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

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. Já no tocante a arquitetura de *software*, o surgimento de novos dispositivos móveis como tablets

e a modernização de celulares trazem uma mudança comportamental e também cultural que exige a evolução dos aplicativos. Com isso a arquitetura de *software* precisa se desenvolver, por exemplo trabalhando com computação na nuvem *cloud computing* na preocupação com mobilidade e trabalhando com usabilidade na interação com o usuário se preocupando com UX (User eXperience), a experiência do usuário.

responsividade frameworks JavaScript MVC

Por outro lado a tecnologia segue em constante evolução, fazendo com que tenhamos que acompanhar as inovações tecnológicas, enfrentar novos paradigmas e nos adaptarmos a realidade. Isso vale para a arquitetura de *software*. As páginas de Internet são codificadas em HTML (HyperText Markup Language), que é uma linguagem de marcação. Com o crescimento da Internet surgiram tecnologias para a geração de páginas dinâmicas e a arquitetura de aplicações web tinham linguagens de programação onde o conteúdo das páginas era gerado no lado do servidor (*server-side*), como Java Servlets e JSP (Java Server Pages), montando a saída no formato de página HTML. A grande evolução no desenvolvimento *web* veio com a recomendação do uso do padrão arquitetural MVC (Model-View-Controller) [**?**]. A arquitetura de *sotfware* 1

O JavaScript evolui de forma significativa com a criação de novos frameworks, tornando-se, a cada dia, mais poderoso e utilizado entre os desenvolvedores 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].

aplicações *web* evoluíram para um modelo conhecido como MVC (Model-View-Controller) separando a arquitetura em três camadas bem definidas como mostra a figura 1. Cada componente com sua responsabilidade. Model é o componente responsável pela representação do modelo de dados e mecanismo de persistência, View é o componente de apresentação e interação com o usuário e Controller é o componente intermediário entre os anteriores, sendo responsável por receber e responder eventos e ações entre Model e View. Ainda assim, as linguagens de programação para a Internet consideradas ”ricas”, continuavam sendo executadas no lado servidor

Figura 1. IMAGEM do MAPA JSON

Atualmente, variações deste modelo de arquitetura vem surgindo e novas siglas como MVP, MVVM e MV* precisam ser compreendidas.

A diferença básica é , pois Model e View são

A arquitetura MVC client-side se diferencia do modelo tradicional devido ao local onde são executados os componentes View e Controller oriundos da sigla MVC [SIGLA],

que no caso são executados no lado cliente (client-side), mais especificamente no navegador (browser) e não no lado servidor (*server-side*).

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:

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.

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.

Redução de carga no lado servidor

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.

Facilidade de manutenção

Como aplicações client-side dependem apenas da API fornecida pelo servidor, as manutenções no lado servidor podem ser feitas de forma independente e transparente para o lado cliente (desde que n?o mude a API, obviamente). Da mesma forma, o lado cliente pode ser alterado sem a necessidade de alterar nada no lado servidor.

Gerenciamento de equipes

Como o lado cliente e o lado servidor passam a ser desenvolvidos de forma completamente independente, a ?nica coisa necess?ria para os dois times trabalharem em paralelo é a definição da estrutura da API. Com esta definição pronta ambos os times podem trabalhar em paralelo onde o time front-ent desenvolve toda a parte client-side utilizando uma API com dados fict?cios enquanto o time de back-end desenvolve a parte servidora se preocupando apenas em respeitar a especificação da API acordada entre as equipes.

Facilidade de inclusão de novos front-ends

Como o lado servidor fornece apenas a API, torna-se muito mais f?cil desenvolver novos front-ends para dispositivos espec?ficos como uma aplicação nativa para iOS ou Android, por exemplo.

Os frameworks MVC client-side substituem totalmente os frameworks MVC

server-side?

N?o. Os frameworks server-side passam a fornecer apenas uma API, normalmente no formato REST, para ser consumida pela aplicação MVC no lado cliente. Toda a parte de controle de rotas, renderização de templates e validação ficam no lado cliente. A parte servidora fica respons?vel pelas validações (novamente? sim, as validações devem ocorrer tanto no lado cliente quanto no lado servidor) e por armazenar e recuperar os dados em um banco de dados.

Frameworks

Existem diversos frameworks MVC client-side mas alguns que vem ganhando mais destaque ultimamente são:

Backbone.js Ember.js Angular.js

Alguns outros frameworks que possuem a parte MVC e mais alguns componentes gr?ficos.

Sencha Touch ExtJS

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.

2. Ferramentas

2.1. 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.

2.1.1. OLAP

OLAP (On-Line Analytical Processing)

Dimensões .
Medidas .

2.1.2. Cubo de dados

A estrutura de dados interpretada pelo HyperCube é um mapa no formato JSON (JavaScript Object Notation), onde seus registros contém tempo, fatos e medidas.

Figura 2. IMAGEM do MAPA JSON

Esta estrutura de dados é convertida em um cubo de dados pesquisável, ou seja, apto a ser filtrado e consumido por funções matemáticas.

Figura 3. IMAGEM do RESULTADO NO CONSOLE JAVASCRIPT in Section 2.1.

2.1.3. 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

2.2. AngularJS

. MVC client-site. Dentro deste conceito, está o framework AngularJS, da empresa Google. [Oliveira 2013] . Falar sobre REST Falar sobre MVVM

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

. https://angularjs.org/

2.3. 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 4. 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 4. 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 5.

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.

Figura 5. SERVIÇO EXECUTADO NO NAVEGADOR

3. Mozaic

3.1. Apresentação

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.

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 [REF], 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.

3.2. Dados

Para servir de exemplo da implementação do sistema, foi escolhida uma base de dados de histórico de copas do mundo, fornecida pelo site da FIFA (Fédération Internationale de Football Association). Esta base de dados é usada para a montagem do objeto de dados mapeado no formato JSON [SIGLA] como mostra a figura 6, que é a estrutura conhecida para que o HyperCube faça a geração do cubo de dados multi-dimensional para aplicar suas funções estatísticas.

Figura 6. IMAGEM DO JSONEDITOR ONLINE]

3.3. Arquitetura

O Mozaic foi desenvolvido usando o conceito *single-page application*, usando o framework AngularJS da Google no *front-end* da aplicação. A aplicação consume uma API de serviços REST [SIGLA] escritos 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

3.3.1. 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 8. 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 8. 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

3.4. 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)

4. 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.

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*.

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