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. *The present paper reports a case study of the implementation of an information system for data analysis, aiming to demonstrate the analysis of multidimensional data cubes on the client side. This case study also includes an evaluation of the proposed architecture for the development of information system, focused on the front end, involves the concept SPA (Single-Page Application) and the MVC (Model-View-Controller) client-side. In Article scope are detailed in the characteristics and differences of this information and its system architecture, as well as the form of manipulation of data cube.*

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 multidimensionais no lado cliente. Este estudo de caso também abrange 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-side. No escopo do artigo serão detalhadas 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 **Statz**. Antes de apresentar a arquitetura proposta e a implementação do Statz, descritas na seção 5.1, 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 Statz.

2. Referencial teórico

Nesta seção estão descritos os conceitos e as tecnologias envolvidas no desenvolvimento do Statz e na arquitetura proposta. Com isso, ficará mais fácil compreender a proposta deste artigo e a relação dos conceitos com a arquitetura do Statz.

2.1. OLAP

Quando falamos de análise de dados, um leque de opções e níveis de análise podem aparecer. Em um contexto mais analítico e interativo, voltado para tomada de decisão rápida sobre grandes volumes de dados, o cenário se estreita a análises sobre cubo de dados multidimensional.

Nesse sentido, entra a tecnologia OLAP (On-Line Analytical Processing) que significa processamento analítico online. OLAP nada mais é do que uma tecnologia de banco de dados otimizado na forma de cubo de dados multidimensional, diferente de um banco de dados relacional baseado em linhas e colunas. Os dados são visualizados sob diversos ângulos chamados de dimensões, onde uma dimensão que sempre existe é o tempo.

2.2. Conceitos de cubo de dados

Um cubo de dados é uma estrutura de dados que combina dados em várias dimensões (multidimensional), uma hierarquia em vários níveis de granularidades como demonstrado na figura 1.

Normalmente possui dados históricos agregados e sumarizados, facilitando o acesso as informações. É chamado de cubo devido ao seu conceito complexo, sem relação com o sentido matemático onde cubo é uma geometria de lados iguais.

Um modelo de dados multidimensional normalmente é representado no formato de estrela (Star Schema), composto por tabelas de fato e dimensões como mostra a figura 2.

É importante destacar alguns conceitos de cubo de dados. Entre eles destaco fato, medida e dimensão. Fato representa uma coleção de dados relacionados que podem servir

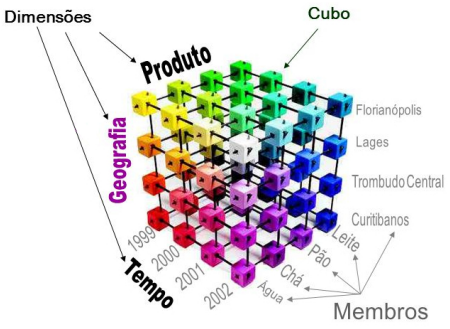

Figura 1. Representação de um cubo de dados multidimensional

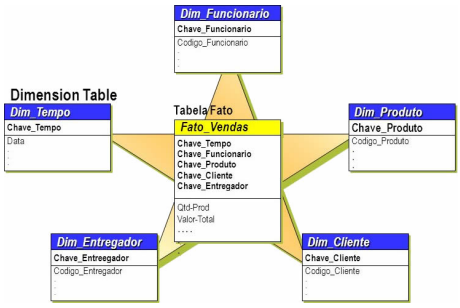

Figura 2. Exemplo de um modelo de dados multidimensional no formato de estrela

a análise. Uma tabela fato possui os dados agregados baseados em uma entidade e seus relacionamentos.

2.2.1. Medida

Medidas correspondem a atributos de um fato cubo de dados. Normalmente que representam indicadores, pois são dados quantitativos pré-calculados.

2.2.2. Dimensão

As dimensões de um cubo de dados representam visões ou contextos diferentes sobre um fato. É como se fosse uma coordenada de análise sobre o cubo de dados multidimensional. Existe uma dimensão especial relativa ao tempo (temporal) que caracteriza uma data/hora. Para exemplificar, dimensões para uma medida ”Total de vendas”poderiam ser ”cliente”, ”cidade”, ”data”, ”produto”e etc, que nada mais são do que visões diferentes sobre uma determinada medida.

2.2.3. Membro

Membros correspondem a elementos usados para determinar o dado de uma dimensão. Por exemplo ”mês Fevereiro”e ”ano 2014”são membros de uma dimensão temporal. Em algumas situações, podem existir membros calculados.

2.2.4. Hierarquia

Este componente apenas organiza o cubo de dados, definindo níveis de agregação e granularidade do cubo, relacionando membros a dimensões.

2.3. Operações sobre cubo de dados

Abaixo estão detalhadas algumas operações básicas realizadas sobre cubos de dados multidimensionais.

2.3.1. Drill down

Drill down é a operação utilizada para detalhar mais o dado. Segundo [Turban et al. 2009], é a capacidade de chegar a mais detalhes em um ou vários níveis.

2.3.2. Roll up

Roll up representa a operação utilizada para abstrair mais os dados, reduzindo o nível de detalhamento atual. O funcionamento é exatamente contrário a operação anterior. Quando maior o nível de granularidade, menor o nível de detalhamento.

2.3.3. Slice

A operação utilizada para a redução do universo de dados do cubo chama-se *slice*. Na prática, filtra ou fatia o cubo de dados multidimensional.

2.3.4. Dice

Outra operação que reduz o universo de dados do cubo é a *dice*, porém esta operação descarta parte do cubo de dados.

2.4. 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 3. 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 compatível com a maioria dos navegadores e respectivas versões.

```
<!doctype html>
<html>
<head>
    <title>STATZ</title>
</head>

<body>

    <script src="statz.js"></script>

</body>
</html>
```

Figura 3. Exemplo de inclusão de uma biblioteca JavaScript em uma página HTML

2.5. 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 4.

```
{
  "array": [
    1,
    2,
    3
  ],
  "boolean": true,
  "null": null,
  "number": 123,
  "object": {
    "a": "b",
    "c": "d",
    "e": "f"
  },
  "string": "Hello World"
}
```

Figura 4. 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 (REpresentational State Transfer), em substituição ao formato XML (eXtensible Markup Language) que é mais verboso como se pode perceber no comparativo representado na figura 5.

JSON

```
{"widget": {
    "debug": "on",
    "window": {
        "title": "Sample Widget",
        "name": "main_window",
        "width": 500,
        "height": 500
    },
    "image": {
        "src": "Images/Sun.png",
        "name": "sun1",
        "hOffset": 250,
        "vOffset": 250,
        "alignment": "center"
    },
    "text": {
        "data": "Click Here",
        "size": 36,
        "name": "text1",
        "hOffset": 250,
        "vOffset": 100,
        "alignment": "center",
    }
}}
```

XML

```
<widget>
    <debug>on</debug>
    <window title="Sample Widget">
        <name>main_window</name>
        <width>500</width>
        <height>500</height>
    </window>
    <image src="Images/Sun.png" name="sun1">
        <hOffset>250</hOffset>
        <vOffset>250</vOffset>
        <alignment>center</alignment>
    </image>
    <text data="Click Here" size="36">
        <name>text1</name>
        <hOffset>250</hOffset>
        <vOffset>100</vOffset>
        <alignment>center</alignment>
    </text>
</widget>
```

Figura 5. Comparativo entre os formatos JSON e XML

2.6. 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 6.

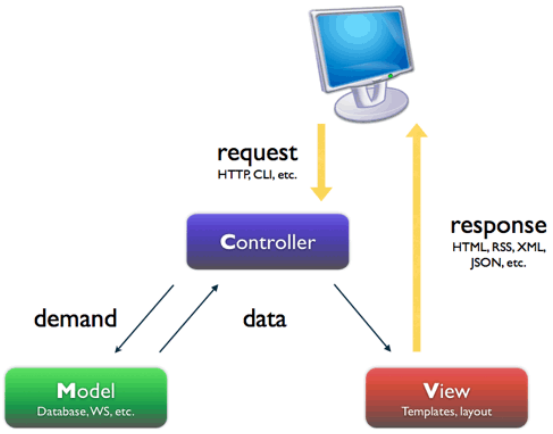

Figura 6. Desenho do modelo MVC e a interação entre as camadas

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].

2.7. MVC client-side

MVC client-side nada mais é do que o modelo MVC descrito na seção 2.6 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).

2.8. Single-Page Application

Single-Page Application (SPA) é uma abordagem atual utilizada em páginas de Internet consideradas ”ricas”. Como o próprio nome diz, significa ter a aplicação em uma única página tendo o seu conteúdo é carregado dinamicamente. Essa abordagem traz uma melhor experiência ao usuário, a medida que evita a troca ou refresh de página, e além disso evita a carga completa da página, o que pode também ser visto como uma melhora de desempenho.

2.9. MVP

MVP (Model-View-Presenter) possui um mecanismo derivado do padrão MVC, diferenciando-se apenas no conceito. Neste padrão, Presenter é a camada intermediária com o conhecimento das extremidades View e Model, tendo função semelhante ao Controller do padrão MVC. A camada Presenter é encarregada de atualizar a camada View quando a camada Model sofre alterações e também encarregada de sincronizar a camada Model em relação a View.

2.10. MVVM

Um novo padrão de *design* de *software* é o MVVM. Significa Model-View-ViewModel e é uma variação dos padrões MVC e MVP vistos anteriormente. A novidade aqui é a camada ViewModel, que surge com a responsabilidade de disponibilizar para a camada View uma lógica de apresentação. Além disso, coordena interações entre as camadas Model e View sem ter nenhum conhecimento sobre a View. Outra vantagem é a sua testabilidade, uma vez que independe de Model e View.

3. Status quo

A ideia desta seção é comentar a respeito de ferramentas existentes que oferecem soluções semelhantes ou que tenham relação com o presente artigo e o sistema de informação implementado.

3.0.1. API olap4j

O olap4j é uma API escrita em Java usada para o acesso a dados de cubos multidimensionais. Esta API é uma especialização das especificações 3 e 4 da API JDBC (Java Database Connectivity), portanto o olap4j é conhecido como o JDBC para OLAP.

3.0.2. Mondrian

Mondrian é um motor de análise OLAP escrito em Java. Com ele é possível construir soluções de BI, utilizando seu motor para análises sobre cubo de dados multidimensionais. É parte do pacote Pentaho BI, uma solução de BI bastante conhecida.

Ao trabalhar com bancos de dados relacionais, a linguagem padrão para consultas é a SQL (Structured Query Language). O Mondrian realiza consultas sobre cubo de dados usando MDX (MultiDimensional eXpressions), que é considerado o SQL para OLAP.

3.0.3. Saiku

O Saiku é uma ferramenta para análise dinâmica de dados em cubos de dados multimensionais. Com essa ferramenta é possível cruzar informações de dimensões e medidas e visualizar os resultados em tabelas e gráficos. Tudo de forma fácil e prática, arrastando elementos (*"drag and drop"*) para a análise no lugar desejado, como linha e coluna. Ainda é possível aplicar filtros sobre a análise realizada.

3.0.4. HyperCube

HyperCube é uma biblioteca escrita em JavaScript que proporciona a criação de cubos de dados multidimensional, permite a aplicação de filtros e agregações, gerando dados estatísticos 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 7. 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 8 mostra o trecho de código JavaScript que realiza a geração do cubo de dados.

3.0.5. 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.

```
var exemplo = [
    {
        "facts": {"ano": 2010, "pais": "Argentina", "continente": "América"},
        "measures": {
            "jogos": 5, "vitorias": 4, "empates": 0, "derrotas": 1
        }
    },
    {
        "facts": {"ano": 2006, "pais": "Argentina", "continente": "América"},
        "measures": {
            "jogos": 5, "vitorias": 3, "empates": 2, "derrotas": 0
        }
    },
    {
        "facts": {"ano": 2010, "pais": "Brasil", "continente": "América"},
        "measures": {
            "jogos": 5, "vitorias": 3, "empates": 1, "derrotas": 1
        }
    },
    {
        "facts": {"ano": 2006, "pais": "Brasil", "continente": "América"},
        "measures": {
            "jogos": 5, "vitorias": 4, "empates": 0, "derrotas": 1
        }
    }
];
```

Figura 7. Exemplo de um mapa de dados no formato JSON no formato compreendido pelo HyperCube

```
var medidas = ['jogos', 'vitorias', 'empates', 'derrotas'];
var cubo = ps.Cube.deserialize(examplo, medidas);
```

Figura 8. Código JavaScript responsável pela transformação do mapa em um cubo de dados do HyperCube

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

3.1. Frameworks MVC client-side

Aplicações *client-side* trazem a experiência de aplicações *desktop* para aplicações *web*. Atualmente existem diversos *frameworks* MVC *client-side*. A maioria deles visam melhorar a estrutura de aplicações *web*, envolvendo recursos de MVC, rotas e mecanismos de *templating*. Cada um tem suas próprias particularidades, diferenciando principalmente em relação a camada de apresentação (View). Em seguida, serão apresentados alguns dos populares *frameworks* MVC *client-side*.

3.1.1. Ember.js

Um dos *frameworks* MVC *client-side* populares para se criar aplicações de página única (SPA) é o Ember.js, já utilizado por empresas como Yahoo! e Groupon. O Ember.js segue

o conceito de CoC (Convention over Configuration), onde o desenvolvedor configura a aplicação por meio de convenções, podendo focar no que realmente importa.

Para a criação dos *templates*, o Ember.js usa por padrão a biblioteca JavaScript Handlebars, mas permite o uso de outras formas de *templating*. O fator desempenho tem sido um dos alvos do Ember.js. Mecanismos de cache e pré-compilação de *templates*, ajudam a aplicação a carregar e rodar com boa performance.

3.1.2. Backbone.js

Criado em 2010, o Backbone.js é um *framework* MVC *client-side* muito leve, utilizado por serviços como Pinterest, AirBNB e Flixster. Possui código simples e é fortemente documentado.

Entre as suas principais características, destaco a sua flexibilidade e facilidade de extensão, permitindo que o desenvolvedor crie seu próprio *framework*, baseado no Backbone.js. Sua curva de aprendizado para usar o Backbone.js pode ser menor, caso o desenvolvedor já tenha experiências com o *framework* Underscore.js e jQuery.

3.1.3. AngularJS

AngularJS é um *framework* MVC *client-side* criado em 2009 de bastante destaque sendo atualmente patrocinado pela empresa Google. Vem trazendo conceitos inovadores para os desenvolvedores de aplicações *web*, por exemplo o conceito *Two-way data binding* que mantém informações sincronizadas entre as camadas Model e View. Uma vez que um dado é alterado na página, o mesmo é refletido no modelo. Da mesma forma, uma vez que o dado é modificado no modelo, se reflate automaticamente na página. Em seu site oficial, é mencionado como *"Superheroic JavaScript MVW Framework"*, devido ao seu poder e abrangência dos paradigmas MVC e MVVM.

O 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]. Uma das suas características é sua arquitetura de *building blocks*, que mantém categorias distintas de código como diretivas e filtros, além dos tradicionais *controllers* e *views*.

4. Tomada de decisão

Esta seção aborda as escolhas realizadas para o desenvolvimento do projeto deste artigo, mencionando as justificativas das decisões após avaliações e algumas comparações entre as ferramentas e *frameworks* relacionados.

4.1. Front-end da aplicação

Considerando que a solução proposta é uma aplicação *web* codificada em HTML, com mecanismo JavaScript envolvendo modelo de dados no formato JSON, após uma avaliação de ferramentas de mercado, optou-se pela utilização do *framework* AngularJS.

Entre os fatores determinantes para essa decisão estão o mecanismo próprio (*engine templating*), o tamanho e dependências do AngularJS e o paradigma SPA. Além

disso, o fato do *framework* ter sido desenvolvido pela Google e experiências positivas do autor utilizando o mesmo contribuíram para a escolha, fazendo com que o *framework* Durandal fosse descartado na comparação por ser bastante semelhante e do mesmo nível do AngularJS em relação a MVVM (*data binding*) e SPA.

A codificação utilizando o AngularJS é feita através de *tags* e diretivas como se fosse uma extensão do próprio HTML, enquanto a codificação utilizando os demais lembram mais outras linguagens *server-side* como ASP e Velocity do que HTML, uma vez que o Backbone.js é integrado com Underscore.js e o Ember.js possui dependência do Handlebars (que já é uma extensão do Mustache). Particularmente, alguns trechos de código do Backbone.js e do Ember.js me lembram respectivamente sintaxes das linguagens ASP e Velocity.

4.2. Análise sobre o cubo de dados

Uma vez que a proposta deste artigo é focada na análise diretamente no lado cliente, a opção foi modelar os dados no formato JSON, não importando a origem da leitura dos mesmos. O importante é poder trabalhar com os dados no JavaScript, ignorando outros fatores como requisições HTTP, consumo de serviços REST e Data Warehouse. Para a montagem do cubo de dados em JavaScript, sem dependência de outras APIs ou servidores de aplicação, a solução encontrada foi utilizar a API do Hypercube. Outras opções seriam mais indicadas para um BI mais robusto ou para trabalhar os dados no lado servidor.

5. Descrição da solução

A solução será descrita nesta seção, apresentando os detalhes da arquitetura e do sistema de informação implementado para estudo de caso deste artigo, que chama-se Statz. Além da teoria sobre análise de dados, serão mostrados detalhes da implementação, bem como o funcionamento das análises e as telas do sistema.

5.1. Statz

O Statz 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 multidimensionais (OLAP), portanto através de filtros e segmentações o Statz apresenta os dados para análise através de *dashboards*.

5.1.1. Apresentação

Esta seção apresenta a interface visual do sistema de informação Statz em sua tela inicial (figura 9), onde é possível conhecer o logo do Statz, posicionado no cabeçalho à esquerda, e os links para as visões distintas de análise localizados na parte superior à direita. As páginas de análise, podem possuir uma área para filtros à esquerda, permitindo que o usuário realize as segmentações desejadas.

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

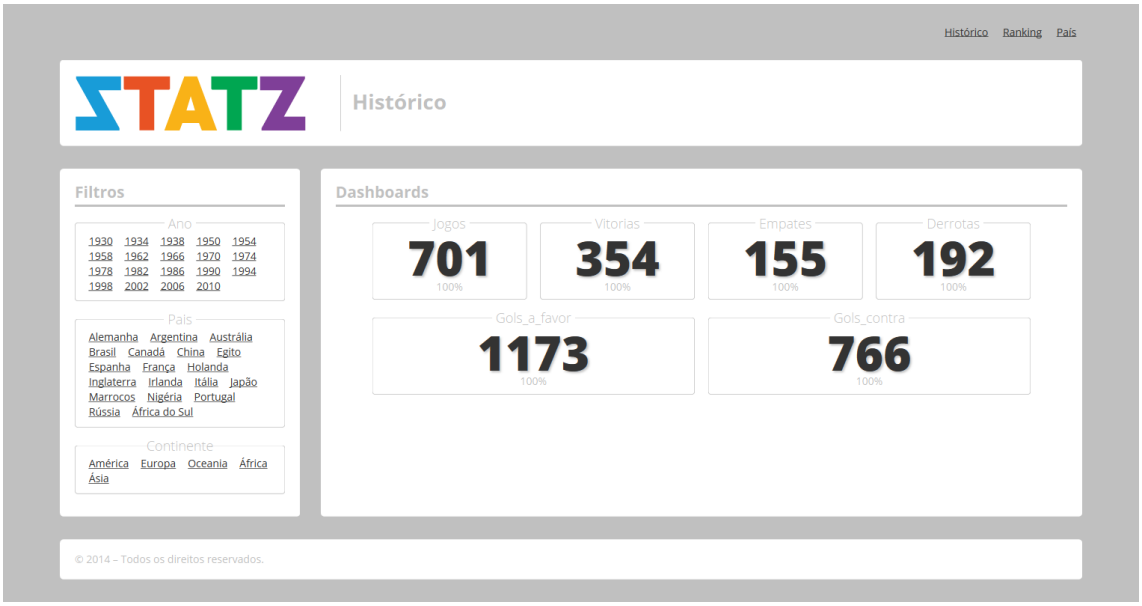

Figura 9. Interface do sistema, mostrando a primeira página de análise

5.1.2. Universo de dados

Para servir de exemplo da análise de dados usando sistema Statz, foi escolhida uma base de dados de histórico de copas do mundo de futebol até 2010, fornecida pelo site da FIFA (Fédération Internationale de Football Association) como mostra a figura 10. O Statz trabalha com os dados no lado cliente, para isso precisa carregá-los como um objeto JavaScript do tipo *array*, no formato JSON.

Copa do Mundo da FIFA™ - Rankings históricos

Rank.	Equipe	PJ	V	E	D	GF	GC	Pts	Méd. Pts.	Apps.
1	Brasil	104	70	17	17	221	102	227	2.18	20
2	Alemanha	106	66	20	20	224	121	218	2.06	18
3	Itália	83	45	21	17	128	77	156	1.88	18
4	Argentina	77	42	14	21	131	84	140	1.82	15
5	Espanha	59	29	12	18	92	66	99	1.68	14
6	Inglaterra	62	26	20	16	79	56	98	1.58	14
7	França	59	28	12	19	106	71	96	1.63	14
8	Holanda	50	27	12	11	86	48	93	1.86	10
9	Uruguai	51	20	12	19	80	71	72	1.41	12
10	Suécia	46	16	13	17	74	69	61	1.33	11
11	Rússia	40	17	8	15	66	47	59	1.48	9
12	Sérvia	43	17	8	18	64	59	59	1.37	11
13	México	53	14	14	25	57	92	56	1.06	15
14	Bélgica	41	14	9	18	52	66	51	1.08	12
15	Polônia	31	15	5	11	44	40	50	1.61	7
16	Hungria	32	15	3	14	87	57	48	1.50	9
17	Portugal	26	13	4	9	43	29	43	1.65	6
18	República Tcheca	33	12	5	16	47	49	41	1.24	9
19	Áustria	29	12	4	13	43	47	40	1.38	7
20	Chile	33	11	7	15	40	49	40	1.24	9

Brasil

Edição	J	V	E	D	GF	GC	MGF	MGC
Copa do Mundo da FIFA África do Sul 2010	5	3	1	1	9	4	1.8	0.8
Copa do Mundo da FIFA Alemanha 2006	5	4	0	1	10	2	2.0	0.4
Copa do Mundo da FIFA Coreia do Sul/Japão 2002	7	7	0	0	18	4	2.6	0.6
Copa do Mundo da FIFA França 1998	7	4	1	2	14	10	2.0	1.4
Copa do Mundo da FIFA EUA 1994	7	5	2	0	11	3	1.6	0.4
Copa do Mundo da FIFA Itália 1990	4	3	0	1	4	2	1.0	0.5
Copa do Mundo da FIFA México 1986	5	4	1	0	10	1	2.0	0.2
Copa do Mundo da FIFA Espanha 1982	5	4	0	1	15	6	3.0	1.2
Copa do Mundo da FIFA Argentina 1978	7	4	3	0	10	3	1.4	0.4
Copa do Mundo da FIFA Alemanha 1974	7	3	2	2	6	4	0.9	0.6
Copa do Mundo da FIFA México 1970	6	6	0	0	19	7	3.2	1.2
Copa do Mundo da FIFA Inglaterra 1966	3	1	0	2	4	6	1.3	2.0
Copa do Mundo da FIFA Chile 1962	6	5	1	0	14	5	2.3	0.8
Copa do Mundo da FIFA Suécia 1958	6	5	1	0	16	4	2.7	0.7
Copa do Mundo da FIFA Suíça 1954	3	1	1	1	8	5	2.7	1.7
Copa do Mundo da FIFA Brasil 1950	6	4	1	1	22	6	3.7	1.0
Copa do Mundo da FIFA França 1938	5	3	1	1	14	11	2.8	2.2
Copa do Mundo da FIFA Itália 1934	1	0	0	1	1	3	1.0	3.0
Copa do Mundo da FIFA Uruguai 1930	2	1	0	1	5	2	2.5	1.0

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

5.1.3. Arquitetura

A arquitetura do Statz 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. Como o foco da solução é a manipulação dos dados diretamente no *front-end*,

optou-se por trabalhar com dados de exemplo fixos (*hard-coded*), não desviando a atenção do escopo do presente artigo. Dessa forma, a aplicação não precisa buscar os dados, eles já estão presentes em um objeto JavaScript no formato JSON e darão origem ao cubo de dados gerado através da API do HyperCube. Para que isso ocorra, é necessário que o objeto JavaScript esteja em uma estrutura de dados predefinida contendo o momento do dado, fatos e medidas conforme é demonstrado na figura 11. Só assim, o HyperCube consegue interpretar e deserializar o objeto, montando o cubo de dados e disponibilizando o mesmo para ser filtrado e calculado também em JavaScript.

```
var copaParcicipacoes = ps.obj();
copaParcicipacoes.data = [
{
    "facts": {"ano": 2010, "pais": "África do Sul", "continente": "África"},
    "measures": {"jogos": 3, "vitorias": 1, "empates": 1, "derrotas": 1, "gols_a_favor": 3, "gols_contra": 5}
},
{
    "facts": {"ano": 2002, "pais": "África do Sul", "continente": "África"},
    "measures": {"jogos": 3, "vitorias": 1, "empates": 1, "derrotas": 1, "gols_a_favor": 5, "gols_contra": 5}
},
{
    "facts": {"ano": 1998, "pais": "África do Sul", "continente": "África"},
    "measures": {"jogos": 3, "vitorias": 0, "empates": 2, "derrotas": 1, "gols_a_favor": 3, "gols_contra": 6}
},
{
    "facts": {"ano": 2010, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 7, "vitorias": 5, "empates": 0, "derrotas": 2, "gols_a_favor": 16, "gols_contra": 5}
},
{
    "facts": {"ano": 2006, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 7, "vitorias": 5, "empates": 1, "derrotas": 1, "gols_a_favor": 14, "gols_contra": 6}
},
{
    "facts": {"ano": 2002, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 7, "vitorias": 5, "empates": 1, "derrotas": 1, "gols_a_favor": 14, "gols_contra": 3}
},
{
    "facts": {"ano": 1998, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 5, "vitorias": 3, "empates": 1, "derrotas": 1, "gols_a_favor": 8, "gols_contra": 6}
},
{
    "facts": {"ano": 1994, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 5, "vitorias": 3, "empates": 1, "derrotas": 1, "gols_a_favor": 9, "gols_contra": 7}
},
{
    "facts": {"ano": 1990, "pais": "Alemanha", "continente": "Europa"},
    "measures": {"jogos": 7, "vitorias": 5, "empates": 2, "derrotas": 0, "gols_a_favor": 15, "gols_contra": 5}
}
//...
];
```

Figura 11. Dados de copas do mundo obtidas junto a FIFA, no formato JSON

Uma vez gerado o cubo de dados, o Statz consegue aplicar funções de estatística do HyperCube sobre o cubo, extraindo os indicadores que são apresentados nos *dashboards* da aplicação. Além dos indicadores, os valores carregador na montagem dinâmica dos filtros também são extraídos a partir da API do HyperCubo, porém não são operações de estatística.

Embora a principal característica do Statz esteja na sua ideia de arquitetura e mecanismo de análise (seção 5.1.4), a *interface* da aplicação é também um ponto importantíssimo. É no *front-end* que são montados os filtros de análise e *dashboards* com dados e indicadores consolidados, lembrando **Model** e **View** se mantém sincronizados em tempo real, devido ao paradigma MVVM do AngularJS.

5.1.4. Mecanismo de análise

O mecanismo de análise se dá através da aplicação de operações de estatística sobre o cubo de dados extraindo dados para apresentar na tela. Cada página de análise possui um conjunto de análise específica, onde os dados são extraídos de acordo com o contexto da página de análise.

Entenda como conjunto de análise o contexto de dados multidimensional, ou seja, fatos, dimensões e medidas relacionadas com a análise que são segmentadas na tela. As páginas de análise atualmente disponibilizadas são: Histórico, Ranking e País. Estas páginas de análise serão detalhadas a seguir e ficará mais claro seus respectivos contextos de análise.

Ao abrir uma página de análise, por padrão é carregado o seu *dashboard* e seus filtros, caso exista. O *dashboard* nada mais é do que os dados do cubo de dados consolidados em indicadores. Os filtros, são os possíveis valores que podem ser usados para segmentar as medidas. Exemplos de indicadores são: total de vitórias, total de empates e total de derrotas. Exemplos de filtros são: ano, País e continente. Sendo assim, podemos segmentar os indicadores através dos filtros.

Na prática, a visão inicial do indicadores é geral, pois não possuem filtros. Ao aplicar um filtro, os indicadores serão automaticamente atualizados, para considerar o filtro aplicado e assim restringir o universo de dados do cubo de dados da respectiva análise. Limpando os filtros, os indicadores são atualizados novamente, voltando ao estado original.

5.1.5. Análise histórica

A tela inicial do Statz é uma visão histórica das informações de jogos das copas do mundo, onde são apresentados os dados estatísticos calculados a partir do cubo de dados. Nela constam as informações de vitórias, empates, derrotas, gols a favor e gols contra, além do número de jogos.

Estas informações são mapeadas no cubo de dados como **measures** (medidas). Os filtros possíveis, mapeados como **facts** (fatos), são o ano, o país e o continente. Através deles, é possível segmentar a análise sobre as informações do cubo de dados.

A figura 12 apresenta a tela de histórico com alguns filtros selecionados à esquerda (realçados em amarelo) e na área principal mostra os dados já filtrados considerando os respectivos filtros. O resumo do filtro aplicado é: dados de copas do mundo do ano de 2006, do continente americano, mas apenas os Países Argentina e Brasil.

A API do Hypercube possibilita filtrar o cubo de dados com a função **slice** passando um mapa de filtros com os valores de cada ”fato”como ”ano”, ”pais”e ”continente”. Inicialmente, como não há filtros selecionados, nenhuma função é aplicada, fazendo com que seja considerado todo o universo de dados contido no cubo de dados. Se quisermos filtrar por mais de um ano, País ou continente, o resultado será um cubo de dados vazio. A explicação é simples: uma vez que os dados de histórico no detalhe sempre estão relacionados a algum ano específico, apenas de um país, que por sua vez pertence a um determinado continente.

Portanto para agrupar os dados o Statz usa a função **dice**, passando também um mapa de filtros com valores de ”fato”. Esta função gera um novo cubo de dados desconsiderando a presença de tais filtros. Como esta tela já possui quais são os filtros selecionados, o Statz aplica a função **dice** nos valores não selecionados, obtendo o resultado esperado para demonstrar na tela.

Figura 12. Tela de histórico do Statz apresentando dashboards com os filtros aplicados

5.1.6. Análise de ranking

A tela de ranking é mais simples que a anterior. Como se pode ver na figura 13, a mesma não possui filtros e apenas mostra uma relação de Países ordenadas por pontuação, ou seja, em primeiro lugar o País de melhor participação em todas as copas do mundo considerando a pontuação de vitórias e empates, e assim por diante até os Países de pior participação.

O cubo de dados contém no detalhe o histórico de todos os Países em todas as participações. O que o Statz faz para criar o ranking é sumarizar os dados que dão origem a pontuação, dado calculado em tempo de execução considerando as medidas ”vitorias”e ”empates”. Vale ressaltar que a pontuação foi contabilizada considerando 3 (três) pontos por vitória e 1 (um) ponto por empate, ignorando o fato de a FIFA ter alterado a regra pontuação de vitória ao longo do tempo.

Com a pontuação calculada um novo cubo de dados é gerado pelo Statz tendo em sua estrutura o País como fato e a pontuação como medida, o que resulta no mapeamento da pontuação de cada País. Aplicando a função **topSum** da API do Hypercube, passando o fato ”pais”e a medida ”pontos”, temos os Países ordenados pela pontuação gerando assim o ranking apresentado na tela.

5.1.7. Análise específica de um País

Na tela de País é possível realizar uma análise focada em um determinado País, sem a interferência de outros dados relacionados, como mostra a figura 14. Isso possibilita uma

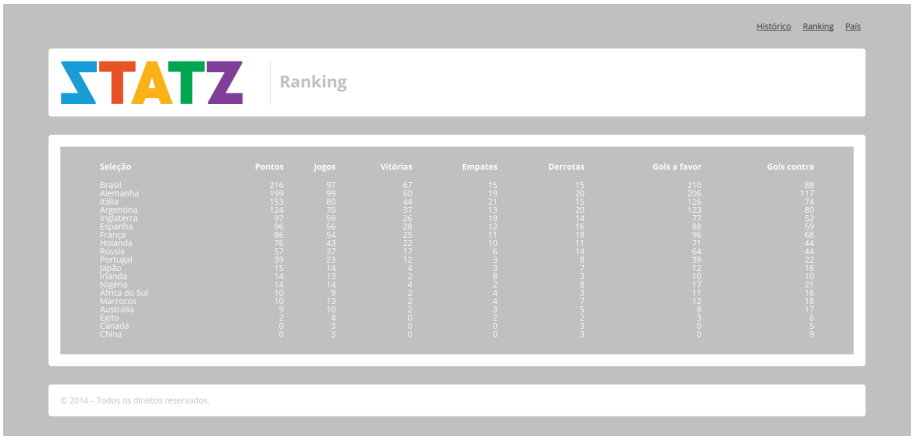

Figura 13. Tela de ranking do Statz

comparação entre um País e outro, sobre a participação dos mesmos nas copas do mundo.

O Statz nesta tela aguarda o usuário realizar a escolha do País, para filtrar o cubo de dados e apresentar as informações do País. Mediante a seleção do País, o sistema aplica a função **slice** da API do Hypercube, passando um mapa de filtros. No caso, apenas o fato ”pais”é passado como filtro. Isso faz com que seja gerado um novo cubo de dados, contendo apenas os dados do País selecionado.

Com os dados do País, o sistema apresenta na área principal os dados do País para que seja realizada a análise. Alterando o filtro para outro País, o processo se repete mostrando novos dados, possibilitando a comparação entre Países.

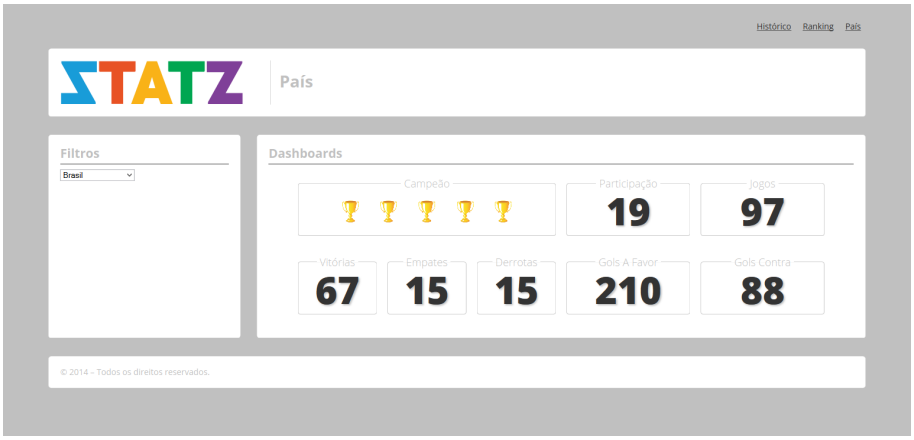

Figura 14. Tela para análise especifica de um País

6. Próximos passos

A solução pode evoluir bastante ainda em alguns aspectos, tanto na parte de negócio com a inclusão de novos contextos de análise quanto na parte técnica em diversos aspectos. Por exemplo, o *front-end* poderia melhorar a identidade visual e incorporar gráficos de vários tipos para melhor análise dos indicadores e com possibilidade de *drill down*, poderia implementar um controle de acesso com login e senha para proteger o acesso aos dados, além de possibilitar exportar os dados da análise para o formato de planilha ou imagem.

Na questão dos dados que atualmente são fixos, seria interessante avaliar o uso de um DW e pensar em automatizar a carga dos mesmos com alguma ferramenta de ETL, adaptando o sistema para um ambiente mais profissional (corporativo) onde a atualização dos dados se faz necessária com periodicidade às vezes até diária. Nesse caso, o arquivo JavaScript seria atualizado diariamente. Lembrando que não as análise são sobre dado histórico (BI/Big Data), não devem ser em tempo real.

Já pensando em carregar os dados em tempo de execução, penso que o melhor caminho seria evoluir para que o sistema consuma um serviço REST através de uma requisição AJAX. Esse serviço seria responsável pela consulta em algum banco de dados e pela carga dos mesmos no formato esperado pelo JavaScript e na estrutura esperada pelo HyperCube.

Sobre o formato dos dados, como a aplicação trabalha no formato JSON e o Hypercube precisa de uma estrutura de dados específica, uma excelente abordagem é utilizar um banco de dados NoSQL, que são bancos de dados orientados a documentos, diferente do modelo tradicional que são bancos de dados relacional.

Existem bancos de dados NoSQL que já trabalham no formato JSON como o MongoDB e CouchDB. Seria possível armazenar nestes bancos de dados documentos já na estrutura de dados esperada pela aplicação para a geração do cubo de dados. Optando pelo uso do CouchDB, abre-se uma oportunidade diferente. Como este possui uma API totalmente baseada no protocolo HTTP, seria possível que o consumo fosse feito diretamente nele, sem a necessidade de outro serviço REST.

Outra abordagem interessante seria manter o dado em tempo real via WebSocket, onde a aplicação se inscreveria em uma fila e receberia eventos que poderiam ser utilizados para atualizar os dashboards das páginas de análise. Como sugestão, a ferramenta RabbitMQ poderia ser utilizada, pois é uma relativamente fácil de usar, além de possuir plugins que facilitam a integração com sistemas.

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 Statz 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 Statz 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 Statz, 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 NoSQL com documentos no formato JSON.

Recomendo a utilização do Statz no meio corporativo. Colocando-o em ambiente de produção será importante para sua evolução, 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.

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

Pires, C. E. Data wareshousing.

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.

Turban, E., Sharda, R., Aronson, J. E., and King, D. (2009). *Business Intelligence: Um enfoque gerencial para a inteligência do negócio*. O'Reilly Media.