Também são apresentadas outras informações sobre eles, como a quantidade de websites que usam atualmente cada um deles, comunidade e popularidade de desenvolvedores no GitHub, informações que também foram levantadas como relevantes para os desenvolvedores entrevistados no artigo "*What leads developers towards the choice of a JavaScript framework?*".

Tabela 3: Comparação de características.

	Angular	**React**	**Knockout**
Opinativo	Fortemente opinativo	Fracamente opinativo	Fracamente opinativo
Curva de aprendizado	Íngrime	Suave	Suave
Abstração	Baixa	Alta	Alta
Manipulação de DOM	DOM Real	Virtual DOM	DOM Real
Desenvolvimento Mobile	Exige biblioteca de terceiros	Suporte nativo com React Native	Exige biblioteca de terceiros
Usado em	522,818 Websites	101,549 Websites	3,207 Websites
Popularidade no GitHub	57.241 estrelas	77.758 estrelas	8.470 estrelas
Comunidade de desenvolvedores	1.603 contribuidores	1.102 contribuidores	72 contribuidores

Fonte: SimilarTech <https://www.similartech.com/>.

O Angular é fortemente opinativo, o que significa que sua estrutura e regras são todas adaptadas para se adequar rigidamente a uma maneira específica de programação. Ele impõe muitas regras e estruturas complexas que os desenvolvedores precisam aprender. Isso pode resultar em curvas de aprendizado consideravelmente íngremes e pode levar muito tempo antes que seja possível iniciar o desenvolvimento de um produto. Em contraste, o React e o Knockout são significativamente mais fáceis para aprender, pois existem conceitos e estruturas menos complexas que os desenvolvedores precisam entender, devido à sua natureza fracamente opinativa.

O Angular exige que os desenvolvedores tenham uma compreensão completa de sua estrutura interna, a fim de utilizar plenamente todas as suas características. Isso implica que Angular tem um baixo nível de abstração. Por outro lado, o React essencialmente esconde todos os detalhes do funcionamento interno de suas funcionalidades por trás de suas bibliotecas e APIs, e os desenvolvedores raramente precisam aprofundar seus códigos. De forma semelhante, o Knockout não exige uma estrutura específica para o desenvolvimento.

O Angular e Knockout, por padrão, são usados apenas para o desenvolvimento de aplicações para web. Se os desenvolvedores desejarem usá-los para aplicativos móveis, é necessário usar uma biblioteca de terceiros, como a Ionic, para permitir que aplicativos móveis executem códigos do Angular e do Knockout. React, no entanto, permite que os códigos sejam executados tanto em navegadores da Web como em aplicativos móveis através do React Native, uma extensão de sua estrutura web. Isso significa que os desenvolvedores podem escrever uma única estrutura de código do React e executá-la na Web e em dispositivos móveis com modificações de código mínimas e com a eficiência e otimização do código nativo.

6. Conclusão

O desenvolvimento do presente trabalho possibilitou uma análise mais profunda sobre aplicações web e suas ferramentas. Há diferenças significativas nos resultados encontrados, visto que para uma mesma aplicação obteve-se resultados diferentes entre o framework e as bibliotecas testadas. Além disso, este trabalho permitiu identificar aspectos que impactam diretamente no desempenho de uma aplicação, como o custo de processamento, o tamanho dos arquivos que o *browser* deve carregar e a eficiência, que envolve a quantidade de código necessário para executar a mesma aplicação.

As diferenças entre o framework e as bibliotecas ficam claras por meio do código implementado para representar uma mesma aplicação. JavaScript puro, por exemplo, não apresenta uma arquitetura rígida como os outros, assim sua flexibilidade se torna maior. Porém, em termos de isolamento e modularidade é o menos eficiente. O React se mostrou mais modular devido ao fato de ser mais legível, possuir melhor manutenção e apresentar desempenho superior.

Foi identificado como a manipulação do DOM influencia no desempenho final da mesma aplicação. O DOM representa como todos os documentos são organizados em uma estrutura. Dessa forma, o React, ao utilizar o Virtual DOM em vez do DOM real, apresentou um desempenho superior aos outros. O JavaScript puro demonstrou um bom desempenho, entretanto, como não possui funções que podem ser estendidas, como uma biblioteca, tende a aumentar a quantidade de código rapidamente. O Angular, por ser um framework mais completo, não exige uma quantidade de código expressiva para rodar, porém, a curva de aprendizado é a mais íngreme em relação aos outros. Já no caso do Knockout, este levou um tempo mais longo para executar a mesma aplicação no *browser* Firefox, e no geral foi o mais lento.

Portanto, se o desempenho for a principal motivação, o React demonstra ser a melhor opção para uma estrutura de aplicação web, especialmente para grandes quantidades de dados ou no caso de haver manipulação excessiva de dados com o DOM. Se o tempo de desenvolvimento for a principal motivação, o Angular se mostra como mais ágil, por apresentar uma estrutura em que não é necessária uma grande quantidade de código para o desenvolvimento em comparação com as bibliotecas testadas e JavaScript puro.

6.1 Trabalhos futuros

Seguem abaixo algumas sugestões para trabalhos futuros:

- Desenvolvimento com outro framework ou biblioteca que manipula o DOM de uma forma diferente das mencionadas neste trabalho, como por exemplo o VueJS, que utiliza o Shadow DOM.
- Extensão da aplicação para poder avaliar outros critérios além do desempenho, modularidade, flexibilidade e custo.

Bibliografia

BERNERS-LEE, Tim. **Information Management: A Proposal - W3C**. Março, 1989.

CORMODE, Graham e KRISHNAMURTHY, Balachander. **Key differences between Web 1.0 and Web 2.0.** Junho, 2008. Disponível em <http://firstmonday.org/article/view/2125/1972> Primeiro acesso em: 12/08/2016.

Documentação do React.
<https://facebook.github.io/react/docs/hello-world.html> Primeiro acesso em: 10/04/2017

Documentação do Knockout.
<http://knockoutjs.com/documentation/introduction.html> Primeiro acesso em: 10/04/2017

GINIGE, Athula. **Consolidating Web Engineering as a Discipline**. Abril, 2002.

HENDRICKSON, Elizabeth e FOWLER, Martin. **The Software Engineering of Internet Software.** Março, 2002.

HAVERBEKE, Marijn. **Eloquent Javascript - 2° edição**. Março, 2014.

JQUERY COMMUNITY EXPERTS. **JQuery Cookbook - 1° edição**. Novembro, 2009.

KERER, Clemens e KIRDA, Engin. **A Methodology for XML-based Web Engineering with Components and Aspects.** 2001.

MUNRO, Jamie. Knockout.js - **Buinding Dynamic client-side web applications.** Janeiro, 2015

OSMANI, Addy. **Learning JavaScript Design Patterns**. Disponível em : <https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html> Primeiro acesso em: 24/09/2016.

PRESSMAN, Roger S. **Engenharia de software**. 2006.

PANO, Amantia , GRAZIOTIN, Daniel e ABRAHAMSSON, Pekka. **What leads developers towards the choice of a JavaScript framework?** <https://www.researchgate.net/publication/303221742_What_leads_developers_towards_the_choice_of_a_JavaScript_framework> Primeiro acesso em: 11/06/2016.

RESIG, John. **The secrets of JavaScript Ninja - 1° edição.** Abril, 2013.

RESIG, John. **PRO JavaScript Techniques - 1° edição**. 2006.

RESIG, John. **JQuery in Action**. 2008.

SESHADRI, Shyam e GREEN, Brad. **Desenvolvendo com AngularJs**. Novembro, 2014.

SAMY SILVA, Maurício. **JavaScript, guia do programador - 1° edição.** Setembro, 2010.

TAKADA, Mikito. **Single page app book**. Disponível em : <http://singlepageappbook.com/goal.html> Primeiro acesso em: 01/05/2016.

ZAKAS, Nicholas. **Professional JavaScript for Web Developers - 3° edição.** Janeiro, 2012.

APÊNDICE A – ARTIGO

Avaliação do framework Angular e das bibliotecas React e Knockout para desenvolvimento do Frontend de uma aplicação Web.

Alexandre Cechinel
Departamento de Informática e Estatística, UFSC.
Campus Reitor João D. F. Lima, Trindade – CEP 88040-970, Florianópolis (SC) – Brasil.

Abstract. *Originally thought to aid Web development, JavaScript frameworks have become very popular among developers for Web application builds as Web pages have grown to become full-blown client-side applications.The present work focused on evaluating a framework and two JavaScript libraries, as well as comparing them with pure JavaScript, as well as identifying the reasons that motivate developers to choose a Frontend tool.*

Resumo. Originalmente pensados para auxiliar no desenvolvimento Web, frameworks JavaScript se tornaram muito populares entre desenvolvedores para construções de aplicações Web na medida em que as páginas Web cresceram e se tornaram aplicações completas ao lado do cliente.O presente trabalho se concentrou em avaliar um framework e duas bibliotecas JavaScript, e também compará-las com JavaScript puro, bem como, identificar as razões que impulsionam os desenvolvedores para a escolha de uma ferramenta Frontend.

PALAVRAS-CHAVE: Web, JavaScript, HTML, CSS, Framework, Biblioteca, Single page applications, Frontend, Angular, React, Knockout.

Introdução

Desenvolvimento Web refere-se ao processo de construção e testes do software específico para a Web, com a finalidade de obter-se um conjunto de programas, que satisfazem as funções pretendidas, quer em termos de usabilidade dos usuários ou compatibilidade com outros programas existentes, podendo variar desde simples páginas estáticas a aplicações ricas, comércios eletrônicos ou redes sociais (SAMY SILVA, 2010). Neste contexto, este trabalho identifica alguns problemas existentes na área de desenvolvimento Web e as soluções mais utilizadas.

Muitos aspectos relativos ao desenvolvimento Web o tornam diferente do desenvolvimento de um software tradicional. Temos a escalabilidade e a necessidade das mudanças contínuas em seu conteúdo como os dois atributos-chave (GINIGE, 2001). De extrema relevância temos a multidisciplinaridade, e entre as características críticas temos a segurança, a usabilidade e também a manutenabilidade.

Esse tipo de sistema não exige que a sua execução seja realizada mediante sua prévia instalação local, assim como é feito com aplicações *desktop*. Muito pelo

contrário, a execução da grande maioria de aplicações Web acontece somente com o uso de um navegador Web como, por exemplo, o Mozilla Firefox. Aplicações Web são sistemas que possuem um alto grau de interação (KERER e KIRDA, 2001), além de atender simultaneamente diversos usuários, distribuídos em locais distintos fisicamente com a necessidade de disponibilização contínua e rápida de tais aplicações (HENDRICKSON e FOWLER, 2002).

Conceitos fundamentais

Em 1989, um pequeno grupo de pessoas liderado por Tim Berners-Lee no CERN (Organização Europeia para a Pesquisa Nuclear) propôs um novo protocolo para a internet, bem como um sistema de documentos de acesso para usá-lo. A intenção desse novo sistema, que o grupo chamou de *World Wide Web*, era permitir que cientistas do mundo todo usassem a internet para trocar documentos descrevendo seus trabalhos (SEBESTA, 2012).

O novo sistema proposto foi projetado para permitir que um usuário, em qualquer lugar, conectado à internet, possa pesquisar e recuperar documentos em qualquer número de diferentes computadores conectados à internet (SEBESTA, 2012).

A Web é um sistema de informação, baseado em documentos de hipertexto interligados, construído sobre a infraestrutura de comunicação provida pela Internet. Servidores gerenciam a maioria dos processos e armazenam dados. Um cliente solicita dados ou processos específicos. A saída do processo acontece do servidor para o cliente. Os clientes, por vezes, lidam com o processamento, mas exigem recursos de dados do servidor para a conclusão (SEBESTA, 2012).

2.1 Modelo cliente-servidor

Documentos fornecidos pelos servidores na web são solicitados pelos navegadores. O navegador é um cliente na Web porque ele inicia a comunicação com o servidor, que aguarda um pedido do cliente antes de fazer qualquer coisa. No caso mais simples, um navegador faz o pedido de um documento estático ao servidor. O servidor localiza o documento entre os seus documentos veiculáveis e envia para o navegador, que exibe para o usuário. No entanto situações mais complicadas são comuns.

2.2 Web 2.0

A Web 2.0 geralmente refere-se a um conjunto de políticas sociais, arquiteturais e padrões de design, resultando em uma migração em massa de negócios para a internet como uma plataforma. Esses padrões focam na interação de modelos entre comunidades, pessoas, computadores e software. Interações Humanas são um importante aspecto da arquitetura do software e, mais especificamente, do conjunto de sites e aplicações baseadas na Web construídas em torno do núcleo de um conjunto de padrões de design que mistura a experiência humana com a tecnologia (GOVERNOR, HINCHCLIFFE, NICKULL, 2009).

2.3 JavaScript

Quando o JavaScript apareceu pela primeira vez em 1995, seu objetivo principal era manipular algumas das validações de entrada que haviam sido deixadas anteriormente para linguagens do lado do servidor, como o Perl. Era necessária uma chamada ao servidor para determinar se um campo necessário havia sido deixado em branco ou se um valor inserido era inválido. O navegador

Netscape procurou alterar isso com a introdução do JavaScript. A capacidade de lidar com alguma validação básica no cliente foi um novo recurso emocionante em um momento em que o uso de *modens* telefônicos era generalizado. As velocidades lentas associadas transformaram cada chamada ao servidor em um exercício de paciência (ZAKAS, 2012).

A ascenção do JavaScript de um validador de entrada simples para uma poderosa linguagem de programação não poderia ter sido prevista. JavaScript é ao mesmo tempo uma linguagem muito simples e muito complicada que leva minutos para aprender, porém anos para dominar (ZAKAS, 2012).

Framework para Desenvolvimentos de Aplicações Web

O JavaScript percorreu um longo caminho, partindo de uma simples linguagem de scripting, usada somente para realizar validações menos significativas, até se tornar uma linguagem de programação completa. A JQuery realizou boa parte do trabalho de base para garantir a compatibilidade com os navegadores, oferecendo uma API sólida e estável para trabalhar com todos eles e interagir com o DOM. À medida que a complexidade e o tamanho das aplicações aumentaram, a JQuery, que é uma camada de manipulação de DOM, tornou-se insuficiente para oferecer, por si só, um framework sólido, modular, testável e de fácil compreensão para o desenvolvimento de aplicações. Cada projeto JQuery era totalmente diferente um do outro (SESHADRI, GREEN, 2014).

Desenvolvimento

Está seção trata do desenvolmento da aplicação de exemplo para conferir os estilos e diferenças de implementação entre o framework Angular e das bibliotecas React e Knockout. Para fins de comparação, JavaScript puro também é utilizado. Será especificado o que cada um tem de exclusivo e suas principais funcionalidades que auxiliam o desenvolvedor.

As versões utilizadas foram: AngularJS - 1.5.8, React - 15.16.1 e Knockout - 3.2.0.

Aplicação de teste

Para a aplicação de teste criou-se um algoritmo que gera um array de objetos (que pode ser alterado de tamanho), cujos valores são determinados aleatoriamente com base em valores contidos em três arrays diferentes, para verificar o desempenho em pequenos e grandes arrays. Os itens serão renderizados em uma *tag* não ordenada (
) e, ao mesmo tempo, será verificada a hora local exata com o objeto *Date()* do JavaScript. Ao final da renderização nos respectivos framework e bibliotecas, será chamado novamente o objeto *Date()* para calcular a diferença de tempo entre o início da renderização e o seu término, como demonstrado na Figura 1.

Figura 1. Código de construção do array de objetos.

```
6
7       // função que constroí o array que será renderizado
8       function run(size) {
9           // quantidade de elementos criados
10          size = size || 200;
11
12          // arrays de elementos
13          var citys = ['Florianópolis', 'São Paulo', 'Porto Alegre', 'Palhoça', 'Florença', 'Roma', 'Mendoza', 'Córdoba',
14                       'Blumenau', 'Criciúma', 'Barcelona', 'Londres', 'Paris', 'Xangai', 'Osaka', 'Pequin',
15                       'Salvador', 'Los Angeles', 'Las Vegas', 'Pato Branco', 'Curitiba', 'Buenos Aires', 'Manaus',
16                       'Nápoles', 'Porto'];
17          var states = ['Santa Catarina', 'Paraná', 'Bahia', 'São Paulo', 'Amazonas', 'Mato Grosso', 'Tocantins',
18                        'Pernambuco', 'Paraíba', 'Roraima', 'Acre'];
19          var countrys = ['Brasil', 'Argentina', 'Chile', 'Itália', 'Peru', 'Bolívia', 'França', 'Espanha', 'China',
20                      'Escócia', 'Irlanda', 'Grécia', 'Austrália'];
21
22          var data = [];
23
24          // o elementos são randomicamente adicionados no array data
25          for (var i = 0; i < size; i++)
26              data.push({ id: i + 1,
27                  label: citys[random(citys.length)] + " | " +
28                         states[random(states.length)] + " | " +
29                         countrys[random(countrys.length)] });
30
31          return data;
32      }
33
34      // função recebe a quantidade de elementos que irá criar
35      function random(max) {
36          return Math.round(Math.random() * 1000) % max;
37      }
```

Embora as diferenças entre Angular, React e Knockout sejam grandes, ambos podem realizar o mesmo. Angular possui uma estrutura muito mais completa do que o React e Knockout, mas isso não significa muito quando não há a necessidade da maioria dos recursos que o Angular fornece.

O tempo de execução foi apurado em três diferentes *browsers*: Chrome, Firefox e Safari. O algoritmo utilizado para teste gera um array de 1000 objetos em que é calculado o tempo desde o início da geração do array até a visualização do usuário no *browser.* Para cada execução é anotado o tempo e calculada a média geral, representada na Tabela 1. Foram feitas 10 execuções do algoritmo.

Tabela 1: Cálculo desvio padrão e média.

Chrome - Versão 60.0.3112.113		**Angular**	**React**	**Knockout**	**JS**
	Desvio padrão	51,81400302	31,70173497	79,99295424	35,76895556
	Média	307 ms	82 ms	468 ms	102 ms
Firefox - Versão 55.0.2		**Angular**	**React**	**Knockout**	**JS**
	Desvio padrão	75,40822236	23,50009671	41,46170193	27,94410004
	Média	440 ms	105 ms	801 ms	111 ms
Safari - Versão 11.0		**Angular**	**React**	**Knockout**	**JS**
	Desvio padrão	10,56753175	7,462634192	15,22557657	14,86117578
	Média	107 ms	29 ms	168 ms	57 ms

Com a medição do desvio padrão é possível observar que o React tem uma menor variação de acordo com a média nos três *browsers* testados, além de ter uma menor média de tempo de execução.

O Angular é fortemente opinativo, o que significa que sua estrutura e regras são todas adaptadas para se adequar rigidamente a uma maneira específica de programação. Ele impõe muitas regras e estruturas complexas que os

desenvolvedores precisam aprender. Isso pode resultar em curvas de aprendizado consideravelmente íngremes e pode levar muito tempo antes que seja possível iniciar o desenvolvimento de um produto. Em contraste, o React e o Knockout são significativamente mais fáceis para aprender, pois existem conceitos e estruturas menos complexas que os desenvolvedores precisam entender, devido à sua natureza fracamente opinativa.

O Angular exige que os desenvolvedores tenham uma compreensão completa de sua estrutura interna, a fim de utilizar plenamente todas as suas características. Isso implica que Angular tem um baixo nível de abstração. Por outro lado, o React essencialmente esconde todos os detalhes do funcionamento interno de suas funcionalidades por trás de suas bibliotecas e APIs, e os desenvolvedores raramente precisam aprofundar seus códigos. De forma semelhante, o Knockout não exige uma estrutura específica para o desenvolvimento.

O Angular e Knockout, por padrão, são usados apenas para o desenvolvimento de aplicações para web. Se os desenvolvedores desejarem usá-los para aplicativos móveis, é necessário usar uma biblioteca de terceiros, como a Ionic, para permitir que aplicativos móveis executem códigos do Angular e do Knockout. React, no entanto, permite que os códigos sejam executados tanto em navegadores da Web como em aplicativos móveis através do React Native, uma extensão de sua estrutura web. Isso significa que os desenvolvedores podem escrever uma única estrutura de código do React e executá-la na Web e em dispositivos móveis com modificações de código mínimas e com a eficiência e otimização do código nativo.

Conclusão

O desenvolvimento do presente trabalho possibilitou uma análise mais profunda sobre aplicações web e suas ferramentas. Há diferenças significativas nos resultados encontrados, visto que para uma mesma aplicação obteve-se resultados diferentes entre o framework e as bibliotecas testadas. Além disso, este trabalho permitiu identificar aspectos que impactam diretamente no desempenho de uma aplicação, como o custo de processamento, o tamanho dos arquivos que o *browser* deve carregar e a eficiência, que envolve a quantidade de código necessário para executar a mesma aplicação.

As diferenças entre o framework e as bibliotecas ficam claras por meio do código implementado para representar uma mesma aplicação. JavaScript puro, por exemplo, não apresenta uma arquitetura rígida como os outros, assim sua flexibilidade se torna maior. Porém, em termos de isolamento e modularidade é o menos eficiente. O React se mostrou mais modular devido ao fato de ser mais legível, possuir melhor manutenção e apresentar desempenho superior.

Foi identificado como a manipulação do DOM influencia no desempenho final da mesma aplicação. O DOM representa como todos os documentos são organizados em uma estrutura. Dessa forma, o React, ao utilizar o Virtual DOM em vez do DOM real, apresentou um desempenho superior aos outros. O JavaScript puro demonstrou um bom desempenho, entretanto, como não possui funções que podem ser estendidas, como uma biblioteca, tende a aumentar a quantidade de código rapidamente. O Angular, por ser um framework mais completo, não exige uma quantidade de código expressiva para rodar, porém, a curva de aprendizado é a mais íngreme em relação aos outros. Já no caso do Knockout, este levou um tempo mais longo para executar a mesma aplicação no *browser* Firefox, e no geral foi o mais lento.

Portanto, se o desempenho for a principal motivação, o React demonstra ser a melhor opção para uma estrutura de aplicação web, especialmente para grandes quantidades de dados ou no caso de haver manipulação excessiva de dados com o DOM. Se o tempo de desenvolvimento for a principal motivação, o Angular se mostra como mais ágil, por apresentar uma estrutura em que não é necessária uma grande quantidade de código para o desenvolvimento em comparação com as bibliotecas testadas e JavaScript puro.

Referências

BERNERS-LEE, Tim. **Information Management: A Proposal - W3C**. Março, 1989.

CORMODE, Graham e KRISHNAMURTHY, Balachander. **Key differences between Web 1.0 and Web 2.0.** Junho, 2008. Disponível em <http://firstmonday.org/article/view/2125/1972> Primeiro acesso em: 12/08/2016.

Documentação do React.
<https://facebook.github.io/react/docs/hello-world.html> Primeiro acesso em: 10/04/2017

Documentação do Knockout.
<http://knockoutjs.com/documentation/introduction.html> Primeiro acesso em: 10/04/2017

GINIGE, Athula. **Consolidating Web Engineering as a Discipline**. Abril, 2002.

HENDRICKSON, Elizabeth e FOWLER, Martin. **The Software Engineering of Internet Software.** Março, 2002.

HAVERBEKE, Marijn. **Eloquent Javascript - 2° edição**. Março, 2014.

JQUERY COMMUNITY EXPERTS. **JQuery Cookbook - 1° edição**. Novembro, 2009.

KERER, Clemens e KIRDA, Engin. **A Methodology for XML-based Web Engineering with Components and Aspects.** 2001.

MUNRO, Jamie. Knockout.js - **Buinding Dynamic client-side web applications.** Janeiro, 2015

OSMANI, Addy. **Learning JavaScript Design Patterns**. Disponível em : <https://addyosmani.com/resources/essentialjsdesignpatterns/book/index.html> Primeiro acesso em: 24/09/2016.

PRESSMAN, Roger S. **Engenharia de software**. 2006.

PANO, Amantia , GRAZIOTIN, Daniel e ABRAHAMSSON, Pekka. **What leads developers towards the choice of a JavaScript framework?** <https://www.researchgate.net/publication/303221742_What_leads_developers_towards_the_choice_of_a_JavaScript_framework> Primeiro acesso em: 11/06/2016.

RESIG, John. **The secrets of JavaScript Ninja - 1° edição.** Abril, 2013.

RESIG, John. **PRO JavaScript Techniques - 1° edição**. 2006.

RESIG, John. **JQuery in Action**. 2008.

SESHADRI, Shyam e GREEN, Brad. **Desenvolvendo com AngularJs**. Novembro, 2014.

SAMY SILVA, Maurício. **JavaScript, guia do programador - 1° edição.** Setembro, 2010.

TAKADA, Mikito. **Single page app book**. Disponível em : <http://singlepageappbook.com/goal.html> Primeiro acesso em: 01/05/2016.

ZAKAS, Nicholas. Professional JavaScript for Web Developers - 3° edição. Janeiro, 2012.

ANEXO A – CÓDIGO DA APLICAÇÃO

index.html

```
<!DOCTYPE html>
<html ng-app="app">
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <title>TCC</title>
      <link href="libs/bootstrap.css" rel="stylesheet" />
      <link rel="stylesheet" type="text/css" href="style.css"></link>
      <!-- LIBS -->
      <script type="text/javascript" src="libs/knockout-min.js"></script>
      <script type="text/javascript" src="libs/angular.min.js"></script>
      <script type="text/javascript" src="libs/react.min.js"></script>
      <!-- SCRIPTS -->
      <script type="text/javascript" src="angular.js"></script>
      <script type="text/javascript" src="react.js"></script>
      <script type="text/javascript" src="knockout.js"></script>
      <script type="text/javascript" src="javascript.js"></script>
   </head>

   <body ng-controller="controller">
      <div class="container-fluid">
         <div class="row">
            <div class="col-md-12">
               <h2>Tempo de renderização com React, Angular, Knockout e JS</h2>
            </div>
         </div>

         <!-- REACT -->
         <div class="col-md-3 band">
            <div class="row">
               <div class="col-md-7">
                  <h3>React</h3>
               </div>
               <div class="col-md-5 text-right time" id="run-react">Iniciar</div>
            </div>
            <div id="react"></div>
         </div>

         <!-- ANGULAR -->
         <div class="col-md-3 band2">
            <div class="row">
               <div class="col-md-7">
                  <h3>Angular</h3>
```

```
                </div>
    <div class="col-md-5 text-right time" id="run-angular" ng-click="initialize()">Iniciar</div>
                </div>
                <div>
                    <div class="row" ng-repeat="item in data">
                        <div class="col-md-12 test-data">
                            <span ng-class="{ selected: item.id === $parent.selected }" ng-
click="select(item)">
                                {{item.label}}
                            </span>
                        </div>
                    </div>
                </div>
            </div>

            <!-- KNOCKOUT -->
            <div id="knockout" class="col-md-3 band">
                <div class="row">
                    <div class="col-md-7">
                        <h3>Knockout</h3>
                    </div>
                    <div class="col-md-5 text-right time" id="run-knockout" data-bind="click: run">
                        Iniciar
                    </div>
                </div>
                <div data-bind="foreach: data">
                    <div class="row">
                        <div class="col-md-12 test-data">
                            <span data-bind="click: $root.select.bind($root, $data),
                                text: $data.label, css: { selected: $data.id === $root.selected() }">
                            </span>
                        </div>
                    </div>
                </div>
            </div>

            <!-- JAVASCRIPT PURO -->
            <div class="col-md-3 band2">
                <div class="row">
                    <div class="col-md-7">
                        <h3>JS puro</h3>
                    </div>
                    <div class="col-md-5 text-right time" id="run-raw">
                        Iniciar
                    </div>
                </div>
                <div id="raw"></div>
            </div>
```

```
    </div>
    <script type="text/html" id="raw-template">
      <div class="row">
        <div class="col-md-12 test-data">
          <span class="{{className}}">{{label}}</span>
        </div>
      </div>
    </script>
  </body>
</html>
```

index.js

```
function run(size) {
    size = size || 200;
    var citys = ['Florianópolis', 'São Paulo', 'Porto Alegre', 'Palhoça', 'Florença', 'Roma',
'Mendoza', 'Córdoba', 'Blumenau', 'Criciúma', 'Barcelona', 'Londres', 'Paris', 'Xangai', 'Osaka',
'Pequin', 'Salvador', 'Los Angeles', 'Las Vegas', 'Pato Branco', 'Curitiba', 'Buenos Aires',
'Manaus',  'Nápoles', 'Porto'];
    var states = ['Santa Catarina', 'Paraná', 'Bahia', 'São Paulo', 'Amazonas', 'Mato Grosso',
'Tocantins',  'Pernambuco', 'Paraíba', 'Roraima', 'Acre'];
    var countrys = ['Brasil', 'Argentina', 'Chile', 'Itália', 'Peru', 'Bolívia', 'França', 'Espanha',
'China',  'Escócia', 'Irlanda', 'Grécia', 'Austrália'];

    var data = [];
    for (var i = 0; i < size; i++)
      data.push({ id: i + 1,
        label: citys[random(citys.length)] + " | " +
          states[random(states.length)] + " | " +
          countrys[random(countrys.length)] });

    return data;
  }

  function random(max) {
    return Math.round(Math.random() * 1000) % max;
  }
```

angular.js

```
  function _angular(data) {
    angular.module('app', []).controller('controller', function($scope) {

      $scope.run = function() {
        var data = run(),
          date = new Date();
```

```
            $scope.selected = null;
            $scope.$$postDigest(function() {
            document.getElementById('run-angular').innerHTML = (new Date() - date) + ' ms';
            });

            $scope.data = data;
        };

        $scope.select = function(item) {
            $scope.selected = item.id;
        };
    });
}
```

react.js

```
function _react() {
    var Class = React.createClass({
        select: function(data) {
            this.props.selected = data.id;
            //this.forceUpdate();
        },
        render: function() {
            var items = [];
            for (var i = 0; i < this.props.data.length; i++) {
                items.push(React.createElement("div", { className: "row" },
                    React.createElement("div", { className: "col-md-12 test-data" },
                        React.createElement("span", {
                            className: this.props.selected === this.props.data[i].id ? "selected" : "",
                            onClick: this.select.bind(null, this.props.data[i]) }, this.props.data[i].label)
                    )
                ));
            }
            return React.createElement("div", null, items);
        }
    });
    var runReact = document.getElementById("run-react");
    runReact.addEventListener("click", function() {
        var data = run(),
            date = new Date();
        React.render(new Class({ data: data, selected: null }),
document.getElementById("react"));
        runReact.innerHTML = (new Date() - date) + " ms";
    });
}
```

knockout.js

```
 function _knockout() {
        ko.applyBindings({
            selected: ko.observable(),
            data: ko.observableArray(),

            select: function(item) {
                this.selected(item.id);
            },

            run: function() {
                var data = run(),
                    date = new Date();

                this.selected(null);
                this.data(data);
                document.getElementById("run-knockout").innerHTML = (new Date() - date) + "
ms";
            }
        }, document.getElementById("knockout"));
    }

    ko.observableArray.fn.reset = function(values) {
        var array = this();
        this.valueWillMutate();
        ko.utils.arrayPushAll(array, values);
        this.valueHasMutated();
    };
```

javascript.js

```
 function _raw() {
        var container = document.getElementById("raw"),
            template = document.getElementById("raw-template").innerHTML;

        document.getElementById("run-raw").addEventListener("click", function() {
            var data = run(),
                date = new Date(),
                html = "";

            for (var i = 0; i < data.length; i++) {
                var render = template;
                render = render.replace("{{className}}", "");
                render = render.replace("{{label}}", data[i].label);
                html += render;
            }
```

```
        container.innerHTML = html;

        var spans = container.querySelectorAll(".test-data span");

        for (var i = 0; i < spans.length; i++)
            spans[i].addEventListener("click", function() {
                var selected = container.querySelector(".selected");
                if (selected)
                    selected.className = "";
                this.className = "selected";
            });

        document.getElementById("run-raw").innerHTML = (new Date() - date) + " ms";
    });
}
```

style.css

```
* {
        box-sizing:border-box;
}

body {
        padding:30px 0;
}

h2 {
        margin:0;
        margin-bottom:25px;
        text-align: center;
}

h3 {
        margin:0;
        padding:0;
        margin-bottom:12px;
}

.band {
        background-color: #b3c4e0;
}

.band2 {
        background-color: #d3dded;
}

.test-data {
        margin-bottom:3px;
```

```
}

.test-data span {
	padding:3px 10px;
	background:#EEE;
	width:100%;
	float:left;
	cursor:pointer;
}

.test-data span:hover {
	background:#DDD;
}

.test-data span.selected {
	background: #5b94ef; color:white;
}

.time {
	font-weight:bold;
	height:26px;
	line-height:26px;
	vertical-align:middle;
	display:inline-block;
	cursor:pointer;
	text-decoration:underline;
}
```