Utilizando MEAN Stack para o desenvolvimento de API REST

Adriano F. De Araújo1, Leonardo Sommariva1

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

flachadriano@gmail.com, lsommariva@gmail.com

**Abstract.** This paper has the object to presents the development of an Application Programa Interface (API) applying the architetural model Representational State Transfer (REST) using the MEAN Stach framework.

**Resumo.** Este artigo tem como objetivo apresentar o desenvolvimento de uma Application Program Interface (API) aplicando o modelo arquitetural REpresentational State Transfer (REST) utilizando o framework MEAN Stack.

# 1. MEAN Stack *framework*

Durante muitos anos JavaScript foi considerada 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 *web*, muitas linguagens, *frameworks* e APIs entram em voga. Várias dessas opções atenderam as espectativas 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 *browser*, 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 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 que 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 ao 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 por 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 realizada 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. Configurando a aplicação

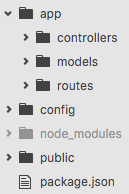
A primeira etapa é instalar o NodeJS, basta acessar o *site* <https://nodejs.org/en/download/>, baixar o instalador e executá-lo. Para verificar se tudo coorreu conforme esperado basta executar no terminal o seguinte comando: *node -v*.

Conforme mencionado no item 1, o NPM é o gerenciador de pacotes utilizado pelo NodeJS para baixar as bibliotecas necessárias para o desenvolvimento das aplicação. Porém, o NPM precisa ter uma configuração em cada aplicação que for utilizar. Aconselha-se que para cada aplicação desenvolvida seja criada uma nova pasta no sistema. Para configurar é necessário abrir o terminal, acessar a pasta onde a aplicação será desenvolvida e executar o comando: *npm init*. Serão realizadas algumas perguntas e ao final será criado um arquivo com nome *package.json*. Para este artigo resultado é apresentado no quadro 1.

Quadro 1. package.json

Após configurado o NPM, pode ser solicitada a instalação da primeira biblioteca, que será o express, para isto basta executar o comando: *npm install* [*express@4.13.4*](mailto:express@4.13.4) *–-save*. A chave *install* informa que está sendo solicitada a instalação de uma biblioteca, em seguida, *express* é o nome da biblioteca que deve ser localizada, *@4.13.4* denota qual versão da biblioteca deve ser instalada, para este artigo será utilizada a versão 4.13.4, foi adicionado ao fim do comando a informçaão –-*save* que informa ao NPM que esta biblioteca deve ser salva como uma dependência da aplicação. Caso verifique novamente agora o arquivo *package.json*, deve ser apresenta uma nova chave (*dependencies*), com o *express* dentro e a versão especificada.

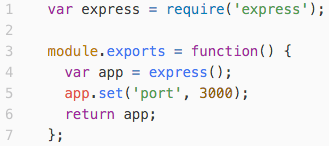
Almeida (2015) sugere que seja criada uma organização de pastas no diretório onde o projeto será desenvolvido, conforme o quadro 2.

Quadro 2. Organização das pastas

Ainda conforme Almeida (2015) cada pasta deve conter os seguintes conteúdos:

* controllers: controladores chamados pelas rotas da aplicação
* models: *models* que representam o domínio do problema
* routes: rotas da aplicação
* views: *views* da máquina geradora de templates
* config: configuração do express, banco de dados, etc
* public: todos os arquivos acessíveis diretamente pelo navegador

Agora é necessário realizar a configuração do servidor. Para isto, será necessário criar um arquivo na pasta *config*, que será responsável por configurar o express, que realizará o tratamento das requisições recebidas pela API, deverá ser criado um arquivo com o nome *express.js* nesta pasta. No quadro 3 pode ser observado o conteúdo que este arquivo deve conter, onde na primeira linha é realizada a importação do *framework express.* Na terceira linha, pode ser verificado que é realizada uma chamada *module.exports* que recebe uma *function* como atribuição, ao fazer isto, está sendo informado ao *node* que ao realizar a importação deste arquivo em outro lugar deverá ser retornado o que estiver dentro de *exports*. Na linha 4 está sendo realizada chamada ao *express* para inicializar uma nova aplicação. Na linha 5, é feita uma chamada ao método *set* da aplicação que está sendo inicializada, este método serve para armazenar alguma informação dentro da aplicação, neste caso está guardando o valor *3000* com a chave *port.*

Quadro 3. arquivo config/express.js

Agora que já foi criado o arquivo que instancia uma nova aplicação do *express*, é necessário criar um servidor *node* para levantar esta aplicação. Para isto deve ser criado um arquivo chamado *server.js* na pasta raiz do projeto, seu conteúdo deve ser conforme o quadro 4. Onde na primeira linha é realizada importação da biblioteca *http*, que é responsável por interagir com a rede do computador. Na segunda linha é feita a importação do configuração da aplicação que está sendo criada. Na linha 4, é primeiramente chamado o método *createServer* da bilbioteca *http*, isto faz com que tenhamos uma instância de servidor do node, ao chamar em seguida o método *listen*, deve ser informado qual porta de rede a aplicação deve ouvir e se há alguma *callback* para ser executada após a aplicação iniciar. Como pode ser verificado, como primeiro parâmetro foi enviado *app.get(‘port’)*, na configuração da aplicação, foi chamado o método *set*, que guarda algum valor na aplicação, agora é realizada chamada ao *get* para buscar alguma valor guardado na aplicação, então na verdade está sendo enviado o valor *3000* como primeiro parâmetro. Como segundo parâmetro foi enviado uma *function* que irá imprimir no console do servidor ‘Express Server escutando na porta 3000’ se tudo tiver sido iniciado conforme o esperado.

Quadro 4. arquivo server.js

Então, para verificar se a configuração foi toda criada corretamente, basta acessar o terminal, entrar na pasta raiz do projeto e executar o seguinte comando: *node server*. Deve ser executado a mensagem conforme esperado.

# Referências

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

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.

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

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

BRANAS, Rodrigo. **AngularJS Essentials:** Design and construct reusable, maintainable, and modular web applications with AngularJS. Birmigham: Packt Publishing, 2014. 164 p.

NODEJS. **NodeJS.** 2016. Disponível em: <https://nodejs.org>. Acesso em: 24 mar. 2016.

\_\_\_\_\_\_. **NodeJS v5.9.1 Documentation.** 2016. Disponível em: <https://nodejs.org/api/modules.html>. Acesso em: 25 mar. 2016.