ESTUDO E DESENVOLVIMENTO DE CASO DE USO COM FRAMEWORK ANGULARJS

Rafael de Castro Zorzo[1]

Élder F. F. Bernardi[2]

RESUMO

A evolução das técnicas de desenvolvimento de aplicações Web permite aos desenvolvedores melhorar constantemente suas aplicações e sistemas e fornecer melhor experiência ao usuário final. O AngularJS é um framework que fornece funções e uma arquitetura que atende praticamente todas as necessidades no ambiente de programação para a Internet e por isso está sendo aplicado frequentemente pelas empresas. Este artigo apresenta um estudo do framework em sua parte conceitual e posteriormente apresenta resultados obtidos através do desenvolvimento de um caso de uso que permite a avaliação do Angular. O protótipo desenvolvido para avaliação da ferramenta determinou diversos fatores que demonstram a qualidade e os resultados obtidos com a implementação do AngularJS em sistemas Web. O resultado final da avaliação apresentou o retorno esperado, pois a ferramenta trouxe efeitos positivos em todos os pontos avaliados dentro do contexto deste trabalho.

Palavras-chave: Aplicação. Sistema. Desenvolvimento. Abstração. Facilidade.

INTRODUÇÃO

O desenvolvimento de aplicações Web é aprimorado constantemente em todos os aspectos, desde a arquitetura e as metodologias empregadas até as linguagens de programação que o compõe. A arquitetura de uma aplicação Web está deixando de ser composta por grandes quantidades de códigos *server-side*, onde todo o processamento é feito no servidor, os códigos são fechados e o consumo de banda é maior aos proprietários do sistema, e passando a ter maior utilização de códigos *client-side*, onde o processamento dos scripts da página ocorre na máquina do cliente e somente quando há necessidade de interação com o banco de dados são feitas requisições ao servidor.

De forma a facilitar essa transição de códigos *server-side* para *client-side*, foi desenvolvido pela Google o framework AngularJS, cujo objetivo é trazer ao lado cliente as ferramentas e funcionalidades que são tradicionalmente implementadas em nível servidor, sem que haja risco à sensibilidade dos dados trafegados. Como consequência desse modo de

[1] Rafael de Castro Zorzo, formando no curso de Tecnologia em Sistemas para Internet pelo Instituto Federal de Educação, Ciência e Tecnologia Sul-rio-grandense, câmpus Passo Fundo (IFSul - Passo Fundo - RS). E-mail: rafael@rafaelzorzo.com.br

[2] Orientador, professor do IFSul. E-mail: elder.bernardi@passofundo.ifsul.edu.br

desenvolvimento, páginas mais rápidas, intuitivas e interativas melhoram a experiência do usuário na utilização do sistema. Ainda, o AngularJS proporciona aos desenvolvedores uma arquitetura facilmente escalável, reutilizável, testável e de fácil manutenção, aumentando a produtividade e a facilidade na depuração do sistema. Este aborda o uso de *Single Page Application* (SPA), técnica já difundida e aplicada na produção de sistemas para Internet. (ANGULARJS)

O objetivo deste artigo é demonstrar o que é e como o AngularJS pode ser usado no auxílio ao desenvolvimento de novos sistemas que pretendem atender a essa evolução na forma de criação das aplicações, bem como sua capacidade de aumentar a produtividade dos desenvolvedores, reduzir a utilização de recursos dos servidores, promover o desenvolvimento ágil e a produzir melhor experiência ao usuário final.

Para isto, será realizado um estudo da organização dos componentes que compõe a arquitetura do framework e será desenvolvido um protótipo baseado em um caso de uso para por em prática os conceitos levantados, sendo analisados ao final os resultados obtidos durante o desenvolvimento. De forma mais específica, os principais pontos abordados serão a organização da aplicação baseada no framework, a utilização de *templates*[3] e abstrações nas requisições AJAX, o uso do *Data Binding*, a utilização do desenvolvimento em página única (SPA), os filtros *built-in* e personalizados e a experiência de desenvolvimento com o framework.

1 REFERENCIAL TEÓRICO

Este referencial apresenta os conceitos estudados que são necessários para o entendimento deste artigo. Na seção 1.1, serão abordados os modelos básicos de desenvolvimento Web utilizadas atualmente. Na seção 1.2, serão descritos os conceitos que estão diretamente ligados ao Framework AngularJS, objeto de estudo do trabalho.

1.1 Aplicações e desenvolvimento Web

Segundo Nations (s. d.), “Uma aplicação Web é qualquer aplicação que utiliza o navegador como cliente, seja uma aplicação simples como um quadro de mensagens ou um complexo processador de palavras”. As aplicações Web atuais se utilizam de ferramentas

[3] *Templates* são trechos de código HTML separados do arquivo principal, com a finalidade de organizar a aplicação.

padrão de desenvolvimento, como a *HyperText Markup Language* (HTML) (ROBBINS, 2013, p.1), *Cascading Style Sheet*, (CSS) (OLSSON, p. XIX) e geralmente usam combinações de scripts *server-side* (como PHP e JSP) e *client-side* (como JavaScript), juntamente a uma tecnologia de banco de dados, como MySQL. (WELLING e THOMSON, 2009, p. 3)

O desenvolvimento Web é a programação que permite a funcionalidade dos websites, conforme necessidade do proprietário. Normalmente, trabalha diretamente com a parte de codificação e marcação, não adentrando a parte de projeto do sistema. (JANSSEN, s. d.)

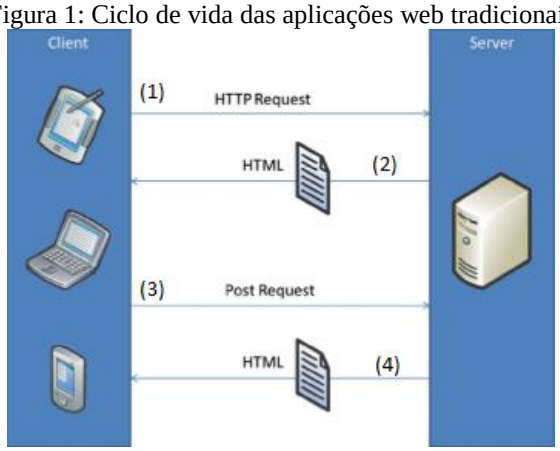
A arquitetura de desenvolvimento para este ambiente mostra-se principalmente em dois amplos e distintos tipos: tradicional e moderno, que serão conceituados a seguir.

1.1.1 Modelo de desenvolvimento web tradicional

As páginas se tornaram dinâmicas graças aos *webservers*[4] e às linguagens *server-side*, que recebem as requisições e enviam o conteúdo requisitado junto a resposta para o cliente. (FINK; FLATOW, 2014, p. 3)

O ciclo de vida destas aplicações está disposto na Figura 1:

Figura 1: Ciclo de vida das aplicações web tradicionais



Fonte: FINK; FLATOW, 2014

Quando o usuário abre a página, este faz uma requisição HTTP ao servidor (1), o qual a responde com páginas HTML (2). Através da submissão de um formulário ou link para outra página, o usuário submete uma nova requisição HTTP ao servidor (3), e este retorna um novo HTML, sendo necessária a recarga completa da página (4). Isso ocorre a toda interação

[4] *Webservers* são computadores que provêm arquivos conforme requisições recebidas. Basicamente, recebem uma solicitação, decodificam essa solicitação e verificam qual arquivo está sendo solicitado. Caso esse arquivo esteja disponível, ele é então entregue em resposta à solicitação. Exemplos de *webserver*: Apache e NodeJS.

do usuário que faz uma chamada HTTP ao servidor, pois nesse tipo de aplicação, a maior parte do desenvolvimento é controlada pelo servidor. (FINK; FLATOW, 2014, p.7 e 8)

Esta abordagem de desenvolvimento gera um *overhead* [5]desnecessário de cabeçalhos e requisições ao servidor, pois a cada *reload* [6]realizado na página, todos os arquivos HTML, CSS e JavaScript são baixados novamente, tornando a exibição do conteúdo mais lenta e levando à necessidade de um novo modelo de desenvolvimento, que é aqui chamado de modelo de desenvolvimento web moderno e será abordado na próxima seção.

1.1.2 Modelo de desenvolvimento web moderno

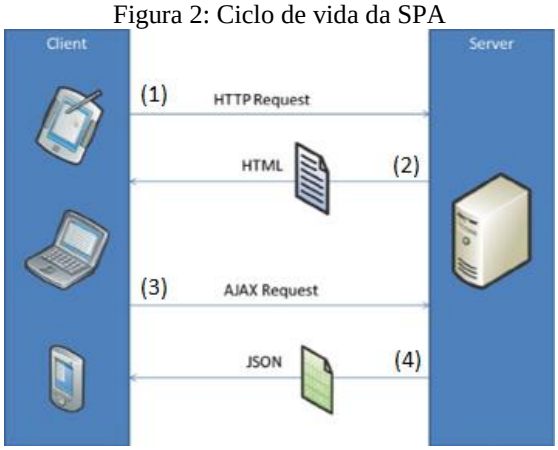
Em contrapartida ao estilo de desenvolvimento tradicional, o modelo de desenvolvimento moderno busca trabalhar de forma assíncrona, sem bloqueio de uso do sistema durante as requisições, onde a manipulação do documento é realizada em linguagens *client-side* nativas e amplamente suportadas (como JavaScript) que não necessitam de instalação de plug-ins e são multiplataforma (no sentido de precisar de somente um navegador que interprete as linguagens *client-side* em qualquer tipo de dispositivo). Esse modelo é composto pela utilização de HTML ou HTML5 e *Asynchronous JavaScript and XML* (AJAX)(W3SCHOOLS), trabalhando de forma independente da linguagem *server-side*, sendo delegada a esta somente receber requisições geradas pelas chamadas AJAX e responder em formato *JavaScript Object Notation*. (JSON)

Este modelo de desenvolvimento se tornou possível do ponto de vista de implementação quando o AJAX emergiu como um padrão de criação de aplicações Web e sua capacidade foi vista por grandes empresas como a Google, abrindo portas para a evolução da linguagem JavaScript. A habilidade do AJAX em fazer requisições assíncronas sem bloquear a interface do usuário e atualizar somente uma seção da página sem precisar recarregá-la revolucionou a experiência do usuário. Posteriormente, as bibliotecas JavaScript de manipulação do HTML como jQuery (JQUERY) trouxeram abstrações às chamadas AJAX, trazendo cada vez mais programadores a esse "novo mundo" das páginas Web. (FINK; FLATOW, 2014, p.4 e 5). A inserção do padrão HTML5 também contribuiu para o melhoramento e evolução do JavaScript. (FINK; FLATOW, 2014, p.5)

[5] *Overhead* é o termo utilizado para descrever o custo excessivo na realização de uma atividade. Na situação apresentada no trabalho, significa o excesso de envio de requisições ao servidor web.
[6] *Reload* é o termo utilizado para designar a recarga completa de uma página.

Nesse ambiente, as *Single Page Applications* (SPA) ganharam espaço. A SPA é uma técnica Web que utiliza uma única página HTML como base para todas as páginas da aplicação nas quais as interações do usuário final são implementadas. (FINK; FLATOW, 2014, p. 3). Seu ciclo de vida está ilustrado na Figura 2:

Figura 2: Ciclo de vida da SPA



Fonte: FINK; FLATOW, 2014

Nas aplicações de página única, uma requisição inicial HTTP de abertura da página é enviada ao servidor (1), que retorna um documento HTML (2). Este HTML inicial, que nunca é recarregado ou reposto, é exibido no navegador e as interações do usuário são guiadas por requisições AJAX. (FREEMAN, 2014, p. 46)

Ao invés de recarregar a página a cada nova requisição, são feitas requisições AJAX ao servidor para solicitar os dados necessários (3), os quais são retornados em JSON ou partes de HTML pré-renderizadas, que serão atualizadas no *Document Object Model* (DOM) e mostradas ao usuário final (4). Outro diferencial das aplicações de página única é que a transição de páginas é realizada no lado cliente da aplicação. (FINK; FLATOW, 2014, p.12)

Portanto, este modelo se difere do tradicional em relação ao *overhead* de requisições, tornando a aplicação mais leve e permite melhor usabilidade para o usuário final, devido ao AJAX não bloquear a manipulação da página durante sua execução. A grande maioria das SPA's é desenvolvida no lado do cliente, em oposição às aplicações tradicionais que interagem com o lado servidor e demandam recargas constantes de páginas inteiras. (FINK; FLATOW, 2014, p. 3)

Dentro deste contexto de desenvolvimento moderno, o AngularJS tem chamado a atenção por sua leveza e pela facilidade com a qual trabalha os conceitos e os paradigmas do modelo moderno. A próxima seção exibirá os conceitos básicos do framework, comentando inicialmente o que ele é e posteriormente suas dependências e sua arquitetura.

1.2 AngularJS

O AngularJS é um framework estrutural para desenvolvimento de aplicativos Web dinâmicos. Ele permite a utilização do HTML como um *template*, possibilitando que a sintaxe HTML seja estendida para representar de forma mais clara e sucinta os componentes da aplicação. O AngularJS é um arquivo JavaScript, não necessitando procedimentos de instalação. (ANGULARJS)

A construção do AngularJS enfatiza pontos importantes do desenvolvimento. São eles:

- Expansão: facilidade no entendimento de complexos aplicativos desenvolvidos, uma vez que se entenda o básico do Angular. Isso permite que a aplicação cresça e seja incrementada conforme a necessidade do usuário;
- Manutenção: a modularização da aplicação facilita o *debug* do sistema, simplificando a manutenção em longa data;
- Testes: capacidade de encontrar falhas e corrigir estas falhas de maneira simples e rápida antes da implantação ao usuário. (FREEMAN, 2014, p. 3)

O objetivo do AngularJS é trazer grande parte das ferramentas e capacidades que são codificadas no lado servidor para o lado cliente. Com isso, pretende-se facilitar o desenvolvimento, os testes e a manutenção dos sistemas. O Angular permite também a extensão do HTML expressando as funcionalidades através de elementos personalizados, atributos, classes e comentários. (FREEMAN, 2014, p. 45)

O framework faz uso dos conceitos de *Data Binding* e de injeção de dependência, que são importantes para o entendimento da ferramenta. O *Data Binding* é uma técnica que se constitui em modificar algo no modelo (dados dos objetos) da aplicação e refletir no DOM as novas informações ou vice-versa. Bibliotecas como jQuery proveem maneiras de modificar parte do DOM separadamente, sem a necessidade de recarregar a página completamente. Neste caso, são adicionados somente dados no *template*, que são atualizados com a ajuda da função *innerHtml*. A execução desta função alcança o objetivo principal que é a atualização do *template* de maneira assíncrona, sem necessidade de novo carregamento da página, porém demanda cuidados não triviais para o controle tanto dos dados no HTML quanto dos dados nos objetos JavaScript. Com o uso do AngularJS, esse trabalho de verificação é geralmente automático, e a preocupação com essa vinculação é reduzida, além de que ele implementa um tipo de *data binding* chamado *two-way data binding*, que consiste em modificar o DOM e refletir nos objetos do modelo ou modificar o modelo e espelhar a modificação no DOM. (GREEN; SESHADRI, 2013, p. 4). Já a injeção de dependência é um conjunto de princípios e

padrões de software que permite o desenvolvimento de aplicações menos acopladas através da inclusão de bibliotecas/dependências a um módulo quando este for necessário. Segundo Jeremy (2013, p. 2), “Injeção de dependência é tudo relacionado à criação de códigos pouco acoplados”.

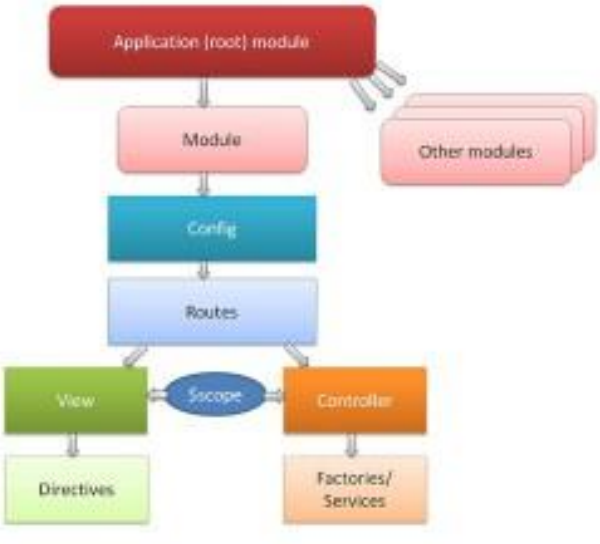
Após apresentadas as técnicas essenciais ao entendimento dos conceitos usados pelo AngularJS, a próxima seção descreverá a arquitetura do framework, juntamente com os a descrição dos componentes que constituem o framework.

1.2.1 Arquitetura do AngularJS

A arquitetura do AngularJS consiste na criação de módulos de sistema que podem trabalhar de forma separada ou conjunta. O Angular trabalha de forma a solicitar informações do lado servidor somente quando necessário, sendo todas as outras atividades da aplicação controladas por ele na camada de visão. (MARCO)

A organização das aplicações criadas com o AngularJS geralmente apresenta a estrutura ilustrada na Figura 3:

Figura 3: Organização de uma aplicação AngularJS

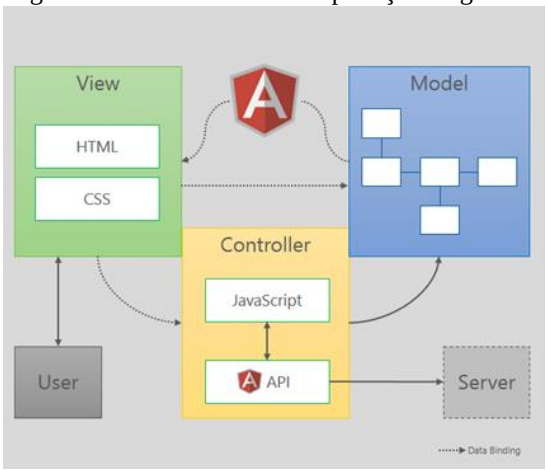


Fonte: MARCO, 2013.

A ideia principal é que cada módulo possa conter suas próprias configurações, rotas e sua própria lista de dependências. Quando o usuário acessa uma determinada URL dentro do sistema, a rota controlada pelo módulo carrega na tela o *template* e o controlador JavaScript

responsável pelo *template*. Feito isso, o usuário passa a manipular o *template* gerando troca de informações e requisições ao servidor. Esse fluxo de dados é representado na Figura 4:

Figura 4: Fluxo de dados em aplicação AngularJS



Fonte: STROPEK, 2013.

Na Figura 4, o usuário interage com a *view*, que está vinculada ao controlador através de diretivas do AngularJS dentro do *template* carregado no momento do roteamento. Quando as interações do usuário resultam em chamadas ao servidor (como salvar dados de um formulário), essa ação é tratada pelo *controller*, que utiliza algum *service* do AngularJS (como o *$http*) para abstrair a interação com o *server*, que retorna os dados conforme a função chamada. Estes dados retornados em formato JSON são espelhados no *model*, que posteriormente são refletidos na *view* através da técnica de *two-way data binding* fornecida pelo Angular. (STROPEK, 2013)

As subseções seguintes conceituam os componentes que possibilitam a organização da aplicação em módulos e o fluxo de dados descrito através da Figura 4.

1.2.1.1 Módulos

Os módulos são basicamente *containers* para as demais partes do aplicativo (como controladores, serviços, filtros, diretivas, entre outros). Eles são fragmentos da aplicação, que se interligam com os demais módulos para formar o sistema. O objetivo da modularização é ter o menor acoplamento possível da aplicação, podendo estes serem utilizados em outros sistemas, caracterizando o reaproveitamento de código. (ANGULARJS)

1.2.1.2 Controladores

Os controladores são funções JavaScript usadas para ampliar o escopo do AngularJS. Eles podem ser usados para iniciar o estado do objeto *$scope* e atribuir comportamento a este objeto. Quando um controlador é atrelado ao DOM através da diretiva *ng-controller*, o Angular vai instanciar um objeto de *Controller*, usando o nome especificado no construtor da função. Neste momento, um objeto $*scope* filho estará disponível como um parâmetro injetável para a função de construção do Controlador. (ANGULARJS)

Segundo Freeman (2014, p. 22), "O *controller* fica entre o modelo e a visão e conecta-os. O controlador responde à interação do usuário, atualizando os dados do modelo e fornecendo à visão os dados de que ela necessita [...]". Ele é responsável por definir como o aplicativo será iniciado. (GREEN; SESHADRI, 2013, p. 142)

1.2.1.3 Diretivas

Segundo Branas (2014, p. 18), "Uma diretiva é uma extensão do vocabulário HTML que permite ao desenvolvedor a criação de novos comportamentos. Essa tecnologia permite a criação de componentes reutilizáveis para toda a aplicação", em forma de atributos, elementos, classes e comentários. É através das diretivas que o controlador recebe as ações geradas pelo usuário, e também através destas que os dados modificados são espelhados na visão.

1.2.1.4 Expressões

Segundo Branas (2014, p. 53), "Uma expressão é uma simples parte de código que será avaliada pelo *Framework* e pode ser escrita entre chaves duplas, como por exemplo {{teste}}. É conhecida como interpolação e permite a fácil interação com qualquer objeto dentro do *scope.*". Ainda, a expressão permite a adição de cálculos, como {{2 + 2}}, e pode trabalhar em conjunto aos filtros para transformação da apresentação do conteúdo na tela. (BRANAS, 2014, p. 54). É através das expressões que o Angular mostra os dados presentes no modelo da aplicação.

1.2.1.5 Filtros

Os filtros são usados nas *views* para formatar os dados apresentados ao usuário. Uma vez que o filtro é definido no módulo, ele pode ser utilizado por todos os *controllers* e *visões* daquele módulo. (FREEMAN, 2014, p. 221)

O AngularJS traz filtros *built-in* (como “currency”, que recebe um número e formata-o para dólar ou “date”, que recebe um *timestamp* e transforma para uma data legível, segundo um formato), e também permite a criação de filtros personalizados. Basicamente, os filtros são funções JavaScript que recebem um dado e formatam este dado para ser mostrado na visão. (FREEMAN, 2014, p. 222)

1.2.1.6 Rotas

As rotas do AngularJS permitem que, ao modificar a URL, seja possível o carregamento e a apresentação de páginas (*templates*), bem como a instanciação de um *controller* que trará o contexto a este *template*. (GREEN; SESHADRI, 2013, p. 38)

A forma como o serviço *$routeProvider* trabalha é de fácil entendimento: quando as modificações no método *$location.path* atingem os mapeamentos realizados, a *view* correspondente é carregada e mostrada na tela. (FREEMAN, 2014, p. 583). O módulo ngRoute inclui uma diretiva chamada *ng-view*, que é o local onde o *template* carregado será renderizado.

1.2.1.7 Serviços

Os serviços do AngularJS são objetos ou funções conectados através do uso de Injeção de dependência. Um exemplo simples de Serviço dentro do AngularJS é o *$http*, cuja finalidade é abstrair chamadas AJAX e facilitar a utilização dos métodos padrão do protocolo HTTP (GET, POST, PUT, DELETE). (ANGULAR)

2 ESTUDO DE CASO

Esta seção apresenta a descrição do estudo de caso abordado juntamente com os pontos a serem avaliados com relação ao desenvolvimento do protótipo. Ainda, estão presentes os objetivos do estudo de caso de forma geral, a metodologia aplicada para

avaliação dos objetivos e a descrição do cenário da aplicação. Por fim, são apresentados de forma mais detalhada os resultados obtidos na utilização do AngularJS, atrelados às considerações sobre o estudo de caso.

2.1 Descrição e objetivos de avaliação

A finalidade de reproduzir um caso de uso é permitir a avaliação dos principais objetivos do trabalho, que se relacionam ao estudo da arquitetura sugerida pelo framework, a forma como este trabalha com AJAX e com a técnica de *Data Binding*, a manipulação de *templates*, a utilização de filtros baseados em *arrays* JSON vinculados ao DOM, o uso de SPA através do framework e a experiência de desenvolvimento resultada ao empregá-lo em um sistema que pode vir a ser colocado em produção.

Na subseção seguinte, será mencionada a metodologia empregada para avaliação dos pontos propostos no estudo de caso.

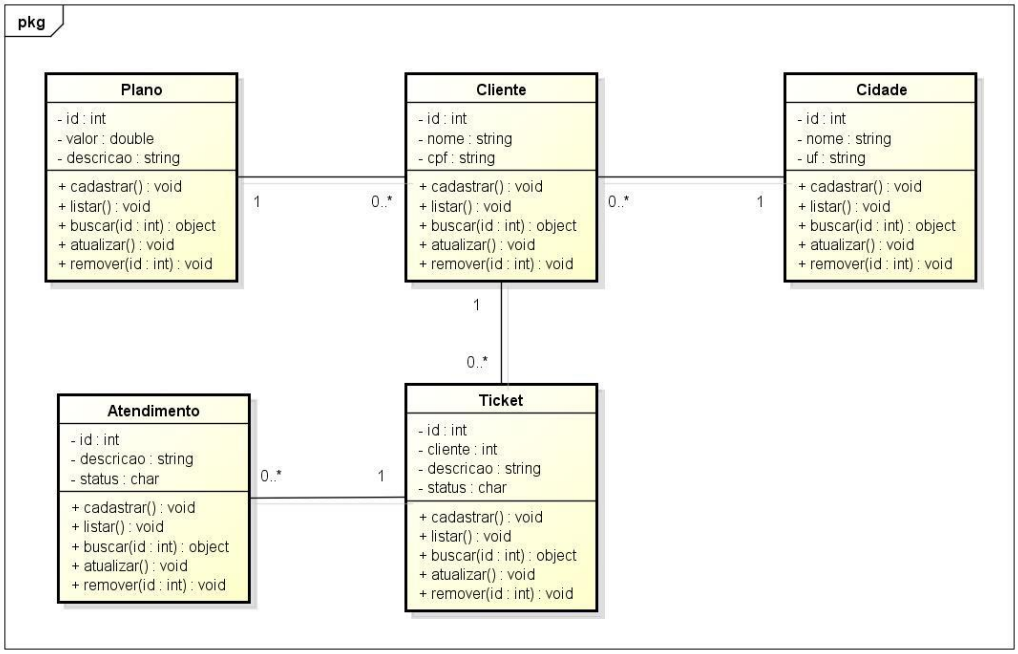
2.2 Metodologia de avaliação

Para avaliar a experiência de desenvolvimento, foi montado e codificado um caso de uso de cadastro de tickets e atendimentos. Dentro deste caso de uso, foi possível a avaliação da arquitetura sugerida pelo framework através da organização dos diretórios do sistema e do uso de SPA, que segue o modelo de desenvolvimento Web moderno onde o AngularJS está inserido. Para análise da forma de trabalho das requisições AJAX, foi montado um *service* dentro do módulo principal através do qual todos os outros módulos comunicam-se com o lado servidor da aplicação para obtenção dos dados armazenados, abordando também a técnica de *Data Binding* para exibição destes dados na camada de visão e a utilização dos componentes de filtro do Angular para modificação da apresentação dos dados obtidos. A manipulação de *templates* é realizada em toda a aplicação e será ilustrada através de figuras da tela principal e demonstração do funcionamento de rotas do Angular.

2.3 Cenário de avaliação

O cenário de avaliação da aplicação baseia-se em um caso de uso de cadastro de tickets de suporte e atendimentos, conforme ilustração do diagrama de classes da Figura 5:

Figura 5: Diagrama de classes do caso de uso



Fonte: Autor

As classes Cidade e Plano são associadas à classe Cliente, a qual obrigatoriamente deve estar atrelada a uma cidade e a um plano. O cliente tem Tickets, através dos quais ele pode informar problemas ocorridos. Os Atendimentos ligam-se diretamente ao Ticket, pois é através dos atendimentos que é definido o Status do Ticket, podendo ser ele: Novo, que representa um Ticket recém-aberto pelo cliente e que não recebeu atendimento; Em andamento, que representa um Ticket que está sendo verificado; Concluído, que representa um Ticket já atendido e resolvido. Para o desenvolvimento do lado cliente do protótipo, foram utilizadas as linguagens padrão de criação de sites (HTML, CSS e JavaScript), em conjunto à linguagem PHP para codificação *server-side* e o Sistema de Gerenciamento de Banco de Dados (SGBD) MySQL para armazenamento dos dados.

Após o desenvolvimento do caso de uso, foi possível avaliar todos os tópicos propostos anteriormente. O detalhamento da avaliação está ilustrado na seção seguinte, denominada Avaliação da utilização do AngularJS.

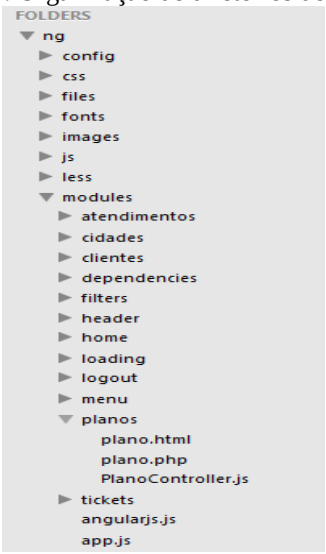
2.4 Avaliação da utilização do AngularJS

Nas seguintes subseções, serão expostas as avaliações referentes a cada ponto abordado no trabalho através da demonstração de cada tópico, ilustração e explicação de sua aplicação e os resultados obtidos.

2.4.1 Organização dos arquivos e componentes da aplicação

Com o desenvolvimento do caso de uso, foi possível analisar a forma como o sistema fica organizado e como ele pode ser expansível. A Figura 6 apresenta a organização dos diretórios do caso de uso criado:

Figura 6: Organização de diretórios do sistema



Fonte: Autor

Com base em tentativas de implementação, foi verificado que o modo ideal é deixar o módulo principal (*app.js*) em um diretório raiz e separar os outros módulos em subdiretórios que contêm todos os arquivos referentes àquele módulo (*templates*, arquivos JavaScript, arquivos de script *server-side*).

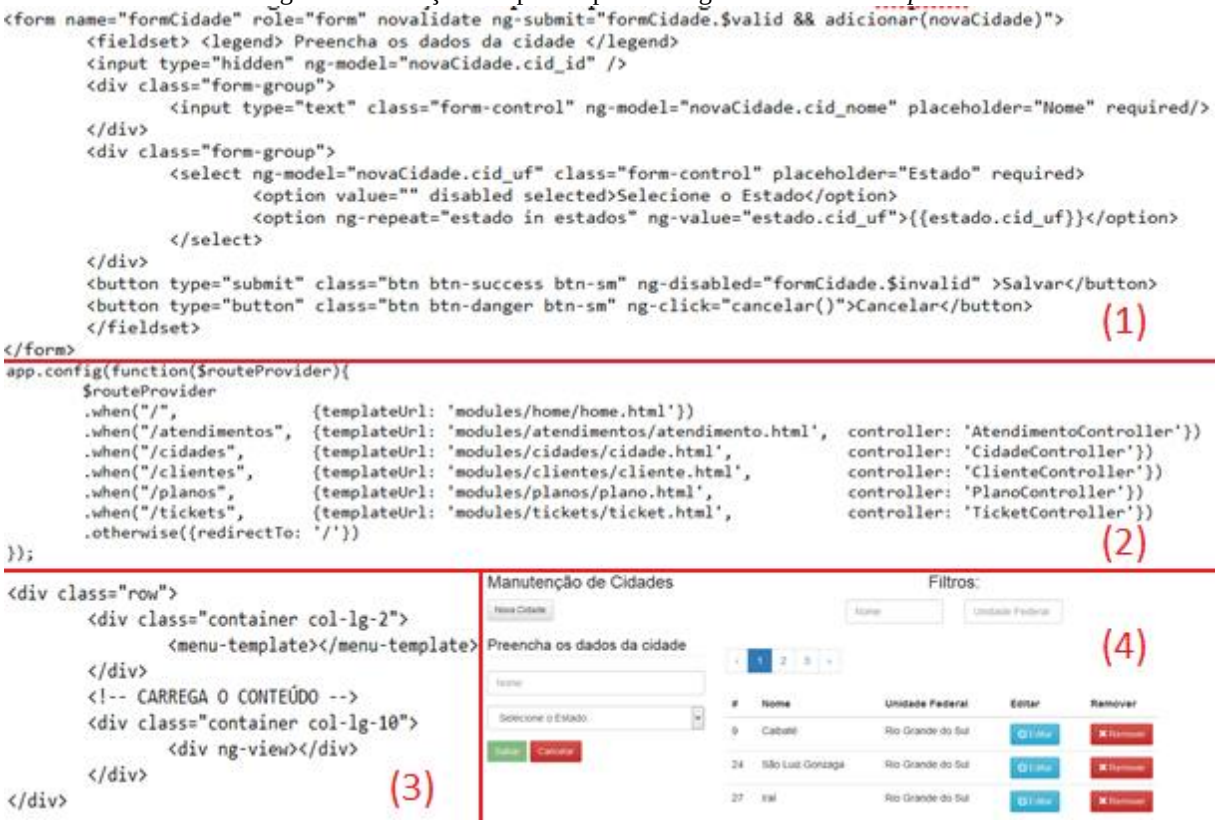
Desta maneira, é possível deixar no módulo principal as funções cabíveis a todos os módulos, e nos módulos "filhos", somente as funções específicas de cada um. Isso permite a inclusão de módulos no sistema sem a necessidade de modificações nos já existentes, melhorando a expansão da aplicação e a possibilidade de reutilização de código.

2.4.2 Utilização de *templates*

O uso de *templates* ocorre em toda a aplicação do caso de uso. Foram utilizados *templates* fixos que não envolvem processamento (como o cabeçalho), *templates* fixos na tela que executam funções (menu) e *templates* dinâmicos, que mudam seus dados conforme ações

do usuário e são carregados sob demanda. O modo de carregamento de um *template* é ilustrado na Figura 7:

Figura 7: Ilustração dos passos para carregamento de um *template*



Fonte: Autor

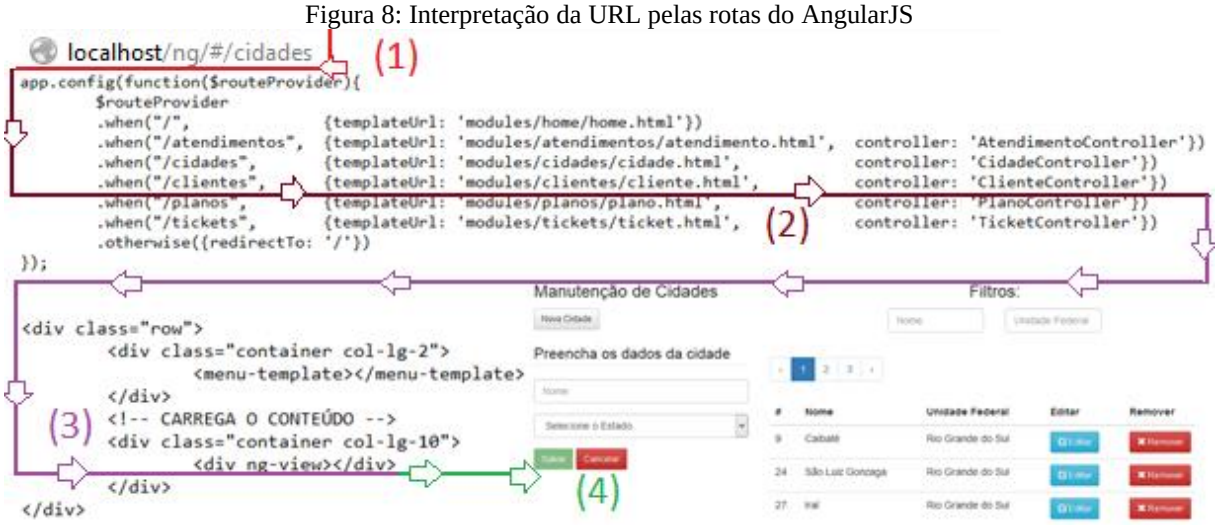
O *template* é criado em um arquivo específico, para separá-lo do código principal (1). Esse arquivo é então referenciado no roteamento da aplicação, através do atributo *templateUrl*, que indica o caminho para o arquivo e o atributo *controller*, que indica o controlador que coordenará esse *template* (2). Quando a URL na barra de endereços do navegador é modificada e atende a um dos roteamentos criados, a ação é realizada e o *template* é renderizado na diretiva *ng-view* (3), sendo finalmente exibido ao usuário final (4).

O AngularJS torna simples a implementação e o uso de *templates* dinâmicos através do uso de rotas. Conforme demonstrado na Figura 7, com poucos trechos de código, pode-se criar diversas rotas em uma aplicação e separar o conteúdo exibido nas páginas em arquivos específicos. Esse modo de implementar *templates* implica na facilidade do emprego da técnica SPA, que será retratada na subseção seguinte.

2.4.3 Utilização de SPA

Todo o caso de uso baseou-se na implementação da técnica de *Single Page Application*. Para uma análise mais adequada dessa técnica, foi implementado um menu lateral que contém links para os módulos. Através das mudanças na URL executadas pela seleção de um módulo no menu, ocorre a modificação no método JavaScript window.location, abstraído pelo AngularJS através do *service $location*. A Figura 8 apresenta o caminho percorrido pelo serviço *$routeProvider* quando é feita qualquer modificação na URL:

Figura 8: Interpretação da URL pelas rotas do AngularJS



Fonte: Autor

Toda a vez que é realizada alguma modificação na URL (1), o serviço *$location* percorre todas as rotas configuradas nos módulos e verifica se alguma condiz com a nova URL, carregando o *template inline* ou o *template* localizado no caminho informado (2). A implantação de rotas é realizada através dos métodos do serviço *$routeProvider* (*when*, que indica "Quando a URL condizer com" e *otherwise*, que é executado caso nenhuma rota combine com o conteúdo da URL), incluindo na diretiva *ng-view* (3), que fica na página principal o *template* carregado (4).

2.4.4 *Data Binding*

O uso do *Data Binding* ocorre em todos os formulários e listagens do caso de uso. Ao realizar qualquer ação sobre um registro (listar, editar, remover ou adicionar) através da visão que resulte em uma modificação dos objetos do *$scope*, essa modificação é automaticamente

espelhada na visão. O mesmo ocorre quando algo é alterado no modelo: feita a alteração, o AngularJS já transmite as modificações à camada de visão. Para que isso ocorra, o AngularJS analisa o *template* ao carregá-lo, verificando as diretivas e as marcações inseridas. A partir disso, o framework produz uma "visão ao vivo", através da qual ele trabalha para transmitir as informações tanto do modelo para a visão quanto da visão para o modelo.

A abstração que o framework fornece para realizar este tipo de manipulação automatiza todo o processo de vinculação e sincronização entre os dados presentes no modelo com os dados mostrados no *template* ao usuário final.

2.4.5 Filtros *built-in* e personalizados

Os filtros da aplicação foram utilizados em todos os módulos que permitem listagem de dados. Estes foram usados para filtragem de dados das listas (buscas por Nome, por exemplo) e juntamente à exibição dos dados (como máscara de Valor, para exibição em casas decimais). Também foi criado um filtro personalizado para exibição do CPF. A Figura 10 mostra o uso do filtro dentro do *template* e o código do filtro personalizado.

Figura 10: Uso do filtro no *template* e código de filtro personalizado

```
<tbody>
        <tr dir-paginate="cliente in clientes | filter: {cli_nome : cliente.cli_nome, cli_cpf: cliente.cli_cpf, pla_descricao:
                                                cliente.pla_descricao, cid_nome: cliente.cid_nome } | itemsPerPage: pageSize ">
                <td>{{cliente.cli_id}}</td>
                <td>{{cliente.cli_nome}}</td>
                <td>{{cliente.cli_cpf | cpf }}</td>
                <td>{{cliente.pla_descricao }}</td>
                <td>{{cliente.cid_nome}}</td>                                                                    (1)
                <td>    <button class="btn btn-info btn-sm" ng-click="buscar(cliente,$index)">
                                <span class="glyphicon glyphicon-edit"></span> Editar
                        </button>
                </td>
                <td>    <button class="btn btn-danger btn-sm"  ng-click="remover(cliente,$index)">
                                <span class="glyphicon glyphicon-remove"></span> Remover
                        </button>
                </td>
        </tr>
</tbody>
app.filter('cpf', function(){
        return function(cpf){                                                                                    (2)
                cpf     = cpf || '';
                out     = "";
                if(cpf.length != 11){
                        out = "CPF inválido";
                } else {
                        out = cpf.substr(0, 3) + '.' + cpf.substr(3, 3) + '.' + cpf.substr(6, 3) + '-' + cpf.substr(9,2);
                }
                return out;
        };
});
```

Fonte: Autor

Durante a execução da diretiva *dir-paginate* no *template* (1), para cada cliente dentro da lista de clientes que está no objeto *$scope.clientes*, o AngularJS submete o atributo *cliente.cli_cpf* para o filtro *cpf* (2). Este, então, formata o parâmetro recebido e retorna ele formatado, sendo então exibido na tela. Um detalhe verificado é que, em listas muito grandes,

há a necessidade de cuidar o uso de filtros que envolvam muito processamento para que estes não venham a travar a execução do script.

2.4.6 Abstração das requisições AJAX

A observação deste item foi realizada através da criação de um *service* inserido no módulo principal do sistema que contém os métodos para as ações de listagem, inserção, edição e deleção de registros do banco de dados. A Figura 11 apresenta o código do *service* em questão:

Figura 11: Código do *service* criado para ilustrar as chamadas AJAX

```
app.factory('Funcoes', function ($http) {
	var resultado = {
		listarDados: function(data){
			var objetos =
				$http.get(data).then(function(retorno){
					return retorno.data;
				});
			return objetos;
		},
		removerDados: function(data){
			var objetos =
				$http.delete(data.url, {params: {objeto: data}}).then(function(retorno){
					return retorno.data;
				});
			return objetos;
		},
		adicionarDados: function(data){
			var objetos =
				$http.post(data.url, $.param({objeto: data}),
				    {headers: {'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8' }})
				    .then(function(retorno){
					return retorno.data;
				});
			return objetos;
		},
		atualizarDados: function(data){
			var objetos =
				$http.put(data.url, data,
				{headers: {'Content-Type': 'application/x-www-form-urlencoded; charset=UTF-8' }})
				.then(function(retorno){
					return retorno.data;
				});
			return objetos;
		}
	};
	return resultado;
});
```

Fonte: Autor

Foi criado um *service* com nome “Funcoes”, que recebe por dependência outro serviço provido pelo AngularJS chamado *$http*. O *$http* executa facilmente chamadas de 4 métodos padrão do protocolo HTTP: GET, POST, PUT e DELETE. No método GET, basta passar a URL do recurso solicitado. Já no método DELETE, é necessário passar a URL e os parâmetros, caso sejam necessários. Devido ao PHP não ter *arrays* específicos para tratamento de JSON Post e chamadas PUT, é necessário informar o tipo de conteúdo a ser

enviado através do parâmetro *headers*. Essa abstração fornecida pelo AngularJS para operações HTTP é de grande valia para aplicações baseadas na arquitetura REST.

2.4.7 Experiência de desenvolvimento

Para avaliar este quesito, foi realizado o desenvolvimento de um caso de uso. Os principais pontos levantados são:

- Expansão da aplicação: uma vez feita a divisão da aplicação em módulos, o framework favoreceu a expansão através de sua abstração na parte de injeção de dependência. Para adicionar um novo módulo na aplicação, por exemplo, basta colocá-lo como dependência dentro do módulo principal ou, caso assim for preferido, colocá-lo somente como dependência nos módulos que o utilizarem;
- Reutilização de código: o AngularJS se mostrou muito eficiente na parte de reutilização do código previamente desenvolvido, pois quando se tem um módulo pronto e bem definido, este módulo geralmente será utilizado em outras aplicações. Mesmo que a nova aplicação contenha campos diferentes, a lógica de criação permanece igual, facilitando sua replicação. Outro detalhe importante é que normalmente todos os módulos seguem o mesmo fluxo de chamadas, sendo possível fazer uma quantidade grande de módulos a partir de um principal e em um espaço reduzido de tempo;
- Mantenabilidade e testabilidade: no que diz respeito a manutenção e testes, a aplicação se tornou limpa e a depuração se tornou simplificada. A modularização auxilia muito a fase de testes do sistema. Como o módulo é praticamente independente dos demais, este pode ser testado separadamente dos outros, agilizando a busca por falhas. Além disso, através da reutilização de código, há ganhos na consistência da aplicação, uma vez que um módulo é previamente testado e então replicado;
- Experiência de uso do sistema: a aplicação desenvolvida com o uso do framework apresentou uma boa experiência de uso. O uso da técnica de SPA permite que o usuário consiga alternar entre as páginas rapidamente, mesmo que ele tenha uma conexão razoável com a internet. Além disso, de forma extremamente abstraída pelo Angular, as buscas (filtros) são quase instantâneas, permitindo melhor visualização dos dados. As validações em formulários também melhoram a usabilidade da

aplicação, uma vez que pode ser desabilitado o botão de submissão do formulário enquanto este não atingir as regras de validação impostas pelo sistema.

Dadas as avaliações acima, a experiência de desenvolvimento com o framework AngularJS foi muito interessante e produtiva. Além de ser altamente otimizado para renderizar e vincular dados, o Angular abstrai diversas atividades que nem sempre são simples de desenvolver ou de controlar.

2.4.8 Considerações da avaliação

O AngularJS se mostrou um ótimo framework ao que ele é proposto. Altamente otimizado, ele fornece à equipe de desenvolvimento praticidade e robustez, juntamente com grandes abstrações em tarefas não triviais para a maior parte dos desenvolvedores. Aliado à organização da aplicação, proporciona simplicidade e qualidade de código, resultando melhor entendimento da estrutura e da codificação do sistema.

CONSIDERAÇÕES FINAIS

Este trabalho propôs um estudo e desenvolvimento de um caso de uso a partir da utilização do framework de desenvolvimento web AngularJS. A aplicação das ferramentas e funcionalidades fornecidas pelo AngularJS permite a criação de sistemas Web rápidos e intuitivos, inserindo o usuário em um ambiente agradável de navegação e manipulação das páginas. Para os desenvolvedores, o uso da ferramenta no escopo do projeto a ser criado traz vantagens relevantes, como a facilidade na codificação, a possibilidade de reutilização de códigos devido à modularidade do projeto e a clareza na leitura do código.

A otimização, abstração e leveza da tecnologia permite sua aplicação em muitos ambientes de desenvolvimento Web, seja em sites demonstrativos (onde somente é exibido conteúdo, sem formulários), geralmente mais simples, ou complexos softwares de gerência, com alta interação do usuário ao sistema. Logicamente, o AngularJS não é a ferramenta mais adequada para todas as demandas de desenvolvimento, porém é um dos principais frameworks quando se quer melhorar a produtividade da equipe de desenvolvimento e fornecer ótima experiência ao usuário final.

STUDY AND DEVELOPMENT OF USE CASE WITH ANGULARJS FRAMEWORK

ABSTRACT

The web application development techniques allows developers to constantly improve your applications and systems and provide better experience to final users. The AngularJS is a framework that provides function and an architecture that support practically all requirements in Internet programming environment and therefore is being applied frequently in companies. This article presents a study of the conceitual part of framework and after presents the obtained results through a case of use that allow AngularJS evaluation. The developed prototype for tool evaluation determined a lot of factors that demonstrate the quality and the results received by AngularJS implementation in Web systems. The final result of evaluation shows the expected return, because the tool took positive effects in all evaluated points.

Keywords: Application. System. Development. Abstraction. Ease.

REFERÊNCIAS

ANGULARJS. **What is Angular?**. Disponível em: <https://docs.angularjs.org/guide/introduction>. Acesso em: 09 nov. 2014.

BRANAS, Rodrigo. **AngularJS Essentials**. Birmingham: Packt Publishing, 2014. Disponível em: <http://it-ebooks.info/book/3899/>. Acesso em: 05 nov. 2014.

FINK, Gil; FLATOW, Ido. **Pro Single Page Application Development**: Using Backbone.js and ASP.NET. Nova Iorque: Apress Media LLC.,2014. Disponível em: <http://www.it-ebooks.info/book/3692/>. Acesso em 08 nov. 2014.

FREEMAN, Adam. **Pro AngularJS**. Nova Iorque: Apress Media LLC., 2014. Disponível em: <http://it-ebooks.info/book/3847/>. Acesso em: 05 nov. 2014.

GREEN, Brad; SESHADRI, Shyam. **AngularJS**. Sebastopol: O'Reilly Media, 2013. Disponível em: <http://it-ebooks.info/book/2076/>. Acesso em: 05 nov. 2014.

JANSSEN, Cory. **Web Development**. Disponível em: <http://www.techopedia.com/definition/23889/web-development>. Acesso em 08 nov. 2014.

JEREMY, Clark. **Dependency Injection**: A Pratical Introdution. 2013. Disponível em: <http://www.jeremybytes.com/downloads/dependencyinjection.pdf>. Acesso em 09 nov. 2014.

JQUERY. **What is jQuery?.** Disponível em: <https://jquery.com/>. Acesso em 18 mai. 2015.

MARCO. **How to structure large angularJS applications**. Disponível em: <http://entwicklertagebuch.com/blog/2013/10/how-to-structure-large-angularjs-applications/>. Acesso em 24 mai. 2015.

NATIONS, Daniel. **Web applications**: What is a Web application?. Disponível em <http://webtrends.about.com/od/webapplications/a/web_application.htm>. Acesso em 09 nov. 2014.

OLSSON, Mikael. **CSS Quick Syntax Reference**. Nova Iorque: Apress Media LLC., 2014. Disponível em: <http://www.it-ebooks.info/book/3810/>. Acesso em 08 nov. 2014.

ROBBINS, Jennifer Nierdest. **HTML5 Pocket Reference**. Sebastopol: O'Reilly Media, 2013. Disponível em: <http://www.it-ebooks.info/book/2516/>. Acesso em 08 nov. 2014.

STROPEK, Rainer. **AngularJS with TypeScript and Windows Azure Mobile Services**, 2013. Disponível em: <http://goo.gl/O2NENn>. Acesso em 24 mai. 2015.

W3SCHOOLS. **AJAX Tutorial.** Disponível em: <http://www.w3schools.com/ajax/>. Acesso em: 18 mai. 2015.

WELLING, Luke; THOMSON, Laura. **PHP and MySQL Web Development**. Indianápolis: Pearson Education, Inc., 2009. Disponível em: <http://goo.gl/PpxW04>. Acesso em 06 nov. 2014.