

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO
DO RIO GRANDE DO NORTE
ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS CORPORATIVOS**

FRANCISCO GENIVAN SILVA

**DENGUEMAP: SISTEMA WEB PARA MAPEAMENTO DOS
CASOS DE DENGUE NA CIDADE DE NATAL UTILIZANDO
A API GOOGLE MAPS**

FRANCISCO GENIVAN SILVA

**DENGUEMAP: SISTEMA WEB PARA MAPEAMENTO DOS
CASOS DE DENGUE NA CIDADE DE NATAL UTILIZANDO
A API GOOGLE MAPS**

Trabalho de Conclusão de Curso apresentado ao Curso de Especialização em Desenvolvimento de Sistemas Corporativos da Faculdade Natalense para o Desenvolvimento do Rio Grande do Norte como requisito parcial para obtenção do grau de Especialista.

Orientador:

Ricardo Wendell Rodrigues da Silveira, Msc.

**LIGA DE ENSINO DO RIO GRANDE DO NORTE
FACULDADE NATALENSE PARA O DESENVOLVIMENTO DO RIO GRANDE DO
NORTE
CURSO DE ESPECIALIZAÇÃO EM DESENVOLVIMENTO DE SISTEMAS
CORPORATIVOS**

Francisco Genivan Silva

**DENGUEMAP: SISTEMA WEB PARA MAPEAMENTO DOS CASOS
DE DENGUE NA CIDADE DE NATAL UTILIZANDO A API GOOGLE
MAPS**

Trabalho de Conclusão de Curso apresentado e aprovado em 06 de outubro de 2011, pela
banca examinadora composta pelos seguintes membros

Ricardo Wendell Rodrigues da Silveira, Msc.
Orientador

Gleydson de Azevedo Ferreira Lima, Msc.
Membro examinador

David Ricardo do Vale Pereira, Msc.
Membro examinador

Natal/RN, ____ de _____ de ____.

Sumário

1	Introdução.....	6
2	Objetivos	7
2.1	Geral	7
2.2	Específicos.....	8
3	Referencial Teórico	8
3.1	Google Maps – Visão Geral	8
3.2	Coordenadas Geográficas e Geocodificação	10
3.3	Google Maps – Ferramentas.....	12
3.3.1	Marcadores.....	14
3.3.2	Tooltip	15
3.3.3	Infowindow	15
3.4	Arrays de Marcadores	17
3.5	Javaserver Faces	17
3.6	Xstream	18
3.7	Jasperreports.....	19
4	A aplicação.....	21
4.1	Visão geral.....	21
4.2	Ambiente de Desenvolvimento	21
4.3	Implementação	22
4.3.1	Adicionando um caso.....	25
4.3.2	Usando Xml para Inicializar o mapa.....	28
4.3.3	Gerenciando os casos	31
5	Conclusões	32
	Referências	33

Resumo

O presente trabalho propõe o desenvolvimento de um sistema web para mapeamento dos casos de dengue da cidade de Natal. Para alcançar esse objetivo, um *mashup* que utiliza as tecnologias Google Maps JavaScript API e o framework PrimeFaces foi desenvolvido. A aplicação permite, dentre outras atividades, a indicação de ocorrência dos casos de dengue através da inserção de marcadores. Os dados gerados são armazenados em um banco de dados PostgreSQL e poderão ser utilizados para geração de relatórios e gráficos. Com isso, a ferramenta desenvolvida visa permitir um melhor planejamento das ações de combate a epidemia, tornando mais dinâmica a representação, interpretação e exploração de dados gerados.

Palavras – chave: Sistema web. Dengue. Mashup. Google Maps JavaScript API. PrimeFaces.

Abstract

This paper proposes the development of a web system for mapping the incidence of *Dengue Fever* in Natal. To achieve this goal, we developed a mashup that runs by using Google Maps JavaScript API technology and PrimeFaces framework. The application allows the indication of *Dengue Fever* occurrence through the insertion of markers. The data generated is stored in a PostgreSQL database and may be used to generate reports and graphs; therefore the developed tool is designed to allow better planning of actions to fight epidemics, which makes representation, interpretation, and exploitation of data more dynamic.

Keywords: web system. Dengue Fever. Mashup. Google Maps JavaScript API. PrimeFaces.

1 INTRODUÇÃO

A Web vem apresentando ao longo dos últimos anos um notável crescimento da popularidade dos serviços colaborativos e coletivos e apresenta a seus usuários novas e estimulantes oportunidades para desenvolvimento de aplicativos que funcionem segundo essa perspectiva de colaboratividade. Diante da tendência de reutilização trazida por essa nova personificação da Web, e somada ao uso em massa das redes sociais, inúmeros aplicativos vêm sendo incorporados ou combinados para atender a novas demandas.

Essa combinação de tecnologias leva o nome de *mashup*, termo que vem da música e que significa misturar. Na web o termo mantém um significado semelhante, é “um web site ou uma aplicação web que usa conteúdo de mais de uma fonte para criar um novo serviço completo” (MERRIL, 2009). Dessa forma, os *mashups* conseguem agregar novas funcionalidades ou se somam para superar as funções originais. O *Mashup* de aplicativos se apresenta como um método de criação ágil e interessante, pois garante um bom aproveitamento do tempo e dos dados de aplicações já disponíveis.

Dentre as ferramentas mais populares e utilizadas para *mashup* na Web, nos dias atuais, estão os mapas. Tradicionalmente usamos mapas para encontrar um endereço, traçar rotas entre pontos, localizar pontos turísticos e outras numerosas utilizações. Google, Yahoo, Microsoft e outros gigantes da Web têm sua ferramenta de mapas baseada na web, porém, em um passado recente, nenhuma dessas ferramentas possuía funcionalidades diferentes das já vistas em aplicações similares. Mesmo com essa “limitação”, alcançaram e satisfizeram o grande público, no entanto, deixaram em aberto uma parcela importante de usuários que precisavam de funcionalidades que se adequassem melhor às suas necessidades.

Uma das características do trabalho com mapas é a facilidade de abstração para adequar-se a uma determinada demanda ou solucionar um problema em específico. Seguindo a natureza prática que este trabalho se propõe a ter, há que se destacar algumas das razões que justificam o desenvolvimento da aplicação que será tratada nos próximos capítulos.

Na cidade de Natal, capital do Rio Grande do Norte, no ano de 2010 houve um incremento de 165% no número de casos de dengue em relação a 2009 (NATAL, 2011). Já neste ano, segundo dados da SESAP/RN, até setembro, já haviam sido notificados 6.225 casos de dengue na cidade.

Assim como nos mais variados ambientes e contextos, este cenário epidemiológico que se apresenta na cidade de Natal provoca o desenvolvimento de estratégias que auxiliem nas diversas tarefas que surgem. Os elevados números apresentados já justificam o uso dos

meios mais diversos para diminuição da incidência de casos de dengue em Natal, dentre os quais, os sistemas computacionais são uma parte importante neste cenário e surgem como parte imprescindível na resolução dos problemas apresentados.

Sendo assim, a proposta do trabalho é desenvolver um sistema web para mapeamento dos casos de dengue em Natal, possibilitando, a partir de seu uso, a articulação de medidas mais eficazes para o combate à doença, uma vez que permitirá a indicação geográfica do local onde os casos ocorrem (rua, bairro, etc.) e que a geração de relatórios gráficos sobre os dados colhidos servirá como guia nas ações de combate à epidemia. Além disso, os mapas são uma das formas mais dinâmicas para representação, interpretação e exploração de dados.

O trabalho se desenvolve a partir de um referencial teórico feito no capítulo 3, em que será abordada detalhadamente a API Google Maps e também onde é tratado o JavaServer Faces e outras tecnologias utilizadas. No capítulo 4 é discutida a implementação da aplicação, detalhando o ambiente de desenvolvimento, e como a aplicação está organizada. As conclusões do trabalho e as perspectivas sobre como ele poderá ser estendido futuramente são apresentadas no capítulo 5.

2 OBJETIVOS

As próximas subseções apresentam os objetivos deste Trabalho de Conclusão de Curso, bem como o que motivou seu desenvolvimento, a metodologia utilizada e a forma como está estruturado.

2.1 GERAL

A proposta deste trabalho é sintetizar os conhecimentos adquiridos na Especialização em Desenvolvimento de Sistemas Corporativos, bem como instrumentalizá-los através do desenvolvimento de um sistema computacional para mapeamento dos casos de dengue na cidade de Natal, Rio Grande do Norte.

A construção do sistema deverá permitir a indicação geográfica do local onde os casos de dengue ocorreram e facilitar a adoção de medidas direcionadas para o combate à doença. Essa indicação se dará por meio de um mapa e de ferramentas visuais de marcação gerados e manipulados através da API Google Maps.

2.2 ESPECÍFICOS

Os objetivos específicos deste TCC são:

- Pesquisar e analisar conceitos de pesquisas e visualização de mapas;
- Conhecer a API Google Maps, seu funcionamento básico e suas principais ferramentas;
- Pesquisar as demais tecnologias necessárias à implementação da aplicação e como será sua integração;
- Implementar e validar os casos de uso levantados;
- Documentar o desenvolvimento da aplicação e os resultados obtidos a partir do seu uso inicial.

3 REFERENCIAL TEÓRICO

3.1 GOOGLE MAPS – VISÃO GERAL

Líder entre as aplicações para visualização de mapas na web desde a metade dos anos 2000, o Google Maps se apresenta como uma ferramenta construída sobre tecnologia Ajax¹ e que oferece uma boa resposta visual a seus usuários. Com sua API pública, possibilitou a customização e permitiu a adição de aplicações de dados específicas aos mapas, abrindo definitivamente as portas aos *mashups*.

Contudo, o Google Maps nem sempre teve sua API aberta. Desenvolvida pelos irmãos Lars e Jens Rasmussen e lançada em fevereiro de 2005, a API chamou a atenção de desenvolvedores, que logo se lançaram na tentativa de hackeá-la e incorporá-la a seus próprios sites. Isso levou a Google a lançar mão de uma API pública em junho do mesmo ano.

Aquele que é considerado frequentemente como primeiro *mashup* da Internet foi uma combinação do Google Maps com anúncios de imóvel do Craigslist.org. Esse *mashup* foi, de fato, criado antes de a API pública ser lançada e foi fruto do esforço do desenvolvedor Paul Rademacher. Na época, esse fato fora considerado revolucionário e inaugurou a era da mistura de informações com uso de fontes diferentes.

¹ O nome é uma abreviação de Asynchronous JavaScript + XML, um misto de “tecnologias que visa eliminar a natureza start-stop-start-stop de interação na Web, introduzindo um mecanismo intermediário entre o usuário e o servidor”. (James, 2005)

Para se ter uma idéia da popularidade de seu uso atual, segundo o programmableweb.com (www.programmableweb.com/apis), com estatísticas de setembro de 2011, o Google Maps é a API mais usada para se fazer *mashups*. Os dados apontam que 40% de todos os *mashups* usam a API da Google. Em comparação, a segunda API mais popular foi o Twitter, com 10%, e a terceira mais popular foi o Youtube, também com 10%.

O Google Maps, na verdade, é composta de uma família de APIs, dentre as quais se pode destacar a Google Earth API, a Google Maps API For Flash e a Google Maps JavaScript API, sendo esta última a mais utilizada em *mashups* e aplicações web.

Durante a Google I/O Conference, em maio de 2009, a versão 3 da Google Maps JavaScript API foi anunciada e em maio de 2010; também durante a Google I/O Conference, foi apresentada não mais como sendo uma versão beta, mas sim como a escolha oficial da Google e sua recomendação para as novas aplicações com Maps. Segundo a definição do site oficial, Google[1] (2010), “a versão 3 dessa API fornece diversos utilitários para manipulação de mapas e adição de conteúdo aos mapas por meio de vários serviços, permitindo criar aplicativos de mapas robustos no seu site.”

Uma vez que tomamos conhecimento do que o Google Maps API é capaz de prover, como uma aplicação ou na combinação com outras ferramentas, torna-se imprescindível saber, mesmo que de forma sintética, como ela funciona.

Quando nos deparamos com a dinâmica natural e a eficácia visual do Google Maps, temos a impressão de que algo muito complexo envolve seu uso, porém nada de mágico o envolve. Por dentro do uso padrão do maps há apenas HTML, CSS e JavaScript trabalhando juntos num background Ajax.

Os mapas que são exibidos na ferramenta são compostos por imagens trazidas por chamadas Ajax e inseridas dentro de uma página HTML. Quando se navega pelo mapa, novas informações sobre as coordenadas e o nível de zoom do mapa são enviadas pela API por meio do Ajax, que retorna novas imagens e carrega-as novamente na página HTML.



Figura 1 - Mapa Google básico

A API por si só consiste basicamente em arquivos JavaScript que contêm classes com métodos e propriedades que podem ser usadas para dizer ao mapa como se comportar. As diferentes maneiras de utilização das classes e de como elas poderão ser manipuladas é que provem aos desenvolvedores a oportunidade de personificação das aplicações e a combinação de tecnologias na criação dos *mashups*.

Apesar de a API da Google cuidar da maior parte de como os mapeamentos funcionam, para compreender com maiores detalhes todos os recursos do Google Maps e como eles são utilizados, alguns conceitos básicos de mapeamento se tornam necessários.

3.2 COORDENADAS GEOGRÁFICAS E GEOCODIFICAÇÃO

Segundo Corrêa (2009), “o sistema de mapeamento da Terra, através de coordenadas geográficas, expressa qualquer posição horizontal sobre o planeta através de duas das três coordenadas existentes num sistema esférico de coordenadas, alinhadas com o eixo de rotação da Terra”.

Existem pelo menos quatro sistemas para assinalar uma localização exata para qualquer ponto no globo terrestre, em três deles o globo é dividido em latitudes, que vão de 0 a 90° (Norte ou Sul), e longitudes, que vão de 0 a 180° (Leste ou Oeste).

Normalmente os valores de latitude e longitude são apresentados em graus, minutos e segundos. No entanto, nos últimos anos a popularização dos GIS (*Geographic Information System*) e de sistemas como o Google Maps fez com que essas coordenadas se tornassem amplamente utilizadas e, por conveniência, optou por se utilizar o formato decimal.

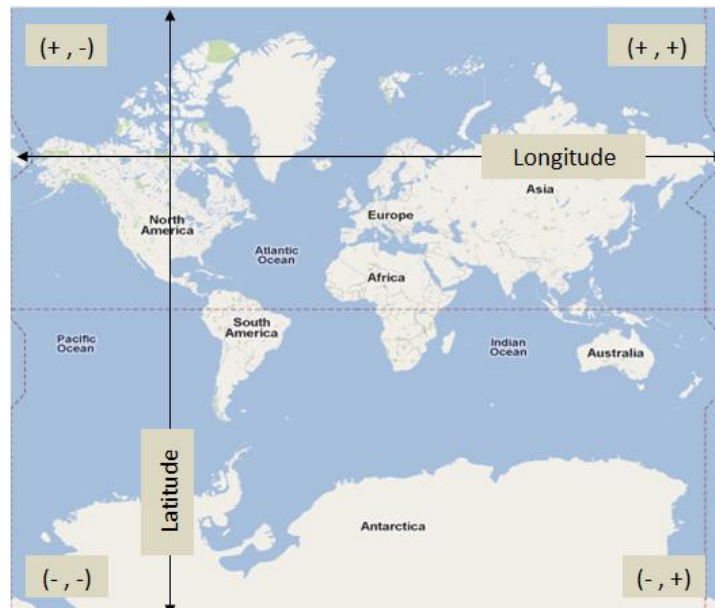


Figura 2 - Latitude e Longitude - adaptado de SVENNERBERG (2001)

Destarte, para se fazer a conversão entre as coordenadas em graus, minutos e segundos para coordenadas decimais, utiliza-se o seguinte cálculo:

$$\text{Graus decimais} = \text{graus} + \frac{\text{minutos}}{60} + \frac{\text{segundos}}{3600}$$

Por exemplo, a seguinte coordenada, convertida para graus decimais, apresenta-se da seguinte maneira: latitude: $-5^{\circ} 48' 2''$ ficaria -5,800556

Apesar do amplo uso das coordenadas decimais pelos sistemas computacionais, nós, seres humanos, não as utilizamos com frequência para representar endereços físicos. Quando necessitamos localizar algum ponto no mapa, utilizamos um endereço (como Av. Hermes Fonseca, Natal, RN, Brasil) e um processo chamado Geocodificação se faz necessário.

Esse processo transforma o endereço literal em coordenadas decimais (como latitude - 5.7920 e longitude -35.1973) e, assim, elas podem ser utilizadas para localizar algum mapa e/ou região, ou, ainda, para utilização de outros recursos do maps. Na figura 3 pode-se ver algumas cidades e suas coordenadas decimais:

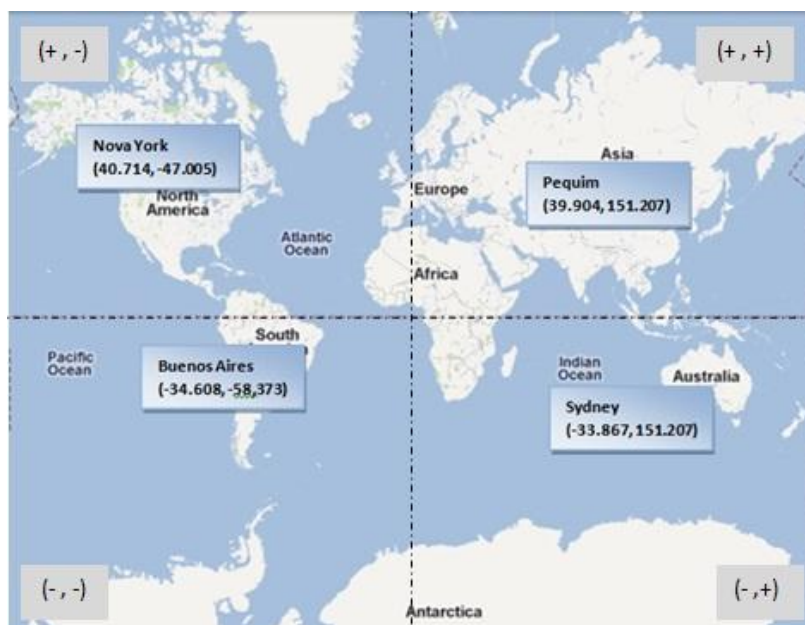


Figura 3 - cidades e suas coordenadas em formato decimal

Essa conversão passa despercebida pelos usuários convencionais, mas é muito útil para o desenvolvimento de aplicações e *mashups*.

3.3 GOOGLE MAPS – FERRAMENTAS

Para começar a trabalhar com a Google Maps API será necessário possuir uma página HTML criada com suas seções básicas. A API nada mais é que um arquivo JavaScript que é armazenado nos servidores Google e é carregada com uma *tag* `<script>` na seção `<head>` da página.

Não é o intuito deste trabalho entrar em detalhes sobre JavaScript, mas é importante lembrar que o elemento `<script>` pode ser usado para adicionar scripts (comandos a serem executados) à página web, o que torna necessária uma breve explanação sobre o uso da *tag*. O elemento em questão possui dois atributos que devem ser usados: o primeiro é o *type* que determinará qual tipo de script será utilizado, cujo valor, neste caso, deverá ser o *text/javascript*; e o outro parâmetro é o *src*, onde será utilizada a URL que apontará para a API Google Maps. Na listagem 1 há um exemplo de como essa referência a API é feita:

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
```

Listagem 1 - Script que referencia a API Google Maps

Na *tag* que carrega a API, nota-se a presença de um parâmetro que complementa sua URL, ele encontra-se após o símbolo de interrogação e se chama *sensor*. “Esse parâmetro é obrigatório quando se referencia a API e indica se a aplicação usa algum equipamento que possui sensor, como um GPS, para indicar a posição do usuário” (SVENNERBERG, 2010). Pelo código acima, nota-se explicitamente que os valores que esse parâmetro pode assumir são *true* e *false*. Se a aplicação usa algum tipo de equipamento com sensor para indicar a posição do usuário, esse valor será *true*, caso contrário seu valor será *false*. Normalmente, equipamentos com sensor são *smartphones* ou outros *gadgets*.

É importante saber que o uso do atributo *sensor* com o valor *true* não resultará em nenhuma mudança no mapa, apenas será utilizado como estatística pela Google, pois é necessário reportar o uso de sensores a seus provedores de dados, porém recomenda-se manter o parâmetro com valor *false* quando não for utilizado em nenhum equipamento com sensor.

Normalmente a API Google Maps tenta determinar automaticamente o idioma nativo da aplicação. No entanto, outro parâmetro poderá ser utilizado para alterar o idioma que será apresentado à interface de usuário. Basta adicionar “*&language=*” ao final do parâmetro, onde após o símbolo de igualdade deverá ser adicionada a abreviação do idioma desejado. Por exemplo, EN, para inglês.

Após carregar a API através do elemento `<script>` será necessário criar o mapa e inseri-lo na página, para isso, será necessário inicializá-lo e isso é feito criando-se uma nova instância do objeto `google.maps.Map`. Dois parâmetros são utilizados para instanciar esse objeto:

- uma referência a sessão HTML que receberá o mapa. Um exemplo é o componente `<div>`, que pode ser utilizado para “receber” o mapa. Esse componente deve possuir um `id` que será utilizado para identificá-lo na página.
- um objeto literal chamado `MapOptions`, que contém os parâmetros para inicialização do mapa, como zoom inicial, qual será o centro do mapa e que tipo de mapa será exibido.

Além de referenciar o elemento no qual será inserido na página HTML, será necessário inicializar o mapa e para isso, como já fora comentado, um objeto literal chamado `MapOptions` ficará responsável por essa função. Esse objeto conterá três propriedades requeridas para que o mapa trabalhe inicialmente. São elas:

- **Center:** define o centro do mapa baseado em coordenadas, para isso será necessário o uso de coordenadas decimais para latitude e longitude. Para latitude os valores são compreendidos entre -90 e +90 graus e para longitude os valores são entre -180 e +180 graus
- **Zoom:** define o nível de zoom do mapa. Esse parâmetro deve estar entre os valores 1 e 23, onde quanto maior o valor, maior é o nível de aproximação no mapa.
- **mapTypeId:** determina o tipo de mapa que será utilizado inicialmente. Para um mapa de ruas padrão esse parâmetro deve assumir o valor **google.maps.MapTypeId.ROADMAP**, para um mapa com imagens de satélite a opção utilizada deve ser **google.maps.MapTypeId.SATELLITE**.

3.3.1 Marcadores

O uso mais frequente de mapas é visualizar a posição geográfica de algum ponto específico, tais como hotéis e restaurantes, pontos turísticos, empresas, dentre outros. Com esse intuito, o Google Maps dispõe de uma ferramenta que permite a identificação desses pontos. Essa ferramenta é chamada marcador, que nada mais é que uma pequena imagem que pode ser posicionada em um lugar específico do mapa. Um marcador pode ser visualizado na figura 4:



Figura 4 – figura do marcador Google Maps

Para inserir um marcador simples no mapa, bastam apenas algumas linhas de código. Para criá-lo será usado o objeto **google.maps.Marker**, utilizando-se apenas um parâmetro, o objeto do tipo **google.maps.MarkerOptions**.

Esse parâmetro possui inúmeras propriedades, porém, para a criação e adição do marcador ao mapa, apenas duas são necessárias: são as propriedades *position*, que definirá as coordenadas do local onde o marcador será posicionado, e a propriedade *map*, que referenciará o mapa em que se deseja que o marcador seja inserido.

3.3.2 ToolTip

Outras opções, além das mencionadas na seção anterior, podem ser exploradas, bem como a propriedade *title*, que adiciona uma “dica de contexto” (no inglês *tooltip*) ao marcador. A *tooltip* é uma caixa de texto que é exibida sempre que o ponteiro do mouse é posicionado sobre o marcador. Na figura 5 um exemplo de *tooltip*:

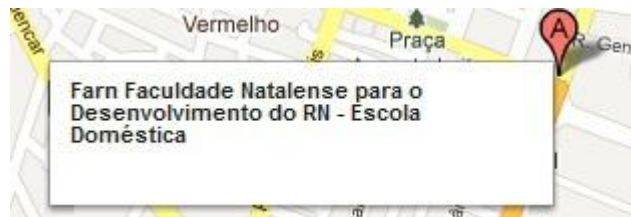


Figura 5 - *ToolTip*

A dica de contexto é uma opção interessante para agregar informação ao marcador, mas não é suficiente quando se deseja acrescentar maiores detalhes relacionados ao lugar que o marcador indica. A Google Maps API fornece uma ferramenta muito útil para esse objetivo, é a *InfoWindow*. Ela funciona como um balão de fala que normalmente aparece quando se clica sobre o marcador.

3.3.3 InfoWindow

A *infoWindow* possui muitas propriedades, mas com certeza a mais importante delas é a *content*, que define o conteúdo a ser inserido dentro dela. Além de texto convencional, a propriedade *content* permite a adição de código HTML. Isso significa que é possível adicionar elementos HTML, como imagens, vídeos, links ou outros componentes que sejam úteis. Pose-se ver um Exemplo de *infoWindow* na figura 6:



Figura 6 - InfoWindow

Mesmo depois de inserido, o texto na infoWindow não será exibido automaticamente no mapa, a menos que seja ligado a um marcador. Além disso, normalmente, adiciona-se um evento de clique sobre o marcador para que a janela seja exibida quando clicado sobre ele.

Um dos métodos utilizados para se capturar e manipular o evento que exibirá a infoWindow é adicionar um *listener* ao objeto, nesse caso, ao marcador. O *listener* é conectado ao objeto e a um determinado evento e quando esse evento acontece o *listener* trata de executar algum comando. Na Google Maps API há um método para adicionar um *listener* que é o `google.maps.events.addListener()`. Ele leva três argumentos:

- o objeto ao qual está ligado;
- o evento que ele deve “ouvir”;
- a função que é chamada quando o evento ocorre.

O código que cria o *listener* é anexado ao evento “click” do marcador que executará algum comando quando esse evento ocorrer. Na maior parte dos casos o objetivo é exibir uma infoWindow ao clicar no marcador, agora basta adicionar o código que exibirá a janela criada.

A infoWindow possui um método chamado `open()`, que abrirá a janela e a exibirá no mapa. Este método leva dois argumentos, um é o mapa no qual a infoWindow será exibida (no caso de haver mais que um mapa) e o outro argumento é o objeto ao qual a infoWindow estará anexada. Neste caso quer-se que a infoWindow fique ligada ao marcador que receberá o “click”. Ao anexar a infoWindow ao objeto em questão, ela será exibida exatamente no local do objeto.

3.4 ARRAYS DE MARCADORES

Obviamente que adicionar um único marcador ao mapa é importante, porém, a maioria das aplicações exige a adição de muitos marcadores, e uma das maneiras de trabalhar com um número maior de marcadores é através de *arrays*. Como a API trabalha essencialmente com JavaScript, a sintaxe de criação e uso de *arrays* vai ser a dessa linguagem.

Um *array* JavaScript é basicamente uma coleção de variáveis e pode conter praticamente tudo que se deseja colocar. Como os *arrays* armazenam qualquer objeto, eles podem ser usados para guardar coordenadas como as dos objetos `google.maps.LatLng`. Dessa maneira, os *arrays* serão usados para criar e posicionar os marcadores.

As principais ferramentas da API Google Maps utilizadas no trabalho foram detalhadas e, em seguida, ainda como parte do referencial teórico, algumas outras tecnologias utilizadas no desenvolvimento da aplicação são descritas.

3.5 JAVASERVER FACES

Desenvolvedores das várias linguagens têm a possibilidade de integrar de forma natural o Google Maps JavaScript API a suas aplicações, o que torna a criação de *mashups* uma tarefa mais comum e dispensa a necessidade de aprofundamento em uma nova linguagem. Focando especificamente a linguagem Java, há a possibilidade de integração da API Maps com uma ferramenta já bem conhecida e muito bem documentada, que é o JavaServer Faces, “uma tecnologia que incorpora características de um framework MVC para WEB e de um modelo de interfaces gráficas baseado em eventos.” (PITANGA, 2004).

O JavaServer Faces foi criado pela própria SUN, lançado em 2005 e é definido como padrão para construção de componentes para a interface de usuário em aplicações Web pela própria SUN e pela JCP (Java Community Process).

Dentre as bibliotecas disponíveis para realizar essa integração Google Maps API e Java Server Faces, pode-se destacar a PrimeFaces, que possibilita a seus usuários a criação simplificada de *mashups* e utilização de todos os recursos dos mapas, como criação de polígonos, linhas e marcadores. Tudo isso com a simplicidade de uso das já conhecidas *tags* JSF.

As vantagens do uso de uma biblioteca que integra JSF com o Google Maps são basicamente as mesmas do uso independente da tecnologia da SUN:

- separação entre visualização e regras de negócio, proveniente do padrão MVC;

- uso de um conjunto de tags flexível para acesso aos componentes da página;
- componentes de fácil utilização;
- agiliza o processo de desenvolvimento.

Apesar de o framework Primefaces possuir os componentes para utilização da API Google Maps, neste trabalho eles não foram utilizados e a geração e manipulação dos mapas foi feita essencialmente através de JavaScript. Essa opção pela linguagem JavaScript foi feita mediante as dificuldades de integração entre a API Google Maps e os componentes Primefaces. Contudo, com exceção do mapa e de suas ferramentas, todo o CRUD foi feito com os componentes do framework.

3.6 XSTREAM

Uma das tecnologias utilizadas no desenvolvimento da aplicação foi a Xstream, “uma biblioteca para serialização de objetos para XML e desserialização de volta” (XSTREAM, 2011).

Segundo Greanier (2000), serialização de objetos é o “processo de salvar o estado de um objeto em uma sequência de bytes, assim como o processo de reconstrução desses bytes em um objeto novamente”. Dentre os motivos para serializar um objeto está a possibilidade de armazenamento desse *array* de bytes em disco, outra maneira de persistência dos dados, além de facilidade de sua transmissão, quando necessário.

A serialização de objetos normalmente é feita para um arquivo XML e a biblioteca Xstream simplifica esse processo através de métodos intuitivos e diretos. Após a definição da classe que terá seus objetos serializados, poucas linhas de código efetuam esse procedimento, como pode ser visto na listagem 2:

```
\\Serializando objeto da classe Person
Person joe = new Person("Joe", "Walnes");
joe.setPhone(new PhoneNumber(123, "1234-456"));
joe.setFax(new PhoneNumber(123, "9999-999"));

XStream xstream = new XStream(new DomDriver());
xstream.alias("person", Person.class);
String xml = xstream.toXML(joe);
```

Listagem 2- Serialização de objetos: retirado de Xstream (2011)

Como pôde ser visto no trecho acima, o método específico da classe XStream responsável pela serialização é o “toXML” que recebe como parâmetro o objeto a ser serializado.

Mesmo não sendo uma tarefa complexa, serializar e desserializar objetos pode tornar-se dispendioso à medida que a aplicação aumenta de tamanho. Uma estratégia adotada pela XStream é permitir que se faça uso de uma técnica que é tendência há algum tempo na linguagem Java: *Annotations*. Para fazer uso dessa tática basta adicionar a anotação “@XStreamAlias” à classe que terá seus objetos serializados e utilizar a propriedade `autodetectAnnotations`, como se observa na listagem 3:

```
@XStreamAlias("message")
class RendezvousMessage {
}

...

XStream stream = new XStream();
stream.autodetectAnnotations(true);
```

Listagem 3 - Serialização com *Annotations*: retirado de Xstream (2011)

Os demais procedimentos para serialização dos objetos se mantêm os mesmos demonstrados na listagem 2.

3.7 JASPERREPORTS

Sendo umas das tarefas mais importantes em aplicativos que trabalham em ambientes corporativos, a geração de relatórios é uma das partes fundamentais deste trabalho e, a seguir, será feito um breve apanhado das tecnologias utilizadas para esse fim.

Uma das bibliotecas mais utilizadas para geração de relatórios em aplicações Java se chama JasperReports. Segundo seu site oficial, JasperForge (2011), o JasperReports é “a biblioteca open-source mais utilizada no mundo, totalmente escrita em Java, capaz de utilizar qualquer fonte de dados e produzir documentos que podem ser visualizados em diversos formatos”.

O design do relatório com todos os itens que farão sua composição são definidos em um arquivo XML e através desse arquivo é possível definir toda sua estrutura, com imagens, campos estáticos, formas geométricas, incluindo os campos que serão dinamicamente preenchidos com informações da base de dados.

Esse arquivo XML deverá ser posteriormente compilado, dando origem a um arquivo com extensão “.jasper” que será utilizado pela aplicação para exibição do relatório, como pode ser visualizado no esquema da figura 7:

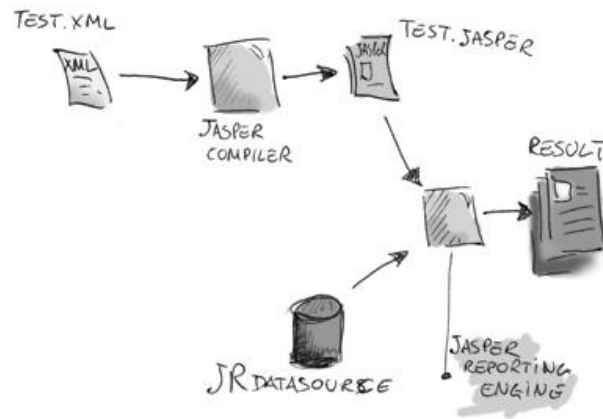


Figura 7 - Geração de relatórios com JasperReports - Retirado de Ireport (2002)

A definição manual do XML pode ser uma tarefa onerosa, contudo, há algumas ferramentas criadas para facilitar essa atividade. Uma das mais utilizadas é a iReport, uma ferramenta que permite a definição do design do relatório utilizando-se um ambiente gráfico. Através desse editor pode-se facilmente determinar qual fonte de dados será utilizada, além de permitir a criação de designs mais complexos para o relatório. Tudo isso de maneira mais rápida e intuitiva. Na figura 8 os recursos visuais da tela do IReport 4.1.1 podem ser visualizados:

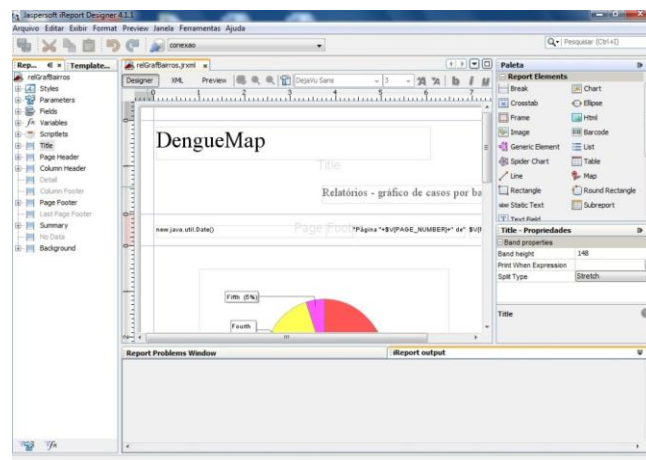


Figura 8 - Tela do iReport

4 A APLICAÇÃO

4.1 VISÃO GERAL

Em suma, a aplicação consiste em um sistema web que utiliza a biblioteca PrimeFaces e a API Google Maps para a geração de mapas personalizados, permitindo a adição de marcadores ao mapa indicando que naquele ponto houve a ocorrência de um caso de dengue.

A aplicação visa a permitir a seus clientes (unidades de saúde, secretarias de saúde, dentre outros) identificar mais precisamente através do mapa os casos de dengue ocorridos, possibilitando, por meio de acesso a relatórios e gráficos, o planejamento de estratégias mais eficazes para o combate à doença. O domínio geral da aplicação pode ser mais bem visualizado no diagrama de casos de uso da figura 9:

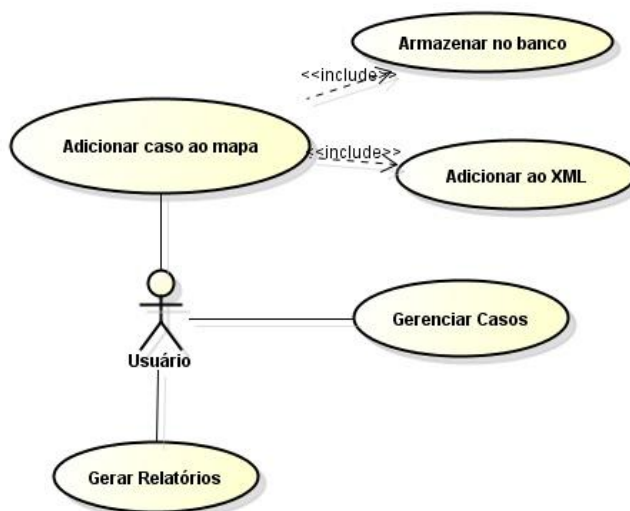


Figura 9 - Diagrama de Casos de Uso

4.2 AMBIENTE DE DESENVOLVIMENTO

A aplicação fora desenvolvida utilizando-se principalmente um computador com processador AMD Athlon 64 X2 4200 e memória RAM de 2 GB, tendo o Windows 7 como seu sistema operacional. A internet utilizada no desenvolvimento da aplicação possuía em média velocidade de 1 MB e, embora a aplicação não tenha sido utilizada em um ambiente com concorrência de múltiplos usuários, seu uso na máquina de desenvolvimento apresentou desempenho satisfatório.

A IDE utilizada para o desenvolvimento da aplicação foi a Eclipse Helios em sua versão 3.6. Já o SGBD utilizado para persistência dos dados e administração do banco gerado foi a PostgreSQL na versão 9.0.

A versão da biblioteca PrimeFaces utilizada é a 3.0M2, liberada em meados do mês de Junho de 2011.

4.3 IMPLEMENTAÇÃO

Antes de partir para os detalhes de implementação da aplicação é importante mostrar como ela está organizada em termos de pacotes e classes, isso dará uma visão geral de como o trabalho está estruturado. Essa organização pode ser vista na figura 10:

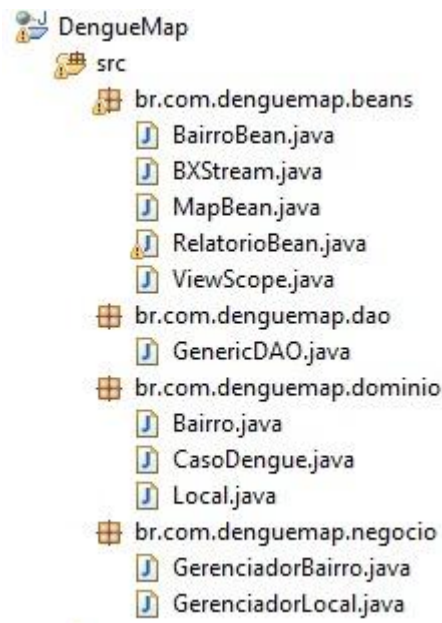


Figura 10 - Estrutura de pacotes e classes

Tentando manter um padrão de organização segundo as funções principais das classes, foram criados quatro pacotes para agrupá-las. A classe CasoDengue relacionada com a classe Local contém os requisitos de domínio da aplicação e os atributos básicos que serão utilizados para manipulação do mapa com a API Google Maps. Os relacionamentos entre as classes são contemplados no diagrama de classes simplificado da figura 11:

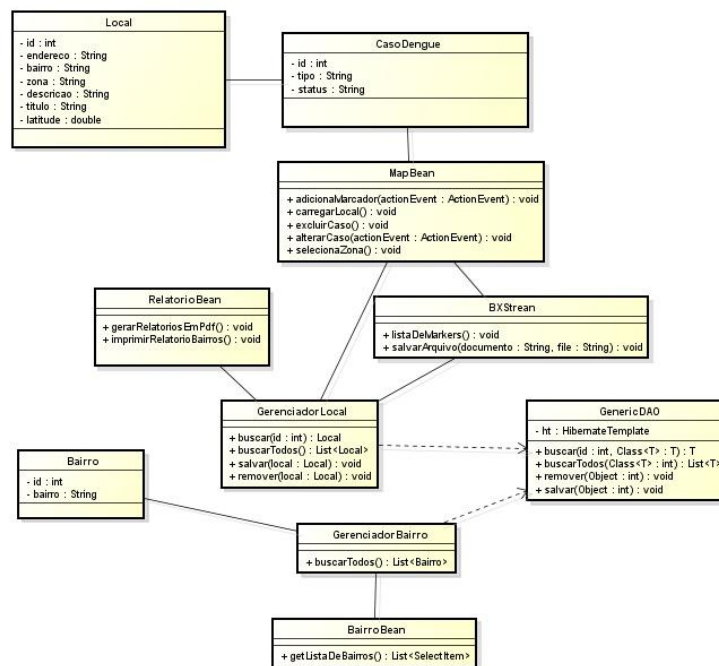


Figura 11 - Diagrama de Classes

A aplicação tem como foco principal a adição ao mapa de marcadores aos locais onde forem notificados os casos de dengue. Seguindo essa linha, a primeira tarefa a ser cumprida na aplicação é referenciar a Google Maps API e inseri-la em uma página JSF (como as páginas foram construídas utilizando JavaServer Faces, serão mencionadas como páginas JSF). Para cumprir essa primeira parte, uma referência a API Google Maps fora inserida na página que servirá para cadastrar os casos de dengue.

Além de referenciar a API é necessário inicializar o mapa e adicioná-lo a uma seção da página JSF. Na listagem 4 parte do código utilizado para esse fim na aplicação:

```

<h:head>
    <script src="http://maps.google.com/maps/api/js?sensor=false"
    type="text/javascript" />
    <script src="js/map.js" type="text/javascript" />
</h:head>
<h:body onload="initialize()">
    <div id="map">
        <div id="map_canvas" style="width: 100%; height: 400px">
    </div>
  
```

Listagem 4 - Referência a API Google Maps e inicialização do mapa

Pelo código acima, observa-se que no carregamento da página (evento *onload*) o método `initialize()` é chamado. Esse método está encapsulado juntamente com os demais métodos JavaScript no arquivo `map.js`, que é referenciado na seção *head* da página. Esse método é responsável pela inicialização do mapa, como pode ser visto no trecho de código da listagem 5:

```
var map;

function initialize() {
    var latlng = new google.maps.LatLng(-5.7820714, -35.198214);
    var myOptions = {
        zoom : 11,
        center : latlng,
        mapTypeId : google.maps.MapTypeId.ROADMAP
    };

    map = new google.maps.Map(document.getElementById('map_canvas'),
                              myOptions);
```

Listagem 5 - método `initialize()`

Para inserir o mapa inicializado na seção `<div>` da página foi necessário definir um id para o elemento, depois referenciá-lo e atribuí-lo a uma variável. No código acima, a variável `map` foi criada e, através do método DOM (Modelo de Objetos do Documento) `getElementById()`, fez-se uma referência à `<div id="map_canvas">`. O método `document.getElementById()` toma uma id, que foi previamente definida (`"map_canvas"`, no caso) e retorna uma referência ao elemento escolhido. Caso o método não encontre o elemento com a id referenciada, ele retornará *null*, ou seja, nada será retornado e, nesse caso, o mapa não seria exibido.

Uma vez que o mapa tenha sido inicializado e inserido na página, a aplicação poderá utilizar todos os recursos e ferramentas de que ele dispõe. Na figura 12 a tela da aplicação que utiliza o mapa inicializado:

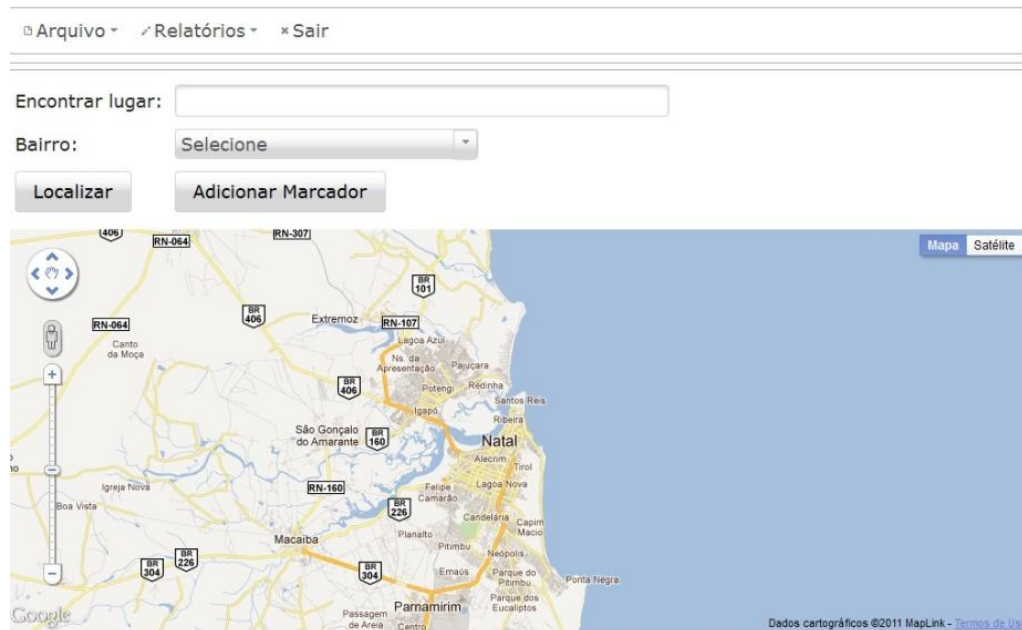


Figura 12 - tela da aplicação com mapa inserido

4.3.1 Adicionando um caso

O cerne da aplicação é a indicação da ocorrência de casos de dengue e a consequente adição de marcadores ao mapa. Os passos que compõem essa atividade podem ser descritos e analisados nos próximos parágrafos.

O usuário informará na caixa de texto da página o endereço do local onde houve a confirmação do caso de dengue e solicitará a busca. Nesse momento o processo de Geocodificação da API do Google Maps é chamado através do método `codeAddress()` (presente no arquivo *map.js*, citado anteriormente). Dentro do código, o acesso a esse serviço de Geocodificação da API do Google Maps foi feito por meio do objeto `google.maps.Geocoder`:

“O método `Geocoder.geocode()` inicia uma solicitação para o serviço de geocodificação, passando um literal de objeto `GeocodeRequest`, que contém os termos de entrada e um método de retorno de chamada a ser executado ao receber a resposta (GOOGLE[2], 2010).

Na aplicação, o parâmetro de entrada para a método `geocode()` fora o endereço digitado pelo usuário e o método de retorno fora o `geocodeResult()`, que é responsável por retornar o status da busca e seus resultados. A interação entre os dois métodos e seus detalhes podem ser melhor analisados no trecho de código da listagem 6:

```

function codeAddress() {
var address = document.getElementById('address').value + ', natal';
    geocoder.geocode({
        'address': address
    }, geocodeResult);
}

function geocodeResult(results, status) {
    if (status == 'OK' && results.length > 0) {
        map.fitBounds(results[0].geometry.viewport);
    } else {
        alert("Endereco nao localizado! Motivo: " + status);
    }
}

```

Listagem 6 - Métodos da Geocodificação

Caso o endereço fornecido pelo usuário seja um endereço válido, ou seja, o campo “status” no objeto de resposta da Geocodificação apresente o valor “OK”, o mapa será posicionado no local encontrado e atualizado com novos dados de centro e zoom, caso contrário uma mensagem de erro será exibida. Na figura 13 observa-se a tela da aplicação com uma busca bem sucedida

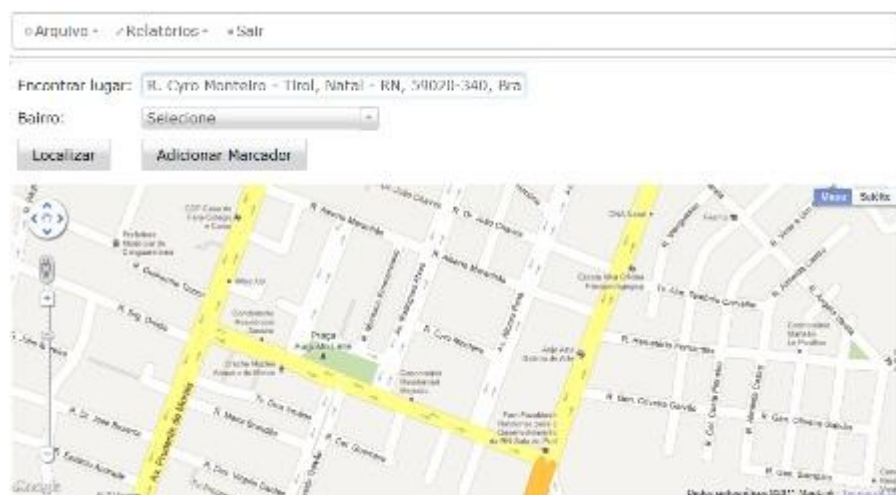


Figura 13 - tela da aplicação com busca bem sucedida

Uma vez que o usuário confirme visualmente que o local desejado é o mostrado no mapa, ele poderá adicionar o marcador ao mapa. As informações sobre o caso a ser inserido e a confirmação de inserção do marcador são feitas através de uma caixa de diálogo que pode ser visualizada na figura 14:



Figura 14 - caixa de diálogo pra inserção de dados

Por fim, a confirmação de inserção do marcador é feita com a adição dos dados do ponto em questão (endereço, latitude, longitude, etc.) no banco de dados PostgreSQL. Esses dados são obtidos através do objeto de resposta da Geocodificação “*results*”. Os dados obtidos são enviados a um Managed Bean², que, dentre outras funções, é responsável por gerenciar a persistência dos dados da aplicação no banco. Na figura 15 é possível visualizar a tela da aplicação com confirmação de adição do marcador e dos dados no banco:

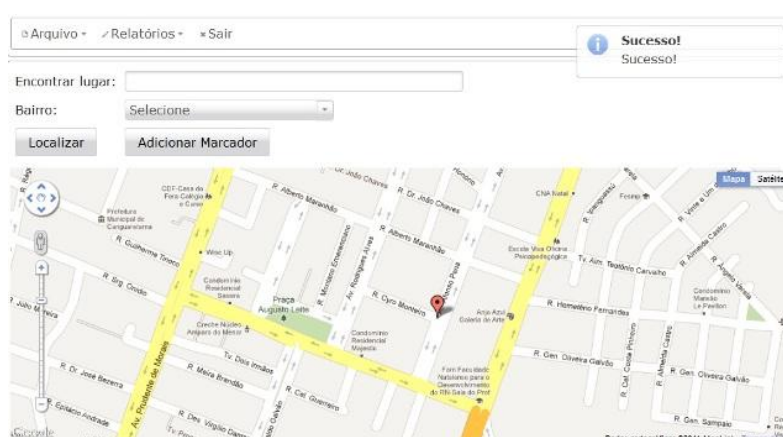


Figura 15 - tela da aplicação após inserção de um novo caso

² De acordo com Geary e Horstmann (2010), “em aplicações JSF, você pode usar managed beans para todos os dados que precisam ser acessíveis a partir de uma página. Os beans são os conduítes entre a interface de usuário e o backend da aplicação”.

O marcador é inserido no mapa através do método `addMarker()` que, através dos dados vindos também da Geocodificação do endereço informado, inserirá o marcador no local desejado. Na listagem 7 um trecho da função `addMarker()`.

```
function addMarker() {
    currentMarker = new google.maps.Marker({
        position : posMarker,
        map : map
    });
}
```

Listagem 7 - método que adiciona o marcador

O marcador será posicionado utilizando-se os pontos de latitude e longitude retornados pela Geocodificação do endereço literal passado na busca. Na função, esses dados são referenciados por uma única variável, “`posMarker`”. Essa variável, na verdade, é um objeto da classe `google.maps.LatLng` que permite a utilização conjunta de valores de latitude e longitude. Além da posição do marcador, o mapa no qual ele vai ser inserido deve ser indicado.

4.3.2 Usando XML para inicializar o mapa

Quando um caso de dengue é inserido no banco de dados, um marcador é adicionado indicando o local de sua ocorrência, porém, esse marcador não será visualizado posteriormente se não houver uma maneira de recuperar essas informações quando o mapa for carregado na página

A API Google Maps utilizada durante o desenvolvimento da aplicação é basicamente um conjunto de classes escritas em JavaScript e, embora haja diversas maneiras de recuperar os dados do banco necessários à inicialização do mapa, a maneira mais prática para executar essa tarefa foi através da linguagem nativa da API. Além disso, recuperar os dados diretamente do banco PostgreSQL e utilizá-los para criar objetos da API Google Maps, como marcadores e janelas de informações mostrou-se uma atividade não tão elementar. Portanto, os passos que se seguem descrevem como esses dados são trazidos e trabalhados na aplicação via um arquivo armazenado em disco e JavaScript.

Através do JavaScript é possível fazer a leitura de arquivos de texto armazenados em disco, dentre eles os arquivos XML, sendo essa operação chamada de *Parsing*. Unindo essa funcionalidade da linguagem com a capacidade de serialização de objetos Java da biblioteca XStream chegou-se a solução encontrada para inicialização do mapa com os dados do banco.

Primeiramente, os dados que estão no banco devem ser serializados para um arquivo XML através da biblioteca XStream, para isso uma classe foi criada e é responsável por essa atividade. Além disso, na aplicação, ao mesmo tempo em que os dados são inseridos no banco, esse arquivo é recriado e atualizado com os dados do novo caso. Na listagem 8, trecho do código utilizado para inserção dos dados do banco no arquivo XML:

```
XStream xstream = new XStream();
    xstream.autodetectAnnotations(true);
    String xml = x.toXML(locais);
```

Listagem 8 - adição de dados ao XML

Como mencionado, a serialização dos objetos se dá cada vez que um novo caso é adicionado. Sempre que isso ocorre, uma busca no banco é feita e uma lista (“locais”) é criada com todos os dados armazenados nele, inclusive os do caso recém adicionado. É essa lista que é passada para o objeto xstream que gerará o XML. O arquivo será armazenado em disco e será utilizado para inicializar o mapa.

A inicialização é feita na ocorrência do evento *onload* da página que receberá o mapa. Esse evento chamará o método `inicialize()` que, além de inicializar o mapa com os dados de centro, zoom e tipo, também será responsável por carregar os marcadores e inseri-los em seus locais corretos. A seguir, na listagem 9, trecho da função `inicialize()` que carregará o XML e inserirá os marcadores:

```
// carrega o xml
xml = xmlLoader("local.xml");
//cria array com dados do XML
markers = xml.documentElement.getElementsByTagName("local");
    for ( var i = 0; i < markers.length; i++) {
        var pos = new google.maps.LatLng(parseFloat(markers[i]
            .getAttribute("latitude")), parseFloat(markers[i]
            .getAttribute("longitude")));
```

```

var marker = new google.maps.Marker({
    position : pos,
    map : map
});
});

```

Listagem 9 - função *initialize()*: carregando XML

No código acima, o laço percorrerá todo o *array* e a cada iteração criará um novo marcador que será adicionado ao mapa.

Além de inseri-lo no mapa, uma dica de contexto e uma *infoWindow*, também criados com dados do banco, são anexados ao marcador em sua inicialização. O trecho de código da listagem 10, a seguir, foi retirado da listagem anterior para facilitar seu entendimento, mas também faz parte da função *initialize()*.

```

var infoWindow = new google.maps.InfoWindow;
var conteudo = markers[i].getAttribute("descricao");
var title = markers[i].getAttribute("titulo");

var marker = new google.maps.Marker({
    position : pos,
    title : title,
    map : map
});
google.maps.event.addListener(marker, 'click', function() {
    infoWindow.setContent(conteudo);
    infoWindow.open(map, marker);
});

```

Listagem 10 - adicionando *InfoWindow* e *tooltip*

Os dados utilizados para inserir conteúdo na *infoWindow* e na dica de contexto são os trazidos do XML e armazenados no *array* *markers*. Essa janela de informações será exibida quando o marcador disparar um evento de clique sobre ele. A figura 16 demonstra a exibição

do mapa com os marcadores vindos do XML, além da infoWindow e da dica de contexto gerados em sua inicialização:

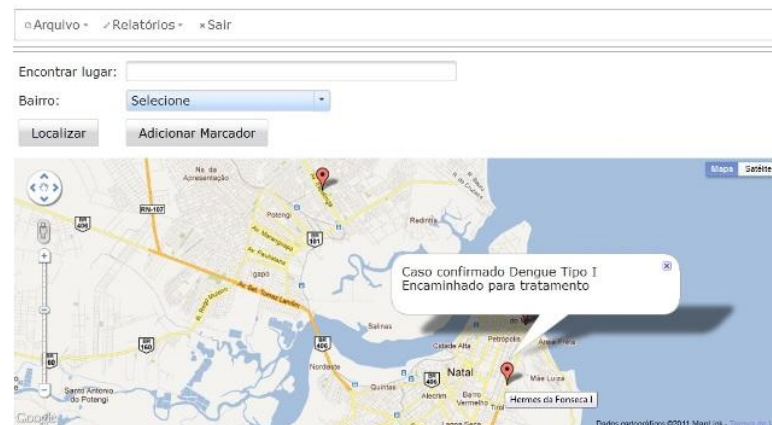


Figura 16 - mapa inicializado com dados do arquivo XML

4.3.3 Gerenciando os casos

Além de permitir a adição de dados ao banco, a aplicação também permite seu gerenciamento através da opção “Gerenciar Casos”, disponível no menu “Arquivo” da aplicação, que pode ser visualizado na figura 17:

Arquivo - Relatórios - Sair				
Gerenciar Casos				
Título	Endereço	Bairro	Zona	
Pte Bandeira I	Av. Pres. Bandeira - Natal - RN, Brasil	Alcirim	Leste	✓ ✕
Itapetinga I	Av. Itapetinga - Potengi, Natal - RN, 59124-400, Brasil	Potengi	Sul	✓ ✕
Hermes da Fonseca I	Av. Hermes da Fonseca - Natal - RN, Brasil	Petrópolis	Leste	✓ ✕
Xavantes	Av. dos Xavantes - Pitimbu, Natal - RN, Brasil	Pitimbu	Sul	✓ ✕
Xavantes II	Av. dos Xavantes - Pitimbu, Natal - RN, Brasil	Pitimbu	Sul	✓ ✕

Figura 17 - Gerenciamento de casos

Um dos módulos da aplicação é a geração de relatórios que servirão para interpretação mais clara dos dados trabalhados. O design dos relatórios foi criado com a ferramenta iReport e são gerados e manipulados em um *bean* através da biblioteca JasperReports. Além de fazer a ligação entre o design do relatório com os dados da aplicação, o *bean* permite que ele seja salvo em disco ou aberto no formato PDF.

5 CONCLUSÕES

A web, nos últimos anos, criou uma demanda por serviços colaborativos e aplicações que trabalhassem conjuntamente. Nesse sentido, a aplicação conseguiu se encaixar nesse papel ao mesclar a API Google Maps com outros frameworks como o PrimeFaces. Este último representou um upgrade no visual da aplicação e tornou seu desenvolvimento mais intuitivo e ágil. Apesar de em algumas situações não se mostrar completamente compatível com algumas demandas do aplicativo e da API Google Maps, o PrimeFaces demonstrou ser um framework muito poderoso no desenvolvimento web.

A aplicação, apesar de experimental e de não ter sido testada com casos reais, pode representar mais uma ferramenta para combater o crescimento dos casos de dengue da cidade de Natal. Trabalhos futuros poderão utilizar sua potencialidade e a urgente demanda por meios que auxiliem nessa tarefa para incrementar suas funcionalidades. Seguindo o mesmo domínio da aplicação, algumas sugestões para estender o trabalho futuramente são:

- permitir a inserção dos casos de dengue divididos por tipos;
- utilizar sobreposição de mapas para uma análise histórica dos casos;
- desenvolvimento de uma versão do aplicativo para dispositivos móveis;
- criar uma interação entre a aplicação e as diversas redes sociais;

Além disso, sua arquitetura poderá ser reusada e facilmente adaptada em muitas outras aplicações de cujo uso demande os serviços da Google Maps API, como o mapeamento de várias outras doenças (epidêmicas ou não), mapas ligados ao setor de turismo e hotelaria, planejamento urbanístico, mercado imobiliário, dentre outros.

REFERÊNCIAS

AGRAWAL, Hitesh. JavaScript XML Parsing on Mozilla Firefox / Opera Browsers, Abril de 2008. Disponível em:

<http://www.hiteshagrawal.com/javascript/javascript-xml-parsing-on-mozilla-firefox-opera-browsers>

Acessado em 02 de setembro de 2011.

ÇIVICI, Çagatay e DARÇIN, Yigit. PrimeFaces User's guide: Covers 1.1 and 2.1 Last Update: 26.07.2010.

CORRÊA, Iran Carlos Stalliviere. Coordenadas Geográficas. Museu de Topografia Prof. Laureano Ibrahim Chaffe - Departamento de Geodésia – IG/UFRGS. Maio de 2009.

GEARY, David; HORSTMANN, Cay. Core JavaServer Faces. Third Edition. Prentice Hall, 2010.

GOOGLE[1]. Documentação da API Google Maps JavaScript, 2010. Disponível em:

<http://code.google.com/intl/pt-BR/apis/maps/documentation/javascript/>

Acessado em 05 de julho de 2011.

GOOGLE[2]. Serviço de Geocodificação da API Google Maps, 2010. Disponível em:

<http://code.google.com/intl/pt-BR/apis/maps/documentation/javascript/examples/geocoding-simple.html>

Acessado em 14 de agosto de 2011.

GREANIER, Todd M. Flatten your objects: Discover the secrets of the Java Serialization API. JavaWorld.com, julho, 2000. Disponível em:

<http://www.javaworld.com/javaworld/jw-07-2000/jw-0714-flatten.html>

Acessado em 02 de setembro de 2011.

IREPORT. A Design Tool for JasperReports, Cap.3, 2002. Disponível em:

<http://ireport.sourceforge.net/cap3.html>

Acessado em 12 de setembro de 2011

JAMES, Jesse. Ajax: A New Approach to Web Applications. February 18, 2005. Disponível em:

<http://www.adaptivepath.com/publications/essays/archives/000385.php>

Acessado em 10 de agosto de 2011.

JASPERFORGE. Página oficial da biblioteca JasperReports, 2011. Disponível em:

www.jasperforge.org/projects/jasperreports.

Acessado em 05 de setembro de 2011.

LLC, Geospatial Training Services. Mashup Mania with Google Maps - Version 5: Updated January 2009.

MERRIL, Duane. Mashups: The new breed of Web app – Julho/2009. Disponível em:

<http://www.ibm.com/developerworks/xml/library/x-mashups/index.html>

Acessado em 18 de julho de 2011.

NATAL, Prefeitura Municipal. Boletim Epidemiológico Dengue, Ano 04 – número 10, 2011. Disponível em:
<http://www.natal.rn.gov.br/sms/paginas/ctd-427.html>
Acessado em 14 de setembro de 2011.

NÓBREGA, Pablo Bruno de Moura. Tutorial: Técnicas de Geração de Relatórios com JasperReports - Uma abordagem utilizando a ferramenta de design iReport. Fortaleza – CE, fevereiro de 2009.

PITANGA, Talita. JavaServer Faces: A mais nova tecnologia Java para desenvolvimento WEB. Grupo de Usuários Java – 2004.

PrimeFaces. Página oficial do framework. Disponível em:
<http://www.primefaces.org/>
Acessado em 26 de setembro de 2011.

RUZYLO, Milena. PrimeFaces Components. Outubro 2010. Disponível em:
<http://www.webchili.pl/articles/10009/primefaces-components>
Acessado em 12 de agosto de 2011.

SESAP/RN, Governo do Estado. Botetom Epidemiológico Dengue – julho de 2011. Disponível em:
<http://www.rncontradengue.rn.gov.br/noticias.php?id=92&titulo=sesap-divulga-novo-boletim-da-dengue>
Acessado em 14 de setembro de 2011.

SVENNERBERG, Gabriel. Beginning Google Maps API 3. Apress – 2010
Usando o XStream para serializar e desserializar objetos Java para XML. Disponível em:
<http://www.meupost.com/2009/03/usando-o-xstream-para-serializar-e-desserializar-objetos-java-para-xml/>
Acessado em 10 de agosto de 2011.

XSTREAM, Página oficial da biblioteca, 2011. Disponível em:
<http://xstream.codehaus.org/>
Acessado em 10 de agosto de 2011