Implementando um sistema de informação de análise de dados focado no front-end usando SPA e MVC client-side

Rafael Andrade de Oliveira[1], **Diogo Lucas**[1]

[1] Pós-Graduação em Tecnologias Aplicadas a Sistemas de Informação com Métodos Ágeis
Centro Universitário Ritter dos Reis (UniRitter)
Caixa Postal 1355 – 90.840-440 – Porto Alegre – RS – Brasil

eu.rafa@gmail.com, diogolucas@gmail.com

Abstract. *[This paper describes a case study on the implementation of an statistical information system using the single-page application concept. The objective of this study is to provide a different system architecture to realize data analysis and sought to join relevant subjects present in the field of Information Technology as data analysis, BI, client-side MVC and REST services. For this study were chosen few tools and frameworks in order to enable the idea of the proposed information system, its features and advantages that are detailed throughout this article.]*

Resumo. *O presente trabalho relata a implementação de um sistema de informação de análise de dados com foco no front-end, tendo como objetivo apresentar uma proposta de arquitetura para realizar análises de dados no lado cliente, usando o conceito SPA (Single-Page Application) aliado ao modelo MVC (Model-View-Controller) client-side. Para viabilizar a ideia do sistema de informação proposto foram escolhidas algumas ferramentas e frameworks existentes. No escopo do artigo consta o detalhamento das características e diferenciais do sistema de informação, a forma de manipulação sobre cubo de dados e as ferramentas utilizadas na composição da arquitetura proposta.*

1. Introdução

Este artigo irá apresentar um sistema de informação de análise de dados com foco no *front-end*, trazendo consigo uma proposta de arquitetura de *software* moderna voltada para a *web*. O objetivo desta implementação é possibilitar a realização de análises de dados diretamente no lado cliente sem a necessidade de requisições HTTP (HyperText Transfer Protocol) adicionais, trocas de mensagens ou consumo de serviços *web* a cada interação.

A escolha do tema justifica-se pela importância da análise de dados para tomada de decisão e competitividade no mercado e a relevância de arquitetura de *software* modernas acompanhando a evolução da Internet e das linguagens de programação. Em relação a análise de dados, percebemos que atualmente muitas empresas estão investindo em soluções de BI (Business Intelligence) e BigData, entendendo que a análise do seu próprio negócio e o levantamento de indicadores é importante para alavancar resultados. Sobre a arquitetura de *software*, sabe-se da necessidade de evolução constante para acompanhar as inovações tecnológicas, além de mudanças comportamentais e culturais da sociedade. O surgimento de novos dispositivos móveis como tablets e a modernização de celulares

(*smartphones*) exigem avanços nos aplicativos fazendo com que a arquitetura de *software* se envolvesse por exemplo com mobilidade, computação na nuvem (*cloud computing*) e design responsivo. Por outro lado, o volume de pessoas com acesso a Internet e a milhares de aplicativos também estimula a competitividade, trazendo a preocupação com UX (*User eXperience*) sem esquecer de pontos óbvios como desempenho e escalabilidade.

Para servir de modelo da implementação de um sistema de informação de análise de dados sobre uma arquitetura de *software* com o foco no lado cliente, foi desenvolvido o sistema **Mozaic**. Antes de apresentar a arquitetura proposta e a implementação do Mozaic, descritas na seção 4, é interessante alinhar alguns pontos facilitando o melhor entendimento do sistema e da arquitetura proposta. As próximas seções abordam conceitos, tecnologias e *frameworks* utilizados na a prova de conceito da arquitetura através do sistema de informação Mozaic.

2. Conceitos e tecnologias

Nesta seção serão descritos conceitos e tecnologias que foram utilizadas na arquitetura proposta e no desenvolvimento do sistema de informação Mozaic. Com isso, será mais fácil compreender a proposta deste artigo e a relação dos conceitos com a arquitetura proposta e as o motivo das tecnologias estarem envolvidas no Mozaic.

2.1. MVC

A busca constante pela melhor forma de desenvolvimento de *software* provoca uma série de estudos. Muitos deles resultaram na criação dos chamados Padrões de Projeto (Design Patterns), que servem de modelos arquiteturais para resolverem problemas comuns de desenvolvimento. Estes padrões focam no reaproveitamento de soluções, seguindo alguns princípios como SoC (Separation Of Concerns), DRY (Don't Repeat Yourself), KISS (Keep It Simple Stupid) que guiam os desenvolvedores e arquitetos de *software* em relação a separação de responsabilidades, não repetir ou querer reinventar a roda e manter códigos simples. Foram pensados, testados e aprimorados pro programadores experientes, dando a confiança necessária para o seu reuso.

A grande evolução no desenvolvimento *web* veio com a recomendação do uso do padrão arquitetural MVC (Model-View-Controller) [Silveira et al. 2012]. Este modelo visa a organização e a padronização da arquitetura de *software*, separando a arquitetura em três camadas como mostra a figura 1.

Figura 1. Desenho do modelo MVC e a interação entre as camadas

No modelo MVC cada um dos componentes tem responsabilidade bem definida. Model é o componente responsável pela representação do modelo de dados e mecanismo de persistência e View é responsável pela apresentação e interação com o usuário final. Já

o Controller é o mecanismo intermediário responsável por receber e responder a eventos e ações entre os componentes Model e View. MVC é um padrão de projeto arquitetônico que incentiva a organização de uma melhor aplicação através de uma separação de interesses [Osmani 2012].

2.2. JavaScript

JavaScript é considerada a linguagem de programação da Internet. Criada inicialmente para os navegadores Netscape, atualmente está presente em todos os navegadores de Internet. As páginas de Internet são baseadas em três tecnologias: HTML (HyperText Markup Language), CSS (Cascading Style Sheets) e JavaScript. Após muito tempo, HTML e CSS sofreram atualizações trazendo respectivamente nas versões HTML5 e CSS3 avanços consideráveis para a geração de páginas de Internet.

Por outro lado, o JavaScript evolui de forma significativa com a criação de novos frameworks, tornando-se, a cada dia, mais poderoso e utilizado entre os desenvolvedores [Schmitz and Lira 2013]. O JavaScript provê uma maior interatividade com a página, por exemplo respondendo a eventos de botões, validação de campos de formulário, manipulação de elementos.

Atualmente existem milhares de bibliotecas JavaScript para fins diversos e para usá-las é necessário apenas declará-las na página com a *tag* HTML <script> como mostra a figura 2. Algumas destas bibliotecas são amplamente utilizadas, como a popular biblioteca chamada jQuery, famosa pela sua facilidade na manipulação de elementos DOM (Document Object Model), estilos CSS, eventos HTML, tratamento de requisições usando AJAX (Asynchronous JavaScript and XML) e ainda de forma compaíivel com a maioria dos navegadores e respectivas versões.

Figura 2. Exemplo de inclusão de uma biblioteca JavaScript em uma página HTML

2.3. JSON

Uma notação derivada da linguagem JavaScript chamada JSON (JavaScript Object Notation) vem ganhando popularidade nos últimos anos. Criada por Douglas Crockford, JSON é uma estrutura de dados auto-descritiva, associativa e fácil de ler como demonstrado na figura 3.

O JSON é um formato de serialização de dados com base em literais de JavaScript [Flanagan 2011]. Este formato de dados tem sido bastante utilizado no desenvolvimento de aplicações *web* e serviços REST (REpresentational State Transfer), em substituição ao formato XML (eXtensible Markup Language) que é mais verboso como se pode perceber no comparativo representado na figura 4.

Figura 3. Exemplo de estrutura de dados no formato JSON

Figura 4. Comparativo entre os formatos JSON e XML

2.4. MVC client-side

MVC client-side nada mais é do que o modelo MVC descrito na seção 2.1 aplicado no lado cliente (*client-side*). Muitos desenvolvedores consideram MVC como o modelo ideal de arquitetura de *software*. Mesmo assim, este modelo de arquitetura vem sofrendo variações. Uma vez que o modelo (Model) e a apresentação (View) são essenciais para os sistemas, estas variações ocorrem no componente intermediário (Controller), dando lugar a novos modelos de arquitetura conhecidos como a família MV* ou MVW (Model-View-Whatever).

2.4.1. MVP

MVP (Model-View-Presenter). Mecanismo semelhante ao MVC, o MVP se diferencia apenas no conceito.

2.4.2. MVVM

MVVM (Model-View-ViewModel)

Binding

Novas siglas surgem como MVP (Model-View-Presenter), MVVM (Model-View-View-Model) e MV* precisam ser compreendidas.

SPA (Single-Page Applications) Quais as desvantagens de utilizar MVC client-side?

A principal dificuldade é a necessidade de aprender mais um (ou as vezes mais do que um) framework espec?fico para trabalhar exclusivamente com o front-end. A inclusão dessa parte da aplicação, apesar de facilitar a manutenção como comentado anteriormente,

adiciona uma nova camada na aplicação, que precisa ser compreendida e respeitada pelo time. Outro fator importante a ser considerado é o fato de que aplicações client-side necessitam da execução do c?digo javascript para gerarem o conte?do html e exibi-lo ao usu?rio. Apesar de praticamente n?o existirem usu?rios com javascript desativado em seus navegadores, os mecanismos de busca ainda tem dificuldade em indexar p?ginas com conte?do gerado dinamicamente no lado cliente. Se o seu projeto exige que o conte?do do seu aplicativo seja indexado por mecanismos de busca, talvez adotar uma arquitetura puramente mvc client-side n?o sejam a melhor opção.

Essa abordagem também é conhecida como *single-page applications*. Significa que, como o próprio nome diz, a aplicação é apresentada em uma página ínica e o seu conteúdo é carregado dinamicamente. Entre as vantagens que esta abordagem traz estão:

Interfaces ricas: Melhor experiência para o usuário, a medida que a aplicação web funciona similar a uma aplicação desktop, além de proporcionar melhor performance por evitar a carga completa de páginas a cada interação.

SPA / Ajax: Falar sobre REST Melhor desempenho na transferência de dados Existe também um ganho considerável em velocidade na transmissão dos dados, pois ao invés de trafegar o conteúdo HTML (HyperText Markup Language) completo a cada interação do usuário, na arquitetura client-side o aplicativo completo é transferido na primeira requisição e as requisições seguintes são responsáveis por trafegar apenas os dados brutos entre o cliente e o servidor, normalmente no formato JSON. Este ganho é visível em aplicações móveis onde a velocidade na transfer?ncia dos dados normalmente é baixa.

Setup / CDN O aplicativo completo passa a ser fornecido através de arquivos html, css e javascript que podem ser comprimidos e distribu?dos através de CDN's com facilidade. Ap?s baixados pela primeira vez esses arquivos são mantidos em cache no browser do usu?rio. O servidor fica respons?vel apenas por fornecer uma API e enviar e receber os dados brutos no formato JSON. Dessa forma todo o processamento respons?vel por parsing dos dados e geração de templates fica no lado cliente e n?o mais no servidor, liberando recursos.

Estes conceitos, aliados a evolução do JavaScript e a facilidade do formato JSON, fizeram surgir diversos *frameworks* JavaScript para trabalhar com MVC *client-side*. Ultimamente estão em destaque os *frameworks* Backbone, Ember e o Angular.

Bootstrap

É importante ressaltar que os *frameworks* MVC *client-side* não substituem os *frameworks* MVC *server-side*. Na verdade os modelos se complementam, podem e devem atuar em conjunto. O lado servidor, também chamado de *back-end*, é responsável por fornecer uma API pela qual o lado cliente irá consumir através de um *front-end* que pode tanto consumir dados para apresentação quanto consumir a API enviando dados por exemplo para serem armazenados em banco de dados.

Existem vantagens desta abordagem, uma vez que as camadas são bem isoladas, o desenvolvimento de cada lado da aplicação se torna independente. Além disso, é possível existir mais de uma aplicação no modelo MVC *client-side* fazendo uso da mesma API disponível na aplicação com modelo MVC *server-side*. Este é um cenário comum atual-

mente, onde existem versões diferentes de uma aplicação para *desktop*, *web* ou *mobile*. Uma desvantagem dessa abordagem poderia ser na questão de validação, que deve ser feita em ambos os lados, garantindo a integridade e qualidade dos sistemas.

2.5. OLAP

Quando falamos de análise de dados, um leque de opções e níveis de análise podem aparecer. Em um contexto mais analítico e interativo, o cenário se estreita a análises sobre cubo de dados multi-dimensional. OLAP (On-Line Analytical Processing)

.

Abaixo um breve descritivo dos componentes básicos contidos em cubos de dados multi-dimensionais:

Fato .
Dimensões .
Medidas .

3. Frameworks

Esta seção aborda os frameworks utilizados no desenvolvimento do sistema Mozaic, descrevendo a importância e características principais, dando uma noção do motivo pelos quais os mesmos foram escolhidos para serem usados na implementação do Mozaic.

3.1. HyperCube

HyperCube é uma biblioteca escrita em JavaScript que proporciona a criação de cubos de dados multi-dimensional, permite a aplicação de filtros e agregações, gerando dados estatíscos para fins de análise. *Open source* sob licença Apache 2.0, o HyperCube está publicado no GitHub, onde está descrito como um banco de dados OLAP (On-Line Analytical Processing) leve escrito em JavaScript, útil para qualquer aplicação que precise extrair métricas para propósitos de gráficos dinâmicos.

A estrutura de dados interpretada pelo HyperCube é um mapa no formato JSON (JavaScript Object Notation), onde seus registros contém tempo, fatos e medidas como mostra a figura 5. Esta estrutura de dados é convertida em um cubo de dados pesquisável, ou seja, apto a ser filtrado e consumido por funções matemáticas. A figura 6 mostra o trecho de código JavaScript que realiza a geração do cubo de dados.

Figura 5. Exemplo de um mapa de dados no formato JSON

Figura 6. Código JavaScript responsável pela transformação do mapa em um cubo de dados do HyperCube

3.1.1. HyperCube API

O HyperCube fornece algumas funções para trabalhar com o cubo de dados e obter as informações para análise. As mais relevantes são:

count Retorna o tamanho do cubo de dados, ou seja, a quantidade de objetos.
getFactNames Retorna uma lista com os fatos contidos no cubo de dados.
getValues Retorna uma lista com os valores distintos contidos em um determinado fato
slice Retorna um cubo de dados filtrado a partir de fatos
sliceDates Retorna um cubo de dados filtrado com dados entre duas datas
dice Retorna um cubo de dados resultante da exclusão de objetos a partir de fatos
merge Retorna a mescla um cubo de dados com outro cubo de dados
sum Retorna a soma das medidas no cubo
avg Retorna a média das medidas no cubo
topSum Retorna as maiores somas das medidas no cubo
serialize Transforma o cubo de dados no formato JSON
deserialize Cria o cubo de dados a partir de um objeto no formato JSON

3.2. AngularJS

AngularJS é um *framework* MVC *client-site* de bastante destaque criado pela da empresa Google. Em seu site, está descrito como AngularJS é construído sobre a ideologia de que a programação declarativa deve ser usada para construção de interfaces e componentes, enquanto que a programação imperativa é excelente para escrever as regras de negócio [Filho et al. 2014].

. Falar sobre MVVM

Para isso, utiliza o conceito SPA (Single-Page Application), o padrão de arquitetura MVC (Model-View-Controller) *client-side* e bibliotecas JavaScript específicas para trabalhar com cubo de dados OLAP (On-Line Analytical Processing).

4. Mozaic

O sistema de informação desenvolvido chama-se Mozaic. Trata-se de uma ferramenta de análise de dados visual que permite aplicar filtros e realizar segmentações sobre cubos de dados multi-dimensionais. É uma aplicação *web* de arquitetura moderna, criada no conceito de página única (*single-page*) usando o *framework* AngularJS. Tem como principal característica a manipulação dos dados diretamente no lado cliente (*front-end*) buscando melhor desempenho fazendo uso de recursos no lado cliente. Entende-se com isso o uso da memória pelo navegador e não tráfego de rede e consultas em banco de dados.

O Mozaic foi desenvolvido usando o conceito *single-page application*, usando o *framework* AngularJS da Google no *front-end* da aplicação. A aplicação consome uma

API de serviços RESTescritos na linguagem Java. A figura 7 apresenta a arquitetura do Mozaic e a forma de integração com a API de serviços.

Figura 7. DESENHO DA ARQUITETURA DO SISTEMA

Mais *client-side* e menos *server-side* é a ideia do Mozaic. Onde o lado servidor só é necessário se a origem dos dados for dinâmica por exemplo tabelas de banco de dados que sofrem atualizações constantes. Diferente de aplicações cliente-servidor que trabalham basicamente com troca de mensagens via requisições HTTP e outras tecnologias, como o Node.JS, processam o JavaScript no lado servidor. Basicamente, o Mozaic realiza apenas requisições para a inclusão das bibliotecas JavaScript utilizadas na página. Para importar os dados e trabalhar as análises, busca um objeto JavaScript no formato JSON para a geração do cubo de dados multi-dimensional. Os dados precisam ser carregados no JavaScript da página (*client-side*), portanto devem ser extraídos de uma requisição adicional para um arquivo, caso os dados sejam estáticos, ou extraídos de uma requisição a um serviço REST que retornará os dados no formato JSON esperado.

Com os dados no lado cliente, o Mozaic já pode montar o cubo de dados multi-dimensional e manipular os dados. Para isso, utiliza a da API (Application Programming Interface) JavaScript do HyperCube, que disponibiliza algumas funções permitindo que se extraia informações do cubo de dados, relacionada tanto a fatos quanto medidas. É dessa forma que o Mozaic acessa os dados e consegue criar os possíveis filtros e análises iniciais para apresentar ao usuário de forma visual através de gráficos e *dashboards*, interagindo com o usuário através da seleção de dados dos filtros na página.

4.1. Cubo de dados

Para servir de exemplo da implementação do sistema, foi escolhida uma base de dados de histórico de copas do mundo de futebol, fornecida pelo site da FIFA (Fédération Internationale de Football Association). Esta base de dados foi mapeada no formato JSON como mostra a figura 8, que é a estrutura conhecida para que o HyperCube faça a geração do cubo de dados multi-dimensional para aplicar suas funções estatísticas.

Figura 8. IMAGEM DO JSONEDITOR ONLINE]

4.2. Interface

Embora a principal característica do Mozaic esteja na sua ideia de arquitetura e mecanismo de análise, a *interface* da aplicação é também um ponto importantíssimo pois é no *front-end* que são montados os filtros de análise e *dashboards* com dados e gráficos consolidados. Além disso, a forma com que os dados se mantém atualizados é baseada no paradigma MVVM do Angular, mantendo **Model** e **View** sincronizados, possibilitando manter dados atualizados em tempo real.

O *design* da aplicação, bem como os gráficos, além de apresentar corretamente as informações devem mostrar os dados de forma clara e objetiva para possibilitar uma análise mais rápida e assertiva.

Configurar, CDN (Content Delivery Network)

https://angularjs.org/

Para este artigo, o Angular foi utilizado via CDN (Content Delivery Network), pelas vantagens de cache, latência e paralelismo.

Foram criadas quatro páginas com análises distintas. São elas:

4.2.1. Análise de dados históricos

.

4.2.2. R

.

4.2.3. Análise de dados histórica

.

5. Integração com MVC server-side

O sistema Mozaic prova que é possível desenvolver uma aplicação *web* para análise de dados diretamente no front-end através de um cubo de dados estático mapeado em JavaScript no formato JSON, porém nem sempre os dados disponibilizados para análise serão fixos. é interessante que o cubo de dados não seja estático.

Consumindo serviços REST

5.0.4. API de serviços REST

Para o desenvolvimento do sistema de informação Mozaic, uma simples API (Application Programming Interface) de serviços REST (Representational State Transfer) foi implementada para que o Mozaic consuma os dados iniciais e monte os dashboards e filtros no front-end. A API (Application Programming Interface) fornece o serviço carregarDados,

suportando o método HTTP (Hypertext Transfer Protocol) **GET** sem esperar parâmetros. Este serviço, escrito em Java e publicado com o *framework* Spark, é responsável por carregar todos os dados dos copas do mundo da FIFA (Fédération Internationale de Football Association) e retornar os mesmos como um objeto JSON como pode ser visto na figura 9. Inicialmente os dados foram colocados em um arquivo físico, pois o foco deste artigo é a manipulação dos dados diretamente no front-end sobre cubo de dados gerado pelo HyperCube. Os dados também poderiam ser obtidos através de um DW (Data Warehouse) em banco de dados relacional ou, melhor ainda, banco de dados não-relacional (NoSQL - Not Only SQL), porém conexão e consulta a banco de dados fogem um pouco do escopo do artigo.

Figura 9. IMAGEM DO OBJETO JSON]

Fato

Medida

HyperCube

https://github.com/thesmart/js-hypercube

Para usar o HyperCube, criamos o cubo de dados através de um objeto javacript no formato JSON[SIGLA], com uma estrutura predefinida contendo o momento do dado, fatos e medidas como é demonstrado na imagem abaixo: [IMG] O HyperCube deserializa o objeto e monta o cubo de dados para ser utilizado, ou seja, disponível para ser filtrado e calcular dados através de funções de agregação

5.1. Spark

Spark é um micro web *framework* desenvolvido que tem como característica viabilizar a criação de aplicações *web* em Java com o mínimo de esforço possível, sem a necessidade de configurações em XML (eXtensible Markup Language).

Inspirado no *framework* Sinatra, o Spark é muito leve e tem como foco a facilidade do desenvolvimento web puramente Java de forma realmente simples e elegante, o que torna divertido para os desenvolvedores.

O Spark é intrigante pelo fato de sua simplicidade [Francisco 2014]. Está na versão 2.0.0, uma versão que foi desenvolvida incluindo adaptações para a versão 8 da linguagem Java, atualizando o *framework* e usando recursos novos como a funcionalidade **Lambda**, deixando assim o *framework* ainda mais elegante.

Para exemplificar a simplicidade de um serviço REST () usando o Spark, o próprio site do Spark apresenta o serviço HelloWorld, respondendo para o mapeamento ”/hello”através do método HTTP GET como mostra a figura 10. Para executar o serviço,

ou seja, colocá-lo no ar ou disponibilizá-lo para consumo, basta executar a classe como um programa Java, pois o Spark possui o servidor de aplicação Jetty embutido.

Figura 10. IMAGEM DO SERVIÇO HELLOWORLD

Com o serviço pronto, o acesso já pode ser realizado através de uma requisição HTTP. No exemplo apresentado no site, a URL do serviço é **http://localhost:4567/hello**, onde o *host* é **localhost** (servidor local) e a porta é **4567**, a porta padrão do Spark que pode ser configurada. Realizando a chamada do serviço no navegador, obtemos o resultado apresentado na figura 11.

Figura 11. SERVIÇO EXECUTADO NO NAVEGADOR

Fora a facilidade do desenvolvimento com Spark, a performance também é um ponto positivo do framework, mesmo rodando sobre a JVM (Java Virtual Machine). Foi colocado a prova e comparado com outras tecnologias e o resultado foi útimo [Rahnema 2014]. Se você quiser saber mais sobre este projeto, acesse o site http://www.sparkjava.com.

6. Próximos passos

Login e controle de acesso WebSocket Filas RabbitMQ Hazelcast NoSQL MongoDB

7. Considerações Finais

O objetivo deste trabalho era propor um sistema de informação especialista em estatística com algumas características específicas. Entre elas está a centralização do mecanismo de manipulação de dados e a análise dos mesmos diretamente na interface, trazendo consigo uma abordagem diferente no que diz respeito a filtros e segmentações em cubo de dados OLAP, e o uso de ferramentas e tecnologias modernas em sua arquitetura beneficiando e enriquecendo o seu desenvolvimento.

Acredito ter sido feliz na escolha do tema, tentando trazer algo inovador e agregando novos conhecimentos ao currículo. A criação do Mozaic foi uma experiência muito positiva, seu desenvolvimento trouxe a oportunidade de aprofundar estudos e realizar provas de conceito das ferramentas e tecnologias escolhidas, resultando ainda em uma excelente ferramenta de análise de dados estatísticos.

Tendo em vista o objetivo alcançado, a ideia é aproveitar o Mozaic em situações mais profissionais, onde a análise dos dados seja útil na tomada de decisões servindo como ferramenta essencial para competitividade no mercado.

Em relação aos próximos passos do Mozaic, pretende-se evoluir ainda mais o sistema no front-end, considerando a atualização do cubo de dados em tempo real através do recurso WebSocket presente no HTML 5 e suportado por navegadores modernos. Outro ponto importante a evoluir é o uso de banco de dados na carga do cubo de dados para a interface. Não foi implementado neste momento, mas é interessante que seja possível, de preferência um banco de dados não-relacional.

Mobile

Recomendo a utilização do Mozaic no meio corporativo. Colocando-o em ambiente de produção será importante para sua evoluição, seja aprimorando a arquitetura ou na adaptação a novos modelos de negócio.

Referências

Filho, A., Luna, B. D., Gondim, C., Marques, D., Eis, D., Shiota, E., Keppelen, G., Real, L. C., Gomes, J., Ferraz, R., and Lopes, S. (2014). *Coletânea Front-end: Uma antologia da comunidade front-end brasileira*. Casa do Código.

Flanagan, D. (2011). *JavaScript: The Definitive Guide*. O'Reilly Media, 6th edition.

Francisco, G. (2014). Construindo aplicações rest utilizando spark 2.0. *We have science*.

Oliveira, E. (2013). Aprenda angularjs com estes 5 exemplos práticos. *Javascript Brasil*.

Osmani, A. (2012). *Learning JavaScript Design Patterns*. O'Reilly Media.

Rahnema, B. (2014). Express vs flask vs go vs sparkjava. *Medium*.

Schmitz, D. and Lira, D. (2013). *AngularJS na prática*. Leanpub.

Silveira, P., Silveira, G., Lopes, S., Moreira, G., Steppat, N., and Kung, F. (2012). *Introdução à Arquitetura e Design de Software: Uma visão sobre a plataforma Java*. Elsevier.