

**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA
SOUZA**

ETEC DR. DEMÉTRIO AZEVEDO JÚNIOR

Técnico em Informática Integrado ao Ensino Médio

CAMARGO, Vitor Bueno de

FERNANDES, Tais Vitória

FRAMEWORK: Meteor.

Itapeva

2016

CAMARGO, Vitor Bueno de
FERNANDES, Tais Vitória

FRAMEWORK: Meteor.

Trabalho de Conclusão de Curso apresentado ao Curso Técnico em Informática Integrado ao Ensino Médio da Etec Dr. Demétrio Azevedo Júnior orientado pelo Prof. Charles Andrei Fabri Proença, Prof. Felipe Augusto de Oliveira e pela Prof.^a. Ana Paula Siqueira Santos de Oliveira, como requisito parcial para obtenção do título de técnico em Informática.

Itapeva

2016

CAMARGO, Vitor Bueno de
FERNANDES, Tais Vitória

FRAMEWORK: Meteor.

Etec Dr. Demétrio Azevedo Júnior

Data: 16 de novembro de 2016

Resultado: _____

COMISSÃO AVALIADORA

Prof. Aquiles Felizardo Filho

Assinatura: _____

Prof. Felipe Augusto de Oliveira

Assinatura: _____

Prof. Danilo Dias Camargo

Assinatura: _____

Itapeva

2016

A todos aqueles que de forma direta e indireta nos ajudaram nesta caminhada, cada um, de forma especial e carinhosa nos deram força e coragem, nos apoiando nos momentos de dificuldades. E a todos aqueles que de alguma forma estiveram e estão próximos de nós, fazendo nossas vidas valerem cada vez mais a pena.

AGRADECIMENTOS

Agradecemos primeiramente a Deus que permitiu que tudo isso acontecesse e por nos dar essa oportunidade. A todos os professores, por sempre estarem dispostos para esclarecimentos e suporte, entre eles, um carinho especial para o nosso professor Felipe Oliveira que nos guiou durante grande parte da nossa trajetória que, sempre mostrou disposição para nos orientar da melhor forma possível e também à professora Ana Paula e o professor Charles Fabri, pela paciência na orientação e incentivo ao nosso trabalho. Agradecemos também ao professor Aquiles Felizardo Filho que em momentos difíceis se dispôs a ajudar, corrigir e sugerir correções ao nosso trabalho.

Somos gratos também à professora Maira Báz e o coordenador do curso Danilo Dias, que se dispuseram a corrigir e sugerir dicas para o aprimoramento deste trabalho.

De maneira muito carinhosa, agradecemos a todos os nossos amigos e pessoas próximas que nos deram apoio para a finalização deste trabalho, em especial à Ana Modenezi que deu todo auxílio necessário.

*“The function of good software is to make
the complex appear to be simple.”*

GRADY BOOCH

RESUMO

Este trabalho apresenta o conceito sobre *frameworks*, abordando desde os primeiros softwares, a sua finalidade, seu contexto histórico, vantagens e desvantagens do uso dessas ferramentas e exemplos de *frameworks* utilizados atualmente, destacando, o *framework Meteor*, – tecnologia que está se tornando ainda mais popular ao decorrer dos anos desde sua criação – assunto no qual, também é apresentado suas vantagens e desvantagens, sua finalidade, sua importância, sua história, seu uso dentro das empresas e seu impacto sobre a produtividade dos programadores que o utilizam, demonstrando então, a evolução da mesma. Esta tecnologia, está em desenvolvimento constante, buscando aprimorar sempre mais a conexão entre o desempenho do programa e o tempo gasto para produzi-los, proporcionando ao usuário uma experiência de desenvolvimento amplo, claro, rápido e que gere resultados adequados. Para demonstrar esse conceito vinculado nas qualificações do *Meteor*, há a necessidade de uma atividade prática que mostre todas as suas ferramentas e funções, o qual é suprido neste trabalho. Ao final, com o auxílio do programa prático é possível comprovar que *Meteor* consegue suprir com a sua finalidade e apesar de seus defeitos é uma ferramenta que auxilia os programadores, proporcionando um tempo pequeno de implementação e resultados rápidos, concisos e profissionais. Transmitindo o consenso básico da maioria dos *frameworks*: a diminuição de tempo de codificação, menos linhas de código, mas com perda no desempenho.

Palavras-chaves: *Frameworks*. *Softwares*. Tecnologia. Programadores. Resultados.

ABSTRACT

This work aims presents the concept of frameworks, covering from the earliest softwares, its purpose, its historical context, advantages and disadvantages of using these tools and examples of frameworks used currently, highlighting, Meteor, framework – technology that is becoming even more popular over the years since its creation – in which, also are its advantages and disadvantages, their purpose , its importance, its history, its use within companies and its impact on the productivity of programmers that use it, demonstrating so, the evolution of the same. This technology is in constant development, always seeking to improve the connection between the performance of the program and the time taken to produce them, providing the user a broad development experience, of course, fast and generate appropriate results. To demonstrate this concept linked in the Meteor's qualifications, there is a need for a practical activity that shows all your tools and functions, which is supplied in this work. At the end, with the aid of practical program it is possible to prove that Meteor can meet with your purpose and despite its shortcomings is a tool that helps programmers, providing a small time of implementation and results fast, concise and professional. Transmitting the basic consensus of most frameworks: the decrease of coding time, fewer lines of code, but with a loss in performance.

Keywords: Frameworks. Softwares. Technology. Programmers. Results.

LISTA DE ILUSTRAÇÕES

Figura 1 - Estrutura do Zachman Framework.....	16
Figura 2 - Funcionamento do MEAN	21
Figura 3 - Logotipo do Meteor	24
Figura 4 - Site da ClassCraft	30
Figura 5 - Uso de Meteor na ClassCraft.....	30
Figura 6 - Modelo Conceitual	33
Figura 7 - Modelo lógico do banco de dados.....	34
Figura 8 - Estrutura do banco de dados	35
Figura 9 - Resultado final do sistema	35
Figura 10 - Exemplo dos templates.....	37
Figura 11 – Estrutura física dos templates	38
Figura 12 - Sistema de login.....	39
Figura 13 – Estrutura lógica dos templates	39

LISTA DE ABREVIATURAS E SIGLAS

API	Interface de Programação de Aplicações
App	Aplicativo
ASP	Active Server Pages
BD	Banco de Dados
CEFET-PR	Centro Federal Tecnológica do Paraná
CGI	Common Gateway Interface
CSS	Cascading Style Sheets
DER	Diagrama Entidade Relacionamento
DOM	Document Object Model
HTML	Hypertext Mark-up Language
IBM	International Business Machines
Inc.	Incorporation
IoC	Inversão de Controle
JS	JavaScript
JSON	JavaScript Object Notation
JSP	JavaServer Pages
JVM	Máquina Virtual Java
MDG	Meteor Development Group
MVC	Modelo-Visão-Controlador
NoSQL	Not Only SQL
ORM	Object-relational mapping
PHP	Hypertext Preprocessor
PUC-Rio	Pontifícia Universidade Católica do Rio de Janeiro
REST	Representacional State Transfer

SGBD	Sistema Gerenciador de Banco de Dados
SQL	Linguagem de Consulta Estruturada
TCC	Trabalho de Conclusão de Curso
TI	Tecnologia da Informação
TTM	Time-to-market
UFCG	Universidade Federal de Campina Grande
UFPE	Universidade Federal de Pernambuco
URL	Uniform Resource Locator

SUMÁRIO

1 INTRODUÇÃO	13
1.1 Objetivo Geral	14
1.2 Objetivos Específicos	14
2 FRAMEWORKS	15
2.1 Contexto Histórico	15
2.2 Finalidade	17
2.3 Vantagens	17
2.4 Desvantagens	18
2.5 Evolução	18
2.6 Exemplos De Frameworks	19
2.6.1 AngularJS	19
2.6.2 MEAN	20
3 METEOR	22
3.1 Contexto Histórico	23
3.2 Finalidade	25
3.3 Vantagens e Desvantagens	25
3.3.1 As Vantagens De Meteor Em Relação A Outros Frameworks	27
3.4 Desenvolvimento Dentro Das Empresas	29
3.5 Desenvolvimento Para Programadores	31
3.6 Meteor em Bancos Relacionais	31
4 DESENVOLVIMENTO	33
4.1 Cenário da Prática	33
4.2 Visão Geral Do Sistema	33
4.3 Funções Básicas	36

4.4 Funções Adicionais	40
5 CONCLUSÃO	42
REFERÊNCIAS.....	43

1 INTRODUÇÃO

Atualmente, com a importância da tecnologia, a forma de programar pelo método convencional se tornou obsoleta. Neste contexto, houve a necessidade de criar um software que transformasse o modo de programar, diminuindo o tempo de codificação e os códigos, aumentando sua produtividade e por consequência a sua eficiência.

Esses *softwares* são chamados de *frameworks*, geralmente esses programas atingem somente alguma linguagem específica dentro da área de TI (Tecnologia da Informação), porém, alguns desses *frameworks* conseguem revolucionar ainda mais a forma de programar. Exemplo disso é o *meteor.js* que, sozinho consegue manipular banco de dados, o *back-end* e o *front-end* de um projeto.

O presente Trabalho de Conclusão de Curso (TCC) tem como intuito demonstrar como *frameworks* podem de uma maneira geral facilitar a vida de um programador, mais concretamente, mostrar suas vantagens e desvantagens do uso de sua aplicação em diversos meios, como, páginas web responsivo, aplicativos *mobile* e banco de dados.

Está organizado em duas partes. Na primeira parte será abordado toda a pesquisa teórica com suas respectivas citações e autores. Na segunda parte será apresentada uma aplicação básica feita com a ajuda do *Meteor* para que fique visível todo conteúdo abordado na primeira parte deste trabalho.

Na segunda parte, foi realizada em cima de uma problemática encontrada, na qual, há a existência de um desenvolvedor sem conhecimento muito grande da linguagem Web e com a necessidade de criar algum site de baixo-médio porte em pouco tempo. Este site será um sistema elaborado de login e cadastramento de usuários. As possíveis soluções para este problema é que, com a implementação do *Meteor* no projeto do programador, ele consiga cumprir com seus objetivos, ou, o *framework* não seja de grande utilidade para ele, compensando a criação do *software* sem a ajuda do *Meteor*.

A metodologia utilizada será a pesquisa bibliográfica, enriquecida com diversos artigos, vídeos, palestras e livros, todos de alta qualidade, autenticidade e atuais. Contudo, será utilizada uma comparação prática entre um projeto já criado sem

nenhum uso de *frameworks* e com o mesmo projeto usando *framework* em destaque, mas também a comparação de projetos criados com 2 *frameworks* diferentes.

1.1 Objetivo Geral

Demonstrar de forma clara e concisa como, um *framework* é capaz de facilitar de diversas formas o modo de programar de desenvolvedores sendo eles, iniciantes ou experientes, em especial, o *framework meteor*.

1.2 Objetivos Específicos

- a) Promover a criação de um software que atenda às exigências da segunda parte da pesquisa teórica;
- b) Criar os devidos diagramas e ferramentas necessárias para o trabalho prático;
- c) Usar de forma clara e abrangente grande parte das funções do *framework: meteor.js*;
- d) Demonstrar as vantagens e desvantagens do uso de *frameworks*, especialmente, do *Meteor*;
- e) Promover uma ligação concisa do projeto prático e teórico que disponibilize ao leitor maior entendimento sobre o tema em questão;
- f) Construir uma aplicação concisa e com a maior quantidade de recursos disponíveis pela ferramenta utilizada para mostrar com clareza todas as informações apresentadas neste TCC.

2 FRAMEWORKS

Na literatura é possível encontrar várias definições para *frameworks*, mas definitivamente não são contraditórias, pelo contrário, todas elas se complementam.

Segundo Fayad, “[...] Um *framework* é um conjunto de classes que constitui um projeto abstrato para a solução de uma família de problemas” (FAYAD, 1999).

Já segundo Buschmann, Pree, Pinto e Johnson & Foote (1996, 1995, 2000, 1988 apud PUC-Rio [Pontifícia Universidade Católica do Rio de Janeiro], 2006) assinala que:

Um *framework* é definido como um *software* parcialmente completo projetado para ser instanciado. O *framework* define uma arquitetura para uma família de subsistemas e oferece os construtores básicos para criá-los. Também são explicitados os lugares ou pontos de extensão (*hot-spots*) nos quais adaptações do código para um funcionamento específico de certos módulos devem ser feitas.

Ou seja, *framework*, em concordância com as mais variadas definições representa um software que pode ser instanciado, um projeto afim de facilitar o modo de programar, diminuindo o tempo de codificação e de códigos.

2.1 Contexto Histórico

Na década de 1980, com a criação da expressão “*enterprise architecture*”¹, recém inventada pelo também desenvolvedor do primeiro *framework*, John Zachman, que recebe um título por iniciar a jornada dos *frameworks* que conhecemos hoje. Este, era baseado em uma espécie de vetor, com seis colunas e linhas. As colunas são relacionadas à descrição de organização e as linhas referem-se aos vários lados das informações que relatam como é a organização, tendo por fim, o objetivo de armazenar alguma informação relevante e específica.

De acordo com John é possível estabelecer conceitos referentes ao termo estudado por ele.

What: refere-se sobre o que a organização precisa de informação? Do que ela trata? Normalmente, essa coluna representa dados mantidos pela organização;

How: trata-se como a organização funciona? Como ela processa seus dados? Esta coluna normalmente refere-se a processos e funções da organização.







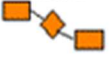
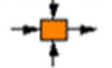




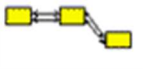
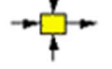
















Where: mostra onde as coisas acontecem? Aqui vão informações geográficas, de localização, entre outras;

¹ Em português: arquitetura corporativa.

Who: demonstra quem está na organização e quem faz o quê? Informações sobre pessoas e estruturas organizacionais estão aqui;
When: aborda quando as coisas acontecem? Questões relativas ao tempo aparecem aqui;
Why: denota o por que as coisas acontecem? Aqui vão as informações relativas às motivações da organização, incluindo seus planos estratégicos de negócio. (ZACHMAN, 2003).

Na imagem abaixo é possível entender melhor os conceitos referentes a expressão “enterprise architecture” relatada na citação anterior de Zachman, onde cada campo da imagem representa um termo da citação (What, how, where, Who, When e Why).

Figura 1 - Estrutura do Zachman Framework

abstractions perspectives	DATA <i>What</i>	FUNCTION <i>How</i>	NETWORK <i>Where</i>	PEOPLE <i>Who</i>	TIME <i>When</i>	MOTIVATION <i>Why</i>
SCOPE Planner contextual	List of Things - Important to the Business 	List of Processes - the Business Performs 	List of Locations - in which the Business Operates 	List of Organizations - Important to the Business 	List of Events - Significant to the Business 	List of Business Goals and Strategies 
ENTERPRISE MODEL Owner conceptual	e.g., Semantic Model 	e.g., Business Process Model 	e.g., Logistics Network 	e.g., Work Flow Model 	e.g., Master Schedule 	e.g., Business Plan 
SYSTEM MODEL Designer logical	e.g., Logical Data Model 	e.g., Application Architecture 	e.g., Distributed System Architecture 	e.g., Human Interface Architecture 	e.g., Processing Structure 	e.g., Business Rule Model 
TECHNOLOGY CONSTRAINED MODEL Builder physical	e.g., Physical Data Model 	e.g., System Design 	e.g., Technical Architecture 	e.g., Presentation Architecture 	e.g., Control Structure 	e.g., Rule Design 
DETAILED REPRESENTATIONS Subcontractor out-of-context	e.g. Data Definition 	e.g. Program 	e.g. Network Architecture 	e.g. Security Architecture 	e.g. Timing Definition 	e.g. Rule Specification 
FUNCTIONING ENTERPRISE	DATA Implementation	FUNCTION Implementation	NETWORK Implementation	ORGANIZATION Implementation	SCHEDULE Implementation	STRATEGY Implementation

Fonte: <http://www.icmgworld.com.asp>

Com a necessidade de uma tecnologia que abrangesse vários tipos de aplicações como: *mobile*, *web* e *desktop*, sendo criada na década de 1990, pelo grupo *Sun Microsystems*, a tecnologia *Java* se mostrou muito mais prático em relação aos outros, fazendo com que atualmente bilhões de desenvolvedores aderissem ao mesmo.

Com o crescimento da internet, as aplicações, atualmente, estão ganhando seu espaço dentro das empresas. Pensando nesse conceito, foram criados *frameworks* e ferramentas que aderiram a esse crescimento repentino e juntas buscam sempre produtividade, menos risco e tornaram o desenvolvimento de aplicações atraente para os desenvolvedores. (ZACHMAN, 2003).

Assim como a tecnologia Java, o uso dos frameworks se tornou cada vez maior, o início da jornada sobre os *frameworks* começa com o Zachman que utiliza a tecnologia Java para a criação e apesar de John Zachman não considerar seu trabalho como um *framework*, ele dá origem a uma linha de outros programas que surgiram após ele.

2.2 Finalidade

Uma das principais finalidades de um *framework* é o reuso. Ou seja, a reutilização de códigos, classes e funções disponíveis na linguagem padrão, como Java e JavaScript.

Para Pinto, “Além destas formas de reutilização, a tecnologia de *frameworks* possibilita que uma família de produtos seja gerada a partir de uma única estrutura que captura os conceitos mais gerais da família de aplicações”. (PINTO, 2000).

Um *framework* relata a estrutura de um sistema orientado a objetos, quais são os objetos e as relações entre os mesmos. Além disto, esses *softwares* possuem o objetivo de auxiliar no desempenho e desenvolvimento, ou seja, melhorar e facilitar a maneira em que o usuário programa.

2.3 Vantagens

Em concordância com Fayad (1999), a utilização de *frameworks* apresenta os seguintes benefícios:

Melhora a modularização – encapsulamento dos detalhes voláteis de implementação através de interfaces estáveis. Aumenta a reutilização – definição de componentes genéricos que podem ser replicados para criar novos sistemas.

Extensibilidade – favorecida pelo uso de métodos *hooks*² que permitem que as aplicações estendam interfaces estáveis. Inversão de Controle (IoC) – o código do desenvolvedor é chamado pelo código do *framework*. Dessa forma, o *framework* controla a estrutura e o fluxo de execução dos programas. (FAYAD, 1999).

Além disto, muitos outros autores e estudiosos, de acordo com suas experiências também escreveram características essenciais dos *frameworks*. Exemplo de Zemel (2009).

Utilidade: o objetivo primeiro dos *frameworks* é auxiliar no desenvolvimento de aplicações e *softwares*. Para tal, eles têm funcionalidades nativas das mais variadas, que ajudam você a

² São técnicas usadas para alterar ou aumentar o comportamento de um sistema.

resolver as questões sobre programação do dia-a-dia com muito mais qualidade e eficiência;

Segurança: os bons *frameworks* são projetados de modo a garantir a segurança de quem programa e, principalmente, de quem usa o que foi feito a partir dele. Não se preocupe mais com aquelas intermináveis linhas de código para evitar um SQL (Linguagem de Consulta Estruturada) *Injection*, por exemplo; com *frameworks*, a parte de segurança já “vem de fábrica”.

Economia de tempo: o que você demoraria algumas horas ou alguns dias para fazer, você encontra pronto em um *framework*. Pense nas quão trabalhosas aquelas funções de manipulação de imagens são, usando um *framework* que tenha isso, você só usa, e pronto.

Ajuda fácil. Os que desenvolvem *frameworks* geralmente disponibilizam material de qualidade na web sites ou repositórios oficiais, com uma vasta documentação a respeito. Além disso, os bons *frameworks* sempre têm uma comunidade de desenvolvedores dispostos a se ajudarem entre si. É um prazer para os que já sabem mexer ajudar os que ainda não sabem, embora a falta de tempo também seja uma realidade. (ZEMEL, 2009).

Por fim, trabalhar com *frameworks*, pode garantir ao programador a diminuição de tempo, uma vez que o mesmo saiba como utilizá-lo de forma correta. É extremamente proveitoso o uso desses softwares, principalmente, na construção de grandes projetos, pois quanto maior o software, o tempo e muitas vezes as dificuldades são muito maiores e o objetivo do *framework* é exatamente ajudar nesses pontos.

2.4 Desvantagens

Contudo, *frameworks* não consistem apenas em vantagens para os usuários, pelo contrário, eles também possuem defeitos e muitas vezes fazem com que o uso de certos *frameworks* não seja tão bom em relação ao método convencional de programação.

Construir um framework é complexo e o reuso não vem sozinho: ele deve ser planejado e bem organizado, muitas vezes é mais complexo e demora mais fazer uma aplicação tendo que construir ou usar um framework em vez de fazer a aplicação do zero. (LEITE, 2006, p.1).

Segundo a explicação de Alexandre Leite, existem casos de *frameworks* que necessitam que o programador o estude tanto que se torna inviável a sua utilização. Existe também casos de *frameworks* que fazem com que o programa perca muito mais desempenho e o programador gaste mais tempo do que em outros softwares.

2.5 Evolução

Toda a base remete à década de 80, é onde começa a evolução, com o *framework* de Zachman, organizado pelo 5W1H.

A grande evolução da tecnologia da informação nas décadas de 80 e 90, em conjunto com o processo de *downsizing*³ dos mainframes e o advento da computação pessoal, fez com que houvesse uma proliferação de sistemas de informação nas organizações, muito deles sendo feitos dentro dos próprios departamentos, no coração das áreas de negócio, sem grandes preocupações com interoperabilidade e reuso de funcionalidades comuns a diversos setores. Além disto, nesta época, a TI em muitos casos era vista como uma atividade secundária dentro da empresa e não tinha um papel determinante na execução do plano estratégico.

O pleno conhecimento de toda a complexidade que uma grande organização pode atingir só pode ser alcançado com a estruturação e definição de uma arquitetura completa da organização, ou seja, de uma arquitetura corporativa. Em tempos de globalização e grandes inovações, onde a tomada de decisão depende da informação, a arquitetura corporativa se torna fundamental na implantação de estratégias para aumentar ou manter as vantagens competitivas (SUL, 2014).

Apesar disto, anos depois, o *software* criado foi considerado ontológico, ou seja, de acordo com o próprio criador não é mais um *framework*.

Após isso, houve o aparecimento de diversos *frameworks* que tratam estas questões e possibilitam a organização da visão arquitetural com seus elementos e as suas relações. Os frameworks também proporcionam a organização das iniciativas de TI, uniformizam termos e linguagens organizacionais, aceleram o funcionamento da TI, definem a organização dos dados e construção de visões. Além disso, incluem o alinhamento estratégico da organização com seus processos de negócio, aplicações, dados e tecnologia. (SUL, 2014).

Após a criação dos primeiros *frameworks*, cada novo lançamento busca o aperfeiçoamento de problemas apresentados nas versões antigas, o que estimula uma constante evolução do mercado de *frameworks*, pois há sempre algo novo para implementar, buscando sempre a facilidade que o *framework* deve trazer.

2.6 Exemplos De Frameworks

Existe uma grande variedade de *frameworks* utilizados no mercado de Tecnologia da Informação, neste sub tópico será abordado dois *frameworks* que se assemelham ao *framework* que será abordado neste trabalho, o Meteor. Além desses *softwares* servirem de base para a criação do *Meteor*, são conhecidos e utilizados no mercado, competindo diretamente com o *meteor*.

2.6.1 AngularJS

³ Significa em tradução livre – enxugamento – mas também pode significar a racionalização de níveis hierárquicos.

AngularJS foi criado em 2009 por Miško Hevery e Adam Abrons, inicialmente direcionado para empresas, porém, esse projeto foi abandonado e deram origem a um *framework* open-souce. Tempos depois, com a deixa de um dos desenvolvedores originais, o *framework* seguiu em desenvolvimento com outros integrantes da empresa Google. E de acordo com sua empresa fundadora (2012):

AngularJS é um *framework* estrutural para aplicações web dinâmicas. Ele permite que você use HTML como seu modelo de linguagem e permite estender a sintaxe do HTML para expressar componentes da sua aplicação de forma clara e sucinta. Ligação de dados do Angular e injeção de dependência elimina grande parte do código que você teria de escrever. E tudo acontece dentro do navegador, tornando-se um parceiro ideal com qualquer tecnologia de servidor. (GOOGLE, 2012, tradução nossa).

Assim, AngularJS se torna uma ferramenta didática, trabalhando como uma intermediadora no conceito de biblioteca e *framework*, abordando dentro de si, características básicas como a ligação de dados, estrutura de controle DOM (*Document Object Model*), suporte para validação de formulários e agrupamentos HTML em componentes reutilizáveis.

Para busca de dados SQL, o Angular pode utilizar PHP, ASP.NET, MySQL, SQL Lite e Access, todos retornando o resultado em JSON.

2.6.2 MEAN

Conhecido pelo nome de MEAN.IO ou simplesmente Mean, criado inicialmente por Amos Haviv na empresa Linnovate e depois desencadeado por Valeri Karpov após a saída de Haviv na empresa. Segundo a sua empresa fundadora, este *framework* foi projetado especialmente para:

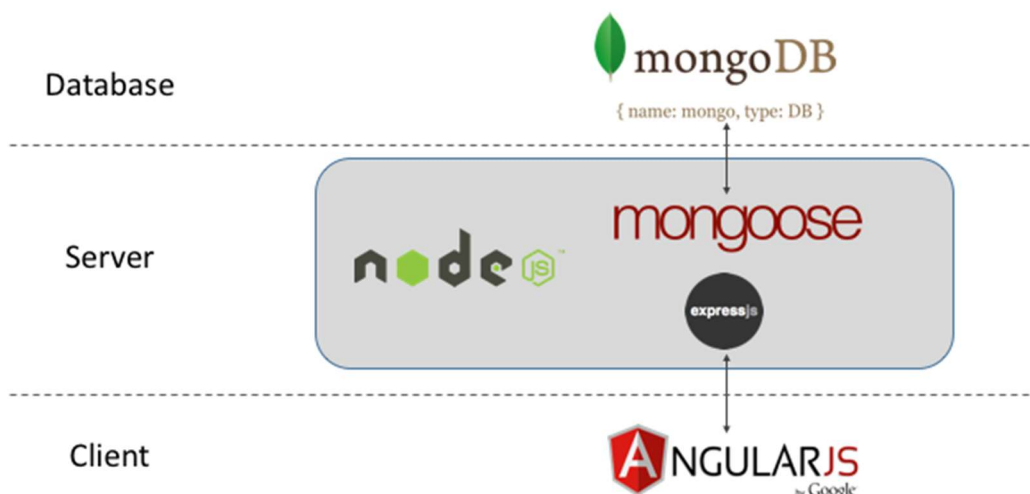
[...] dar-lhe uma maneira rápida e organizada para começar a desenvolver MEAN, baseado aplicações web com módulos úteis como Mongoose e Passport pré-empacotado e configurado. Nós principalmente [Linnovate], tentar cuidar dos pontos de ligação entre os *frameworks* populares existentes e resolver problemas comuns de integração. (LINNOVATE, 2013, grifo nosso)

Ou seja, Mean possibilita a utilização de frameworks empacotados, solucionando problemas entre estes *frameworks* para a facilitação no uso dentro da linguagem Web.

Mean utiliza o sistema MongoDB para banco de dados noSQL, ou seja, um banco não estrutural, também utiliza o *framework* Express.js, usado para aplicações

cliente-server com o Node.js, utiliza AngularJS, um framework MVC (modelo-visão-controlador) já citado anteriormente e Node.js, uma aplicação ideal para todo tipo de projeto por ser baseado em “non blocking I/O” e desempenha muito bem com aplicações de grande volume de dados, pois, pode maximizar o uso de uma única CPU, bem como memória.

Figura 2 - Funcionamento do MEAN



Fonte: <http://slidedeck.io/prakashwaghvani/rails-vs-mean-stack>

Na imagem acima pode se entender toda a parte da funcionalidade do framework Mean, sendo elas: o banco de dados (*Database*), servidor (*Server*) e cliente (*Client*).

3 METEOR

De acordo com Owens (2014, tradução nossa), “*Meteor.js* é uma plataforma de desenvolvimento coeso, um conjunto de bibliotecas e pacotes que estão unidos de forma arrumada para tornar o desenvolvimento web mais fácil [...]”. Porém, *meteor.js* tende a oferecer esses recursos descritos por Owens não só de uma maneira fácil e flexível para construção de um projeto web, mas também, *mobile* e para *desktop*. Uma de suas características fundamentais: ser um *software open source*. Todas essas bibliotecas e pacotes se unem para garantir a melhor experiência possível para o usuário em relação a etapas do desenvolvimento, como o *front-end*.

Node.js e *MongoDB* são as bases de construção da plataforma *Meteor*, ambos atuam diretamente e indiretamente nas fases de desenvolvimento de um software, desde a interface do usuário até o banco de dados e servidor.

Segundo a *The Linux Foundation* (2009), criadora do projeto:

O *node.js* é uma plataforma construída sobre o motor *JavaScript* (JS) do Google Chrome para facilmente construir aplicações de rede rápidas e escaláveis. *Node.js* usa um modelo de I/O direcionada a evento não bloqueante que o torna leve e eficiente, ideal para aplicações em tempo real com troca intensa de dados através de dispositivos distribuídos. (THE LINUX FOUNDATION, 2009)

A primeira estrutura utilizada para a construção de *meteor* e sendo responsável para garantir ao *framework* a flexibilidade de tempo real que ele possui, além de inúmeros outros recursos relacionados a parte lógica de uma aplicação é o *NodeJS*. Já a segunda estrutura que serve de base para *MeteorJS* é o *MongoDB*.

MongoDB torna o trabalho com um simples banco de dados e elegante. Ele usa um modelo de dados JSON [JavaScript Object Notation] que mapeia para as suas aplicações, e tem esquemas dinâmico, que permitem interagir rapidamente. Ele tem drivers para os idiomas que você usa para código. E ele tem uma linguagem expressiva consulta que permite obter, definir, classificar e agregar dados sem escrever código extra. Nós ainda fizemos com que seja fácil de implantar. (MONGODB, 2014).

Segundo seus desenvolvedores, o *MongoDB*, como o próprio nome diz, possui relação com o *Data Base*, ou seja, o banco de dados, ele é responsável pelo total controle da base de dados da aplicação.

O Mongo é um dos poucos softwares que se dedicam em uma das novas tecnologias: O NoSQL (Not Only SQL). Quando comparado com bancos de dados relacionais, bancos de dados NoSQL são mais escaláveis, proporcionam

desempenho superior e, seu modelo de dados aborda várias questões que o modelo relacional não são destinados a abordar, como grandes volumes de rápida mudança de dados estruturados, semiestruturados e não estruturados, são mais fáceis e flexíveis.

Segundo os pesquisadores Bernadette, Hélio e Jonas da UFPE (Universidade Federal de Pernambuco), em 2011:

O MongoDB é um banco de dados orientado a documentos, sendo possível utilizá-lo em diferentes sistemas operacionais, como Windows, Linux, OS X e Solaris. O MongoDB possui drivers para diversas linguagens de programação, entre elas: C, C#, C++, Java, Perl, PHP, Python e Ruby. (LÓSCIO, 2011).

Além do mais, graças a ajuda do *meteor* é possível a implementação do Mongo junto ao NodeJS o que acaba com um dos maiores problemas que um banco de dados NoSQL possui:

Um ponto comum a todas as empresas que têm adotado a tecnologia NoSQL são os problemas enfrentados quando se tem uma grande quantidade de dados, e estes precisam ser compartilhados em tempo real. Para isto, é necessário que as aplicações sejam escaláveis e seus dados tenham alta disponibilidade. (LÓSCIO, 2011).

Para alcançar todo o seu potencial, é necessário de um software que proporcione um ambiente escalável e faça uso de dados transparentes e em tempo real. Ou seja, *Meteor* está firmado em bases que se completam para juntos cumprirem suas funcionalidades e objetivos.

3.1 Contexto Histórico

Com o passar dos anos, assim como qualquer área relacionada a TI evoluiu, a internet também assumiu um grande papel nesta evolução. Do mesmo modo que a História é dividida em partes, a História das aplicações web também teve que ser repartida, em ambos os casos, essa divisão ocorreu e continua acontecendo para entender melhor e conseguir observar o que cada período da história trouxe de novo.

De acordo com o líder e desenvolvedor *full stack*⁴, Frederico Mia Arantes (2016, informação verbal)⁵, existem diversas gerações de aplicações web. A primeira geração era caracterizada por páginas estáticas e CGI (*Common Gateway Interface*) no servidor, que era a maneira padronizada para que o servidor interagisse com

⁴ Pessoa que tenha conhecimento razoável em tecnologias *back-end* e *front-end*

⁵ Dados fornecidos na palestra de Arantes em 2016

programas executáveis instalados no servidor que geravam dinamicamente páginas web. A segunda geração transforma os conceitos já estabelecidos da geração anterior, com páginas dinâmicas e *scripting*⁶ em servidores, como, o ASP (Active Server Pages), JSP (JavaServer Pages) e PHP.

Na terceira geração, as páginas continuaram sendo dinâmicas, porém, além de existir o *scripting* nos servidores, surge também o *scripting* no cliente, onde a linguagem é executada no lado do cliente, normalmente, quando não é possível no lado do servidor. Até esse momento, apesar de já existir os lados servidor e cliente, ambos se relacionavam como um só, porém com a chegada da quarta geração isso acabou, e então, passou a existir a separação do cliente (*front-end*), onde havia o consumo das páginas e o servidor (*back-end*) que recebia as requisições.

A quinta geração fica marcada pela atualização das funções cliente-servidor, a partir dela, o cliente passa a ser um aplicativo escrito em *JavaScript* e o servidor recebe e gera dados via REST (*Representational State Transfer*), em português Transferência de Estado Representacional que faz essa interação por meio de *scripts* e é nada mais que um protocolo comum para o desenvolvimento de serviços web. E por fim, Arantes cita outra geração a qual já fazemos parte, nela é possível que a separação entre cliente-servidor seja apenas física, possibilitando o uso de uma só linguagem e somente uma base de códigos e propagando os chamados dados transparentes.

E é dentro desta última geração que o *meteor.js* se encontra, era necessário que algum *software* pudesse suprir as necessidades que o mercado tinha, ou seja, promover a praticidade e qualidade em uma única plataforma e, baseando-se em outras grandes linguagens, surge definitivamente o *Meteor*.

Figura 3 - Logotipo do Meteor



Fonte: <https://www.meteor.com/>

⁶ Uso de scripts

A sua primeira grande aparição foi em dezembro de 2011, porém, com o nome de *Skybreak* e em 20 de janeiro de 2012, *Skybreak* se torna Meteor. Fundado pelo MDG (*Meteor Development Group*), em português Grupo de Desenvolvimento *Meteor*.

Com o passar dos anos, o MDG focou no aprimoramento de seu maior e mais novo *software*, como, suporte total para Linux em 06 de março, novas ferramentas, maior compatibilidade e atualizações constantes.

3.2 Finalidade

Cada *framework* na atualidade é criado com um propósito único, trabalhando junto para suprir a principal função dos *frameworks* em geral: a facilitação para os usuários. Este sub tópico aborda a principal finalidade do *Meteor*.

Segundo a empresa criadora do *Meteor*: “Deixando de lado todas as características do *Meteor*, nós acreditamos que tudo se resume a uma única coisa: *Meteor* é fácil de aprender [...]” (MDG, 2014).

Ele foi criado para este fim e conseguiu completar esse objetivo com sucesso, todas as ferramentas impostas pelo *meteor.js* são de total fácil aprendizado. Porém, ele não foi criado apenas para isso.

Hoje existem muitas formas para conseguir desenvolver qualquer tipo de web app, a variedade é enorme e a mesma coisa acontece para quem vai programar para *mobile* apps. Algumas das diversas linguagens de programação e de marcação que existem para web apps, são *JavaScript*, HTML (*Hypertext Mark-up Language*), CSS (*Cascading Style Sheets*), PHP, C#, Ruby e o próprio *Java*.

A principal finalidade para a criação do *meteor.js* foi suprir a grande variedade de programas e linguagens para programar. Ou seja, em uma só plataforma conseguir criar um aplicativo *web* completo, com banco de dados, *front-end* e servidor. Com o tempo a sua finalidade aumentou, no final de 2012, *Meteor* se expande para o mercado de *mobile* apps. Devido ao fato do mercado de aplicativos *mobile* ter crescido exponencialmente.

3.3 Vantagens e Desvantagens

Conforme com um artigo realizado pela UFCG (Universidade Federal de Campina Grande), os *frameworks* que vem ganhando cada vez mais o seu espaço,

surpreende com a evolução contínua a cada novo *software*. Isso acontece devido a quantidade gigantesca de vantagens que os frameworks têm.

Meteor estando dentro da classificação de *frameworks* possui um grande diferencial em relação a todos os outros concorrentes. Entre suas principais vantagens estão:

- Maior produção com menos linhas de código: segundo a MDG (2014), o *meteor.js* realiza em 10 linhas o que, de outro modo gastaria 1000, graças a sua integração com *JavaScript* que estende desde o banco de dados até a tela do usuário;
- Construir aplicativos em qualquer *device*: ele pode, usando o mesmo código se você está desenvolvendo para web, iOS, Android ou *desktop*;
- Usar tecnologias já conhecidas: construção de apps usando *frameworks*, ferramentas, recursos e outros softwares já conhecidos pela maioria dos programadores;
- Aplicações em tempo real: a melhor forma de explicar essa vantagem, em conformidade com Tom Coleman e Sacha Greif é utilizando do método prático de experiência:

Imagine que você está abrindo uma pasta em duas diferentes janelas do seu computador. Agora clique dentro de uma das janelas e delete um arquivo. Esse arquivo desapareceu da outra janela também? Você não precisa realmente seguir esses passos para saber disso. Quando nós modificamos algo num sistema de arquivos local, a mudança é aplicada em todo lugar sem necessidade de *refreshes* ou *callbacks*. Apenas acontece.

Entretanto, vamos pensar sobre como o mesmo cenário funcionaria na web. Por exemplo, vamos dizer que você abriu o mesmo site do *WordPress* como admin em duas janelas do *browser* e então criou uma nova postagem numa delas. Ao contrário do *desktop*, não importa o quanto você espere, a outra janela não vai refletir a mudança a não ser que você a atualize.

Ao longo dos anos, nós nos acostumamos com a ideia de que um *website* ser algo com o qual você se comunica apenas com pequenos, separados acessos. Mas *Meteor* é parte de uma nova onda de frameworks e tecnologias que estão procurando desafiar o status ao fazer a web em tempo real e reativa. (2015, p. 1).

Não há realmente outra maneira de demonstrar o visualmente o *on real time* (em tempo real) na construção de suas aplicações. Ele supera a barreira da separação entre *front-end* e *back-end*. De acordo com David Turnbull, essa é a principal característica que *Meteor* tem em destaque, pois:

Ultimamente, empresas como o Twitter e o Facebook já pensam em uma aplicação web em tempo real. É inevitável que, mais cedo do que você provavelmente imagina, os usuários consigam ver as aplicações web quase instantaneamente enquanto estão sendo criadas. Imagino que já há usuários que se sentem mal sempre que um carregamento de página separada é necessário para tarefas simples como alterar as configurações e encerrar a sessão. (TURNBULL, 2015).

Contudo, existem muitas outras vantagens que caracterizam o *meteor.js*, tais como, o desenvolvimento em apenas uma só linguagem, economia de tempo de programação, a comunidade oferecer suporte, sempre atualizado é extremamente fácil de aprender e utilizar e está à frente de tudo que conhecemos.

Apesar disto, como qualquer *framework*, o *Meteor* também possui desvantagens, entre elas está ligada a criação da própria plataforma. De acordo com a pesquisa realizada pela UFCG (2014), a grande desvantagem dos *frameworks* em geral, inclusive, *meteor.js* é o fato de que eles têm como objetivo reinventar cada vez mais o mercado onde ele se encontra, porém, isso pode trazer um grande risco devido ao fato que esses softwares são totalmente novos e não possuem suporte em casos de erros e problemas.

Outra desvantagem que é observado é o chamado "*Not Made Here Syndrome*", em português síndrome do não feito aqui, ou seja, apesar de oferecer a vantagem de ter várias formas de programação em uma, há uma certa dependência desses recursos externos.

Segundo Booch, "O *framework* [...] nunca será utilizado a menos que o custo de compreendê-lo usando suas abstrações seja inferior ao custo percebido do programador de escrevê-las a partir do zero". (BOOCH, 1994).

Essa afirmação mostra que *meteor.js* antes mesmo de ter esse nome, quando era conhecido apenas por *Skybreaker* sofreu muito com esse problema. Pois era muito mais conveniente utilizar o método convencional de programação do que usar as vantagens do *framework*, contudo, após a reformulação total de *Skybreaker* para *Meteor* essa desvantagem deixou de existir.

3.3.1 As Vantagens De Meteor Em Relação A Outros Frameworks

Como já mostrado, *meteor.js* é superior em muitos aspectos, a tabela a seguir mostra a as maiores e principais diferenças entre o *Meteor* e *frameworks* em geral baseando se nas pesquisas de Turnbull (2015), UFCG (2014) e Custódio (2012).

Tabela 1 - As vantagens de *Meteor* em relação a frameworks em geral

Meteor	Frameworks em geral
Redução de custos e linhas de códigos	Redução de custos
Redução do <i>time-to-market</i> ⁷ (TTM)	Redução do <i>time-to-market</i>
Maximização de re-uso	Maximização de re-uso
Menos manutenção e mais atualizações	Menos manutenção e poucas atualizações
Melhor consistência	Melhor consistência
Compatibilidade entre aplicações	Compatibilidade entre aplicações
Extrema facilidade de uso e aprendizado	Facilidade e aprendizado varia de acordo com o <i>framework</i>
Empacotamento de várias linguagens, ferramentas e recursos	Empacotamento de funcionalidades, e normalmente 1 ou 2 linguagens
Aplicações em tempo real	Normalmente, não são em tempo real
Suporte em comunidades	Suporte em comunidades
Qualidade muito bem-conceituada e de alto nível no desenvolvimento	Qualidade no desenvolvimento
Seguro, livre de vírus e ameaças em potencial	Segurança média, com casos de vírus e problemas relacionados
Não está incluso na curva de aprendizado ⁸	Curva de aprendizado
Sem necessidade de requisitos e dependências do sistema	Pequena necessidade de requisitos e dependências do sistema

⁷ Em português, tempo para o mercado, ou seja, o tempo até o software estar pronto para venda.

⁸ Necessidade de um pré conhecimento, devido ao uso de códigos e bases de terceiros.

Simple e com criação de códigos limpos	Muitas vezes mais complexos e com poluição nos códigos
--	--

Fonte: O autor (2016)

De acordo com os 3 autores, o *Meteor* se encaixa perfeitamente na nova fase da evolução dos *softwares*, possibilitando maiores benefícios para aqueles que o utilizam. Apesar de se assemelhar muito com as vantagens dos outros *frameworks*, *Meteor* se diferencia em relação aos seus benefícios exclusivos, ele pega as desvantagens dos outros *softwares* para melhorar a si próprio, por exemplo, o tempo real, que é uma desvantagem para os demais *frameworks*, é uma vantagem exclusiva para o *meteor*.

3.4 Desenvolvimento Dentro Das Empresas

Meteor apesar de fazer parte na nova geração e ser considerado como um *framework* muito novo já vem se expandindo não somente para uso pessoal, mas também no uso empresarial, em muitos casos, a empresa troca a forma de programação convencional para aderir as inúmeras vantagens disponibilizadas pelo *meteor.js*, empresas como, Mazda, Ikea, Honeywell, Qualcomm, PGA Tour e ClassCraft Studios Inc.

De acordo com Shawn Young, desenvolvedor da ClassCraft Studios, “a melhor escolha para nossa empresa foi investir em programas que trouxessem resultado real e rápido para nós, nesse caso: *meteor.js*” (YOUNG, 2016, informação verbal)⁹. Segundo ele, a criação do jogo ClassCraft teve 2 grandes fases, a primeira fase foi a criação da ideia e a segunda, a mudança da programação convencional para *meteor.js*.

⁹ Dados fornecidos em uma palestra de divulgação do ClassCraft em fevereiro de 2016.

A imagem abaixo mostra o site da empresa ClassCraft, onde foi usado, segundo seus criadores, muitas das ferramentas disponíveis pelo *framework*, possibilitando um ambiente flexível e ideal para o desenvolvimento do projeto.

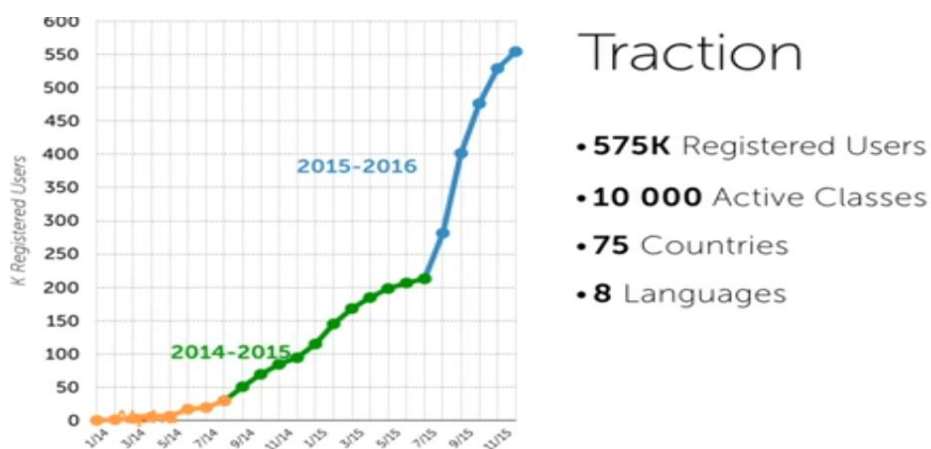
Figura 4 - Site da ClassCraft



Adaptado de: www.classcraft.com/

No gráfico de segmento abaixo, transmitido na palestra de Young, é demonstrado com uma linha amarela a representação da versão beta do jogo, a linha verde mostra o jogo em um período pós-teste e a linha azul a representação do jogo após a implementação do *meteor.js*. É visível como em apenas 2 meses com o *Meteor*, o ClassCraft alcançou a mesma taxa de crescimento que obteve durante os primeiros 1 ano e 7 meses do jogo sem a ajuda do *framework*.

Figura 5 - Uso de Meteor na ClassCraft



Adaptado de: [youtube.com/watch?v=W4mtTwMXnj4](https://www.youtube.com/watch?v=W4mtTwMXnj4)

Também houve o aumento em cerca de 60% do número de usuários, 79% das atividades nas classes do jogo. Com *Meteor* também foi mais fácil a expansão do app

de 1 idioma para 7 diferentes, atingindo mais 59 países que passaram a jogar ClassCraft.

“Foi muito proveitoso para o nosso *software* as vantagens que o *meteor.js* nos proporcionou [...]” (Young, 2016). Segundo ele, a utilização se tornou indispensável tendo em vista a evolução que o *framework* trouxe para o seu projeto.

3.5 Desenvolvimento Para Programadores

Meteor não traz só benefícios para a empresa que o utiliza, ele beneficia também os desenvolvedores que utilizam o *framework* para trabalhar.

O primeiro ponto é sobre a facilidade de aprendizado: para os programadores iniciantes, *meteor.js* é extremamente fácil de manusear. “[...] Eu sabia *Meteor* era especial a partir do momento que eu vi em ação. Ele mudou fundamentalmente como eu aproximar e construir software para a web agora. [...] Muito fácil de programar e encontrar soluções para minhas dúvidas [...]” (OWENS, 2014).

Além de ser de fácil aprendizado, *meteor.js* possui uma grande comunidade de suporte para auxiliar às pessoas que ainda estão conhecendo o software. Em uma pesquisa elaborada pela MDG, já existe cerca de 21794 perguntas com respostas no *Stack Overflow*, um dos maiores sites com perguntas e respostas relacionados à informática.

Rahul Choudhury, desenvolvedor da Mazda, uma das empresas que aderiram ao *Meteor* afirmou que: “Ano passado, antes de entrar na equipe da Mazda tive que desenvolver um *app web*, no qual demorei 2 meses para concluí-lo, porém, neste ano, graças ao *meteor.js* foi possível finalizar um projeto muito mais complexo na metade do tempo” (CHOUDHURY, 2016, informação verbal)¹⁰.

De acordo com Rahul é possível visualizar o quanto *meteor.js* consegue de maneira completa auxiliar os desenvolvedores que utilizam seus recursos.

3.6 Meteor em Bancos Relacionais

Basicamente, um banco de dados relacional significa um banco que modela os dados de uma forma que eles sejam percebidos pelo usuário como tabelas. Exemplos práticos disto é o MySQL, que apresentada um banco de dados estruturado.

¹⁰ Dados fornecidos na entrevista realizada pela MDG em 2016.

Apesar do Meteor trabalhar exclusivamente com o MongoDB, que é um banco de dados não relacional, é possível utilizar dentro dele, bancos de dados estruturados, como, o MySQL. Porém, a utilização do MySQL não é algo disponibilizado pela própria empresa criadora do Meteor, sendo necessário, a instalação de módulos que “burlam” o sistema que privilegia o MongoDB como banco padrão. Para poder utilizar o MySQL, é necessário a instalação do módulo pelo terminal: “meteor add patrickocoffey:mysql”.

Ainda assim, até mesmo para programadores experientes na ferramenta, a utilização de bancos relacionais indiretamente na aplicação Meteor, não é viável, pois, segundo a empresa, pode apresentar problemas em execução, não é à toa, que o MySQL ainda não foi adicionado como uma ferramenta oficial da empresa.

Outro ponto na utilização do MongoDB em relação a bancos relacionais, segundo Ricardo W. Brito, na faculdade de Fortaleza, em 2013, está na superioridade de bancos NoSQL, como o Mongo, para SGBD's.

4 DESENVOLVIMENTO

Para exemplificar a facilidade, praticidade e as vantagens e desvantagens do *framework meteorJS* é necessário a criação de um exemplo prático do mesmo.

4.1 Cenário da Prática

Para o desenvolvimento do mesmo é necessário principalmente o uso da internet para instalar o meteor no computador e baixar os pacotes e ferramentas para usar no projeto, após isso não existe necessidade de ter internet para criar, editar e executar o site.

Os requisitos básicos para a execução do meteor são um sistema Mac OS X de versão 10.7 ou superior, Linux com x86 ou uma arquitetura x86_64 ou um sistema com Windows 7 ou superior. Nesse sentido, foi utilizado somente um notebook para a criação do projeto, possuindo um processador Intel® Core™ i3-4005U CPU @ 1.70GHz, com 4GB de memória RAM e um sistema operacional Windows de 64bits, possibilitando uma execução rápida e sem problemas.

4.2 Visão Geral Do Sistema

Para isso, foi feita uma aplicação web que cria um ambiente de bloco de notas online. O objetivo central do site será fazer um sistema de *login*, onde o usuário só poderá ver e editar, criar ou apagar suas anotações após estar logado no sistema.

Iniciando a modelagem de dados utilizado na aplicação, é necessário entender o funcionamento e o modelo de dados do *MongoDB*. Ele é de fácil entendimento e consiste em uma coleção. Existe o banco de dados que tem como função armazenar coleções, as coleções são uma forma de armazenar um conjunto de documentos, estes por sua vez são a junção de um ou vários campos que se dividem em um par chave-valor, ou seja, em uma chave, que no *framework* é do tipo: *string* e o valor que pode ser dos tipos: texto ou caractere, inteiro ou flutuante, logaritmo de tempo e binário. Um dos pontos que diferenciam o *Mongo* dos outros SGBD (Sistema Gerenciador de Banco de Dados) é o armazenamento de documentos e “*arrays*” de valores.

Figura 6 - Modelo Conceitual

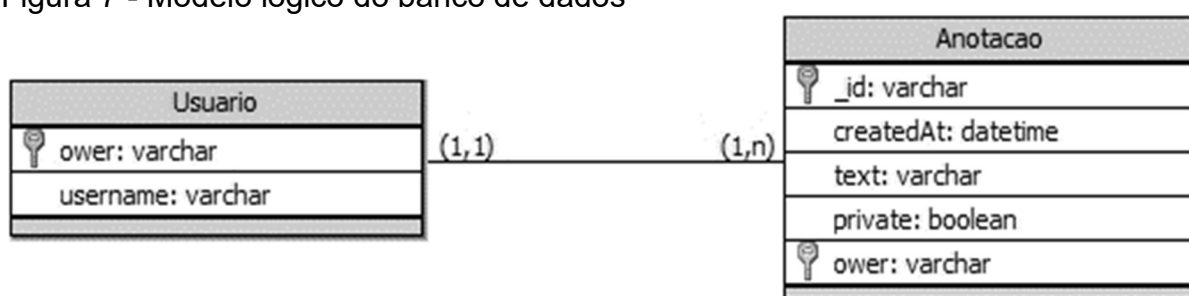


Fonte: O autor (2016)

A imagem anterior ilustra a modelagem conceitual do banco de dados para a aplicação proposta, nele é apresentado o DER (Diagrama Entidade Relacionamento) que descreve os requisitos de dados da aplicação, onde cada usuário possui um “owner” que o identifica unicamente, um “username”, nota-se que o campo “owner” equivale a um campo “id”. Cada anotação possui um “_id”, “createdAt” (data em que a anotação foi criada), “text” (a anotação propriamente dita) e o campo “private” que mostrará se a anotação é de domínio público e todos os usuários poderão ver ou somente o criador.

Na figura abaixo, - o modelo lógico do banco de dados - mostrará de uma forma mais clara o funcionamento e estrutura do *MongoDB* sendo possível entender melhor algumas peculiaridades de se trabalhar com Mongo. A primeira exceção encontrada no *MongoDB* é inserção automática de chaves primárias e a edição automática para o tipo: varchar. Isto só ocorre junto ao *Meteor*, pois o mesmo necessita de uma chave primária e a coloca como varchar para facilitar o seu uso interno, ou seja, ao utilizar uma coleção no Meteor e caso o desenvolvedor não crie o campo _id, o programa automaticamente cria e a cada novo registro armazena uma espécie de chave para o campo. Apesar de no modelo lógico existir duas tabelas, isso não se aplica na execução do *framework*, pois, ele reúne todos em somente uma coleção, acabando com o sistema de chave estrangeira e relacionamentos.

Figura 7 - Modelo lógico do banco de dados



Fonte: O autor (2016)

A seguir está uma imagem de um cadastro feito no Mongo segundo o exemplo dos modelos: conceitual e lógico e, da aplicação. Na imagem, é possível reforçar sobre a peculiaridade do campo _id, neste caso, existe um para a anotação em si (_id) e outro para a usuário (owner). Também é possível ver que não existe uma separação entre as tabelas, trazendo o sentido de que para cada anotação armazenada é necessário que um usuário tenha criado e é por esse motivo que não existe um campo “password”. Outro fato sobre a estrutura é sobre o campo “private”, caso a anotação seja privada, há uma necessidade para criar o campo private, o que ocorre no

Exemplo1, caso seja público, a própria aplicação *Meteor* visualiza que não existe um campo “private”, ou seja, ele é automaticamente não privado.

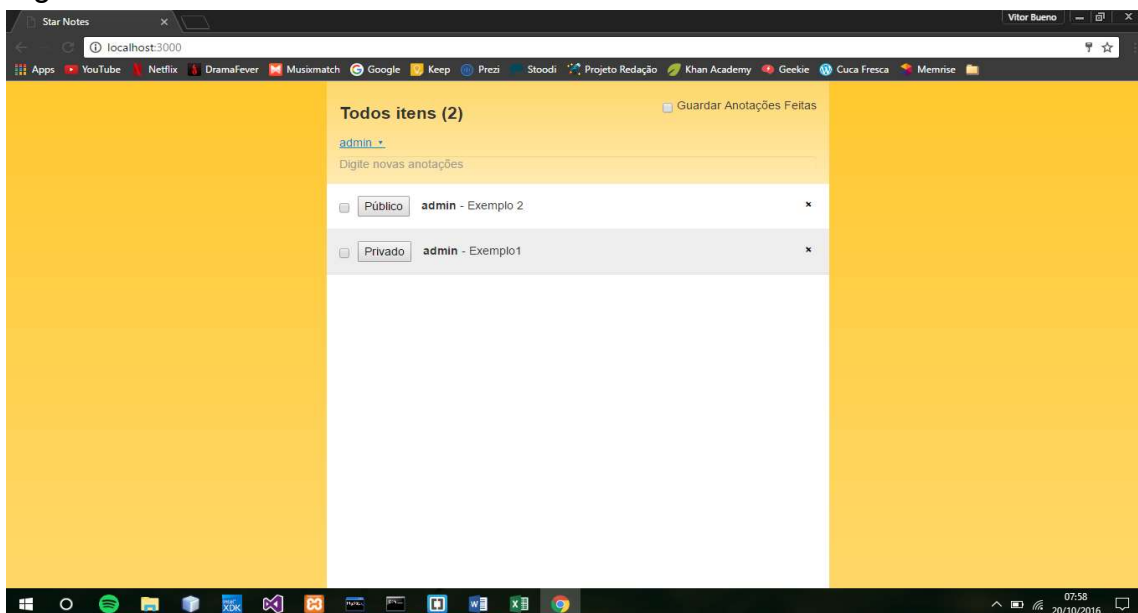
Figura 8 - Estrutura do banco de dados

```
meteor:PRIMARY> db.tasks.findOne();
{
  "_id" : "uyoMgeJcAtqvL95Jd",
  "text" : "Exemplo1",
  "createdAt" : ISODate("2016-10-07T00:53:49.142Z"),
  "owner" : "DZZrzbiFhcNhdCxoi",
  "username" : "admin",
  "private" : true,
  "checked" : false
  "_id" : "sPyoP7HibH2mgTrHr",
  "text" : "Exemplo 2",
  "createdAt" : ISODate("2016-10-20T09:57:58.625Z"),
  "owner" : "DZZrzbiFhcNhdCxoi",
  "username" : "admin",
  "private" : false,
  "checked" : false
```

Fonte: O autor (2016)

Na figura anterior, é possível ver todo o projeto completo e finalizado, onde, a parte lógica e estética e física se complementam.

Figura 9 - Resultado final do sistema



Fonte: O autor (2016)

É possível observar uma interface não muito planejada ou desenvolvida, visando apenas as ferramentas e utilidades disponível pelo meteor.

4.3 Funções Básicas

Para a criação de um aplicativo desenvolvido em *meteor* há a necessidade da utilização da ferramenta *Git Bash*, onde por ela será instalada o *framework*. Para a criação basta abrir o *Git Command Line* e digitar: “meteor npm install”, após isso o *Meteor* será instalado e pronto para uso.

Assim que *Meteor* estiver instalado, o usuário poderá utilizar o Prompt de Comando do computador para criar seus projetos e fazer a simulação da sua aplicação. Para criar a aplicação, foi necessário digitar: “meteor create simple-todos” no Prompt. Esse código ativa o *meteor* e suas funções e cria, neste caso uma pasta chamada “simple-todos” no diretório padrão do Windows: C:\Users\Usuário\.

Em seguida, o *framework* criará dentro da pasta todos os arquivos que um *Meteor app* precisa. São eles:

Tabela 2 - Arquivos necessários para um *Meteor app*

Arquivo	Comentário
client/main.js	Um ponto de entrada <i>JavaScript</i> carregada no cliente;
client/main.html	Um arquivo HTML que define a visualização dos <i>templates</i> ;
client/main.css	Um arquivo CSS que define a estilização do <i>app</i> ;
server/main.js	Um ponto de entrada <i>JavaScript</i> carregada no servidor;
package.json	Um arquivo de controle para instalar pacotes;
.meteor	Arquivos internos do <i>Meteor</i> ;
.gitignore	Um arquivo de controle para o <i>Git</i> .

Fonte: Adaptado de: http://www.addlabs.uff.br/sbse_site/SBSC2011_NoSQL.pdf

É muito interessante destacar que assim como Frederico Mia Arantes (2016), o *Meteor* se encaixa perfeitamente na 6ª geração de aplicativos web. Onde a separação dos lados cliente-servidor é apenas física, como vemos no caso dos arquivos que são lidos pelo cliente e pelo servidor, cada um em sua respectiva pasta, porém, na prática os dois lados trabalham juntos, propagando o que Arantes previa como dados transparentes.

Para simular o aplicativo, após as mudanças iniciais no código do aplicativo basta abrir o Prompt de Comando e alterar o diretório para a pasta em que está a

aplicação e digitar: “meteor”, desta maneira, o *framework*, disponibiliza a simulação, sendo possível visualizá-la abrindo o navegador e se entrar pelo endereço: “localhost:3000”.

Depois de simulado, qualquer alteração feita no aplicativo a partir do ponto em que é salvo, o site atualiza automaticamente mostrando as novas adições, a partir disto é possível ver uma das suas principais vantagens: o tempo real.

Toda a lógica imposta pelo *meteor* é feita pela interação do Node.js e com *Spacebars* onde são responsáveis, respectivamente, por garantir vantagens como tempo real e a lógica dentro do aplicativo, o que antes era feito totalmente pelo *JavaScript* ou *JQuery*. A lógica é feita pelos chamados *Templates*, que segundo David Burley (2014) são instruções que são cercadas por chaves duplas e que podem receber valores de acordo com um sistema próprio de lógica do *Spacebars*, com ele é possível mostrar dados de uma coleção do *MongoDB* de forma mais rápida e prática.

Figura 10 - Exemplo dos *templates*



Adaptado de: meteorcapture.com/spacebars/

Para fazer alguma inserção de dados em algum *template* é necessário criá-lo no corpo do tempo com a *tag*: <template>, nomeando ela para que se possa chamá-la depois. Quando houver necessidade de que os dados sejam em forma de uma matriz é necessário que, após a tag, coloque {{#each}} {{variáveis}} {{/each}}. A partir deles serão inseridos dados dentro da {{variáveis}}. Na parte lógica da imagem é possível entender o chamado da função onde, Template.list.helpers, representa o chamado do *Spacebars* pelo “Template”, a seleção do *template* “list”, que é nome dado à *tag* <template> e, a função *helpers* que adicionará informações para o template selecionado. Dentro da função é chamado o {{#each items}} que é uma matriz onde possui variáveis {{name}} e {{pet}}, as quais receberão dados mostrados na imagem

que, por fim, resultará na última parte da figura, mostrando as variáveis preenchidas pelos devidos dados.

Na aplicação desenvolvida, a lógica dos *templates* se tornam mais complexos, possibilitando um aprofundamento nas ferramentas do framework.

Na figura abaixo está a estrutura na página onde consta os principais elementos da aplicação, dentro delas é possível ver os *templates* utilizados no *app*. Apesar deles não estarem dentro de alguma *tag* `<template>` ainda é possível trabalhar, inserindo dados.

Figura 11 – Estrutura física dos templates

```
<body>
  <div class = "container">
    <header>
      <h1> Todos itens {{incompleteCount}} </h1>

      <label class = "hide-completed">
        <input type = "checkbox" /> Guardar Anotações Feitas
      </label>

      {{> loginButtons}}

      {{#if currentUser}}
        <form class = "new-task">
          <input type = "text" name = "text" placeholder = "Digite novas anotações" />
        </form>
      {{/if}}
    </header>

    <ul>
      {{#each tasks}}
        {{> task}}
      {{/each}}
    </ul>
  </div>
</body>
```

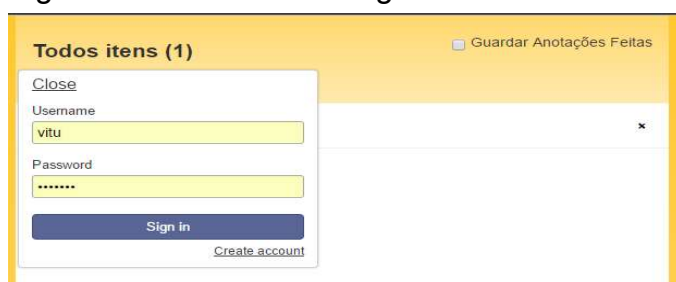
Fonte: O autor (2016)

O `{{incompleteCount}}` mostra a quantidade de anotações ainda não concluídas em cada um dos usuários. Mais abaixo, o *template* `{{each}}` com a variável `{{task}}` dentro são o lugar responsável por guardar todas as anotações criadas pelos usuários.

Outro ponto considerável é o *template* `{{loginButtons}}` e o `{{if currentUser}}`. São estrutura predefinidas do *meteor* para a inserção automática do sistema de login e cadastro de usuários. Graças ao *Meteor*, não há a necessidade de criar funções e códigos para desenvolver um sistema de *login*, isso é possível devido a uma das suas ferramentas adicionais.

Na imagem abaixo é visível a estrutura que o *Meteor* cria para a ferramenta de cadastro de usuários, o *framework* desenvolve toda a lógica e linguagem de marcação necessária, só é necessário que o programador crie um campo de identificação e do nome de *login* no banco de dados, nota-se que não existe necessidade do campo senha, pois, *Meteor* armazena este campo secretamente por motivos de segurança do usuário.

Figura 12 - Sistema de login



Fonte: O autor (2016)

A imagem abaixo ilustra a lógica básica para adicionar dados do banco de dados para os *templates*. Na primeira linha selecionada está a importação da variável *Task*, aonde se encontra o banco de dados do *MongoDB*.

Figura 13 – Estrutura lógica dos *templates*

```
import { Tasks } from '../api/tasks.js';

import './task.js';
import './body.html';

Template.body.onCreated(function bodyOnCreated() {
  this.state = new ReactiveDict();
  Meteor.subscribe('tasks');
});

Template.body.helpers({
  tasks() {
    const instance = Template.instance();

    if (instance.state.get('hideCompleted')) {
      // Filtrar se as anotações estiverem escondidas
      return Tasks.find({ checked: { $ne: true } }, { sort: { createdAt: -1 } });
    }
    // Retorna todas as anotações
    return Tasks.find({}, { sort: { createdAt: -1 } });
  },
  incompleteCount() {
    return Tasks.find({ checked: { $ne: true } }).count();
  },
});

Template.body.events({
  'submit .new-task'(event) {
    event.preventDefault();

    // Pegar o valor do elemento
    const target = event.target;
    const texto = target.text.value;

    // Inserir uma anotação na coleção
    Meteor.call('tasks.insert', texto);

    // Limpar formulário
    target.text.value = '';
  },
  'change .hide-completed input'(event, instance) {
    instance.state.set('hideCompleted', event.target.checked);
  },
});
```

Fonte: O autor (2016)

A segunda parte grifada é função *helpers*, que adiciona dados, é possível visualizar que como não existe nenhuma *tag <template>* é usado o *body*, que coloca como um '*template* pai' e onde se encontra todos os outros *templates*. Neste caso, para adicionar dados é necessário usar o nome do *template* em forma de função, por exemplo, *tasks()*, que colocará informações no `{{#each tasks}}`. No exemplo mostrado, caso o *checkbox* esteja marcado, ele mostra apenas dados completos, caso contrário mostra todas as anotações disponíveis no banco de dados para o usuário que estiver *logado*, após isso ele usa a função *incompleteCount()* que atualiza o número de anotações disponíveis. Para se conectar com o banco e encontrar informações, é utilizado a linha: *Tasks.find()*, ele é responsável por buscar as informações de acordo com os parâmetros utilizados.

4.4 Funções Adicionais

Além das funcionalidades básicas que o Meteor apresenta, existem outras características que definem seu trabalho. A primeira delas diz respeito à emulação para dispositivos móveis, é possível com Meteor, a transformação do site, feito para *desktop* em *mobile*, sendo disponível apenas para Android e iOS.

Primeiramente, para conseguir fazer a aplicação rodar em aparelhos mobile é necessário utilizar o *Meteor* no sistema operacional Linux ou utilizar um Macbook, pois, esta ferramenta não está disponível para Windows. Respectivamente, Linux e Mac são capazes de emular para o sistema Android e iOS.

Meteor foi projetado para trabalhar em diferentes plataformas e, para rodar a aplicação em iOS apenas é necessário a utilização da sintaxe "*meteor install-sdk ios*", este código será executado durante a configuração necessária para construir um aplicativo iOS do projeto. Quando estiver pronto, digite: "*meteor add-platform ios*" e em seguida: "*meteor run ios*". Após estes passos, todas as ferramentas necessárias para emular já estarão instaladas e o emulador aparecerá na tela.

Para a emulação em Android também só é necessário ir no terminal do computador e instalar a ferramenta de emulação com este código: "*meteor install-sdk android*", depois "*meteor add-platform android*" que adicionará a plataforma Android e "*meteor run android*" que carregará o emulador Android.

Outra funcionalidade presente no Meteor é a implantação do site nos seus servidores. Porém, esta ferramenta não deve ser usada para hospedar aplicações do

mundo real, e sim, para compartilhar protótipos e projetos simples com amigos e familiares. Isso ocorre porque o desenvolvedor não tem nenhum controle sobre o servidor, o serviço pode desaparecer a qualquer momento e não há garantia de escalas e de desempenho.

Para implantar este serviço, primeiramente precisa estar logado em uma conta do Meteor que seja autorizada para a implantação. Após isso só é necessário ir no terminal, na pasta do projeto e digitar: “meteor deploy nomedaaplicacao.meteor.com”, logo em seguida, aceitar os termos de uso da ferramenta e logar de novo na conta Meteor. Assim será possível visualizar o site no link digitado anteriormente.

5 CONCLUSÃO

Durante todo o processo de criação do trabalho, não houve qualquer obstáculo que atrapalhasse na execução do mesmo em toda a teoria exposta neste trabalho, pois, o número de materiais disponíveis em relação ao tema é muito abrangente, por se tratar de um assunto atual e muito popular na área de TI.

Porém, apesar da ferramenta utilizada na parte prática seja de fácil integração e manuseio, alguns erros relacionados a programação e execução do projeto dificultaram a finalização dele, ainda sim, a quantidade de benefícios foi muito maior do que os problemas enfrentados.

Durante a criação do projeto, apesar do aplicativo ser simples, foi possível observar o ponto principal de um *framework* em relação a programação normal: o *framework* ganha em tempo e simplicidade de criação, porém, perde em desempenho para o método convencional de programação. Exemplo disto é que, apesar de uma das maiores vantagens apresentado pelo *Meteor* – o tempo real – traga benefícios para o projeto, a demora para que a ação aconteça naturalmente ultrapassa o esperado, sendo de aproximadamente 10 segundos dependendo da mudança feita no projeto.

O tempo total desde a criação da pasta do projeto até a conclusão do mesmo foi de apenas 2 dias, tempo o qual demoraria muito mais, levando em consideração a falta de conhecimento para criar o aplicativo sem a ajuda do *framework*. Entretanto, devido ao fato, de que o site criado é de porte pequeno, se torna inviável uma comparação clara de todas as vantagens e desvantagens do *framework* utilizado. Ainda assim, teoricamente, *Meteor* consegue demonstrar suas qualidades independente da complexidade do projeto, como, a aplicação pequena desenvolvida e o exemplo da empresa Class Craft, como uma grande aplicação.

Apesar de tudo, levando em conta a problemática, foi mais proveitoso o uso de *Meteor*, devido à falta de tempo dos desenvolvedores para concluir a programação, demonstrando então, a superioridade do *framework* em projetos pequenos.

REFERÊNCIAS

- ARANTES, Frederico Mia. **Apresentando Meteor**. 2016. Disponível em: <<http://pt.slideshare.net/fredmaia/apresentando-meteor-join-community-goinia>>. Acesso em: 12 jun. 2016.
- BOOCCH, Grady. **The Booch Method**. 1994. 10 p. Disponível em: <<http://www.grosmax.uqam.ca/Martin/files/Life%20cycle%20booch.pdf>>. Acesso em: 01 jun. 2016.
- BOOCH, Grady; RUMBAUGH, James; JACOBSON, Ivar. **UML Guia do Usuário**. Editora Campus. 2012. 271 p.
- BOOTSTRAP. **History of Bootstrap**. 2011. Disponível em: <<http://getbootstrap.com/about/>>. Acesso em: 07 set. 2016.
- BRITO, Ricardo W. **Banco de Dados NoSQL x SGBDs Relacionais: Análise Comparativa**. 2013. 6p. Disponível em: <<http://www.infobrasil.inf.br/userfiles/27-05-S4-1-68840-Bancos%20de%20Dados%20NoSQL.pdf>>. Acesso em 09 nov. 2016.
- BURLES, David. **Understanding Spacebars**. 2014. Disponível em: <<http://meteorcapture.com/spacebars/>>. Acesso em: 19 out. 2016.
- COLEMAN, Tom; GREIF, Sacha. **Discover Meteor: Building real-time JavaScript web apps**. 2015. 126 p. Disponível em: <<http://pt.discovermeteor.com/pdf>>. Acesso em: 26 abr. 2016.
- CUSTÓDIO, Glaucio. **Porque usar um framework?** 2012. Disponível em: <<http://glaucocustodio.com/2012/07/31/porque-usar-um-framework/>>. Acesso em: 28 mai. 2016.
- EIS, Diego. **O básico: O que é HTML?** 2011. Disponível em: <<http://tableless.com.br/o-que-html-basico/>>. Acesso em: 26 ago. 2016.
- ELLISLAB. **Codeigniter Web Framework**. 2013. Disponível em: <<https://www.codeigniter.com/>>. Acesso em: 10 mai. 2016.
- GOOGLE Inc. **What Is AngularJS**. 2012. Disponível em: <<https://angular.io/>>. Acesso em: 09 nov. 2016.
- IVANOV, Alex. **Top 21 Best Free CSS3 Frameworks for Web Development 2016**. 2016. Disponível em: <<https://colorlib.com/wp/free-css3-frameworks/>>. Acesso em: 20 out. 2016.
- LEITE, Alexandre Ferreira. **Frameworks e Padrões de Projeto**. 2006. Disponível em: <<http://www.devmedia.com.br/frameworks-e-padroes-de-projeto/1111>>. Acesso em: 08 jun. 2016.
- LINNOVA. **Open Source at Its Best**. 2013. Disponível em: <<http://www.linnovate.net/>>. Acesso em: 09 nov. 2016.
- LOPES, S. **O que é PHP?** 2007. Disponível em: <https://www.oficinadanet.com.br/artigo/659/o_que_e_php>. Acesso em: 05 set. 2016.
- LÓSCIO, Bernadette Farias et al. **NoSQL no desenvolvimento de aplicações Web Colaborativas**. 2011. 17 p.

MARCOTTE, Ethan. **Frameworks**. Responsive Design. 2015. Disponível em: <<http://alistapart.com/article/frameworks>>. Acesso em: 02 jun. 2016.

MATTSSON, Michael. **Evolution and composition of object-oriented frameworks**. 2000.

MATTSSON, Michael; LUNDBERG, Christer. **Using Legacy Components with Object-Oriented Frameworks**. 1996.

MATTSSON, Michael; BOSCH, Jan; FAYAD, Mohamed E. **Framework integration problems, causes, solutions**. 1999.

METEOR DEVELOPMENT GROUP. **Introduzindo Meteor API Docs**. 2014. Disponível em: <<http://docs.meteor.com/#/full/>>. Acesso em: 02 jun. 2016.

MONGODB. **What is MongoDB**. 2016. Disponível em: <<https://www.mongodb.com/what-is-mongodb>>. Acesso em: 14 jun. 2016.

PINTO, Álvaro Vieira. **Sete lições sobre educação de adultos**. 11 Edição. São Paulo. Cortez, 2000.

OLIVEIRA, Eric C. M. **O Universo dos Frameworks Java**. 2014. Disponível em: <<http://www.linhadecodigo.com.br/artigo/758/o-universo-dos-frameworks-java.aspx>>. Acesso em: 20 mai. 2016.

OWENS, Josh. **What is Meteor.js?** 2014. Disponível em: <<http://jshowens.me/what-is-meteor-js/>>. Acesso em: 12 jun. 2016.

POPE, Stephen. **Can Meteor and MySQL Play Nicely Together**. 2014. Disponível em: <<https://projectricochet.com/blog/can-meteor-and-mysql-play-nicely-together#.WCOpNvkrLIW>>. Acesso em: 09 nov. 2016.

PRESSMAN, Roger S. **Engenharia de Software**. 6ª edição McGraw-Hill. 2006. Acesso em: 27 mai. 2016.

RODRIGUES, Joel. **Modelo Entidade Relacionamento (MER) e Diagrama Entidade-Relacionamento (DER)**. 2010. Disponível em: <<http://www.devmedia.com.br/modelo-entidade-relacionamento-mer-e-diagrama-entidade-relacionamento-der/14332>>. Acesso em: 07 set. 2016.

RODRIGUES, Jorge. **Meteor.js: Construindo aplicações web com Node.js e MongoDB**. 2015. Disponível em: <<http://www.devmedia.com.br/meteor-js-construindo-aplicacoes-web-com-node-js-e-mongodb/32953>>. Acesso em: 29 mai. 2016.

SANTOS, Antônio Henrique; CARVALHO, Nelson Ribeiro. **Frameworks e seus Benefícios no Desenvolvimento de Software**. 2011. 16 p. Disponível em: <http://revistapensar.com.br/tecnologia/pasta_upload/artigos/a95.pdf>. Acesso em: 08 jun. 2016.

SILVA, Ricardo Pereira e. **Suporte ao Desenvolvimento e uso de Frameworks e Componentes**. 2000. 262 p. Disponível em: <<http://www.inf.ufsc.br/~ricardo/download/tese.pdf>>. Acesso em: 11 jun. 2016.

SOUZA, Givanaldo Rocha de. **Diagrama de Caso de Uso**. 2012. 45 p. Disponível em: <<http://docente.ifrn.edu.br/givanaldorochoa/disciplinas/engenharia-de-software-licenciatura-em-informatica/diagrama-de-caso-de-uso-tecnico>>. Acesso em: 20 ago. 2016.

STADZISZ, Paulo César. **Projeto de Software usando a UML**. 2002. 69 p. Disponível em: <<http://www.etelg.com.br/paginaete/downloads/informatica/apostila2uml.pdf>>. Acesso em: 07 set. 2016.

SUL, Ricardo. **Frameworks de Arquitetura Corporativa**. 2014. Disponível em: <<http://www.architectonics.com.br/frameworks-de-arquitetura-corporativa/>>. Acesso em: 01 mai. 2016.

THE JQUERY FOUNDATION. **JQuery**. 2014. Disponível em: <<https://jquery.com/>>. Acesso em: 06 ago. 2016.

THE LINUX FOUNDATION. **Node.js**. 2009. Disponível em: <<http://collabprojects.linuxfoundation.org/>>. Acesso em: 04 mai. 2016.

THE PHP GROUP. **PHP: What Is PHP?** 2001. Disponível em: <<http://php.net/manual/en/intro-what-is.php>>. Acesso em: 20 ago. 2016.

TURNBULL, David. **Your First Meteor Application**: A complete beginner's guide to meteor.js. 2015. 149 p. Disponível em: <http://art.yale.edu/file_columns/0000/8802/book.pdf>. Acesso em: 09 jun. 2016.

TURNBULL, David. **7 Reasons to Develop Your Next Web App with Meteor**. 2014. Disponível em: <<https://www.sitepoint.com/7-reasons-develop-next-web-app-meteor/>>. Acesso em: 10 jun. 2016.

W3CHOOLS. **CSS Tutorial**. 1999. Disponível em: <<http://www.w3schools.com/css/default.asp>>. Acesso em: 05 set. 2016.

ZACHMAN, John A. **Primer for Enterprise Engineering and Manufacturing**. 2003.

ZEMEL, Richard S et al. **Learning as a practical achievement: An interactional perspective**. 2009. 122 p.