FILIPE LEUCH BONFIM
MICHAEL LIANG

APLICAÇÕES ESCALÁVEIS COM *MEAN STACK*

Trabalho de Graduação apresentado à disciplina CI083 - Trabalho de Graduação em Arquitetura e Organização de Computadores II como requisito à conclusão do curso de Bacharelado em Ciência da Computação, Setor de Ciências Exatas da Universidade Federal do Paraná.
Orientador: Bruno Müller Junior.

CURITIBA

2014

SUMÁRIO

LISTA DE FIGURAS

RESUMO

Com o aumento significativo da quantidade de usuários da internet [29], surge cada vez mais a preocupação com escalabilidade ao se desenvolver aplicações web. O MEAN *Stack* é um conjunto de tecnologias com foco no desenvolvimento de aplicações web escaláveis. Que é composta pelas seguintes ferramentas: MongoDB, Express.js, AngularJS e Node.js.

Esta monografia tem como objetivo apresentar o MEAN *Stack* como uma opção para o desenvolvimento de aplicações que dependam da escalabilidade, além de mostrar como é feita a integração dos componentes que o compõem.

Palavras-chave: MEAN, Javascript, Node.js, Angular.js, MongoDB, Express.js, Escalabilidade.

ABSTRACT

With the significant increase of Internet users [29], it is also increasing concern the scalability to develop web applications. The MEAN Stack is a set of technologies focusing on the development of scalable web applications, which consists of the following tools: MongoDB, Express.js, AngularJS and Node.js.

This paper aims to present the MEAN Stack as an option for the development of applications that depend on scalability, and show how is the integration of the components that comprise it.

Keywords: MEAN, Javascript, Node.js, Angular.js, MongoDB, Express.js, Scalability.

CAPÍTULO 1

INTRODUÇÃO

Uma aplicação web pode ser resumida em requisições de um cliente ao servidor. Após o servidor ter recebido a requisição, ele envia uma resposta ao cliente. Esta resposta pode ser desde uma página HTML até uma página de erro.

Considere a seguinte situação, muitos clientes requisitando uma página HTML ao mesmo tempo e o servidor tendo que responder a todos eles. Se o servidor tratar esses pedidos como uma fila e somente responder a próxima requisição quando a anterior estiver terminada, os clientes que tiveram suas requisições no fim desta fila, terão maior tempo de resposta implicando em demora para carregar a página do lado do cliente.

Para diminuir o tempo de duração de múltiplas requisições de clientes ao servidor, existem diversas soluções que utilizam técnicas de escalabilidade, alguns exemplos são melhorar o hardware do servidor, com a substituição do hardware antigo por um mais moderno com um poder de processamento maior, e no caso do software, é checar se as ferramentas que estão sendo utilizadas possuem sinergia e foram desenvolvidas para serem escaláveis.

A proposta deste texto é apresentar uma solução via software que visa ajudar a resolver o problema de escalabilidade. O MEAN *Stack*. O MEAN é um acrônimo das quatro tecnologias que são o banco de dados MongoDB, o Node.js e o Express.js na parte do servidor e por último o AngularJS no lado do cliente.

1.1 Objetivo

Nesta monografia serão apresentados conceitos e funcionalidades de um conjunto de tecnologias com o foco em resolver os problemas de aplicações web altamente escaláveis. Esse conjunto é conhecido como MEAN *Stack*[1], que é basicamente a utilização de quatro

[1] *Stack* no sentido de pilha, no caso uma pilha de aplicações.

tecnologias baseadas em Javascript: **M**ongoDB, **E**xpress.js, **A**ngularJS e **N**ode.js. Será apresentado como o MEAN *Stack* lida com a escalabilidade e como é o desempenho de alguns de seus componentes, através da análise de testes de desempenho.

Para demonstrar como uma aplicação MEAN é implementada e como seus componentes interagem entre si foi desenvolvida uma aplicação usando MEAN *Stack* e o Socket.io baseada na aplicação *Math Race*. A idéia do *Math Race* é realizar uma competição em tempo real para ver qual jogador acerta mais contas de matemáticas em determinado tempo. A partir desta implementação, testes de escalabilidade foram feitos para análise dos resultados.

1.2 Organização do trabalho

Este trabalho está dividido em quatro partes. A primeira parte trata os conceitos que foram utilizados para o desenvolvimento do tema, como aplicações web escaláveis, programação orientada a eventos, Javascript e NoSQL.

Na segunda parte explicamos mais detalhadamente as tecnologias do MEAN que foram propostas, enfatizando algumas características que contribuem para a escalabilidade.

A terceira parte aborda como os componentes do MEAN são integrados, além de mostrar um comparativo de uma aplicação MEAN com o LAMP. O LAMP é outra tecnologia bastante difundida, que utiliza o sistema Linux, o servidor Apache, o banco de dados MySQL e a linguagem de programação PHP[2]. Os testes de desempenho também serão apresentados nesta parte.

A conclusão do trabalho e os possíveis trabalhos futuros são apresentados na quarta e última parte.

[2]Podendo haver váriações com outras linguagens de programação como Python e Perl

CAPÍTULO 2

CONCEITUAÇÃO E IDÉIA GERAL

As seções seguintes descrevem alguns conceitos importantes para que se possa compreender melhor o tema desta monografia. A seção 2.1 descreve o que são aplicações web escaláveis, a seção 2.2 descreve a programação orientada a eventos, a seção 2.3 Javascript e a seção2.4 o NoSQL.

2.1 Aplicações web escaláveis

Escalabilidade

Cada vez mais nos deparamos com um aumento do número de usuários da internet. A cada ano essa quantidade aumenta ainda mais devido a fatores como a popularização da banda larga, os programas sociais, entre outros.

Na figura 2.1 [29] é demonstrado um aumento expressivo[1] do número de usuários de Internet. Para isso foi feita uma comparação com a quantidade de usuários entre os anos de 1993 até 2014.

[1] Aproximadamente de 650%.

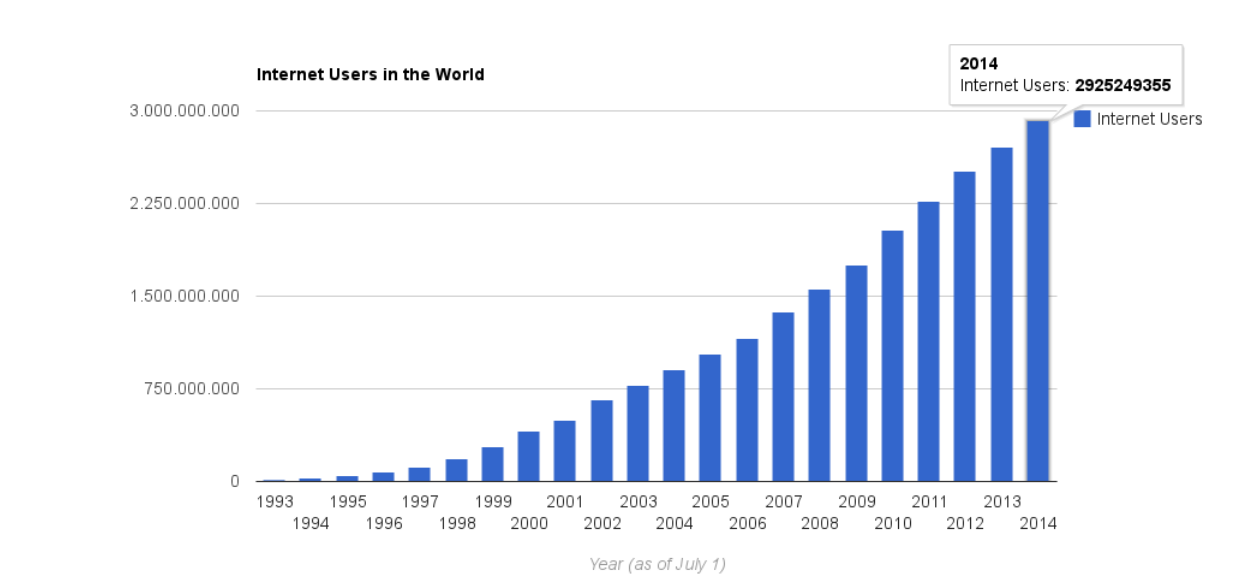

Figura 2.1: Quantidade de usuários da internet entre 1993 e 2014

Aplicações web

A aplicação web refere-se a sistemas de informática que são executados através de navegadores com internet. Geralmente aplicações que utilizam-se de tecnologias web como HTML, Javascript e CSS.[36]

Quando é desenvolvida uma aplicação web que visa atender a este aumento significativo de acessos, surge-se a preocupação com a escalabilidade do sistema. Escalabilidade vem a ser "a habilidade de uma aplicação manter o desempenho[2] quando a carga de trabalho aumenta" [22], podendo ser considerada uma junção entre capacidade[3] e desempenho.

Escalabilidade em aplicações web

Existem duas formas de escalabilidades, relacionadas as páginas web, a escalabilidade vertical e a horizontal.

Na escalabilidade vertical, há uma melhoria do hardware de um único servidor, quando é adicionada mais memória, ou o processador é trocado por outro mais potente, entre outros aspectos.

[2]Que significa "a habilidade que uma aplicação tem de atingir um objetivo, como por exemplo responder no menor tempo possível" [22]

[3]Que significa "a carga total que uma aplicação pode suportar"[22]

A escalabilidade horizontal acontece quando são adicionados mais servidores, permitindo a criação de um *cluster*[4].

Estas duas formas de escalabilidade são geralmente relacionadas ao hardware, sendo utilizadas por páginas web que têm uma grande quantidade de acessos, e precisam continuar respondendo as requisições de maneira satisfatória[5].

Na figura 2.2 pode-se observar de maneira mais clara a diferenciação entre escalabilidade vertical e horizontal.

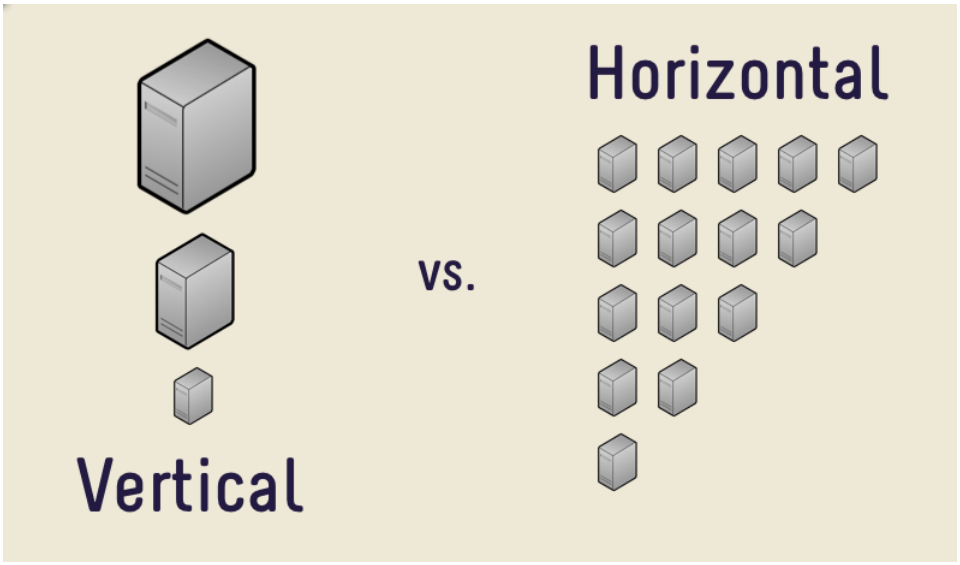

Figura 2.2: Escalabilidade vertical e horizontal [15]

[4]É um conjunto de computadores que subdividem o processamento do sistema para o aumento de desempenho, disponibilidade e balanceamento de carga.

[5]Por exemplo sem que o usuário perceba atrasos no carregamento da página

Existem situações em que a aquisição ou o melhoramento de servidores se torna algo muito custoso ou até mesmo inviável. Para essas situações são aplicadas técnicas de escalabilidade relacionadas ao software, que podem se subdividir em duas partes, que serão apresentadas a seguir:

- A primeira é quando é realizado um simples acesso à página, ou seja, existem requisições do tipo *http get*, neste caso para lidarmos com escalabilidade uma das soluções é a utilização de um proxy reverso[6], que pode funcionar como uma *cache* para conteúdos estáticos[7] e dinâmicos[8].
- A segunda parte é quando dados são enviados para serem persistidos através da página. Nesse caso estas requisições são do tipo *http post*, e não há como efetuar *cache* destes dados, pois são dados novos. As soluções encontradas nesse caso, são desde enfileirar as entradas para o banco de dados, até a utilização de banco de dados NoSQL.

2.2 Programação orientada a eventos

A programação orientada a eventos ou programação dirigida a eventos é considerada um paradigma da programação[9] . A compreensão da orientação a eventos é importante para que se entenda algumas características do Node.JS.

Entre outros paradigmas de programação pode-se citar [16]:

- Imperativa: é um paradigma de programação que descreve a computação como ações, enunciados ou comandos que mudam o estado (variáveis) de um programa.
- Estruturada: é uma forma de programação de computadores que preconiza que todos os programas possíveis podem ser reduzidos a apenas três estruturas: sequência,

[6]"Enquanto um proxy, no modelo convencional, intercepta requisições originadas na rede local (LAN – Local Área Network) com destino à Internet, um proxy reverso intercepta requisições originadas na Internet com destino à rede local." [13]

[7]Musicas, imagens, vídeos, etc.

[8]Que é quando um código é executado do lado do servidor para atender uma requisição, podendo realizar uma série de tarefas, como por exemplo acessar um banco de dados, sendo que este código é escrito em uma linguagem de programação como PHP, Java, Python, entre outras.

[9]Um paradigma de programação fornece e determina a visão que o programador possui sobre a estruturação e execução do programa.

decisão e iteração (esta última também é chamada de repetição).

- Orientada a objetos: é um modelo de análise, projeto e programação de sistemas de software baseado na composição e interação entre diversas unidades de software chamadas de objetos.

- Funcional: é um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste da programação imperativa, que enfatiza mudanças no estado do programa.

Um evento é qualquer ação do usuário que interaja com o sistema, por exemplo um clique do mouse em determinado local da aplicação, teclas do teclado sendo pressionadas ou até mesmo um toque na tela, caso o hardware permita como em smartphones e tablets modernos.

Um programa que utiliza orientação a eventos possui um fluxo que é um laço que recebe repetidamente informação para processar e disparam uma função de resposta de acordo com o evento recebido. As informações de entrada podem ser enfileiradas ou registrar uma interrupção, em alguns casos ambos podem ser adotados. Diferentemente do fluxo de linguagens tradicionais como C, onde o fluxo é linear com *loops* no decorrer do caminho.

Para facilitar o entendimento imagine uma aplicação que mostra em sua tela o número zero. Esse número é incrementado ao se pressionar o botão esquerdo do mouse, decrementado ao se pressionar o botão direito do mouse, e volta para zero quando for pressionado o botão do meio. Esses cliques representam eventos, quando ocorrem, o sinal destes eventos são passados ao *Event Loop*, que os trata e retorna o resultado. Nesse caso, o retorno é o incremento, decremento ou zerar o valor.

A figura 2.3 mostra como esse processo ocorre, destacando a interação dos eventos com o *Event Loop*.

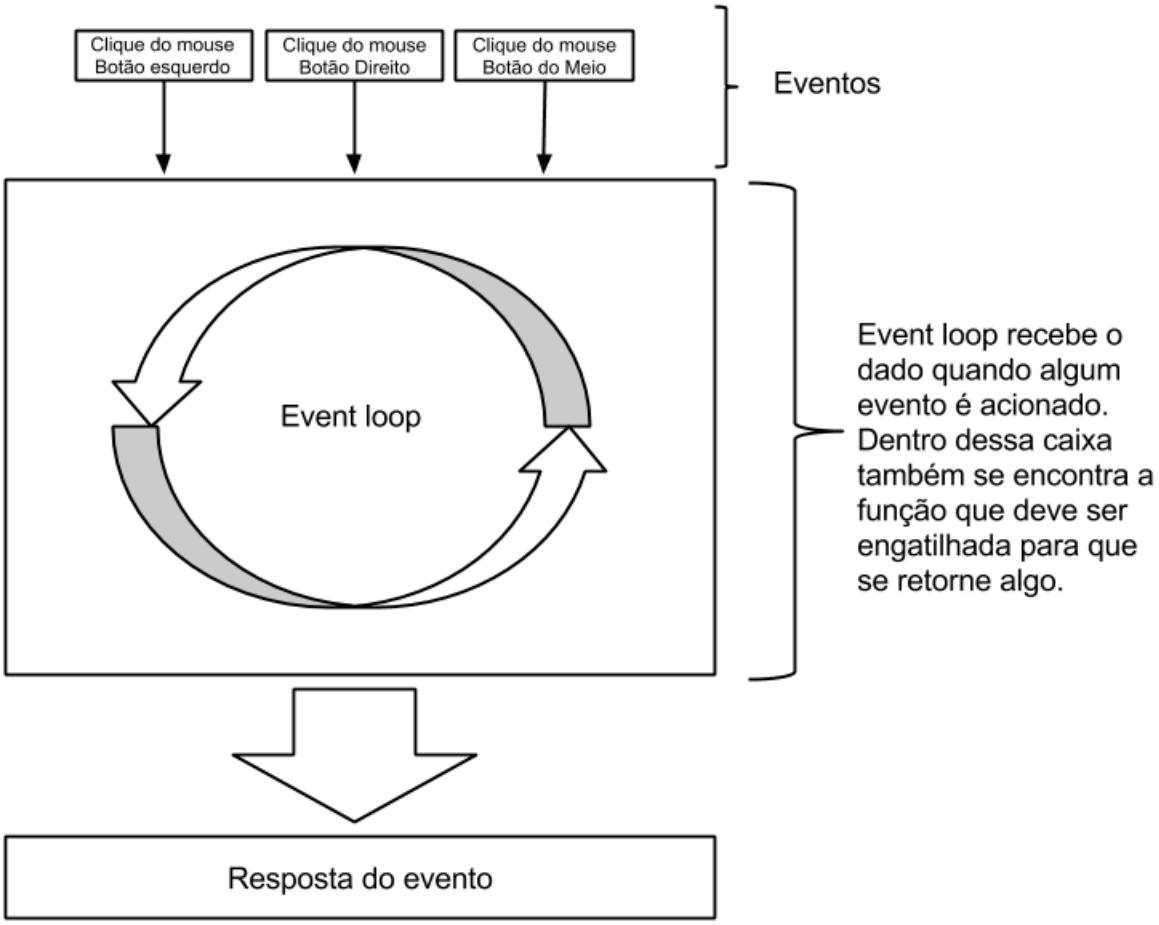

Figura 2.3: Event Loop

2.3 Javascript

O Javascript é uma linguagem de programação interpretada, que foi criada com o intuito de ajudar os navegadores web a executar scripts no lado do cliente e assim permitir a interação com o usuário. Esta linguagem foi baseada na ECMAScript[10] que é padronizada pela Ecma international nas especificacões ECMA-262[11] e ISO/IEC 16262[12]. Abaixo serão descritas algumas das principais características do Javascript:

Imperativa e estruturada

O Javascript possui elementos de sintaxe de linguagens de programação estruturada, elementos que se assemelham por exemplo com a linguagem C como *If*, *while* e

[10]Que é uma linguagem de programação de scripts.
[11]`http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-262.pdf`
[12]`http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=55755`

switch.

Apesar de sintaticamente parecer com C, seu escopo é diferente, Javascript utiliza-se de um escopo a nível de função[13].

Tipagem dinâmica

No Javascript os tipos são associados com valores e não com as variáveis. Por exemplo, uma variável "A" poderia ser associada a um número e posteriormente ser associada a uma string.

Baseada em protótipos

O Javascript utiliza-se de protótipos no lugar das classes para o mecanismo de herança. É possível simular muitas características de orientação a objetos usando-se protótipos.

Linguagem orientada a eventos

No Javascript é possível tratar as interações do usuário em um arquivo HTML e é através de eventos que isso é feito, como explicado na seção 2.2 deste capítulo .

2.4 NoSQL - Not Only SQL

Os bancos de dados relacionais não foram desenvolvidos para resolver problemas enfrentados pelas aplicações web modernas como a escalabilidade e a agilidade [23]. "Foram projetados para serem executados em uma máquina apenas" [6]. "Quanto maior o tamanho, mais custoso se torna essa escalabilidade, seja pelo custo de novas máquinas, seja pelo aumento de especialistas nos bancos de dados utilizados".[18]

Devido às limitações citadas, começaram surgir diversas soluções que funcionam de maneira diferente aos tradicionais banco de dados relacionais. Essas soluções ficaram conhecidas como NoSQL - *Not Only SQL*. Pode-se categoriza em "uma nova classe de banco de dados" [14], formada por "diferentes sistemas de armazenamento "[25], que surgiu

[13]Isso significa que somente funções podem criar novos escopos, blocos como *if*, *while*, *for* não criam novos escopos.

para lidar com situações em que os bancos de dados tradicionais são ineficientes. “Muitas dessas bases apresentam características muito interessantes como alta performance, escalabilidade, replicação, suporte à dados estruturados e sub colunas”[25].

Seguem abaixo algumas características de bancos de dados NoSQL[6]:

- Não usa modelo de dados relacional e portanto não usa a linguagem SQL;
- Costuma ser projetado para ser executado em um *cluster*;
- Costuma ser *open-source*;
- Não possui esquema fixo (*SCHEMA-LESS*), permitindo gravar qualquer dado em qualquer estrutura.

Existem diversas categorias[2] em que se classificam os banco de dados NoSQL que variam de acordo com a forma de armazenar os dados. Algumas das principais categorias que se destacam são: chave/valor, orientado a colunas, orientado a documentos e baseado em grafos [14].

Chave/Valor

Utiliza um *array* associativo como modelo de dados [37]. Nestes *array* os dados podem ser acessados através de chaves, e estas chaves estão associadas a seus valores correspondentes. Alguns bancos desta categoria são: Azure Table Storage, Berkeley DB,Couchbase, DynamoDB, Redis, Riak e Tokyo Cabinet.

Orientado a colunas

Em um banco de dados orientado a colunas “ao invés de cada registro da tabela ficar armazenado em uma linha, o registro passa a ser armazenado em colunas separadas.”[4]. Como exemplo de banco de dados desta categoria são: Hadoop, Cassanda, Hypertable e Amazon SimpleDB.

Orientado a documentos

Nesta categoria de banco de dados, o armazenamento é realizado através de documentos, e o conjunto de documentos forma uma coleção. Geralmente os documentos

são codificados nos formatos JSON, XML e YAML[37]. Alguns dos bancos de dados que pertencem a esta categoria são: MongoDB, CouchDB e EjDB.

Baseado em grafos

Quando se tem dados cujas relações são bem representadas como um grafo, pode se utilizar um banco de dados baseado em grafos [37]. Com este modelo é possível "representar os dados e/ou o esquema dos dados como grafos dirigidos ou como estruturas que generalizem a noção de grafos." [18]

Alguns bancos que fazem parte desta categoria são: Neo4J, Infinite Graph, Titan, e Bigdata.

CAPÍTULO 3

TECNOLOGIAS

Este capítulo tem como foco explicar cada uma das tecnologias propostas neste texto, abordando algumas particularidades que servem para resolver o problema de escalabilidade.

Primeiramente na seção 3.1 será explicado o conceito da sigla MEAN, e as subseções seguintes detalharão cada uma das tecnologias propostas. Na subseção 3.1.1 será explicado o MongoDB. Na subseção 3.1.2 será apresentado o Express.js. Na subseção 3.1.3 aborda-se sobre o AngulaJS. Na subseção 3.1.4 o foco será no Node.js.

3.1 MEAN *Stack*

O nome MEAN é um acrônimo que representa o conjunto de quatro ferramentas: o banco de dados NoSQL **M**ongoDB[1], o *framework back-end* **E**xpress[2] , o framework front-end **A**ngularJS[3], e o servidor **N**odeJS[4]. O MEAN foi concebido através de um debate de usuários em um grupo no Linkedin[5]. Nos subtópicos abaixo serão abordadas as principais características de cada uma dessas ferramentas.

3.1.1 MongoDB

O MongoDB é um banco de dados de código aberto do tipo NoSQL. Foi implementado na linguagem C++ e é orientado a documentos. O formato dos documentos utilizados pelo MongoDB é o JSON[6]/BSON[7]. Utiliza-se este formato pela facilidade na integração de

[1] http://www.mongodb.org/
[2] http://expressjs.com/
[3] https://angularjs.org/
[4] http://nodejs.org/
[5] https://www.linkedin.com/groups/Mean-Stack-5070069
[6] *JavaScript Object Notation*, é um formato de texto baseado no conceito de chave-valor, que serve para troca de informações/dados entre sistemas.
[7] Internamente este formato é convertido para BSON, que é um arquivo JSON no formato binário, que foi criado para tornar mais eficiente a busca e o espaço de armazenamento.

dados em certos tipos de aplicações. O nome Mongo vem da expressão da língua inglesa "humongous", que significa "monstruoso" ou "gigantesco".

Algumas razões por optarem pelo Mongo para funcionar junto com ferramentas Javascript são: o tratamento de objetos JSON, possuir uma boa aceitação por parte da comunidade de desenvolvedores[1], ter um bom desempenho com grande volume de dados [6], a capacidade de executar códigos em Javascript para as consultas, além de possuir uma grande integração[8] com Node.js.

A tabela 3.1 apresenta várias terminologias e conceitos padrões do SQL e as terminologias e conceitos correspondentes que o MongoDB utiliza.

Termos/Conceitos SQL	Termos/Conceitos MongoDB
base de dados	base de dados
tabela	coleção
linha	documento (BSON)
coluna	campo
índice	índice
joins	documentos incorporados (*embedded*)
chave primária	chave primária
Especificar uma coluna ou um conjunto de colunas como chave primária	No MongoDB, a chave primária é automaticamente atribuída ao campo _id

Tabela 3.1: Comparação entre as terminologias do SQL e do MongoDB. [24]

Escalabilidade através do Sharding

O termo *Sharding* é uma técnica de escalabilidade horizontal em que os dados de uma base de dados são divididos em muitos *Shards*, os quais podem ser distribuídos entre várias máquinas [26].

O MongoDB tem suporte a uma técnica conhecida como *Autosharding*, que "permite a construção de um *cluster* de banco de dados escalável horizontalmente projetado para incorporar novas máquinas de forma dinâmica" [5].

O *cluster* é formado de três componentes: blocos de servidores chamados *Shards*, que são reponsáveis pelo armazenamento dos dados, os servidores de configuração chamados de *mongod*, que contêm os metadados e as informações de rotas, e os serviço

[8] Através do próprios drives, e com a utilização da api Mongoose.

de rotas chamados *mongos*, que são responsáveis "pelo roteamento das operações ao destino apropriado"[5].

A figura 3.1 abaixo mostra como os componentes que constituem o *cluster* do MongoDB interagem para prover uma escalabilidade horizontal.

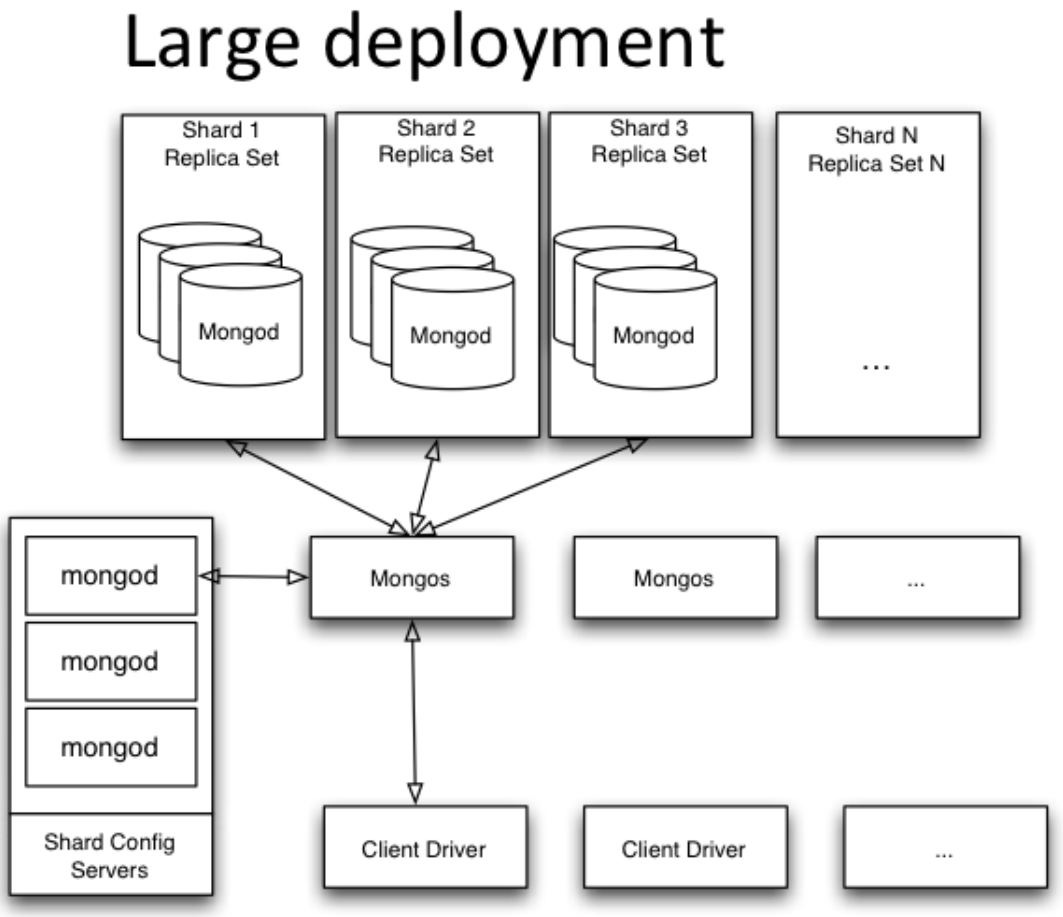

Figura 3.1: Topologia do Sharding. [3]

Armazenamento

O MongoDB "mapeia diretamente um arquivo armazenado em disco para um array de bytes em memória virtual, não memória física RAM, onde a lógica de acesso aos dados é implementado usando aritmética de ponteiro. O uso de arquivos mapeados em memória é eficiente para melhorar o desempenho"[14].

Uma das características do MongoDB é a de ter consistência eventual (*eventual consistency*), que significa que existe um momento apropriado para o sistema tornar-se consistente. [28]

O gerenciador de memória virtual do sistema operacional é responsável por decidir quais partes da base de dados vão para o disco e quais ficam em memória. Este mecanismo faz com que o MongoDB não tenha controle sobre quando os dados são escritos no disco[26].

3.1.2 Express.js

O Express.js é um *framework* que visa facilitar o desenvolvimento de aplicações Web com o Node.js. O Express.js permite criar servidores web e receber requisições HTTP de maneira simples. Ele também permite a criação de um conjunto de diretórios com uma estrutura padrão, além de organizar as rotas dos arquivos para as *views* da aplicação. Geralmente os projetos que utilizam-se do Express também aderem a algum *framework* de *templates* como Jade ou EJS.

3.1.3 AngularJS

AngularJS é um *framework* javascript *client-side* que segue o modelo MVC[9] e é mantido pela Google.

Existem muitos outros frameworks além do AngularJS, e ao meio de tanta competitividade ele tem se destacado. O gráfico apresentado na figura 3.2 é um bom argumento para mostrar a popularidade do AngularJS.

A figura 3.2 foi retirada do site "Info Q [34]", onde foi realizado uma enquete para saber qual *framework* é mais utilizado pelos desenvolvedores. A enquete é realizada como um *drag and drop*, onde o usuário que está votando terá que posicionar o nome da aplicação entre os dois eixos do gráfico de acordo com sua opinião.

A linha vertical, *"Value Proposition"*, representa a importância da aplicação para o usuário que está votando, a linha horizontal, *"Adoption Readiness"* representa o quanto o usuário usa a aplicação na vida real.

[9]MVC (*Model-View-Controller*, que em português é Modelo-Visão-Controlador) é uma forma de estrutura seu projeto/aplicação de forma que a interface de interação (*views*) esteja separada do controle da informação em si (*models*), separação essa que é intermediada por uma outra camada controladora (*controllers*).

O tamanho dos circulos do gráfico represetam a quantidade de votos recebidos.

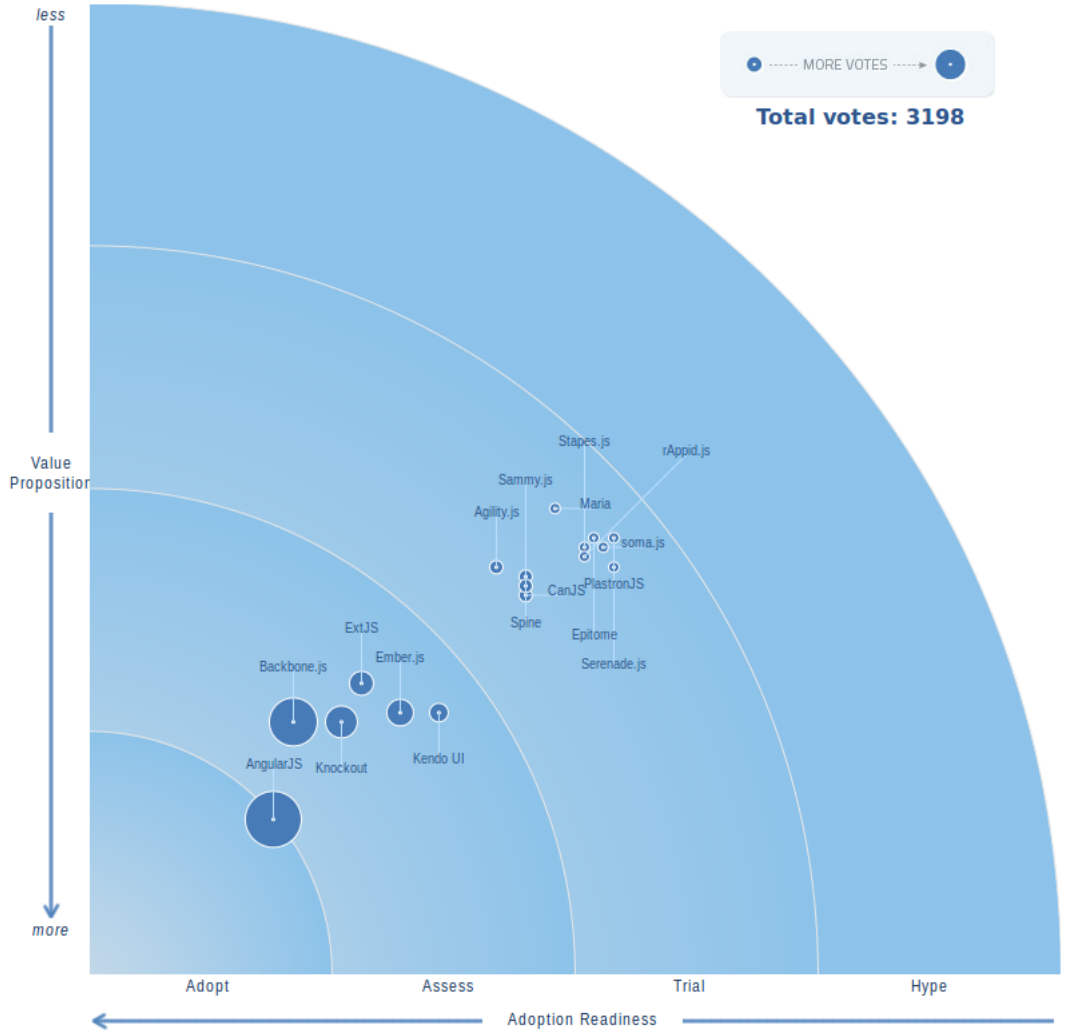

Figura 3.2: Gráfico comparando frameworks Javascript. [34]

Algumas características que se destacam no AngularJS são:

SPA - *Single Page Application*

O conceito de Single Page Application é que cada parte de página é carregada de forma independente, ou seja, quando uma página é carregada, todas as demais atualizações são feitas através de requisições AJAX[10] e renderizações parciais na página.

Através destas renderizações parciais tem-se um aumento de desempenho devido à diminuição da quantidade de dados que precisam ser trafegadas entre o cliente e o servidor.

Two-Way Data Biding

É um mecanismo para facilitar o desenvolvimento do sistema, pois reduz a quantidade de código escrito. *Two-way Data biding* funciona por meio de processo que sincroniza os dados na *view* e no *model* como aparesentado na figura 3.3.

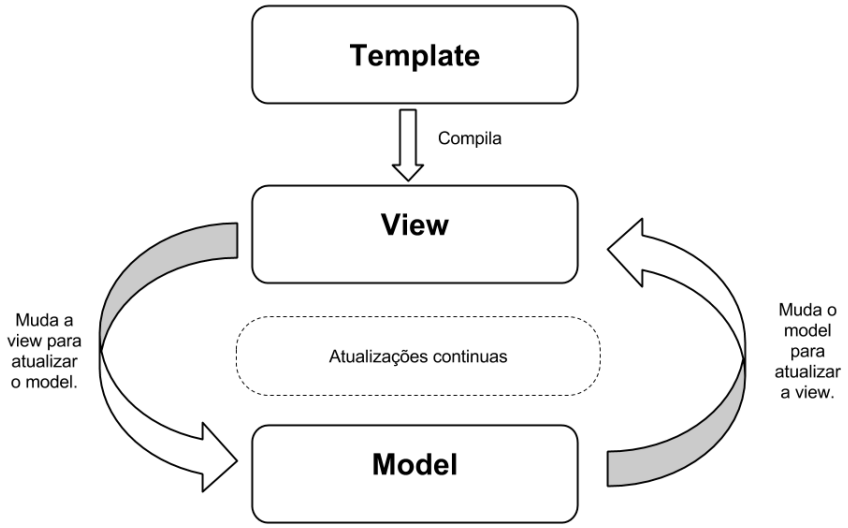

Figura 3.3: Digrama do *Two-way data biding* [35]

[10]AJAX é uma metodologia que utiliza tecnologias como XML e Javascript para fazer requisições assíncronas ao servidor e com as informações retornadas modificar uma página carregada através do DOM, sem a necessidade de recarregar todo seu conteúdo.

Injeção de dependências

A injeção de dependências consiste basicamente em fornecer recursos extras necessários na aplicação de forma transparente ao usuário, de modo que o desenvolvedor somente deverá solicitar um recurso, que será injetado pelo *framework* e ficará disponível para uso.

3.1.4 Node.js

O Node.js é uma plataforma de desenvolvimento web que funciona com a linguagem Javascript no lado do servidor, para criação de aplicações e páginas web de alta escalabilidade.

Foi concebida por Ryan Dahl em 2009 , e desde então vem ganhando muita popularidade entre os desenvolvedores web, e sendo utilizado por grandes empresas e instituições tais como LinkedIn, Microsoft, GitHub, MySpace, entre outras[33].

A escolha da linguagem Javascript foi devido a enorme quantidade de bibliotecas para I/O que a linguagem possui, permitindo assim criar as funções assíncronas para a plataforma.

A arquitetura do Node.js é composta em sua maior parte por componentes desenvolvidos em C e em Javascript . Os principais componentes da parte escrita em C são: a *V8 Javascript Engine*[11], o *Node Bindings*[12], a *Thread Pool*[13], e o *Event Loop*. Para a parte responsável pelo Javascript, foi criada uma biblioteca chamada *Node Standard Library*, para permitir que o Node.js interprete códigos em Javascript.

Abaixo segue a figura 3.4 que concede um modelo visual desta arquitetura, com o intuito de facilitar o entendimento da relação entre os componentes citados do Node.js.

O Node.js foi criado de forma que suas funcionalidades pudessem ser estendidas através de módulos, para implementar diversos componentes *middleware* que facilitem o desenvolvimento de aplicações web. Estes módulos são geralmente instalados através de um

[11]É máquina virtual para Javascript do Google utilizada no navegador Chrome, que "compila e executa código fonte JavaScript, lida com a alocação de memória para objetos e limpa-os quando não são mais necessários"[9].

[12]São códigos executáveis que fazem com que o V8 e a biblioteca em javascript padrão do node sejam capazes de se comunicar.

[13]"É uma coleção de *threads* disponíveis para realizar tarefas"[31]

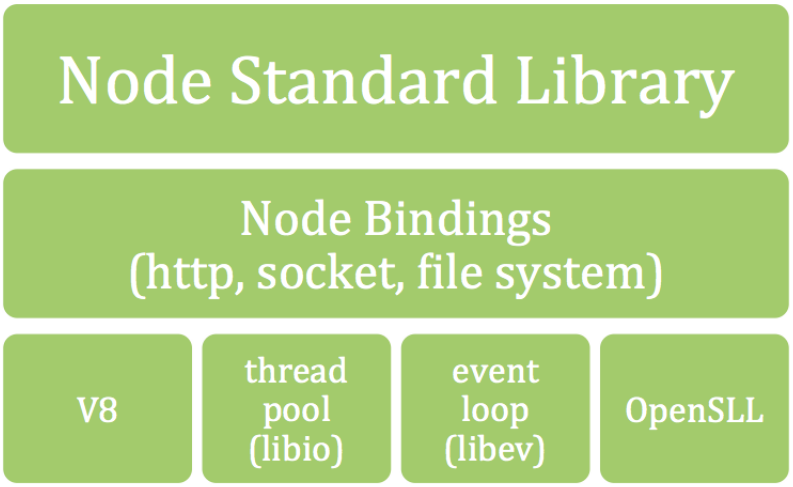

Figura 3.4: Arquitetura do Node.js. [10]

gerenciador de pacotes conhecido como npm, que significa *Node Package Manager*, e serve para facilitar diversos aspectos relacionados à aplicação como a compilação, a instalação e a atualização e também funciona como um gerenciador de dependências.

I/O não bloqueante e orientado a eventos

Quando é necessario acessar[14] os dados de uma aplicação, estes dados podem estar em vários locais, tais como na memória (L1, L2, RAM), no disco ou na rede. Cada local contém uma latência de I/O diferente, no caso para memória quando se necessita acessar a L1 é gasto aproximadamente 3 ciclos, passando para 14 ciclos na L2 e para 250 ciclos na RAM. Já para o disco e para a rede a quantidade de ciclos necessários tem um aumento significativo, sendo aproximadamente 41 e 250 milhões de ciclos, respectivamente. Ou seja, quando utiliza-se a memória é possível considerar que será um acesso rápido, também conhecido como não bloqueante, enquanto para o disco e a rede considera-se que será um acesso lento, também conhecido como bloqueante.

A figura 3.5 demonstra como é o funcionamento do *Event Loop* no Node.js, que tem como entrada uma fila (*queue*) de requisições, e no caso de ser recebido uma requisição bloqueante, esta requisição é passada para uma *Thread Pool*, que irá tratar o evento, caso contrário o evento é tratado no próprio *Event Loop*.

[14]Seja para leitura, escrita, atualização ou remoção.

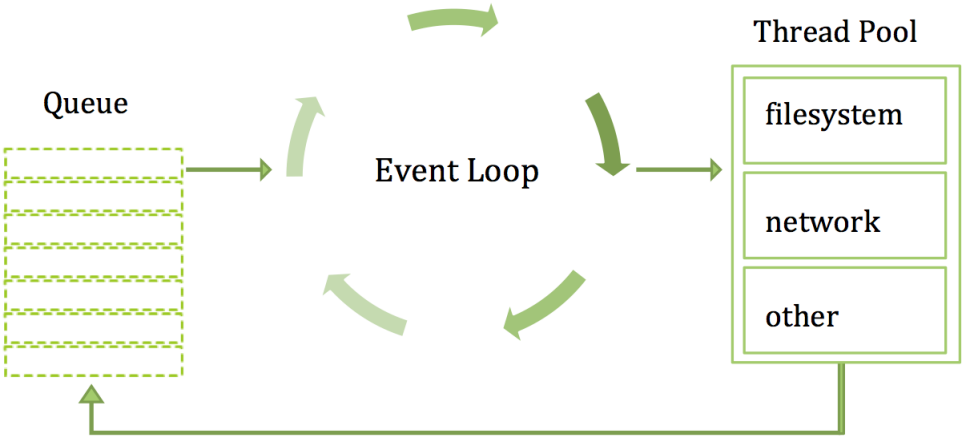

Figura 3.5: Event Loop do Node.js. [10]

O Node.js foi desenvolvido com base no conceito de "I/O não bloqueante" (também conhecido por I/O assíncrono), e isto foi feito através da utilização e adaptação de uma construção de programação conhecida como *Event Loop*, que foi apresentado na seção 2.2 do capítulo 2.

A execução do Node.js geralmente funciona através de uma única *thread* e quando existe a necessidade de I/O bloqueante, uma *thread* pertencente à *Thread Pool* fica responsável por lidar com este tipo de I/O.

Um exemplo do funcionamento do *Event Loop* do Node.js é quando se recebe uma requisição de um evento não bloqueante, sendo que este evento será tratado diretamente pela *thread* principal. Outra situação é quando se recebe uma requisição bloqueante, por exemplo uma leitura de disco, esta requisição então é enviada para a *Thread Pool* do Node.js. A *Thread Pool* irá utilizar uma *thread* para tratá-la, e quando terminar, a *thread* responsável pelo processamento bloqueante envia uma mensagem para a *thread* principal, que então executa a função de *callback*[15] respectiva à requisição.

[15] "é um mecanismo de controle de fluxo que visa beneficiar processos assíncronos."[32]

CAPÍTULO 4

APLICAÇÕES WEB COM MEAN *STACK*

No capitulo três foi apresentado cada elemento do MEAN *Stack* individualmente. Neste capítulo será apresentado e analisado, como estes elementos funcionam em conjunto, formando uma pilha (*Stack*).

Na seção 4.1 é apresentado um comparativo entre o MEAN *Stack* e a arquitetura LAMP. A integração das ferramentas do MEAN *Stack* é apresentada na seção 4.2. Na seção 4.3 são mostrados os testes realizados em relação ao desempenho.

A figura 4.1 ilustra a pilha que é formada unindo as tecnologias já explicadas no capítulo anterior. No banco de dados coloca-se o MongoDB, no lado do servidor existe o Node.js e o Express e por último no lado do cliente o AngularJS. A seta de duplo sentido representa que a passagem de dados acontece em ambos os sentidos. O principal motivo para estas tecnologias funcionarem bem em conjunto, é que todas elas têm como base a linguagem Javascript, sendo que esta característica faz parte da proposta do MEAN *Stack* que é o desenvolvimento de aplicações web escaláveis com a utilização de um conjunto de ferramentas em Javascript.

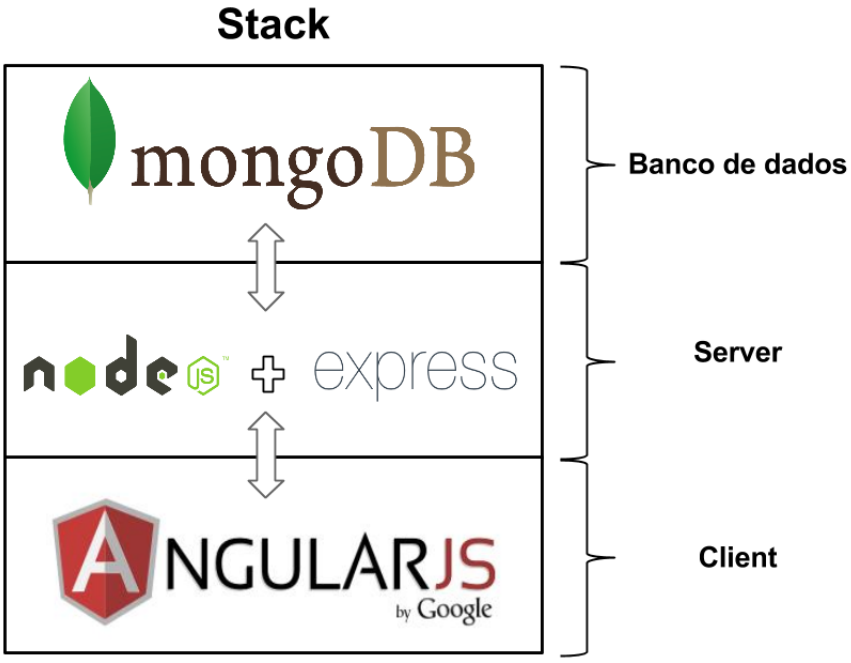

Figura 4.1: MEAN *Stack* visto como uma pilha

4.1 Comparativo em relação ã arquitetura de aplicações LAMP

Para cada componente da pilha existe uma quantidade razoável de tecnologias semelhantes. Se for utilizada como exemplo a figura 3.2, mostrada no capítulo 3, pode-se observar uma grande quantidade de tecnologias concorrentes que existem, cada uma se destacando em alguma situação. Neste cenário torna-se muito complicado concluir o que é melhor, ainda mais que o próprio MEAN pode variar, por exemplo para PEAN quando o MongoDB é substituído pelo PostgreSQL. A comparação do MEAN será feita com o LAMP, pois este tem um destaque quando o assunto se trata de aplicações web. [17]

Assim como o MEAN, o LAMP é uma pilha de software livres com o foco no desenvolvimento web. A sigla LAMP é um acrônimo de Linux, Apache, MySQL e Perl/PHP/Python.

Uma característica que difere os dois conjuntos de ferramentas, é que o LAMP necessita do Linux ou outro sistema operacional, ficando dependente do sistema escolhido. Enquanto o MEAN pode ser executado em qualquer plataforma.

O servidor do MEAN é o Node.js, enquanto ao do LAMP é Apache. O que destaca o Node.js neste caso é o fato de ser totalmente não bloqueante e ser orientado a eventos, como já citado na subseção 3.1.4, o que permite concorrência entre as requisições. Porém

como o Node.js é recente, não existe muitos *plugins* que auxiliam seu uso, o que não acontece com o Apache, que já está há muito tempo no mercado, sem dizer que em muitos casos ele é escolhido como servidor padrão de muitos desenvolvedores.

O Banco de dados do MEAN é o MongoDB, que como já explicado anteriormente pertence a categoria NoSQL. O LAMP utiliza o MySQL, que pertence a categoria SQL. Para comparação entre esses bancos de dados deve-se considerar a situação, por exemplo uma consulta com intuito de retornar um valor no MongoDB é mais rápida do que no MySQL, pois o MongDB não possui *schemas* de tabelas mas sim um arquivo *JSON*. Em contrapartida, a atualização dos dados do banco pode ser muito lenta para o MongoDB, isso considerando que existem muitos dados armazenados no banco.

Para o desenvolvimento da aplicação no lado do cliente, o MEAN e o LAMP seguem abordagens consideravelmente diferentes, enquanto no MEAN *Stack* a ferramenta responsável é o AngularJS, no LAMP não há uma ferramenta padrão definida.

No lado do servidor, o MEAN *Stack* utiliza o Express.js e o Node.js, e o LAMP, pode ser codificado através de três linguagens de programação: Perl, PHP ou Python.

No MEAN, o Express.js serve como a camada de controle, empacotando os dados e enviando para AngularJS que utiliza a informação para realizar alguma ação e renderizar a página. A principal vantagem de se utilizar o AngularJS é que ele funciona totalmente no lado do cliente, como já foi explicado no capítulo anterior.

Todas as características citadas nesta seção podem ser observadas, de forma mais direta, na tabela 4.1.

O fato de todas as tecnologias do MEAN serem em Javascript, garante que o desenvolvedor precise saber apenas uma linguagem de programação. Porém isso também tem seu lado negativo, que é o engessamento do projeto só em Javascript, ou seja seu projeto fica dependente apenas do que o Javascript consegue fazer.

	MEAN	LAMP
Sistema Operacional	Pode ser executada em qualquer plataforma.	Utiliza o Linux como sistema operacional.
Servidor	**Node.js** - Totalmente não bloqueante. - Orientado a eventos. - Não possui muitos plugins para o Node.js, pois é uma tecnologia recente.	**Apache** - Existe há muito tempo, o que garante ferramentas que trabalhem com junto com o Apache. - Geralmente é a escolha padrão de muitos desenvolvedores.
Banco de dados	**MongoDB** - Banco de dados NoSQL. - Utiliza-se de arquivos JSON. - Ganha do MySQL na operação de consulta no banco de dados.	**MySQL** - Banco de dados SQL. - Tabelas baseadas em *Schemas*. - Ganha do MongoDB na atualização dos dados no banco.
Codificação	**AngularJS** - No lado do cliente. **Express.js** e **Node.js** - No lado do servidor	**Pearl**, **PHP** ou **Phyton** - No lado do servidor. - Possuem uma comunidade com muitos desenvolvedores.

Tabela 4.1: Tabela de comparação entre MEAN vs LAMP

4.2 Aplicação *Math Race* com MEAN *Stack*

Nesta seção será apresentada uma aplicação desenvolvida com o MEAN *Stack*. A subseção 4.2.1 descreve a proposta de desenvolvimento da aplicação. A subseção 4.2.2 mostra como é a utilização e o funcionamento da aplicação. A subseção 4.2.3 aborda como foi desenvolvida a aplicação.

4.2.1 Proposta da aplicação

Uma aplicação simples foi desenvolvida, com o intuito de demonstrar diversos aspectos do MEAN *Stack*, como a organização e a integração das ferramentas que compõem o MEAN *Stack*. A aplicação foi baseada na implententação do *Math Race*, desenvolvida por Iván Loire[20], possuindo apenas o Node.js e o Socket.io como ferramentas em comum.

4.2.2 Utilização e funcionamento da aplicação

A aplicação é um jogo, cujo objetivo é realizar uma competição em tempo real, para ver qual jogador acerta mais operações de matemáticas aleatórias de adição e subtração, em determinado tempo.

Na figura 4.2 é mostrada a interface da aplicação, que contém a operação aleatória, o campo para entrada de dados, o marcador de tempo (*Time*), o campo de pontuação (*score*) e o Hall da fama (*Hall of fame*).

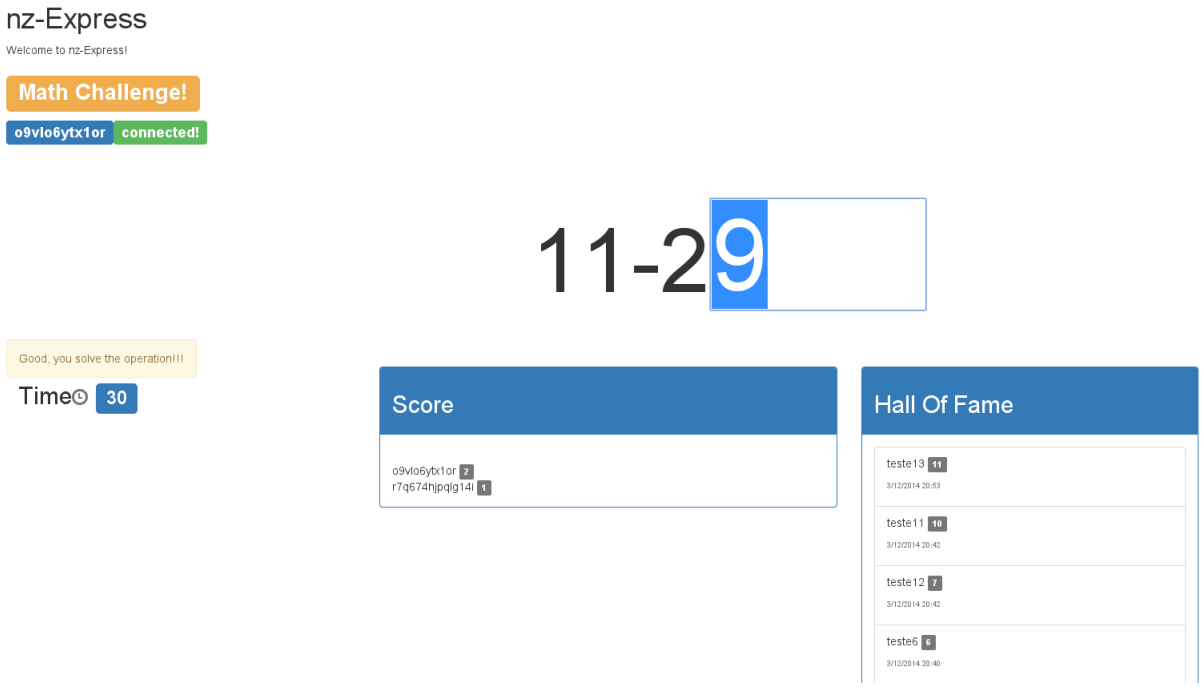

Figura 4.2: Imagem da interface da aplicação

Todos os jogadores possuem o mesmo tempo para responder a mesma operação matemática. A operação matemática muda para todos os jogadores após a resposta correta da operação anterior ter sido preenchida por qualquer jogador.

O jogo funciona através de rodadas, e para cada rodada o usuário tem um tempo limite pré-determinado para acertar o resultado da conta. A cada resultado correto o valor da pontuação do usuário, que começa em zero, é incrementado com mais um ponto. Ao final de uma rodada os usuários que efetuaram alguma pontuação são adicionados no "Hall da fama", que é ordenado pelos dez usuários com mais pontos obtidos em uma única rodada, e as pontuações dos usuários são zeradas, para que uma nova rodada se inicie.

Na figura 4.3 é demonstrado, através de um diagrama, o funcionameto da aplicação.

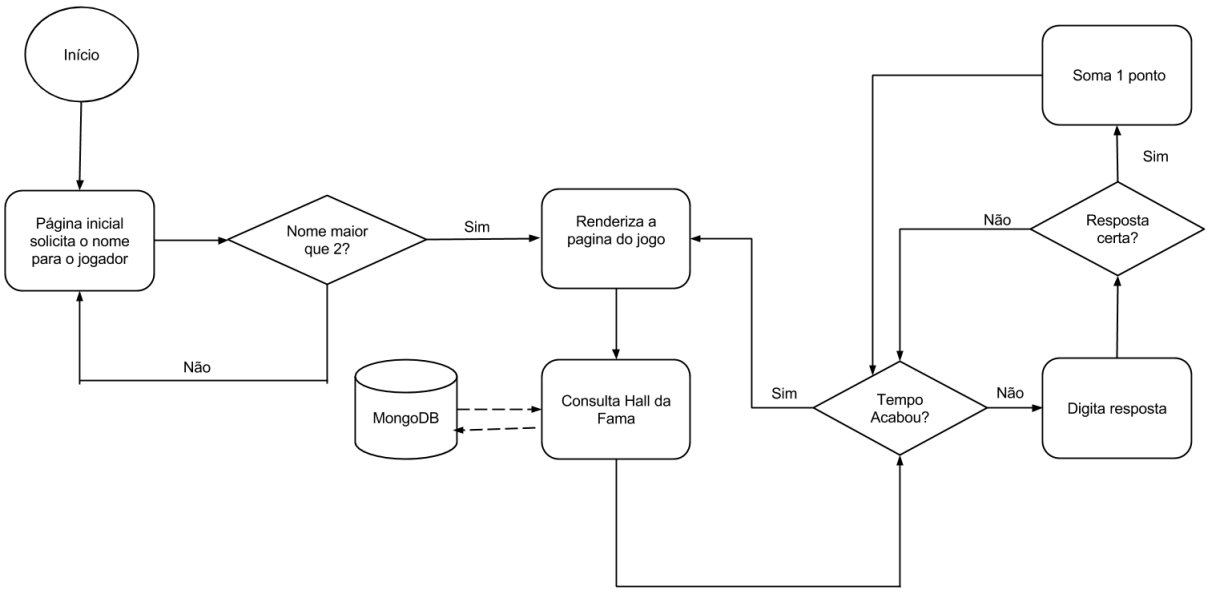

Figura 4.3: Diagrama do funcionamento da aplicação

4.2.3 Desenvolvimento da aplicação

Nesta subseção será abordado como é o desenvolvimento de uma aplicação baseado no MEAN *Stack*. Partindo de como foi realizada a escolha da estrutura da aplicação, e finalizando com a integração das ferramentas do MEAN *Stack*.

Estrutura da aplicação

No início do desenvolvimento da aplicação, verificou-se as possibilidades em relação à estrutura e organização de arquivos que seria utilizada, pois não exite uma abordagem padrão em relação a este assunto.

Nas pesquisas iniciais as primeiras possibilidades que apareceram foram através do MEAN.js e o MEAN.io que são geradores automáticos de estruturas de arquivos para o MEAN *Stack*. Apesar de fornecerem uma estrutura pronta, ao lidar com geradores, é necessário que se programe de uma maneira pré-determinada de acordo com o gerador escolhido, o que torna a aplicação um pouco mais complexa de ser apresentada, e detalhada, e foge do escopo desta monografia.

A opção escolhida para a criação da estrutura de diretórios, foi através de um submódulo do Express.js chamado *express-generator*. A figura 4.4 mostra como as pastas e os arquivos são organizados neste gerador, ao total são 7 pastas e 9 arquivos.

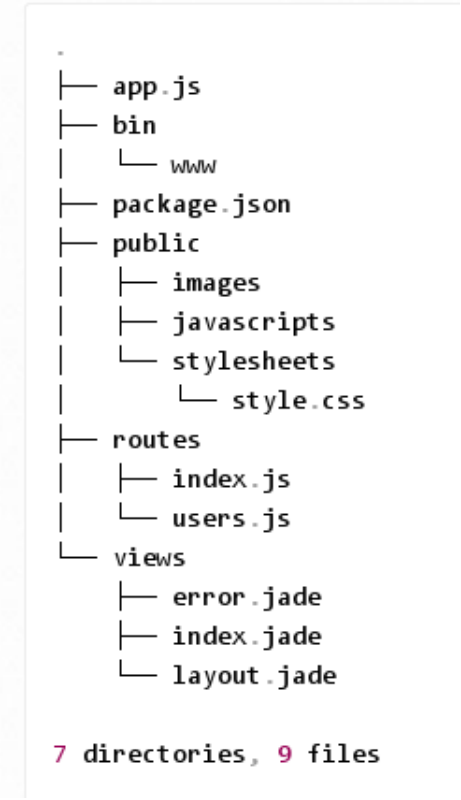

Figura 4.4: Estrutura criada pelo express-generator [11]

A pasta *public* contém todos os arquivos de imagem, Javascript e CSS que serão enviados para o cliente, como por exemplo os arquivos Javascript do Angular.js, e o arquivo CSS chamado style.css.

Na pasta *views* são definidas as páginas da aplicação, contendo a página do *layout*, do *index*, e uma página de erros. Quando o usuário acessa a aplicação, primeiramente é carregada a página de *layout*, que então carrega a página *index*, e caso algum erro ocorra é enviada uma mensagem de erro.

Para as rotas que definem como serão tratadas as requisições que podem ser realizadas pela aplicação, o *express-generator* cria uma pasta chamada *routes*. Ao receber requisições, o servidor através de funções definidas na pasta *routes*, pode desde renderizar as páginas solicitadas, até encaminhar as requisições para outras

rotas, afim de, por exemplo, realizar uma consulta em um banco de dados.

A parte do servidor que o Node.js é responsável, fica no arquivo www da pasta *bin*, e no arquivo app.js na raiz da aplicação.

Além da estrutura criada pelo *express-generator*, foram criadas mais duas pastas, que são a *models* e a *lib*. Na *models*, ficam os arquivos responsáveis pela conexão e pelos acessos ao MongoDB. A *lib* contém as principais funções da aplicação responsáveis pelo funcionamento do jogo *Math Race*, e da comunicação cliente/servidor, que é feita através do Socket.io, sendo que a sua explicação foi colocada na seção 6.1 do apêndice A.

Integração das ferramentas do MEAN *Stack*

Quando o usuário acessa a aplicação, ocorre uma série de mensagens, entre o lado do cliente e do servidor, afim de informar o servidor que há um novo usuário conectado, e fazer com que o cliente obtenha os dados da partida.

O diagrama da figura 4.5, demonstra a sequência de mensagens que ocorrem quando um usuário acessa a aplicação (mensagem 1).

No lado do servidor, o Node.js envia os arquivos da pasta *public* e o *index* da aplicação (mensagem 2). No lado do cliente, o Angular.js envia uma solicitação de conexão através da função *connect* do Socket.io (mensagem 3), e o Node.js responde com uma mensagem avisando que o usuário está conectado (mensagem 4). Ao receber essa mensagem o Angular.js envia outra mensagem chamada "*join*" (mensagem 5), e o Node.js envia os dados referente a operação, a pontuação e o *hall* da fama (mensagem 6, 7 e 8). Por último o Angular.js faz um requisição ao MongoDB para obter o hall da fama atualizado (mensagem 9).

A cada final de rodada, o Node.js envia uma nova operação, e o Angular.js envia uma mensagem ao Node.js solicitando o *hall* da fama atualizado. Essa mensagem é enviada através de um módulo do Angular.js chamado *ng-resource*, sendo que através deste módulo é possível interagir com o Node.js utilizando o RESTFul[1].

[1] "RESTful é um serviço web que utiliza o paradigma de arquitetura do REST, ou seja, é o termo

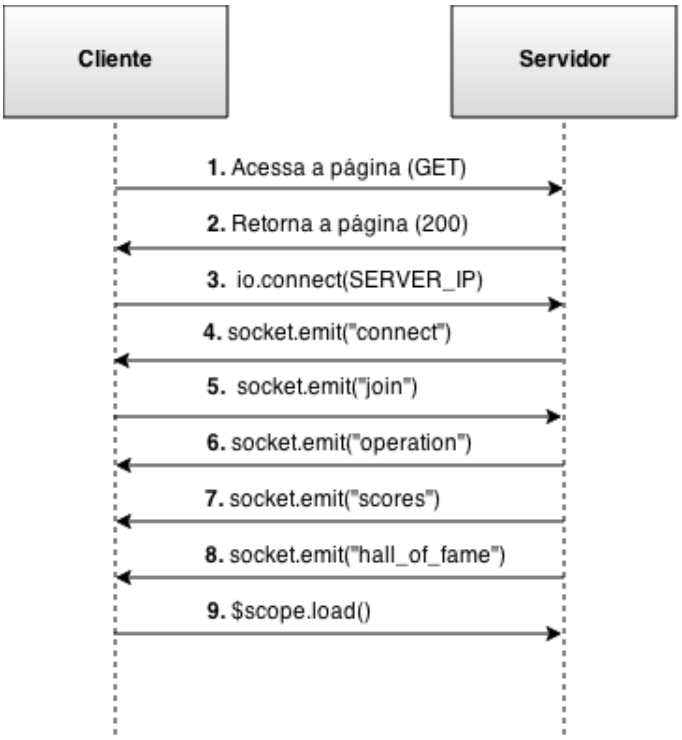

Figura 4.5: Diagrama de sequência do acesso da aplicação

Caso o usuário tenha efetuado alguma pontuação na rodada, o Angular.js envia uma mensagem ao Node.js, contendo um objeto JSON com o nome e a pontuação efetuada. O Node.js então faz a inserção no MongoDB, se for um novo usuário, ou atualiza a pontuação, se o nome do usuário já estiver cadastrado no MongoDB, e a nova pontuação for maior do que a antiga.

4.3 Testes de desempenho

Nesta seção serão apresentados os testes que foram realizados através de duas aplicações, o *Math Race* e o "Encurtador de URL".

No *Math Race*, foi efetuado um teste simples através do envio de um conjunto de requisições para consultas ao banco de dados, obtendo-se o tempo de duração para o servidor responder cada conjunto de requisições, além da média de requisições por segundo.

A aplicação "Encurtador de URL" foi implementada em diversos ambientes, definidos na tabela 4.3, afim de avaliar o consumo da memória, a utilização da CPU, e a quantidade

normalmente usado para se referir a implementação de Web Services que utilizam tal arquitetura."[38]

de requisições realizadas por tempo.

4.3.1 Testes na Aplicação *Math Race*

Os testes foram realizados aumentando gradativamente a quantidade de requisições e requisições concorrentes, afim de verificar a latência de resposta do servidor. A cada requisição é realizada uma consulta no banco de dados para obtenção da lista dos dez primeiros usuários e suas pontuações no *hall* da fama.

Na figura 4.6, podemos observar parte da lista de objetos retornada pela consulta ao banco de dados, que cada cliente realiza ao acessar a aplicação durante os testes.

```
{ _id: 547f788612e68d671c64b48b,
__v: 0,
timestamp: '3/12/2014 20:53',
score: 11,
player: 'teste13' },
{ _id: 547f75e9ff97ddc81ba16cf2,
__v: 0,
timestamp: '3/12/2014 20:42',
score: 10,
player: 'teste11' },
...
```

Figura 4.6: Lista de objetos retornada pelo MongoDB

A máquina utilizada nos testes tem como configurações um Core i7 de 3.6GHZ, e 8GB de memória RAM, utilizando o Fedora 20 de 64 *bits* como sistema operacional.

Na tabela 4.2 são mostrados os resultados dos testes realizados na aplicação, através de uma ferramenta específica para teste de carga (*Load test*) conhecida como *weighttp*[2]. No *weighttp* é possível definir parâmetros como a quantidade de requisições que serão enviadas ao servidor e o número de requisições concorrentes que ocorrerão durante o teste.

Para estes testes foram imaginados dois cenários diferentes, sendo determinada uma quantidade de requisições concorrentes fixa (100 e 1000) em cada cenário. Para cada cenário variou-se a quantidade de requisições de 1.000 até 100.000, obtendo-se o tempo demorado pelo servidor para atender todas estas requisições e a taxa de requisições por segundo.

[2]http://redmine.lighttpd.net/projects/weighttp/wiki

# requisições concorrentes	# requisições	tempo(seg)	req/seg
100	1000	0,69	1458
	2000	1,357	1472
	5000	3,437	1454
	10000	7,690	1414
	30000	20,928	1433
	50000	34,986	1429
	70000	47,102	1486
	100000	67,967	1471
1000	1000	3,250	330
	2000	3,350	658
	5000	3,691	1354
	10000	7,503	1332
	30000	21,963	1365
	50000	37,327	1339
	70000	52,157	1342
	100000	74,914	1334

Tabela 4.2: Carga na quantidade requisições

Pode-se notar através da tabela 4.2, que à medida que carga de requisições aumenta, o tempo total para o servidor responder estas requisições cresce consideravelmente, independentemente do número de requisições concorrentes, mas mantém, em média, a mesma taxa de requisições por segundo. No gráfico da figura 4.7, pode-se visualizar de maneira mais clara o comportamento da aplicação, durante a carga de requisições.

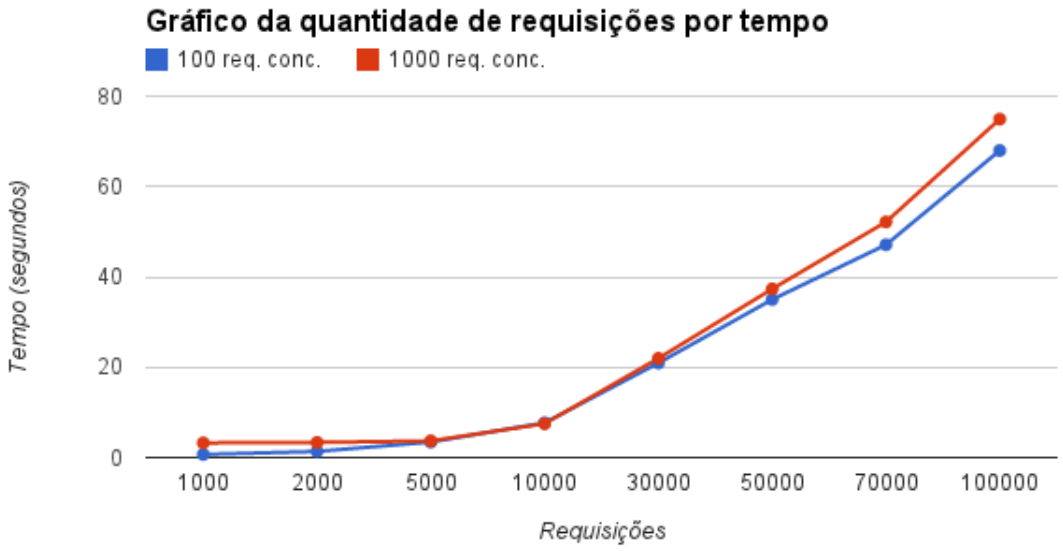

Figura 4.7: Gráfico de requisições por tempo

4.3.2 Análise de testes de desempenho do Node.js e do MongoDB com outros ambientes

Os principais componentes responsáveis pela a escalabilidade no MEAN *Stack* são o MongoDB, através do *autosharding*, e do mapeamento dos dados em memória, e o NodeJS, através do *Event Loop*, e da *Thread Pool*, como mencionado no capítulo 3.

Existem diversas referências que comparam o desempenho destes dois componentes juntos com outros ambientes, variando desde a linguagem de programação até a base de dados utilizada.

A referência que esta monografia irá utilizar como base é a do artigo de Ricardo Schroeder e Fernando dos Santos, intitulado "Arquitetura e testes de serviços web de alto desempenho com Node.js e Mongodb" [19] que implementou um "Encurtador de URL" como aplicação.

A aplicação do "Encurtador de URL" serve para mapear uma *hash*, de 6 caracteres, para uma URL. Cada *hash* é gerada de maneira única, e aleatória, e foram alocados um milhão de URL's para os testes.

A cada acesso é feita a verificação se a *hash* utilizada é válida. Primeiro é verificado se a *hash* tem 6 caracteres e depois se ela existe no banco de dados, caso alguma destas verificações falhe é retornado somente um código de resposta de erro 404.

Para os testes, a aplicação definida foi implementada em 6 ambientes, que foram descritos na tabela 4.3. Cada aplicação de um ambiente efetua a mesma requisição e será testada utilizando uma base de amostras contendo duas mil *hashs*. A duração foi estipulada em 60 segundos, sendo que nos 5 segundos iniciais, foram criados quarenta usuários que irão se conectar de maneira incremental, permanecendo ativos durantes os próximos quarenta e cinco segundos e reduzindo durante os dez segundos finais.

Os testes realizados na aplicação foram executados em um ambiente virtualizado contendo as seguintes características: Sistema Operacional CentOS 6.2 x86, 1Gb de memória RAM, 2 núcleos de 2,4Ghz Intel Core I5. A aplicação utilizada para os testes foi o JMeter, que é um software que serve para realizar testes de desempenho, carga e stress,

Servidor	Linguagem de Programação	Banco de Dados
Node.JS	Javascript	MongoDB
Node.Js	Javascript	PostgresSQL
Netty	Java	MongoDB
Netty	Java	PostgresSQL
Apache	PHP	MongoDB
Apache	PHP	PostgresSQL

Tabela 4.3: Ambientes dos testes [19]

desenvolvido pela Apache[3] . Os resultados são mostrados nas tabelas 4.8, 4.9 e 4.10.

[3]http://jmeter.apache.org/

Consumo de Memória RAM

A memória RAM é um dos princípais pontos de análise em um servidor web, pois está ligada diretamente ao número de requisições que o servidor é capaz de atender em determinado tempo.[19] O resultado do teste demonstrou o uso de memória constante e estabilidade por parte de cada plataforma. Os piores resultados ficaram com os ambientes que utilizam o servidor Apache e a linguagem PHP, enquanto o ambiente que usa Node.js e o MongoDB foi o segundo melhor no índice de consumo de memória.

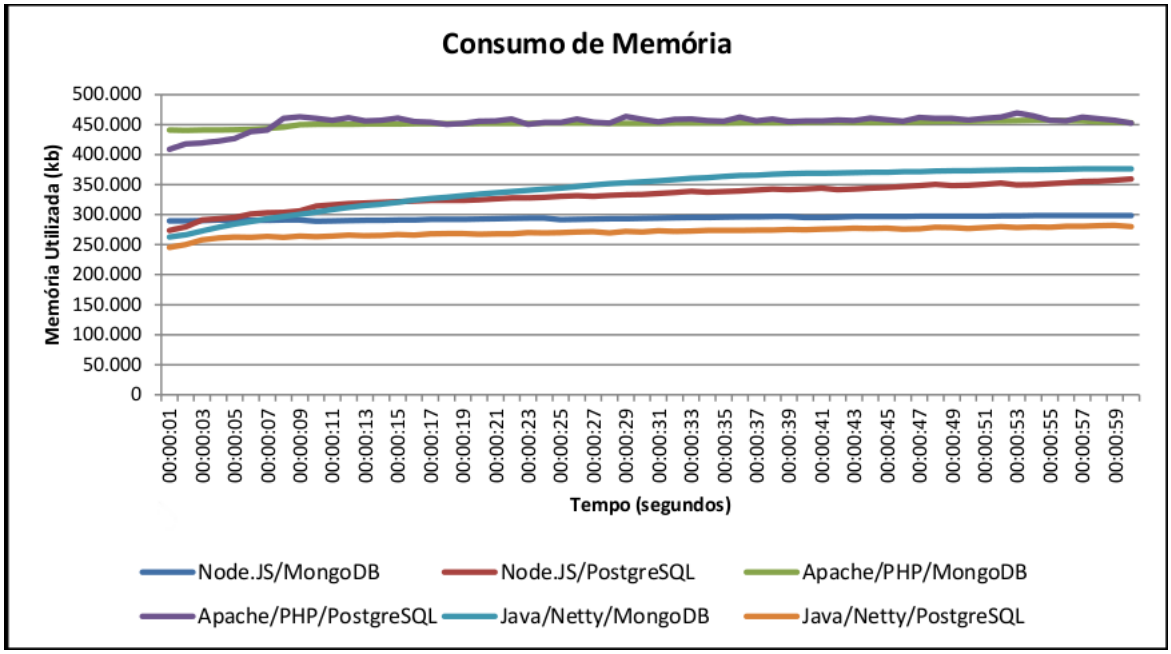

Figura 4.8: Consumo de memória RAM durante a execução dos testes [19]

Utilização de CPU

Analisar a utilização de CPU é um fator relevante para que se tenha uma noção da carga gerada pelas requisições efetuadas pelos clientes. Quanto mais próximo de 100% estiver o uso da CPU, maiores serão as chances do servidor não conseguir atender as requisições recebidas.

No gráfico da figura 4.9 nota-se um melhor desempenho do Node.js junto ao MongoDB em relação aos outros ambientes, ficando com 40% de uma média de uso, enquanto em outros ambientes como o que utilizou Java/Netty/PostgreSQL este consumo passou para aproximadamente 90%.

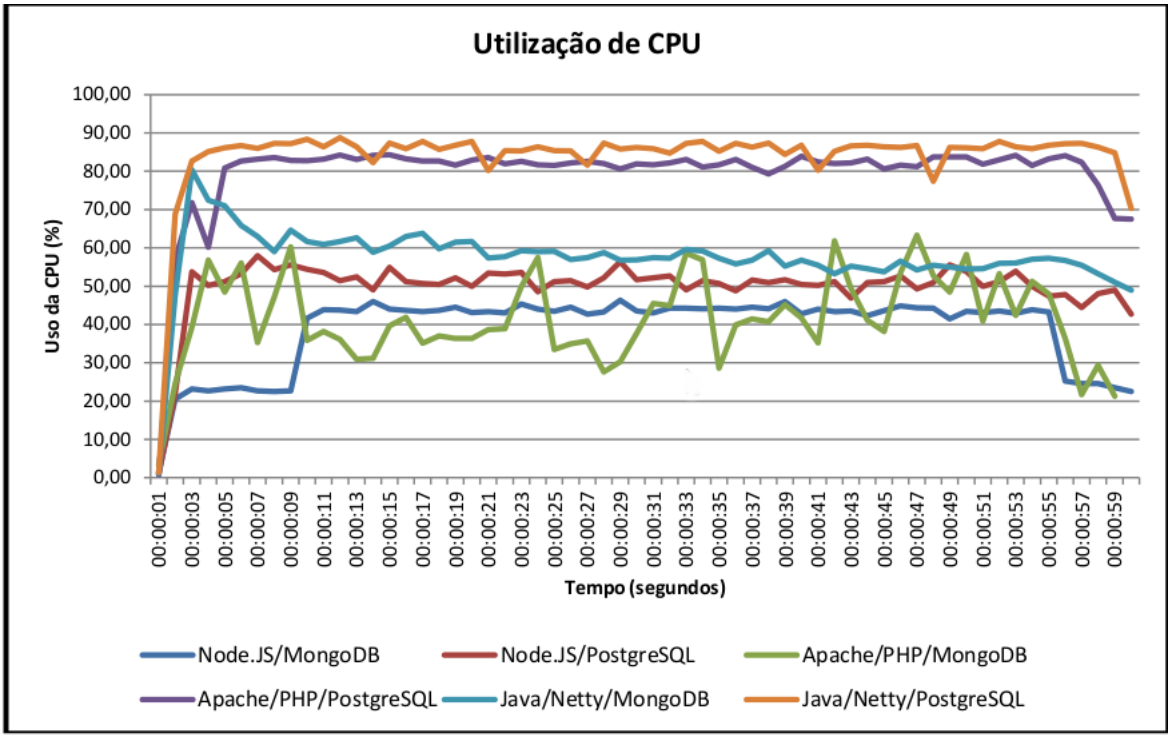

Figura 4.9: Consumo de CPU (%) durante a execução do teste. [19]

Quantidade de requisições por tempo

O teste de requisições por tempo basicamente indica o quanto de usuários (requisições) o servidor é capaz de absorver.[19]

Na figura 4.10 é possível observar que os dois ambientes que mais responderam requisições, que no caso foram o Node.JS/MongoDB e o Netty/MongoDB, utilizaram o mesmo banco de dados. Mas o crédito desse resultado positivo, também se deve aos servidores de alto desempenho, que possuem como característica atender o maior numero de requisições simultâneas. Outro ponto que chama atenção, é como os ambientes com MongoDB se destacaram sobre os ambientes com PostgresSQL.

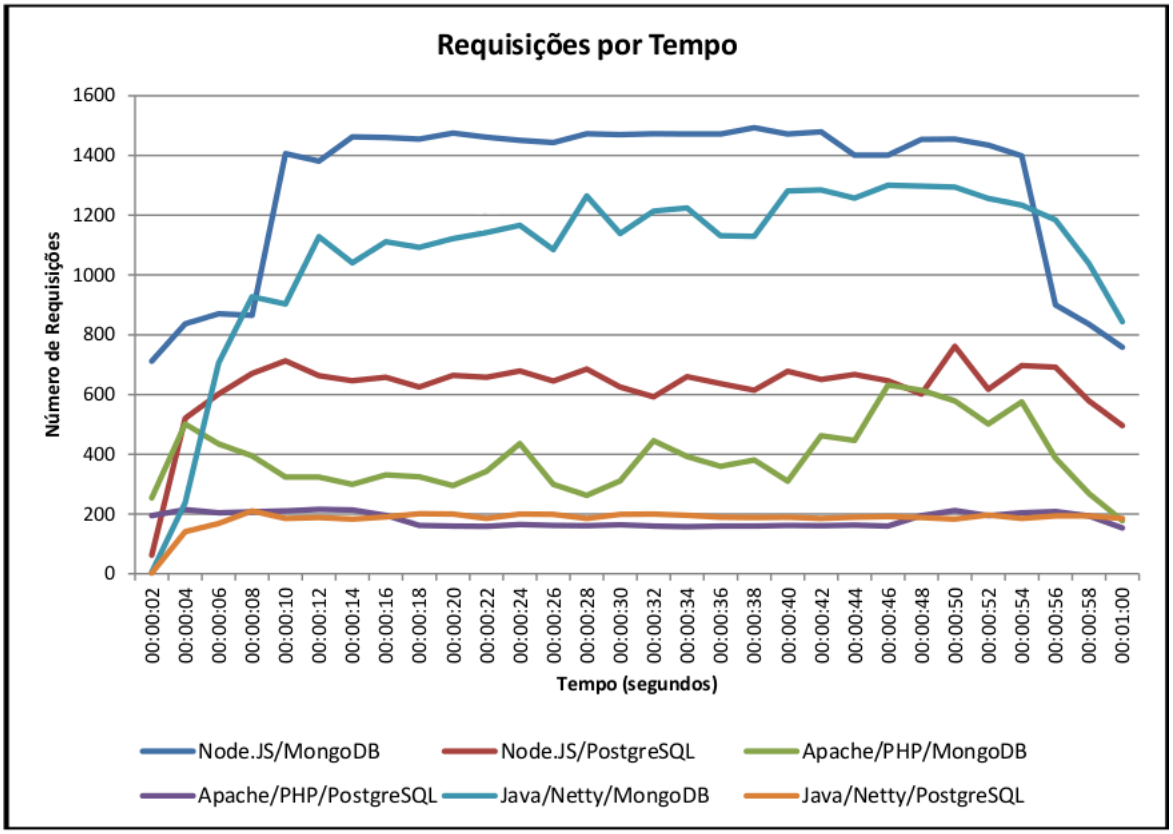

Figura 4.10: Requisições por tempo. [19]

Após ser realizada uma análise dos testes, tanto no *Math Race* quanto no "Encurtador de URL", concluímos que realmente a utilização do Node.js junto ao MongoDB, proporciona um ganho de desempenho considerável, em relação as tecnologias comparadas. Este ganho se deve ao fato das particularidades que cada tecnologia adota, para lidar com escalabilidade, no caso o Node.js com o *Event Loop* e o MongoDb com o *Autosharding*.

CAPÍTULO 5

CONCLUSÃO

Esta monografia apresentou os conceitos e as tecnologias do MEAN *Stack*, além de mostrar como é feita a integração destas ferramentas para se criar uma aplicação web. Foram apresentados também os resultados de análises com aplicações implementadas neste ambiente, com o intuito de se alcançar uma conclusão sobre a escalabilidade através da utilização das ferramentas do MEAN *Stack*.

Através da análise dos resultados propostos foi concluído que o MEAN *Stack* é de fato uma ótima opção para o desenvolvimento de aplicações web escaláveis, principalmente através de dois de seus componentes o Node.js e o MongoDB. O Angular.js e o Express.js atuam como facilitadores no processo de desenvolvimento, sem estes componentes, dependendo da complexidade da aplicação, o tempo gasto no desenvolvimento da aplicação do lado do cliente pode aumentar drasticamente, e a utilização do Node.js pode ser consideravelmente mais trabalhosa.

5.1 Trabalhos Futuros

Para os trabalhos futuros, seria interessante uma abordagem de maneira mais ampla, através da comparação do comportamento de diversas aplicações, com diferentes funcionalidades, e em diversos ambientes como Ruby e Python, além dos que foram utilizados nesta monografia.

BIBLIOGRAFIA

[1] Db-engines web site. http://db-engines.com/en/ranking, 2014.

[2] NoSQL Archive. List of nosql databases. http://nosql-database.org/, 2014.

[3] Naren Arya. Mongo db for python developers. https://impythonist.wordpress.com/tag/nosql/, 2014.

[4] Isaias Barroso. Banco de dados orientado a colunas. http://isaiasbarroso.wordpress.com/2012/06/20/banco-de-dados-orientado-a-colunas/, 2012.

[5] Ricardo W. Brito. Bancos de dados nosql x sgbds relacionais:análise comparativa. 2010.

[6] Vinícius Maran Cristiano Politowski. Comparação de performance entre postgresql e mongodb. http://www.lbd.dcc.ufmg.br/colecoes/erbd/2014/003.pdf, 2014.

[7] Douglas Crockford. Javascript: The good parts. O'Reilly Media, 2008.

[8] Francisco de Assis Ribeiro Junior. Programação orientada a eventos no lado do servidor utilizando node.js. http://www.infobrasil.inf.br/userfiles/16-S3-3-97136-Programa%C3%A7%C3%A3o%20Orientada___.pdf.

[9] Google Developers. Chrome v8 - introduction. https://developers.google.com/v8/intro, 2012.

[10] Johann du Toit. Node meet intro. http://johanndutoit.net/presentations/2013/05/node-meetup-intro-29-may-2013/, 2013.

[11] Express.js. Express application generator. http://expressjs.com/starter/generator.html, 2014.

[12] Stephen Ferg. Event-driven programming: Introduction, tutorial, history. http://ufpr.dl.sourceforge.net/project/eventdrivenpgm/event_driven_programming.pdf.

[13] Donizete Fidelis. Proxy reverso. http://ti.crinfo.com.br/wp-content/plugins/downloads-manager/upload/Proxy%20Reverso%20por%20Donizete%20Fidelis.pdf, 2013.

[14] Ivan Isaías Friess. AnÁlise de bancos de dados nosql e desenvolvimento de uma aplicaÇÃo. http://www-app.inf.ufsm.br/bdtg/arquivo.php?id=171, 2013.

[15] Georgi Georgiev. What is vertical scaling and horizontal scaling – vertical and horizontal hardware / services scaling. http://www.pc-freak.net/, 2014.

[16] Cristian Machado Goulart Gustavo Jungthon. Paradigma de programação. http://petry.pro.br/sistemas/programacao1/materiais/artigo_paradigmas_de_programacao.pdf, 2010.

[17] Itay Herskovits. Mean vs lamp – how do they stack up? http://blog.backand.com/mean-vs-lamp/, 2014.

[18] Vinicius Ianni. Introdução aos bancos de dados nosql. http://www.devmedia.com.br/introducao-aos-bancos-de-dados-nosql/26044#ixzz3Kq0arjUf, 2013.

[19] Fernando dos Santos icardo Schroeder. Arquitetura e testes de serviços web de alto desempenho com node.js e mongodb. 2014.

[20] Iván Loire. Math race. https://github.com/iloire/math-race/, 2012.

[21] T.J. Holowaychuk Nathan Rajlich Mike Cantelon, Marc Harter. Node.js in action. Manning, 2014.

[22] Elton Luís Minetto. Desenvolvendo aplicações web escaláveis. http://eltonminetto.net/docs/app_web_escalaveis.pdf.

[23] MongoDB. Nosql databases explained. http://www.mongodb.com/nosql-explained, 2013.

[24] MongoDB. Sql to aggregation mapping chart. http://docs.mongodb.org/manual/reference/sql-aggregation-comparison/, 2014.

[25] Jean Nascimento. Nosql – você realmente sabe do que estamos falando? http://imasters.com.br/artigo/17043/banco-de-dados/nosql-voce-realmente-sabe-do-que-estamos-falando/, 2010.

[26] Kai Orend. Analysis and classification of nosql databases and evaluation of their ability to replace an object-relational persistence layer. 2010.

[27] Caio Ribeiro Pereira. Aplicações web real-time com node.js. Casa do Código, 2014.

[28] Dan Pritchett. Base: An acid alternative. ACM Queue, 2008.

[29] Real Time Statistics Project. Internet live stats. http://www.internetlivestats.com/internet-users/#sources, 2014.

[30] Fernando dos Santos Ricardo Schroeder. Arquitetura e testes de serviços web de alto desempenho com node.js e mongodb. http://www.ceavi.udesc.br/arquivos/id_submenu/787/ricardo_schroeder_versao_final_.pdf.

[31] Dr. Jacques Philippe Sauvé. Pools de threads. http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/threads/pool.html, 2014.

[32] Stackoverflow. O que é callback? http://pt.stackoverflow.com/questions/27177/o-que-Ã-callback, 2013.

[33] Dio Synodinos. Projects, applications, and companies using node. https://github.com/joyent/node/wiki/Projects,-Applications,-and-Companies-Using-Node, 2013.

[34] Dio Synodinos. Top javascript mvc frameworks. http://www.infoq.com/research/top-javascript-mvc-frameworks, 2013.

[35] Kaio Valente. Porque utilizar angularjs no seu próximo projeto. http://tasafo.wordpress.com/2014/11/26/porque-utilizar-angularjs-no-seu-proximo-projeto/, 2014.

[36] Wikipedia. Aplicações web. http://pt.wikipedia.org/wiki/Aplica%C3%A7%C3%A3o_web, 2013.

[37] Wikipedia. Nosql. http://en.wikipedia.org/wiki/NoSQL, 2013.

[38] Wikipedia. Rest. http://pt.wikipedia.org/wiki/REST, 2013.

[39] Jim R. Wilson. Node.js the right way: Practical, server-side javascript that scales. Pragmatic Bookshelf, 2013.

CAPÍTULO 6

APÊNDICE A

6.1 Socket.IO

O Socket.io é uma API em Javascript que permite que a comunicação entre o servidor e o cliente ocorra sem dificuldades e em tempo real. Ele abstrai e utiliza o protocolo *WebSocket*[1], e também possui outras alternativas caso seja um ambiente que não suporte *WebSocket*.

[1]É um protocolo orientado a eventos que permite abrir uma sessão de comunicação interativa entre o navegador do cliente e o sevidor, sendo possível enviar mensagens para um servidor e receber respostas sem ter que consultar o servidor para uma resposta.