Utilizando MEAN Stack para o desenvolvimento de aplicações *web*

Adriano F. de Araújo[1], Leonardo Sommariva[2]

[1]Departamento de Sistemas e Computação – Universidade Regional de Blumenau (FURB) – Blumenau, SC - Brazil

[2] Departamento de Sistemas e Computação – Universidade Regional de Blumenau (FURB) – Blumenau, SC - Brazil

flachadriano@gmail.com, lsommariva@gmail.com

Abstract. *This paper presents the MEAN Stack framework, developed to minimize the need to know different programming languages to the development of a web application, using the JavaScript. MongoDB é used to data persist, the back-end development of the application is used Express, NodeJS is the platform of execution. AngularJS é applied on front-end construction of the application. In the end of the paper presents an application example using the MEAN Stack framework.*

Resumo. *Este artigo apresenta o framework MEAN Stack, desenvolvido para minimizar a necessidade do conhecimento de diferentes linguagens de programação para o desenvolvimento de aplicações web, utilizando a linguagem JavaScript. Para a persistência dos dados é utilizado MongoDB, o desenvolvimento do back-end da aplicação é realizado com o Express, sendo NodeJS utilizado como plataforma para sua execução. AngularJS é aplicado na construção do front-end da aplicação. No final do artigo é apresentada uma aplicação exemplo utilizando o framework apresentado.*

1. Introdução

Durante muitos anos o JavaScript foi considerado por muitos como uma linguagem para amadores, porém sua arquitetura de desenvolvimento e potencial fez com que seus desenvolvedores mostrassem o poder desta linguagem. Com o surgimento do AJAX vislumbrou-se a possibilidade de transformar *sites* simples em aplicações *web*, o que inspirou o desenvolvimento de bibliotecas utilitárias, como jQuery e Prototype, para agilizar o desenvolvimento dessas aplicações. Google contribuiu para o contínuo crescimento da linguagem com o Chrome V8 (HAVIV, 2014). Este último, lançado em 2008, é uma máquina interpretadora de código JavaScript feita em C++, possibilitando o desenvolvimento de código JavaScript em processadores que suportem a linguagem C++ (GOOGLE, 2015).

Hoje em dia, JavaScript tornou-se ubíquo para o desenvolvimento de aplicações *web* client-side, porém para o desenvolvimento do servidor dessas aplicações, muitas linguagens, frameworks e APIs entram em voga. Várias dessas opções atenderam às expectativas e estão decolando entre os desenvolvedores e empresas, enquanto outras ficaram obsoletas com o tempo. Em 2009, as pessoas já haviam se dado conta do potencial que JavaScript tinha como linguagem para o desenvolvimento de aplicações para o

navegador, quando Ryan Dahl vislumbrou o potencial que esta linguagem tinha para o desenvolvimento de aplicações no servidor, então nascia o Node (BROWN, 2014).

Conforme Almeida (2015), O acrônimo MEAN foi cunhado em 2013 por Valeri Karpov do time do MongoDB para denotar o uso de uma stack completa para o desenvolvimento de aplicações, incluindo MongoDB, Express, AngularJS e Node.js.

A letra M do termo MEAN denota o MongoDB, um banco de dados orientado a documentos, que traz um novo conceito de armazenamento de dados, onde não há um esquema fixo definindo como cada dado armazenado deve ser (CHODOROW, 2013). A forma de armazenamento utilizada é muito similar ao JavaScript Object Notation (JSON) o que ajuda a realizar o armazenamento e reaver os dados, pois JSON é o formato comumente utilizado para prover e consumir APIs. Este formato de armazenamento realiza poucas validações em relação aos dados recebidos, tendo a aplicação a maior parte da responsabilidade de validar estes dados (ALMEIDA, 2015). Mantendo funcionalidades disponibilizadas por um banco relacional, como por exemplo índices e ordenação.

Express, criado em 2009, é responsável pela organização da aplicação no lado do servidor, utilizando a arquitetura MVC (ALMEIDA, 2015). Inspirado no *framework* Sinatra, desenvolvido em Ruby, que preza pelo desenvolvimento rápido, eficiente e manutenível. Seguindo esta ideia, Express disponibiliza uma camada mínima para o desenvolvimento da aplicação, porém, sua grande força está em permitir que sejam acoplados *middlewares*, que são responsáveis por executar alguma tarefa maior para a aplicação. Permitindo assim, que o framework evolua constantemente através de seus *middlewares*, assim como ocorre com Sinatra (BROW, 2014).

AngularJS implementa a letra A do MEAN Stack, que é responsável pelo desenvolvimento de aplicações no lado do cliente, utilizando conceito de Single Page Application (SPA), onde a aplicação necessita ser carregada completamente apenas uma vez, sendo as demais chamadas realizadas apenas para buscar partes necessárias para realizar a apresentação dos dados ao usuário (ALMEIDA, 2015). Este *framework* preza por desenvolver a aplicação de forma declarativa, extendendo as tags HyperText Markable Language (HTML), as tags adicionadas através deste *framework* são responsáveis por modularizar a aplicação (BRANAS, 2014). Este artigo focará no desenvolvimento da parte de servidor de uma aplicação web, sendo assim não será abordado o funcionamento do angular no MEAN Stack.

NodeJS é uma plataforma para aplicações JavaScript que roda sobre o Chrome V8. Nesta plataforma é possível utilizar bibliotecas desenvolvidas pela comunidade através do gerenciador de pacotes NPM. NodeJS seria o core da aplicação web.

2. Persistindo dados com MongoDB

MongoDB é um banco de dados orientado a documentos e não um banco relacional, desta forma o conceito de “linha” de uma tabela é substituído pelo modelo de “documento”, assim é possível prover maior facilidade e flexibilidade para alterações na modelagem dos dados, podendo armazenar um documento com uma lista de outros documentos em um mesmo registro, algo muito comum de ser observado no desenvolvimento orientado a objetos. Este banco se utiliza de esquemas não fixos onde chaves e valores não tem tamanhos ou tipos fixos, o que permite que facilmente seja realizada adição e remoção de campos, fazendo com que seja facilitado escalar este banco (CHODOROW, 2013).

Documento é um conjunto de chaves associadas a valores, representado de diferentes formas em diversas linguagens, em formas de mapas, *hash*s, dicionários, entre outros, em JSON são representados como objetos, conforme é apresentado no Quadro 1, neste documento pode ser verificado que há três chaves: nome, framework e versão. Como pode ser verificado na chave versão, este formato comporta diferentes tipos de dados, onde os dois primeiros são textos e o terceiro é numérico (CHODOROW, 2013).

```
{“nome” : “Adriano”, “framework” : “MEAN Stack”, “versao” : 1}
```

Quadro 1. JSON

O formato de armazenamento utilizado pelo MongoDB funciona de forma muito similar ao apresentado no Quadro 1, sendo que devem ser observadas algumas regras e orientações sobre o funcionamento:

- Chaves não devem conter o caracter *\0* (caracter conhecido como *null*), pois é utilizado para identificar o fim do identificador de uma chave;
- Os caracteres “.” e “$” são comumente utilizados como chave para identificar propriedades especiais, tendo sua utilização restringida a alguns casos;
- As chaves são *case-sensitive*, ou seja, quando utilizado letras maiúsculas como chave será diferente de se utilizar somente caracteres minúsculos;
- Não são aceitas chaves repetidas dentro de um documento;
- O *schema* não deve depender da ordem em que os campos são ordenados, pois o MongoDB pode realizar alterações em alguns casos.

Conjuntos de documentos que representem a modelagem de algo são agrupados em coleções. Assim como um documento está para uma linha no banco relacional, uma coleção está para uma tabela. Porém o MongoDB não controla o que é adicionado em uma coleção, sendo assim, é possível que haja dentro de uma mesma coleção dois documentos com chaves totalmente diferentes entre si, desta forma, conforme a massa de dados evolui podem observadas dificuldades na identificação de objetos dentro de uma coleção, acarretando na necessidade de haver restrições muito genéricas para varrer uma grande massa de dados. Para garantir que tal comportamento não afete negativamente as consultas realizadas, são utilizadas bibliotecas que controlam as coleções através de modelos (CHODOROW, 2013).

Este banco de dados organiza as coleções de documentos em bases de dados, podendo, uma instância do MongoDB armazenar várias bases de dados. Sendo que cada uma das bases de dados tem permissões de acesso específicas que são salvas em arquivos separados do disco. Bases de dados são comumente utilizadas para armazenar separadamente os dados de diferentes aplicações ou de usuários (CHODOROW, 2013).

Conforme apresentado nesta seção o MongoDB trabalha com uma estrutura de organização onde as bases de dados armazenam coleções, que por sua vez armazenam documentos. Ao inicializar o MongoDB, por padrão é criada uma base de dados chamada *test*, para verificar isto basta executar o comando: `show dbs`.

A inserção de um documento em uma coleção é apresentada no Quadro 2. Como pode ser verificado na segunda linha, é realizado acesso a `contatos` que está em `db`, e nesta coleção é adicionado o documento necessário e por fim na terceira linha é

apresentada informação de que o dado foi inserido corretamente, porém como pode ser verificado, foi inserido um documento em uma coleção sem que fosse necessário criá-la anteriormente, isto se deve ao fato que o MongoDB realiza esta criação, caso identifique que a mesma não existe entre as coleções disponíveis.

```
  var contato = {"nome":"Adriano"}
> db.contatos.insert(contato)
WriteResult({ "nInserted" : 1 })
```

Quadro 2. Inserindo documento

No Quadro 3 é apresentado como listar documentos de uma determinada coleção. Como pode ser verificado na segunda linha, foi armazenado o contato salvo, porém foram adicionados mais dados que os enviados. A informação "_id" é criada e controlada pelo MongoDB, esta informação cria uma chave única para o documento dentro da coleção em que se encontra, isso significa dizer que o em outra coleção pode haver uma chave idêntica, porém não haverá outra igual dentro da mesma coleção.

O valor da chave "_id" está dentro de um objeto, chamado de ObjectId, que é um dos tipos de dados suportados pelo MongoDB. Enquanto o formato JSON disponibiliza o suporte a seis tipos de dados (Array, Boolean, null, Number, Object e String) o BSON (Binary JSON) suporta mais de 15 tipos de dados (ALMEIDA, 2015). BSON é o nome dados ao formato de dados com o qual o MongoDB armazena os dados.

```
  db.contatos.find()
{ "_id" : ObjectId("571905ce729c4c6ae9b45f60"), "nome" : "Adriano" }
```

Quadro 3. Listando documentos

3. O servidor de aplicações *web*, Node

Node apresenta características semelhantes a outros servidores web utilizados no mercado, como Internet Information Services (IIS) e Apache. Porém, Brown (2014) destaca duas características que diferenciam as metodologias aplicadas ao Node e estes servidores. Há profissionais especializados que, através dos anos dominaram as características do servidores *web* IIS ou Apache, sendo que na metodologia do Node a busca em ser minimalista para simplificar a configuração e inicialização do servidor. A outra característica é o tratamento *single thread* realizado pelo Node, esta forma de tratamento simplifica o controle das regras de negócio em aplicações *web*, e caso seja necessário que a aplicação seja *mulithread,* é possível levantar outras instâncias do Node, obtendo o mesmo desempenho de uma aplicação *mulithread*.

Entre outras características destacadas por Brown (2014) estão:

- Publicação da aplicação: enquanto linguagens como Java e .NET necessitam serem traduzidas para linguagem de máquina, para que assim possam ser executadas no servidor de aplicação, outras linguagens como PHP e Ruby tem o próprio código programado enviado para o servidor para que seja executado. O que é muito semelhante ao que ocorre com JavaScript, que se utiliza da máquina V8 da Google para compilar o código de forma transparente, sem que haja a necessidade que este passo tenha alguma interação por parte do desenvolvedor para que ocorra, e assim possa ser executado pelo servidor de aplicação;

- Plataforma independente: para rodar aplicações .NET no sistema operacional Linux, é possível rodar através do *software* Mono, porém é um trabalho muito árduo. Algo que não ocorre com outras linguagens como PHP, que têm este processo simplificado, com pouca diferença entre as plataformas. Tendo o Node um processo mais simplificado e uniforme entre as plataformas mais comuns (Windows, OS X e Linux). Esta característica é principalmente notada em times multi disciplinares, onde há designers e programadores trabalhando em um mesmo, onde geralmente se utilizam de computadores com o sistema OS X e Windows respectivamente.

Assim como Java tem o Maven para controle de bibliotecas de terceiros, o Node também se utiliza de um pacote de controle de pacotes e dependências, conhecido como npm. Através deste gerenciador de pacotes que será realizada a instalação no framework para o lado do servidor, Express, o qual é explicado na seção 4.

```
1   var http = require('http');
2
3   http.createServer(function(req,res){
4     var url = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
5     switch(url) {
6       case '':
7         res.writeHead(200, { 'Content-Type': 'text/plain' });
8         res.end('Página inicial');
9         break;
10      case '/sobre':
11        res.writeHead(200, { 'Content-Type': 'text/plain' });
12        res.end('Sobre');
13        break;
14      default:
15        res.writeHead(404, { 'Content-Type': 'text/plain' });
16        res.end('Não encontrado');
17        break;
18    }
19  }).listen(3000);
20
21  console.log('Servidor iniciado em http://localhost:3000');
```

Quadro 4. Criando um servidor Node

No Quadro 4 é apresentado como criar um servidor em Node respondendo ao acesso as páginas. Na linha 1 do Quadro 4 é realizada a importação da biblioteca `http`, que é responsável por controlar os acessos a rede, na linha 3 é solicitada a criação do servidor através da chamada ao método `createServer`, é enviada uma função como parâmetro, que será executada para cada chamada a aplicação. Na lnhas 7 é chamado o método *writeHead* da variável *res*, este método realiza a escrita no cabeçalho de resposta à solicitação informando o *status* 200, que significa que o processo foi executado com sucesso e que a resposta da solicitação deve ter conteúdo em forma de texto (`text/plain`) e na linha seguinte, ao chamar o método `end` informa que deve ser retornado o texto

"Página inicial" ao usuário. Por fim, na linha 19, informa que a aplicação estará interceptando qualquer chamada à porta 3000.

4. Aplicação *server-side* com Express

Express é descrito em seu *web-site* da seguinte forma: "O Express é um framework para aplicativo da web do Node.js mínimo e flexível que fornece um conjunto robusto de recursos para aplicativos web e móvel" (EXPRESS, 2016). Brown (2014), apresenta as características deste framework em 5 aspectos: *minimal*, *flexible*, *web application framework*, *single-page web applications* e *multipage and hybrid web applications*.

Segundo Brown (2014) o aspecto *minimal* é o mais forte do *framework* Express, pois a filosofia do *framework* é prover uma camada mínima entre o que se deseja e o que é desenvolvido. O que significa que é esperado que o *framework* dê a liberdade para que o desenvolvedor possa expressar suas ideias através do código, porém disponibilizando funções úteis para alcançar o objetivo.

O aspecto *flexible* é descrito por Brown (2014) como outro ponto chave da filosofia do Express, onde é disponibilizada uma fina camada para o desenvolvimento da aplicação e permitindo que em diferentes partes do *framework* possam ser adicionadas quaisquer funcionalidades necessárias, através de bibliotecas.

O termo *framework* para aplicações *web* é utilizado para descrever o *framework* Express, porém, Brown (2014) reitera que isto não significa afirmar que este framework somente pode ser utilizado para o desenvolvimento de aplicações *web*, mas que também pode ser utilizado para o desenvolvimento de *websites* e páginas *web*, informando que um *website* é um tipo de aplicação *web*, assim como uma página *web*. Sendo que uma aplicação permite *web* permite desenvolver outras funcionalidades, como prover uma funcionalidade para outra aplicação *web*.

Conforme descrito por Brown (2014), em uma aplicação *web* comum é realizado uma requisição para o servidor toda vez que o usuário navega para uma página diferente. Sendo que em aplicações *web single-page*, isto não ocorre. Em uma *single-page web application*, quando o usuário acessa a aplicação é baixado todo o código necessário para construir as telas. Desta forma, a navegação pela aplicação se torna mais rápida.

Por fim, o aspecto *multipage and hybrid web applications* é descrito por Brown (2014) como a forma mais tradicional de desenvolvimento de *websites*. Onde cada página é baixada em uma solicitação nova ao servidor. Ainda conforme Brown (2014), isto não significa dizer que por esta forma ser mais antiga, deve ser considerada inferior à *single-page*, apenas que o framework disponibiliza mais opções para o desenvolvimento da solução necessária, inclusive é possível mesclá-las para que atenda às necessidades.

No Quadro 5 é apresentado o desenvolvimento da mesma aplicação do Quadro 4, porém utilizando o framework Express. Sendo na linha 5 realizada chamada ao método `get` da variável `app`, este último armazena todas as configurações para a aplicação, enquanto o método `get` informa que quando for realizada uma chamada à rota recebida como primeiro parâmetro a função enviada como segundo parâmetro deve ser executada. Na linha 6 informado que o tipo da resposta deve ter seu conteúdo em forma de texto (`text/plain`) e na linha seguindo é informado o conteúdo da resposta. Na linha 17 é informado que o status para a resposta deve ser 404, o que significa que a `url` solicitada não está entre as suportadas pela aplicação. Por fim, na linha 21 é informado que deve ser

iniciado o servidor interceptando a porta 3000, configurado no primeiro parâmetro e no segundo parâmetro é informada a função que deve ser executado ao iniciar o servidor com sucesso.

```
1   var express = require('express');
2   var app = express();
3   app.set('port', process.env.PORT || 3000);
4
5   app.get('/', function(req, res){
6     res.type('text/plain');
7     res.send('Página inicial');
8   });
9
10  app.get('/sobre', function(req, res){
11    res.type('text/plain');
12    res.send('Sobre');
13  });
14
15  app.use(function(req, res){
16    res.type('text/plain');
17    res.status(404);
18    res.send('Não encontrado');
19  });
20
21  app.listen(app.get('port'), function(){
22    console.log('Express iniciado em http://localhost:' + app.get('port'));
23  });
```

Quadro 5. Criando um servidor Node com Express

5. Persistindo dados da aplicação com Mongoose

O Node dispõe de um *driver* para ligar a aplicação hospedada pelo Node ao MongoDB, porém é um *driver* de baixo nível, para isto o MongoDB disponibiliza de forma oficial um *driver* que realiza essa persistência de dados de uma forma chamada de "object document mapper" (ODM), conhecido como Mongoose.

O JavaScript permite que sejam adicionadas propriedades e métodos sem a necessidade de alterar classes ou configurações. Isto pode influenciar de forma negativa ao banco de dados por tornar difícil a tarefa de otimização no tratamento dos dados. Mongoose tem o objetivo de balancear este ponto, sendo introduzidos os conceitos de *schemas* e modelos, que podem ser comparados às classes de configuração em linguagens de programação orientadas a objetos, porém permite que os controles dessas configurações se mantenham flexíveis para alterações, provendo a estrutura necessária ao banco de dados (BROWN, 2014).

No Quadro 6 é apresentado um exemplo de como configurar um modelo utilizando Mongoose, onde na linha 1 é realizada a importação da biblioteca. Entre as linhas 4 e 8 é definido um mapa onde as chaves são os nomes dos campos que serão suportados para o modelo e os valores são os tipos de dados que estes campos devem ter. Na linha 10 o modelo definido é registrado no Mongoose.

```
1   var mongoose = require('mongoose');
2
3   module.exports = function() {
4     var schema = mongoose.Schema({
5       data: Date,
6       valor: Number,
7       descricao: String
8     });
9
10    return mongoose.model('Transacao', schema);
11  };
```

Quadro 6. Schema utilizando Mongoose

6. Aplicação exemplo

Para demonstrar a utilização do *framework*, foi desenvolvida uma aplicação exemplo onde será possível cadastrar despesas pessoais e visualizar os dados cadastrados.

```
1   var express = require('express');
2   var bodyParser = require('body-parser');
3   var mongoose = require('mongoose');
4
5   var app = express();
6   app.set('port', process.env.PORT || 3000);
7   app.use(bodyParser.urlencoded({
8     extended: true
9   }));
10  app.use(bodyParser.json());
11  mongoose.connect('mongodb://localhost/mymony');
12  var Transacao = require('./models/transacao')();
```

Quadro 7. Configurando servidor Node

No Quadro 7 é iniciada a configuração de um servidor Node, porém na linha 2 está sendo importada uma biblioteca ainda não instalada, o body-parser serve para configurar que as solicitações utilizarão o formato JSON para transportar dados, entre as linhas 7 e 10 é realizada a configuração desta biblioteca. Na linha 11 é solicitado ao MongoDB que se conecte no banco de dados mymony. Na linha 12 é importado o *schema* para o documento de transação, para isto foi criado um novo arquivo chamado transação em uma nova pasta (*models*), apresentado no Quadro 8.

```
1   var mongoose = require('mongoose');
2
3   module.exports = function() {
4     var schema = mongoose.Schema({
5       data: Date,
6       valor: Number,
7       descricao: String
8     });
9
10    return mongoose.model('Transacao', schema);
11  };
```

Quadro 8. Schema utilizando Mongoose

No Quadro 9 é apresentada a continuação da configuração do servidor de Node. Na linha 2 é utilizado o *schema* de `Transacao` para realizar manipulações na coleção de transações, ao chamar o método `find` é realizada a busca de todos os registros desta coleção, ao encadear com o método `exec` é realizada a criação de uma *promise*, o que significa dizer que ao finalizar a busca dos documentos da coleção de transações será executada a função enviada como parâmetro para o método `then`. São enviada duas funções para o método `then`, sendo a primeira executada em caso de sucesso para obter os dados da consulta e a segunda em caso de falha. Na linha 4 é realizada a chamada ao método `json`, que fará a transformação do conteúdo recebido como parâmetro em JSON. Na linha 12 é realizada a chamada ao método `post` de `app`, desta forma está sendo informado ao Express que a função ligada a esta rota somente deve ser executada quando for realizada uma chamada HTTP do tipo POST.

```
app.get('/', function(req, res) {
  Transacao.find().exec().then(
    function(transacoes) {
      res.json(transacoes);
    },
    function(erro) {
      res.status(500).json(erro);
    }
  );
});

app.post('/criar', function(req, res) {
  Transacao.create(req.body).then(
    function success(transacao) {
      res.json(transacao);
    },
    function(erro) {
      res.status(500).json(erro);
    }
  );
});

app.use(function(req, res) {
  res.type('text/plain');
  res.status(404);
  res.send('Não encontrado');
});

app.listen(app.get('port'), function() {
  console.log('Express iniciado em http://localhost:' + app.get('port'));
});
```

Quadro 9. Configurando e executando servidor Node

Para realizar a execução da aplicação deve ser executado o comando node seguido do nome dado ao arquivo de configuração do servidor, por exemplo `node server`.

7. Considerações finais

O framework se propõe a disponibilizar o desenvolvimento completo de uma aplicação web utilizando ferramentas que possam ser configuradas e implementadas utilizando-se da linguagem de programação JavaScript.

O desenvolvimento deste artigo apresentou o desenvolvimento do *back-end* de uma aplicação web, onde é possível identificar que para o seu completo desenvolvimento com este *framework* é necessário adicionar outras bibliotecas, utilizando-se da flexibilidade para o acoplamento de bibliotecas de terceiros, possibilitando que o framework possa manter sua evolução através destas ferramentas auxiliares.

O artigo não contempla o desenvolvimento do *front-end* da aplicação, que seria responsável por consumir a API criada e apresentar outra ferramenta que compõe o

MEAN Stack, o AngularJS. Isto ocorreu devido ao embasamento necessário para que possa ser apresentado o desenvolvimento completo do *front-end* de uma aplicação *web*.

Durante o desenvolvimento da aplicação pode ser verificado que as ferramentas utilizadas não impõem uma estrutura fixa para a organização de arquivos ou pastas, o que facilita na rapidez de construção de uma aplicação, por outro lado, permite que em projetos grandes esta liberdade acarrete na dificuldade de manutenabilidade do projeto e na entrada de novos desenvolvedores por não terem conhecimento da localização dos diversos componentes da aplicação.

Um ponto comum para um desenvolvedor *web front-end*, assincronismo. Com este *framework* é levado ao desenvolvimento do *back-end*. Desta forma, em diversos momentos é necessária a aplicação de tratamentos de *promises*, pois não será obtida uma resposta para a execução de uma chamada logo em sequência, mas sim em algum momento no futuro. Sendo necessário atentar aos retornos das chamadas, que em alguns momento retornarão com um resultado e em outros com uma *promise*.

No desenvolvimento deste aplicativo foi utilizado o editor Atom. Os códigos fontes desenvolvidos neste artigo estão disponíveis no endereço: https://github.com/flachadriano/pos-desenv-web/tree/master/artigo/projeto.

Referências

ALMEIDA, Flávio. MEAN: **Full stack JavaScript para aplicações web com MongoDB, Express, Angular e Node**. São Paulo: Casa do Código, 2015. 377 p.

BROWN, Ethan. Web Development with Node & Express: **Leveraging the JavaScript Stack. Sebastopol**: O'Reilly Media, 2014. 306 p.

CHODOROW, Kristina. MongoDB: **The Definitive Guide**. 2. ed. Sebastopol: O'Reilly Media, 2013. 410 p.

BROWN, Ethan. Web Development with Node & Express: **Leveraging the JavaScript Stack**. Sebastopol: O'Reilly Media, 2014. 306 p.

GOOGLE. Chrome V8: **Google's high performance, open source, JavaScript engine**. 2015. Disponível em: <https://developers.google.com/v8/>. Acesso em: 24 mar. 2016.

HAVIV, Amos Q. MEAN Web Development: **Master real-time web application development using a mean combination of MongoDB, Express, AngularJS, and Node.js**. Birmigham: Packt Publishing, 2014. 456 p.

EXPRESS. **Express:** Framework de aplicativo da web Node.js. 2016. Disponível em: <http://expressjs.com/pt-br/>. Acesso em: 21 abr. 2016.