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*.

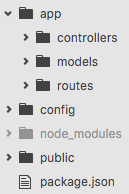
# 2. Configurando a aplicação

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.

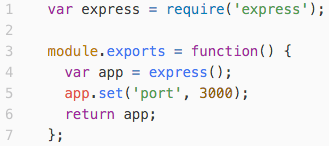
Almeida (2015) sugere que seja criada uma organização de pastas no diretório onde o projeto será desenvolvido, conforme o quadro 2.

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

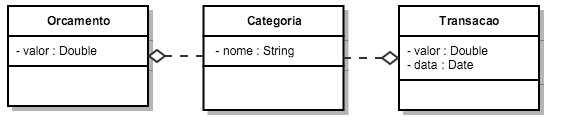
Agora que já foi criado o arquivo que instancia uma nova aplicação do *express*, é necessário criar um servidor *node* para levantar esta aplicação. Para isto deve ser criado um arquivo chamado *server.js* na pasta raiz do projeto, seu conteúdo deve ser conforme o quadro 4. Onde na primeira linha é realizada 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.

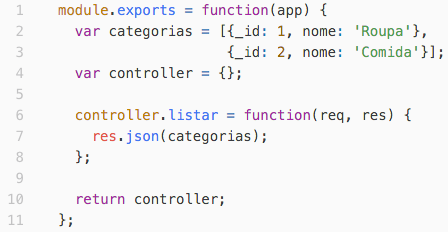
**3. Desenvolvendo a aplicação**

Foi criada a infraestrutura para uma aplicação MEAN, então agora faz-se necessário definir o tipo de aplicação que será desenvolvida. Será uma aplicação web para controle financeiro pessoal, onde terá categorias que são responsáveis por categorizar os tipos de despesas, terão orçamentos, que indicam qual o valor máximo desejado para gastar mensalmente com as categorias e por fim as transações, que são as despesas. Pode ser verificado um diagrama no quadro 5.



Quadro 5. UML

Para iniciar o desenvolvimento será utilizado a categoria, pois não tem dependência entre os demais objetos. Então para isto deve ser criado um arquivo com o nome *categoria.js* na pasta controllers. Este arquivo terá o código responável por executar as rotas chamadas pelo navegador. Neste primeiro momento não será realizada a persistência dos dados no banco. Sendo assim, como primeira etapa será implementado o método responsável por listar algumas categorias fixas, conforme pode ser verificado no quadro 6.

 Quadro 6. controllers/categoria.js

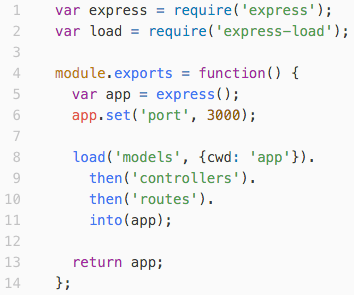
Assim como ocorreu com o arquivo *express.js*, para este arquivo também será exportada uma functiona que ao final retorna um objeto. Neste objeto, que será chamado de controller deverá conter *functions* que executarão a ação responável por alguma rota chamada pelo navegador. Como pode ser verificado na linha 6 do quadro 6, foi adicionado a chave *listar* uma *function* que recebe dois parâmetros, todas as funtions de *controller* no *express* podem receber estes dois parâmetros, *req* identifica os dados da requisição enviada pelo navegador ao acessar algum endereço da aplicação, enquanto *res* contém os dados de resposta para a requisição. Na linha 7 é chamado o método *json* do objeto *res*, isto fará com que a resposta enviada a solicitação seja um objeto JSON.

Após o desenvolvimento do *controller* é necessário identificar qual rota da aplicação corresponderá a este método do *controller*, para isto é necessário criar um novo arquivo, desta vez na pasta *routes*, deve ter o nome de *categoria.js* e seu conteúdo deve ser idêntico ao quadro 7.

 Quadro 7. routes/categoria.js

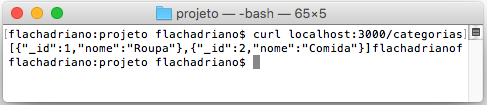
Assim como nos *controllers*, as rotas também podem receber como parâmetro a instância da aplicação que está sendo desenvolvida, utilizando esta instância, na linha dois é armazenado em uma variável o *controller* de categoria. Na linha 4, está sendo chamado o método *route* do *app*, que irá criar um recurso atrelado à esta rota, então a partir disto pode ser realizada chamada a recursos REST (get, put, post, update). Após obter este recurso é adicionado uma ligação com o método *get* enviando como parâmetro qual método do controller corresponde à rota especificada.

Após implementação do *controller* e da rota, a aplicação ainda não poderá rodar, pois de alguma forma na linha 2 do quadro 7, foi acessado um *controller* da aplicação, porém não especificado a configuração de onde este *controller* é localizado. Para isto é necessário alterar o conteúdo do arquivo *express.js* que está na pasta *config*, conforme pode ser verificado no quadro 8.

 Quadro 8. config/express.js

Na linha dois pode ser verificado que está sendo importada uma nova bilioteca, porém ela ainda não foi instalada pelo NPM, para isto basta executar no seguinte comando na pasta raiz do projeto: *npm install* [*expres-load@1.1.15*](mailto:expres-load@1.1.15). Agora que está importada, pode-se verificar sua utilização entre as linhas 8 e 11, nestas linhas está sendo indicado ao *express-load* que tudo que estiver nas pastas *models*, *controllers* e *routes* deve carregado para dentro da aplicação, isto significa que não será necessário executar o método *require* em nenhum dos arquivos da aplicação para acessar outro arquivo da mesma aplicação, isto será carregado para dentro da aplicação na inicialização do *express*.

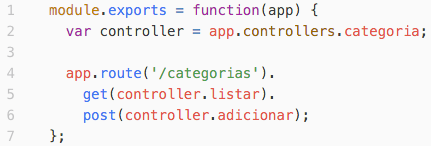
Agora a aplicação está funcional, para poder testar o que foi desenvolvido será utilizado cURL, que é uma biblioteca de código aberto para transferência de dados com a sintaxe de Uniform Resource Locator (URL) (CURL, 2016). Para utilizar esta biblioteca é necessário realizar o download, no endereço: <https://curl.haxx.se/download.html>. Então, após acessar a pasta da aplicação pelo terminal e executar novamente o comando: *node server*. Pode ser executado o cURL para verificar se estão sendo retornadas as categorias quando acessada a rota, para executá-lo é necessário abrir um novo terminal e executar o comando: *curl localhost:3000/categorias*. O resultado deve ser idêntico ao quadro 9.

 Quadro 9. executando aplicação

Ao utilizar a aplicação, um usuário deve poder não só visualizar as categorias disponíveis, mas também criar, alterar e deletá-las. Para que isto ocorra é necessário desenvolver novos métodos no *controller* que corresponderão à rotas. Então, seguindo o processo do usuário, primeiramente seria necessário adicionar as categorias desejadas, sendo assim, será implementado o método responsável por adicionar uma nova categoria na lista de categorias disponíveis, para isso deve ser alterado o arquivo de *controller* de categoria, implementando um método (linha 4 do quadro 10) para adicionar mais um item nas lista de categorias. Como pode ser visto no quadro 10, na linha 6, é adicionado o corpo (*body*) da requisição dentro da lista de categorias. Na linha 7 é retornado o status 201, informando que a categoria foi criada com sucesso e o objeto que foi adicionado à lista.

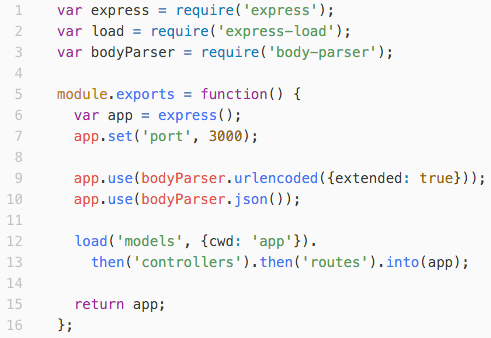
 Quadro 10. executando aplicação

Deve ainda ser ajustado o arquivo de rotas de categoria (*categoria.js*) para responder a rota de criação de categoria com este novo metodo, conforme é apresentado no quadro 11.

 Quadro 11. executando aplicação

Como pode ser verificado na linha 6 do quadro 11, foi encadeado a chamada do método *post* após o método *get*, quando se tem um recurso de rota, qualquer método REST pode ser chamado em seguida, neste caso está sendo informado que a rota */categorias* quando for chamada através do método *post* deve ser executada a function de adicionar.

O método adicionar espera que no corpo da requisição retorne dados em JSON, porém por padrão isto não ocorre, portanto será necessário a configuração do express para que isto ocorra foram adicionadas as linhas 3, 9 e 10, conforme pode ser verificado no quadro 12. Na linha 2 é realizada a importação de uma biblioteca que ainda não existe no projeto, portanto faz-se necessário solicitar a instalação ao NPM através do comando: *npm install* [*body-parser@1.15.0*](mailto:body-parser@1.15.0) *–- save*. Como apresentado anteriormente, uma das forças do express é sua gama de plugins disponíveis, na linha 9 é utilizado o método *use* da aplicação, este método permite que sejam adicionados módulos externos à aplicação, nesta linha a biblioteca está configurando o express para que aceite URLs extendidas e na linha 10 está informando que deve converter estes dados para um objeto JSON que ocupará o corpo da requisição.

 Quadro 12. arquivo config/express.js

Assim, após estas alterações, pode ser executado o comando no cURL para criar categoria. Reinicie o servidor da aplicação que está executando e execute em outro terminal o seguinte comando: *curl –-data “\_id=4&nome=Livros” localhost:3000/categorias*. Se, após este comando, for executado novamente a listagem de categorias, irá apresentar esta nova categoria.

Ainda restam as rotas para alterar uma categoria, buscar e deletar. Sua implementação pode ser encontrada no quadro 13 e devida configuração das rotas no quadro 14. A implementação dos demais arquivos podem ser encontrados no endereço <https://github.com/flachadriano/pos-desenv-web/tree/master/artigo/projeto>.

 Quadro 13. arquivo controllers/categoria.js

 Quadro 14. arquivo routes/categoria.js

**4. Persistindo os dados**

# 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.

CURL. **CURL:** groks those URLs. 2016. Disponível em: <https://curl.haxx.se/>. Acesso em: 25 mar. 2016.