Implementando um sistema de informação para análise de dados 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. *Pendente...*

Resumo. *O presente trabalho relata um estudo de caso referente a implementação de um sistema de informação para análise de dados, tendo como objetivo demonstrar a realização de análises sobre cubos de dados multi-dimensionais no lado cliente. O estudo de caso abrange também uma avaliação da arquitetura proposta para o desenvolvimento deste sistema de informação que, focado no front-end, envolve o conceito SPA (Single-Page Application) e o modelo MVC (Model-View-Controller) client-sidea, além de serviços REST (REpresentational State Transfer). No escopo do artigo consta o detalhamento das características e diferenciais deste sistema de informação e sua arquitetura, bem como a forma de manipulação sobre cubo de dados.*

1. Introdução

Este artigo irá apresentar a implementação de um sistema de informação para 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 sobre cubo de dados diretamente no lado cliente, evitando requisições HTTP (HyperText Transfer Protocol) adicionais, trocas de mensagens ou consumo de serviços *web* a cada interação do usuário ao aplicar filtros e segmentações.

A escolha do tema justifica-se pela importância da análise de dados para tomada de decisão aliada a competitividade no mercado corporativo e também a relevância do uso de uma arquitetura de *software* moderna, que acompanhe a evolução da Internet e linguagens de programação.

Em relação a análise de dados, percebemos que atualmente muitas empresas estão investindo em soluções de apoio de decisão como BI (Business Intelligence) e BigData, entendendo que a análise do seu próprio negócio e o levantamento de indicadores é importante para alcançar melhores 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. Além disso,

o volume de pessoas com acesso a Internet e a milhares de aplicativos estimula a competitividade, fazendo com que a arquitetura tenha também a preocupação com UX (User eXperience), sem esquecer necessidades óbvias como desempenho e escalabilidade.

Para servir de modelo da implementação de um sistema de informação para 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 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. OLAP

Pendente... Apresentação

Pendente... Tempo

Pendente... Dimensões

Pendente... Medidas

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 1. 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 1. 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 2.

Figura 2. Exemplo de estrutura de dados no formato JSON

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 (descrito na seção 2.4), em substituição ao formato XML (eXtensible Markup Language) que é mais verboso como se pode perceber no comparativo representado na figura 3.

Figura 3. Comparativo entre os formatos JSON e XML

2.4. REST

Pendente...

2.5. 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 4.

Figura 4. 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].

2.6. MVC client-side

MVC client-side nada mais é do que o modelo MVC descrito na seção 2.5 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).

Pendente... SPA (Single-Page Application)

Pendente... Vantagens

2.6.1. MVP

Pendente... MVP (Model-View-Presenter).

2.6.2. MVVM

Pendente... MVVM (Model-View-ViewModel)

Pendente... Binding

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 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, 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].

Pendente... link com MVVM / MVW / MV*

Pendente... Controladores

Pendente... Modelos

Pendente... Escopo

Pendente... Diretivas

Pendente... Filtros

3.3. 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 7. 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 7. 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 8.

Figura 8. 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.

4. Mozaic

O sistema de informação implementado para o estudo de caso deste artigo chama-se Mozaic. O Mozaic é uma aplicação *web* no conceito de página única (*single-page*) que serve para auxiliar a análise de dados estatísticos, manipulando dados em tempo real diretamente no *front-end*. Tem como característica possibilitar esta análise sobre cubos de dados multi-dimensionais (OLAP), portanto através de filtros e segmentações o Mozaic apresenta os dados para análise através de *dashboards*.

A arquitetura do Mozaic é moderna, sendo desenvolvida no modelo MVC *client-side*, mais precisamente o modelo MVVM com o uso do *framework* AngularJS no *front-end* da aplicação. O Mozaic é apoiado por uma API (Application Programming Interface) de serviços REST escritos na linguagem Java, onde para o presente estudo de caso artigo foi criado apenas um serviço chamado **carregarDados**. Este serviço, publicado com o *micro-framework* Spark, suporta o método HTTP (Hypertext Transfer Protocol) **GET** sem esperar parâmetros e é responsável por carregar os dados iniciais necessários para a montagem do cubo de dados multi-dimensional. A figura 9 apresenta a arquitetura do Mozaic e a forma de integração com a API de serviços.

Figura 9. Desenho da arquitetura proposta, utilizada pelo Mozaic

Uma vez gerado o cubo de dados, o Mozaic consegue executar as funções de estatística do HyperCube. É desta forma que ele extrai os indicadores que são apresentados nos *dashboards* da aplicação, além da montagem dinâmica dos filtros que permitem a interação com o usuário.

Para servir de exemplo da análise de dados usando sistema Mozaic, foi escolhida uma base de dados de histórico de copas do mundo de futebol, fornecida pelo site da FIFA (Fédération Internationale de Football Association) como mostra a figura 10. O Mozaic trabalha com os dados no lado cliente, para isso precisa carregá-los no JavaScript. Inicialmente foi pensado em colocar os dados em um arquivo JavaScript externo e importá-lo na página HTML, porém para enriquecer a arquitetura uma API de serviços REST foi criada, onde o serviço **carregarDados** foi disponibilizado e através dele os dados são carregados no lado cliente, como um objeto JavaScript to tipo *array*, no formato JSON.

Figura 10. Estatísticas de copas do mundo disponibilizados no site da FIFA

Atualmente se optou por trabalhar com um universo de dados de exemplo fixo (*hard-coded*). Neste cenário, o Mozaic consume o serviço REST, publicado com o Spark, no JavaScript (via AJAX) para carregar os dados estatísticos no lado cliente no formato JSON. As figuras 11 e 12 mostram respectivamente o retorno do serviço **carregarDados** diretamente no navegador e o código JavaScript consumindo o serviço.

Figura 11. Retorno do serviço no navegador

Figura 12. Consumo do serviço REST no JavaScript via AJAX

Como o escopo do artigo é a manipulação dos dados diretamente no *front-end* sobre o cubo de dados gerado pelo HyperCube, optou-se por utilizar os dados fixos. Mas a carga de dados poderia ser feita de diversas maneiras. Os dados poderiam ser obtidos através de um DBMS (DataBase Management System), em português SGBD (Sistema de Gerenciamento de Banco de Dados), no tradicional banco de dados relacional ou em banco de dados NoSQL (Not Only SQL) que é um banco de dados não-relacional, mais flexível e de alta performance. Em uma abordagem mais profissional, é interessante trabalhar com os dados de um DW (Data Warehouse), onde o modelo de dados é desnormalizado, normalmente os dados já foram transformados, agrupados e agregados por uma ferramenta de ETL (Extract, Transform, Load) deixando-os prontos para serem apresentados.

4.1. Dashboards

Pendente... Apresentar a interface do Mozaic

4.1.1. Histórico

Pendente... Descrever interface 13

Figura 13. Tela de histórico do Mozaic com filtros e dashboards

Pendente... Filtros

Pendente... Tela 14

4.1.2. Ranking

Pendente... Dados

Pendente... Tela 15

Figura 14. Tela de histórico do Mozaic apresentando dashboards com os filtros aplicados

Figura 15. Tela de ranking do Mozaic

4.1.3. País

Pendente... Filtros

- Pendente... Dados
- Pendente... Tela 16

Figura 16. Tela do Mozaic para análise especifica de um País

4.1.4. Confronto direto

Pendente... Filtros

- Pendente... Dados
- Pendente... Tela 17

5. Próximos passos

Pendente... Login e controle de acesso

- Pendente... WebSocket

Figura 17. Tela do Mozaic para análise de um confronto entre Países

Pendente... Filas Rabbit

Pendente... Hazelcast

Pendente... NoSQL

Pendente... MongoDB

6. Considerações Finais

Pendente...

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*.

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.