UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

ITAMAR FRANCISCO DE SALES JUNIOR

SISTEMA GEORREFERENCIADO PARA PARTICIPAÇÃO POPULAR NA COMUNICAÇÃO DE PROBLEMAS DE INFRAESTRUTURA URBANA

São Luís
2012

ITAMAR FRANCISCO DE SALES JUNIOR

SISTEMA GEORREFERENCIADO PARA PARTICIPAÇÃO POPULAR NA COMUNICAÇÃO DE PROBLEMAS DE INFRAESTRUTURA URBANA

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dr. Anselmo Cardoso de Paiva

São Luís

2012

ITAMAR FRANCISCO DE SALES JUNIOR

SISTEMA GEORREFERENCIADO PARA PARTICIPAÇÃO POPULAR NA COMUNICAÇÃO DE PROBLEMAS DE INFRAESTRUTURA URBANA

Monografia apresentada ao Curso de Ciência da Computação da Universidade Federal do Maranhão, para obtenção do grau de Bacharelado em Ciência da Computação.

Aprovada em / /

BANCA EXAMINADORA

Prof. Anselmo Cardoso Paiva (Orientador)
Doutor em Ciência da Computação
Pontifícia Universidade Católica do Rio de Janeiro

1º Examinador
Prof. Carlos de Salles Soares Neto
Doutor em Ciência da Computação
Pontifícia Universidade Católica do Rio de Janeiro

2º Examinador
Prof. Geraldo Braz Junior
Mestre em Engenharia de Eletricidade
Universidade Federal do Maranhão

AGRADECIMENTOS

A Deus, pelas dádivas nessa caminhada de vida.

A meus pais, que sempre estiveram ao meu lado.

Aos colegas de universidade que, ao longo dessa jornada acadêmica, contribuíram com conhecimento e apoio para a conclusão desse trabalho.

Ao Prof. Anselmo Cardoso de Paiva, pela orientação segura e idealização na elaboração da monografia.

RESUMO

Desenvolvimento de um sistema de cadastro de postagens que possuem imagens de buracos nas avenidas e ruas da cidade de São Luís – MA. Apresenta a utilização da API do Google Maps, do Framework *Web* MVC em Java Mentawai, da biblioteca JavaScript, jQuery e do sistema gerenciador de banco de dados MySQL. Exibe a integração dessas tecnologias *Web* 2.0 com o provedor de conteúdo de mapas Google Maps e descreve a modelagem do sistema desenvolvido.

Palavras-chave: Desenvolvimento *Web*. Mapas. Java. Google maps. Mentawai.

ABSTRACT

Development of a registration system of posts that have images of holes in the avenues and streets of the city of São Luís - MA. In the use of Google Maps API, the Java MVC Web Framework Mentawai, the jQuery JavaScript library and management system MySQL database. Displays the integration of these Web 2.0 technologies with the content provider Google Maps mapping and describes the modeling system developed.

Keywords: Web Development. Maps. Java. Google Maps. Mentawai.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AJAX	– Asynchronous JavaScript and XML
API	– Application Programming Interface
CSS	– Cascading Style Sheets
HTML	– Hypertext Markup Language
IDE	– Integrated Development Environment
KISS	– Keep It Simple
MVC	– Model View Controller
XML	– Extensible Markup Language
MIT	– Massachusetts Institute of Technology
GPL	– GNU General Public License
JSP	– JavaServer Pages

SUMÁRIO

1 INTRODUÇÃO

A segunda geração de serviços na Web, também conhecido como Web 2.0, busca ampliar as formas de produzir e compartilhar informações online. Dessa forma, o foco das aplicações foi alterado para uma maior integração de Websites e a possibilidade da troca de conteúdo. Tais aplicações começaram a se integrar e a trocar conteúdo através de serviços disponibilizados por APIs, tornando também a Web uma plataforma de desenvolvimento. Além dessa plataforma de desenvolvimento, a Web 2.0 trouxe uma mudança na interação dos usuários. Os usuário que utilizam aplicações desenvolvidas com essas novas tecnologias passaram a ser produzir as informações, ou seja, mudou-se a forma de interação dos usuários. O que antes era realizado de forma passiva, as informações eram disponibilizadas e consumidas pelos usuários, passou-se a ser realizados também de forma ativa, os usuários produzindo informações que alimentas as mais diversas aplicações. Podemos citar como exemplo de comparação entre a Web 1.0 e a 2.0:

Web 1.0 havia os sites pessoais, já na Web 2.0 há os blogs, os quais tem como fundamental característica a capacidade de leitores escreverem comentários de forma a interagir com o autor do blog e outros leitores. Também como exemplo, na Web 1.0 há as enciclopédias online, cujo conteúdo é produzidos autores de dicionários, gramáticos, etc. Mas na Web 2.0, existe a Wikipédia, enciclopédia desenvolvida de forma colaborativa na qual qualquer usuário pode participar produzindo conteúdo.

Os princípios que normalmente são seguidos por aplicações desenvolvidas para a Web 2.0 são o uso da internet como plataforma para processar, produzir ou consumir informação e a existência de uma conexão computador – internet como ferramenta principal de trabalho.

Assim, diante desse novo paradigma de interação dos usuários com a Web e dentre os mais diversos problemas de infraestrutura de um centro urbano, idealizou-se o desenvolvimento de uma aplicação Web 2.0, cujo foco é o problema da falta de manutenção das avenidas e ruas da cidade de São Luís - MA. Também focou-se no engajamento cívico dos usuário, uma vez que esses usuários é que irão alimentar o sistema com imagens dos buracos das avenidas e ruas da cidade, indo ao encontro, assim, da nova forma de paradigma de utilização da Web 2.0.

Este trabalho tem dois objetivos gerais: o primeiro é o estudo e posteriormente a escolha de tecnologias Web e Web 2.0, levando em consideração as documentações disponíveis de cada tecnologia, seus termos de licenças, arquiteturas e particularidades. O segundo objetivo geral é a integração das tecnologias escolhidas. Já referente ao objetivo específico, esse trabalho tem o objetivo de desenvolver uma aplicação Web 2.0 para o cadastro de imagens de buracos nas avenidas e ruas da cidade de São Luís - MA.

Esta monografia está estruturada da seguinte forma. No capítulo 2 tem-se uma visão geral das tecnologias utilizadas, mostrando suas principais características e funcionalidades. O capítulo 3 fala da modelagem do sistema, exibindo a descrição da aplicação, requisitos funcionais, diagramação de classes e sequência, bem como as interfaces da aplicação. Por fim, no capítulo 4, tem-se a conclusão do trabalho.

2 TECNOLOGIAS UTILIZADAS

2.1 Descrição geral do Framework Mentawai

O Mentawai foi o primeiro *Framework* Web MVC em Java a adotar, implementar, documentar e incentivar exclusivamente toda e qualquer tipo de configuração através de configuração programática, abolindo por completo o uso de XML e *Annotations* para as configurações do *Framework*.

Outro pilar em que o Mentawai se apoiou desde o início foi o comprometimento em abstrair e simplificar as principais tarefas recorrentes de todo projeto Web. Ao invés de direcionar o usuário para qualquer outro *Framework*, o Mentawai oferece soluções ou abstrações para as funcionalidades básicas de toda aplicação Web como: pool de conexões com o banco de dados, autenticação, autorização, envio de e-mail, *upload* de arquivo, paginação, tags e outros (MENTAWAI, 2011).

O Mentawai possui uma arquitetura baseada no princípio *KISS*, o qual fundamenta a simplicidade como ordem, evitando, assim, complexidades nos projetos de software. Esse *Framework* também possui como paradigma a noção de ação (MENTAWAI, 2011).

Nesse paradigma, uma ação possui uma entrada de dados (*input*) e uma saída, o resultado (*output*). Esse *input* recebe os dados através de uma requisição Web*,* o qual produz resultados acessados através do *output e* são exibidos numa página JSP, por exemplo.

As ações também têm acesso ao contexto da requisição, tais como dados de sessão e *cookies*. Toda ação no *Framework* Mentawai gera um resultado, que, por sua vez, gera uma conseqüência. Tal conseqüência pode ser configurada para gerar um redirecionamento para uma página qualquer (*redirect*) - nesse caso todos os dados da requisição são perdidos -, um redirecionamento para uma ação específica (*forward*) - aqui há o aproveitamento dos dados da requisição -, como por exemplo, uma ação de validação de dados, ou um encadeamento de ações (*chain*), onde os dados da requisição são processados por duas ações e uma terceira invocando uma conseqüência.

O Mentawai vem com outras conseqüências, como o retorno de uma imagem ou um arquivo binário, ou a opção de codificação das suas próprias consequências.

Assim, as características do paradigma de ação do Mentawai são:

- Uma ação tem uma entrada e uma saída;
- Após a execução de uma ação, é produzido um resultado. Esse resultado é uma *String* (*java.lang.String*) que é interpretada pelo *Framework* produzindo uma conseqüência. Assim, a cada resultado produzido há uma conseqüência.
- Uma ação tem acesso aos contextos. Contextos para um aplicativo Web são geralmente um contexto de sessão e um contexto de aplicação, mas há a possibilidade de criação de contextos próprios.

2.1.1 Instalação

Para utilização do *Framework,* basta adicionar o arquivo mentawai.jar no diretório /*WEB*-INF/lib da sua aplicação e configurar o arquivo *Web*.xml conforme Figura 2.1.

```
<!-- O controlador do Mentawai-->
<servlet>
    <servlet-name>Controller</servlet-name>
    <servlet-class>org.mentawai.core.Controller</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
<!-- Extensão da ação do Mentawai -->
<servlet-mapping>
    <servlet-name>Controller</servlet-name>
    <url-pattern>*.mtw</url-pattern>
</servlet-mapping>
```

Figura 2.1: Instalação do *Framework* Mentawai

2.1.2 Exemplo de *Action*

```
import org.mentawai.core.*;

public class HelloMentawai extends BaseAction {

    public String execute() throws ActionException {
        String username = input.getStringValue("username");
        if (username == null || username.trim().equals("")) {
            return ERROR;
        }
        output.setValue("username", username.toUpperCase());
        return SUCCESS;

    }

}
```

Figura 2.2: Exemplo de *action* do *Framework* Mentawai

Conforme Figura 2.2, a ação procura por um parâmetro de entrada (*input*) *username* e envia para o parâmetro de saída (*output*). Se não encontrar o parâmetro *username*, o resultado retornado será a *String* ERROR , caso contrário, SUCCESS.

Esse retorno será interpretado, conforme configuração do controlador do *Framework*. A Figura 2.3, exibe essa configuração e as conseqüências de cada retorno efetuado pela *action*. A classe *HelloMentawai* herda de *BaseAction* , que é parte integrante do *Framework* e implementa a interface *Action,* também integrante do *Framework*.

Para todo resultado há uma conseqüência associada. Toda configuração dessa associação é realizada por uma classe que estenda de *org.mentawai.core.ApplicationManager*.

2.1.3 Configuração do controlador do *Framework* Mentawai

Segue abaixo um exemplo de uma classe e sua configuração.

```
public class ApplicationManager extends org.mentawai.core.ApplicationManager {

    public void loadActions() {
    ActionConfig ac = new ActionConfig("/HelloWorld", HelloMentawai.class);
    ac.addConsequence(HelloMentawai.SUCCESS, new Forward("/hello.jsp"));
    ac.addConsequence(HelloMentawai.ERROR, new Forward("/username.jsp"));
    addActionConfig(ac);
    }

}
```

Figura 2.3: Configuração do controlador do *Framework* Mentawai

No exemplo acima, a conseqüência de um *SUCCESS* é um *forward* para a renderização da página hello.jsp. Já a conseqüência de *ERROR* é um *forward* para username.jsp. O tratamento de erros pode ser realizado por uma classe que implemente a interface *Validatable.* A interface *Validateble* possui um método *prepareValidator* que recebe, por exemplo, um objeto Validator e um objeto String, que é o nome da *action* chamada.

O Mentawai possui três tidos de ações de contexto. Uma ação de contexto nada mais é que um *Map* com algumas particularidades. Existe o contexto da aplicação, um *Map* que que pode ser utilizado para armazenar informações acessíveis a toda a aplicação. Há o contexto de sessão, utilizado para persistir informações de requisição feitas pelo mesmo usuário. Por fim, existe o contexto de *Cookies* tratados automaticamente pelo *Framework*, mas que pode ser acessado, bem como modificado pelo desenvolvedor. Esse contexto são utilizados para, por exemplo, passar informações num *Forward* de uma *action*.

Quando o container Web inicia, o controlador Mentawai chama o método loadActions() . Através de *ac*, instância da classe *ActionConfig*, foram adicionados diferentes consequências para a implementação da mesma ação. O controlador de Mentawai encontra ações pelo nome, então "/HelloWorld" será chamado pelo http://www.myapp.org/HelloWorld.mtw (MENTAWAI, 2011).

2.1.4 Inclusão de tags HTML.

Exemplo dessa ação numa página jsp, pode ser visualizado na Figura 2.4. Nesse exemplo é possível visualizar a ação num formulário HTML, bem como o campo que de *input*, *username*, o qual será tratado na *action*, conforme Figura 2.2.

```
<html>
    <body>
        <h1>Hello Metawai!</h1>
        <form action="HelloWorld.mtw" method="post">
            Type an username: <input name="username" size="25" />
            <input type="submit" value="Send">
        </form>
    </body>
</html>
```

Figura 2.4: Código de um exemplo de uma página jsp.

Já na Figura 2.5, é exibido o exemplo da página com o resultado do campo de saída *output.* Observa-se o uso da tag Mentawai <mtw:out value=”username”>, o qual foi utilizado para a exibição do resultado de saída.

```
<%@ taglib uri="/WEB-INF/lib/mentawai.jar" prefix="mtw" %>
<html>
    <body>
        <h3>Hello <mtw:out value="username" /> from Mentawai!</h3>
    </body>
</html>
```

Figura 2.5: Código de um exemplo de uma página jsp com um campo de saída.

2.1.5 Filtros

Grande parte das funcionalidades do *Framework* é implementada através dos filtros. Um filtro intercepta uma ação antes que a mesma seja executada e pode modificar os dados de entrada (*input*) e os dados de saída (*output*), ou seja, o filtro pode modificar os dados antes e depois da execução da ação.

Pode-se configurar os filtros na classe *ApplicationManager* por filtro global, ou seja, utilizado por todas as classes ou configurar um filtro por classe. O *Framework* já possui vários filtros, mas podem outros filtros podem ser criados de acordo com a necessidade do usuário.

2.1.6 Arquitetura

A Figura 2.6 mostra a arquitetura do Mentawai, que como se pode notar, os filtros têm acesso a todos os dados de entrada e saída. Também se observa o fluxo de dados do *Framework*: ação, resultado e consequência.

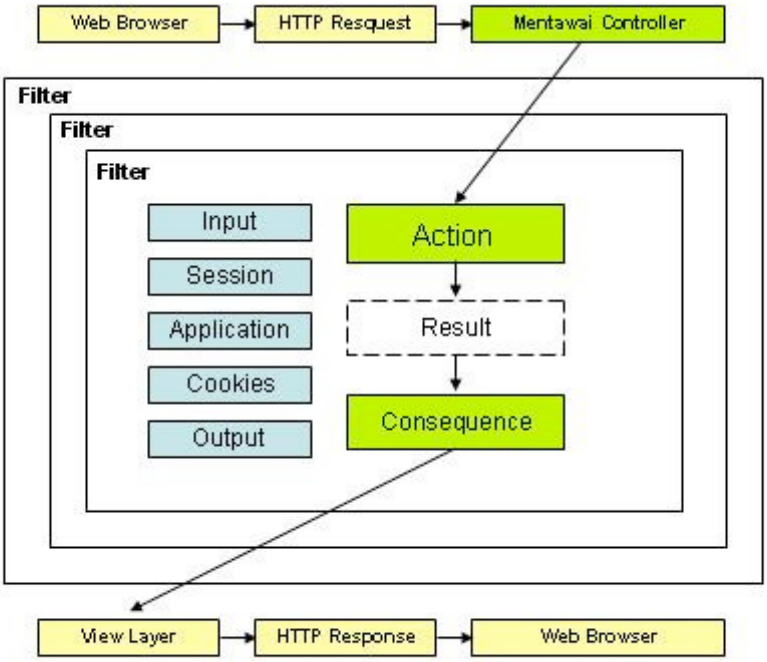

Figura 2.6: Arquitetura do *Framework* Mentawai
Fonte: (MENTAWAI, 2011)

A utilização do *framework* aconteceu devido a disponibilização de uma pilha completa de funcionalidade como autenticação, validação, acesso a banco de dados, envio de e-mails e *tags* da camada de visão. Além da disponibilizar fácil configuração, a qual elimina o uso de arquivos XML, o *framework* provê vasta documentação para consulta.

2.2 Google Maps

Google Maps foi originalmente desenvolvido por dois irmãos dinamarqueses, Lars e Jens Rasmussen. Eles foram co-fundadores da Where 2 Technologies, uma empresa que se dedicava à criação de soluções de mapeamento. A empresa foi adquirida pela Google em outubro de 2004.

Google Maps foi introduzido em um *post* no Google em fevereiro de 2005, revolucionando a forma como aplicações Web poderiam interagir com mapas, pois não era mais exigido o uso de servidores de mapas caros e específicos.

Durante a conferência Google I / O em maio de 2009, a versão três do API, que esta aplicação utiliza, foi anunciada. E em maio de 2010, também durante a conferência Google I / O, ela foi anunciada como versão beta. Agora é a escolha recomendada para novas Aplicações Google Maps.

O Google Maps é um serviço do Google que oferece uma poderosa e amigável tecnologia de mapas e informações sobre empresas locais, incluindo endereços, informações de contato e rotas para as empresas. (GOOGLE, 2011)

O Google Maps JavaScript API V3 é uma interface de desenvolvimento para aplicações baseadas no próprio Google Maps, possibilitando a criação de aplicativos para plataforma Web, a qual permite que o usuário incorpora os serviços do Google Maps às sua páginas Web. Essa API integra mapas e serviços de Geocodificação – processo de conversão de endereços (como “Avenida dos Holandeses, São Luís, MA”) em coordenadas geográficas (latitude 37.09 e longitude -95.71), que podem ser usados para incluir marcadores e posicionar os mapas – aos sites, possibilitando aplicações com conteúdo georreferenciado.

Todo dinamismo do Google Maps é possível devido ao trabalho conjunto do HTML, CSS e JavaScript . Os mapas do *Google Maps* são imagens carregadas em segundo plano com chamadas AJAX que depois são inseridas numa <*div*> HTML. À medida que você navega no mapa, a API do envia informações sobre novas coordenadas e níveis de zoom via chamadas AJAX que retornam novas imagens.

A API, basicamente, consiste em arquivos JavaScript que contêm as classes com métodos e propriedades que você pode usar para dizer o mapa como se comportar.

2.3 Inclusão da API na aplicação

Para incluir a API V3 do Google Maps numa página HTML há alguns pontos a serem observados.

- Incluir a Google Maps JavaScript API V3 usando a *tag* script;
- Criar a div (map);
- Criar um literal de objeto JavaScript contendo as propriedades do mapa;
- implementar uma função JavaScript para criação do objeto map;

- Inicializar através do evento onload da tag body do documento HTML.

A Figura 2.7 exibe o código de exemplo dessa inclusão.

```
<script type="text/javascript" src="http://maps.google.com/maps/api/js?sensor=set_to_true_or_false"></script>
<script type="text/javascript">
  function initialize() {
    var latlng = new google.maps.LatLng(-34.397, 150.644);
    var myOptions = {
      zoom: 8,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
    };
    var map = new google.maps.Map(document.getElementById("map"), myOptions);
  }

</script>
</head>
<body onload="initialize()">
  <div id="map_canvas" style="width:100%; height:100%"></div>
</body>
</html>
```

Figura 2.7 Inclusão da API Google Maps

2.3.1 Serviço de geocodificação

O serviço mais utilizado da Google Maps pela aplicação é a geocodificação, cuja definição é a conversão de endereços em coordenadas geográficas (latitude e longitude). As solicitações ao serviço de geocodificação são realizadas de forma assíncrona, assim a API do Google Maps precisa fazer uma chamada a um servidor externo, tornando necessário um método de retorno de chamada para possa processar os resultados da geocodificação.

Para acessar o serviço, basta instanciar um objeto *google.maps.Geocoder* e chamar o método *Geocoder.gecode()* através desse objeto, o qual vai iniciar uma solicitação ao serviço de geocodificação. Nessa solicitação há a necessidade que seja passado um literal de objeto *GeocodeRequest*, que contém os campos de entrada e um método de retorno de chamada para receber a resposta (resultados da consulta de geocodificação).

Um literal de objeto *GeocodeRequest* contém os seguintes campos:

```
{
 address: string,
 latLng: LatLng,
 bounds: LatLngBounds,
 language: string,
 region: string,
}
```

Figura 2.8 Campos do literal de objeto *GeocodeRequest*

Significado dos principais e obrigatórios campos do objeto *GeocodeRequest*:

- *address* : O endereço a ser codificado;
- *latLng*: latitude e longitude que deseja obter mais próximo;

Caso seja passado uma latitude e longitude(*latLng*), o geolocalizador realizará uma geocodificação reversa, que localizará um endereço baseado no *latLng* fornecido.

Além do literal de objeto passado para o serviço de geolocalização, é necessário um método de retorno de chamada para recuperação dos resultados do geolocalizador. Esse método deve passar dois parâmetros para conter *results* e *status.* Descrição dos parâmetros:

- *results*: Literal do objeto *GeocoderResults*, que pode ser uma matriz (results[]), pois o *Geocoder* pode retornar mais de uma entrada.
- *status*: Código retornado pela chamada ao serviço de geocodificação.
    - *google.maps.GeocoderStatus.OK* indica que houve êxito no geocódigo passado;
    - *google.maps.GeocoderStatus.ZERO_RESULTS* indica que houve êxito, mas não houve retorno do geocódigo passado;
    - *google.maps.GeocoderStatus.OVER_QUERY_LIMIT* indica que a cota de solicitações foi ultrapassada;
    - *google.maps.GeocoderStatus.REQUEST_DENIED* indica negação da solicitação por alguma razão;

- *google.maps.GeocoderStatus.INVALID_REQUEST* indica que está faltando algum campo para o objeto *GeocodeRequest* (*address ou latLng*)

2.4 jQuery

jQuery é uma biblioteca JavaScript que simplifica a varredura de elementos em documentos HTML, a manipulação de eventos, animação e interações AJAX para o desenvolvimento Web. jQuery é regida segundo licença estabelecida pelas regras do MIT e pelo GPL.

Principais funcionalidades do jQuery:

- Problemas de *cross-browser* - incompatibilidade entre os navegadores nas renderizações.
- Redução de código.
- Reutilização do código através de *plugins* e utilização de outros *plugins* criados por outros desenvolvedores, tornando extensível a biblioteca, pois permite a criação e inserção de novas funcionalidades.

2.5 MySQL 5.5

O MySQL é considerado o banco de dados mais rápido para utilização em aplicações Web que necessitam apenas de armazenamento e linguagem SQL.

A versão do MySQL 5.5 vem com suporte a consultas espaciais e capacidade de lidar com um número ilimitado de usuários, além de manipular mais de cinqüenta milhões de registros. Assim, o MySQL 5.5 atende todas as necessidade do sistema.

O sistema gerenciador de banco de dados MySQL é de Licença Dupla. Os usuários podem optar por usar o software MySQL como um produto de código aberto sob os termos da GNU General Public License ou pode comprar uma licença comercial padrão da Oracle.

3 MODELAGEM DO SISTEMA

3.1 Descrição do Sistema

A aplicação Web tem como composição a seguinte arquitetura: Provedor de conteúdo, Componente aplicação, Banco de dados e Aplicação cliente.

Os provedores de conteúdo são responsáveis por fornecer dados através de suas APIs, conjuntos de padrões e rotinas para que o aplicativo utilize suas funcionalidades, abstraindo os detalhes das implementações desses métodos e rotinas. No caso do sistema desenvolvido, o provedor de conteúdo utilizado foi o Google Maps.

O segundo componente, o Componente aplicação, corresponde ao local onde é executada a aplicação, conexões com o banco de dados e processamento das regras de negócio.

O terceiro componente é o banco de dados, que visa à persistência das postagens efetuadas no sistema, bem como o processamento de consultas espaciais e não espaciais.

O quarto componente é a aplicação cliente, o qual se refere à aplicação através do qual os dados oriundos do provedor de conteúdo são disponibilizados. No caso, a aplicação cliente é um navegador Web.

A aplicação tem o foco nas consultas de endereço para a cidade de São Luís – MA. Isso acontece, pois o usuário ao inserir um endereço no campo e clicar no botão de localizar, a aplicação concatena esse endereço com a *String* “sao luis“.

Assim, ao parâmetro ser passado para o serviço de geocodificação do Google Maps, o mesmo se encarrega de retornar os endereços encontrados na cidade de São Luís – MA.

3.2 Análise dos Requisitos

Nessa atividade do estágio do desenvolvimento de software, há a necessidade da descoberta de mais informações sobre o domínio da aplicação, suas funcionalidades e serviços, bem como o seu desempenho exigido e as restrições de hardware. Também há a criação da documentação do que realmente é necessário, chegando a uma noção do que o sistema deve fornecer.

3.2.1 Requisitos Funcionais

Os requisitos funcionais para um sistema descrevem a funcionalidade ou os serviços que se espera que o sistema forneça. Eles dependem do tipo de software que está sendo desenvolvido, dos usuários de software que se espera verificar e do tipo de sistema que está sendo desenvolvido. Quando expressos como requisitos de usuário, eles são normalmente descritos de um modo bastante geral, mas os requisitos funcionais de sistema descrevem a função de sistema detalhadamente, suas entradas e saídas, exceções e etc. (SOMMERVILLE, 2005, p. 84)

Os requisitos funcionais foram obtidos através da entrevista com o orientador do projeto deste trabalho e são apresentados na TABELA 1 a seguir.

REQUISITOS FUNCIONAIS	DESCRIÇÃO
RF01	Cadastrar no Sistema
RF02	Autenticar no Sistema
RF03	Localizar Endereço
RF04	Efetuar Postagem
RF05	Localizar Postagens Próximas

Tabela 1: Requisitos funcionais

3.3 Casos de Uso

Um caso de uso especifica o comportamento de um sistema ou de parte de um sistema e é uma descrição de um conjunto de sequências de ações, incluindo variantes realizadas pelo sistema para produzir um resultado observável do valor do ator. Os casos de usos podem ser aplicados para captar o comportamento pretendido do sistema que está sendo desenvolvido, sem ser necessário especificar como esse comportamento é implementado. (BOOCH, 2005, p. 227)

A Figura 3.1 apresenta o diagrama de caso de uso do sistema.

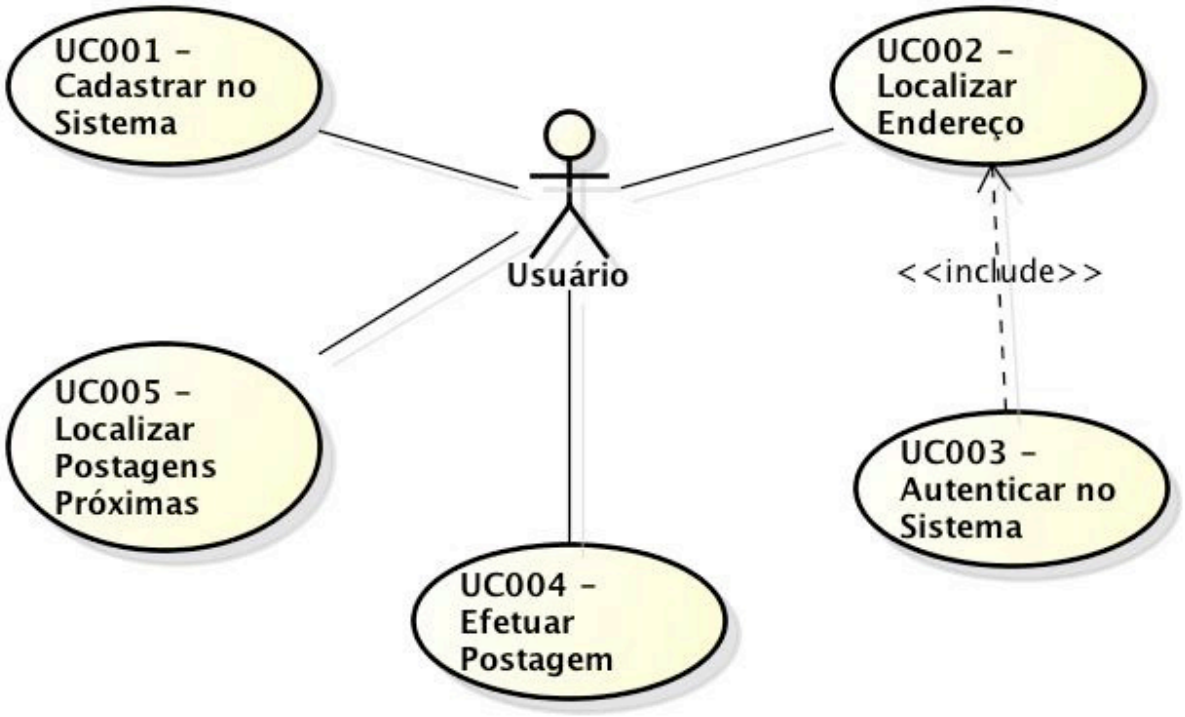

Figura 3.1: Diagrama de Casos de Uso

A descrição funcional dos casos de uso do sistema pode ser visualizada na Tabela 2. Nela, há a representação é realizada pelo nome do caso de uso, ator e descrição, que pode ser dividida em curso básico e curso alternativo. Um ator é um agente externo, o qual pode ser um humano, um dispositivo ou um sistema, que interage com o sistema. No sistema desenvolvido há um ator:

- Usuário: são os usuários que realizam a postagem das imagens dos buracos nas avenidas e ruas de São Luís. Para isso, há a necessidade do cadastro no sistema e login.

Caso de Uso	**Cadastrar no Sistema**
Ator:	Usuário
Curso Básico:	Todo usuário deve ter cadastro no sistema para efetuar postagens.
Curso Alternativo:	Caso o usuário não esteja cadastrado e tentar efetuar uma postagem, o mesmo será avisado sobre a obrigatoriedade do cadastro e será redirecionado para a página de cadastro.

Caso de Uso	**Autenticar no sistema**
Ator:	Usuário
Curso Básico:	Todo usuário precisa se autenticar para ter acesso às funcionalidades do sistema.
Curso Alternativo:	Caso o usuário não esteja cadastrado ou a senha estiver incorreta, o sistema emitirá um aviso.
Caso de Uso	**Localizar Endereço**
Ator:	Usuário
Curso Básico:	O usuário entra com o endereço que deseja localizar. O sistema efetua a geocodificação e insere um marcador da API do *Google Maps* no mapa da aplicação. Caso haja mais de um ponto geocodificado, o sistema criará marcadores para cada endereço geocodificado.
Curso Alternativo:	Caso o endereço seja inválido, o sistema notificará o usuário acerca da impossibilidade da geolocalização.
Caso de Uso	**Efetuar Postagem**
Ator:	Usuário
Curso Básico:	Após caso de uso Localizar Endereço, o usuário clicará no marcador e escolherá a imagem a ser postada. Após seleção da imagem, o usuário efetuará a postagem e será redirecionado página inicial da aplicação.
Curso Alternativo:	O usuário terá que estar autenticado na aplicação para efetuar a postagem. Caso o tamanho máximo permitido (2 MB), o sistema irá alertar o usuário acerca da limitação e não permitirá a postagem da imagem.

Caso de Uso	Localizar Postagem Próxima
Ator:	Usuário
Curso Básico:	Após caso de uso Efetuar Postagem, o sistema irá realizar uma consulta espacial na base de dados, a qual retornará todas as postagens efetuadas num raio de 200m da última postagem realizada. Havendo postagens dentro do raio da consulta, o sistema irá exibir um círculo cujo centro será a última postagem efetuada.
Curso Alternativo:	Caso não haja postagem localizada no raio da consulta espacial, o sistema efetuará a postagem conforme caso de uso Efetuar Postagem.

Tabela 2: Descrição dos casos de uso

3.4 Diagrama de Classes

Os diagramas de classes são os diagramas encontrados com maior frequência na modelagem de sistemas orientados a objetos. Um diagrama de classes mostra um conjunto de classes, interfaces e colaborações e seus relacionamentos.

Utiliza-se para realizar a modelagem estática do projeto de um sistema. A Figura 3.2 apresenta o diagrama de classes do sistema.

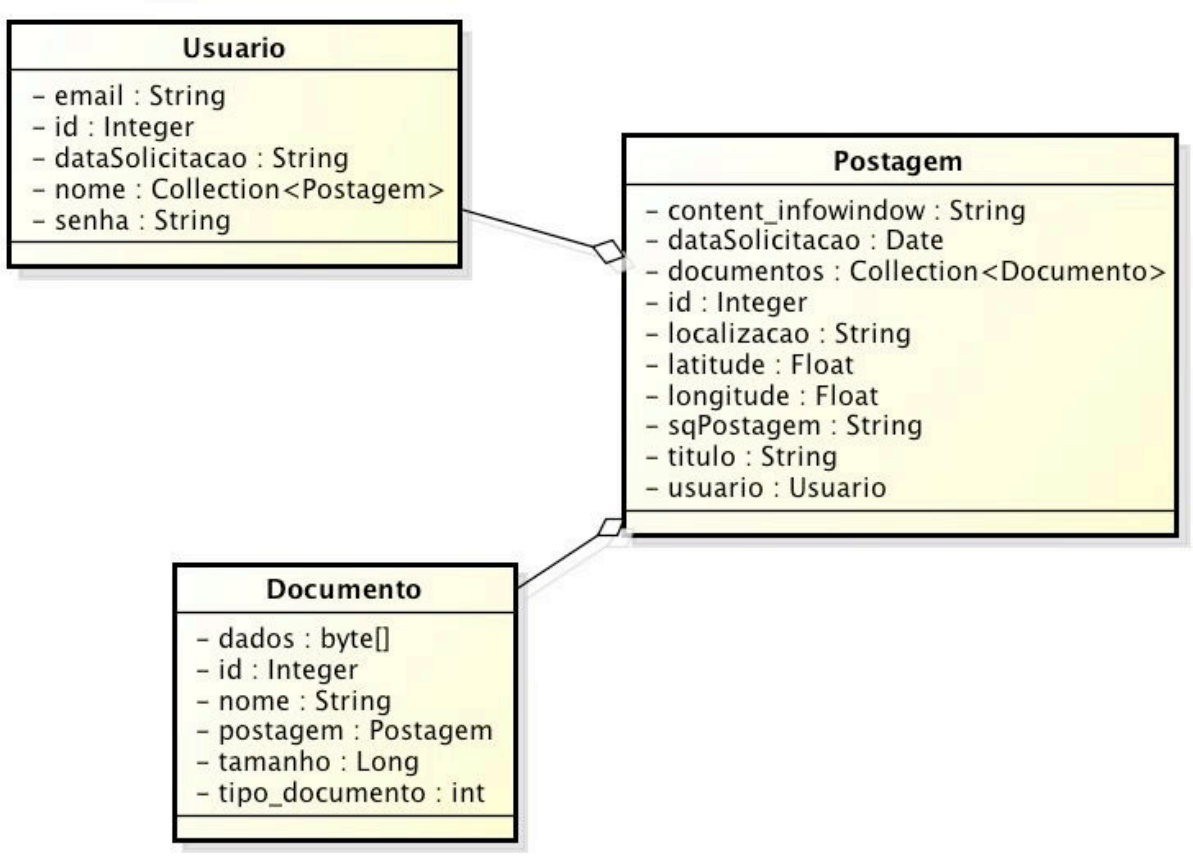

Figura 3.2: Diagrama de Classes

3.5 Diagrama de Sequência

O diagrama de sequência mostra a troca de mensagens entre as classes na realização de um caso de uso, dando ênfase a ordenação temporal das mensagens.

Utiliza-se para mostrar a implementação de um cenário específico de um caso de uso.

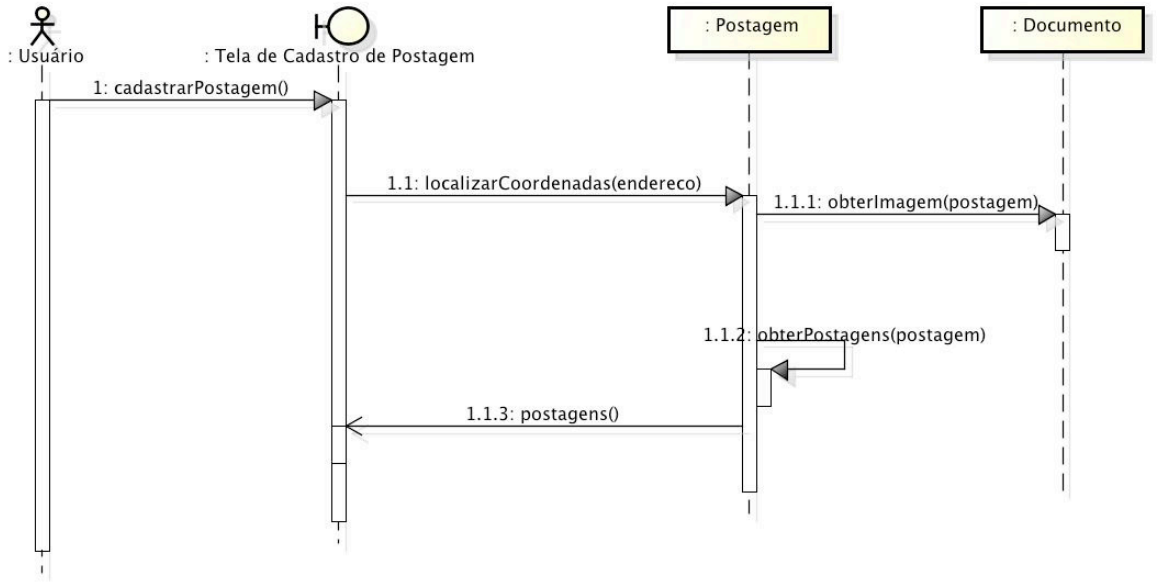

Figura 3.3: Diagrama de Sequência do caso de uso Localizar Postagens Próximas

3.6 Interface

A Figura 3.1 a seguir é a tela inicial do sistema onde é exibido a mapa, que tem como centro a cidade de São Luís – MA. Na referida Figura 3.4, é exibido um aviso informando aos usuários que só será permitida a postagem de usuários cadastrados e autenticados no sistema. Entretanto, nada impede dos usuários localizarem pontos no mapa, bem como visualizarem, ao clicar nos marcadores do mapa, as postagens já efetuadas.

Figura 3.4: Tela inicial do sistema

A Figura 3.5 a seguir é um exemplo de uma consulta de um endereço. No exemplo, foi inserido o endereço “av. Daniel De La touche”. Após a consulta, o sistema marca a localização do último endereço retornado pelo serviço de consultas espaciais do *Google Maps*. Os demais endereços, se houver, são exibidos ao lado do mapa.

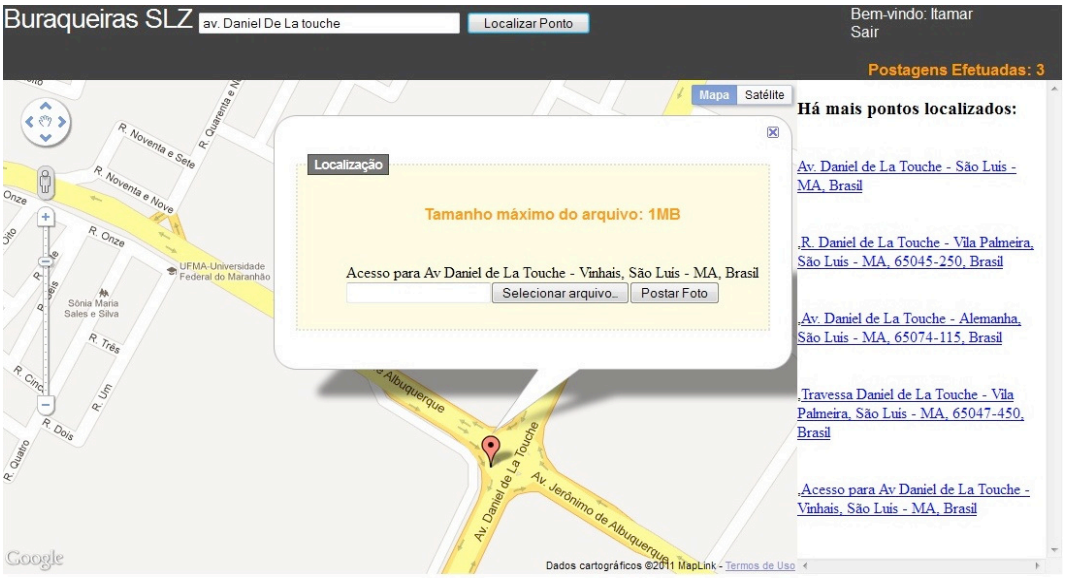

Figura 3.5: Tela de um exemplo de consulta de um endereço do sistema

A Figura 3.6 exibe a postagem efetuada. Na infowWindow – janela que é exibida nos marcadores do *Google Maps*, pode-se visualizar informações da postagem , bem como a imagem do local onde foi registrado o buraco.

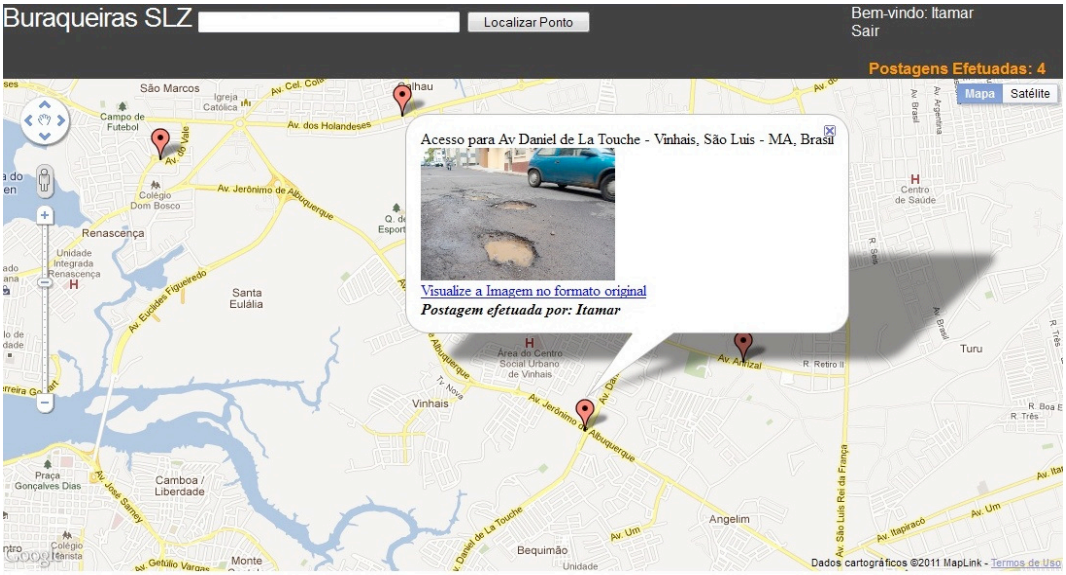

Figura 3.6: Tela de exibição da postagem efetuada

A Figura 3.7 exibe a funcionalidade da localização de postagens que estão dentro de um determinado raio de distância de uma latitude/longitude específica. A consulta (instrução SELECT) é baseada na fórmula de Haversine e retornará todas as postagens efetuadas que estão dentro do raio de 200 metros de distância da latitude e longitude da postagem efetuada. A fórmula de Haversine é usada para calcular distâncias circulares entre dois pares de coordenadas em uma esfera. (GOOGLE, 2011)

Esta é a instrução SQL que retornará todas as postagens que estejam num raio de 200 metros. O valor 6371 é utilizado para retornar a distância em quilômetros.

```
SELECT id_postagem, ( 6371 * acos( cos( radians(lat1) ) * cos( radians( lat2 ) ) *
cos( radians( lng2 ) - radians(lng1) ) + sin( radians(lat1) ) * sin( radians( lat1 ) ) ) )
AS distance FROM postagem HAVING distance < 0.2 ORDER BY distance
```

Figura 3.7: Exibe as postagens próximas a última postagem efetuada

A Figura 3.8 exibe a tela de cadastro de usuário. Após inserir os campos e efetuar o cadastro, o usuário será cadastrado no sistema e redirecionado para a página inicial da aplicação.

Figura 3.8: Tela do formulário de cadastro do sistema

E por fim, a Figura 3.9 mostra a tela de autenticação do usuário. Após a autenticação do usuário, o sistema irá exibir a página inicial com o mapa das postagens efetuadas.

Figura 3.9: Tela de autenticação.

4 CONCLUSÃO

Neste trabalho é apresentado o desenvolvimento de um sistema de cadastro de postagens de imagens de buracos nas ruas e avenidas da cidade de São Luís – MA.

O sistema utiliza tecnologias Web 2.0 e um *Framework* MVC *Web*. Há também a utilização de um provedor de serviço de mapas para *Web* e o gerenciador de banco de dados MySQL 5.5 com suporte a consulta espaciais.

Primeiramente, foram realizados estudos de quais tecnologias seriam utilizadas para o desenvolvimento da aplicação. Assim, o *Framework* MVC Web Mentawai foi escolhido, pois o não uso de arquivo XML para configuração e instalação, fez com que a total integração do *Framework* na aplicação fosse realizada de maneira simples e funcional. O *Framework* Mentawai ainda oferecia uma gama de funcionalidades, tornando dispensável o uso de qualquer outro *Framework* para o desenvolvimento da aplicação. Desde o início, o serviço de mapas Google Maps já tinha sido escolhido, pois com a fácil utilização da API e extensa documentação disponível, a integração dessa tecnologia com as demais foi realizada satisfatoriamente, pois há muitos exemplos de utilização da API em documentos HTML.

Em seguida, chegou-se a descrição da implementação deste trabalho com a exibição de suas funcionalidades, mostrando suas características e detalhes de funcionamento.

Posteriormente, mostrou-se a modelagem do sistema a partir dos requisitos funcionais da aplicação e seus casos de uso, mostrando suas funcionalidades e interações com usuários. Exibiu-se também a estrutura estática da aplicação com o diagrama de classes e o detalhamento do processo de postagem com o diagrama de seqüência.

Assim, o sistema ora apresentado mostra, com suas características e particularidades, a possibilidade de criação de aplicações, que integrem tecnologias Web com provedores de serviços de mapas.

4.1 Trabalhos Futuros

Fica como proposta de continuação deste trabalho, desenvolver um sistema mais abrangente, ou seja, permitir a participação popular em outros problemas urbanos, como falta de água, falta de energia, assaltos ou problemas de

infraestrutura relacionados à saúde e educação, como hospitais superlotados e prédios de escolas mal conservados, respectivamente.

5 REFERÊNCIAS

OSSL 2010, Sinal vermelho para o trânsito de São Luís. Disponível em: <http://www.nossasaoluis.org.br/index.php?option=com_content&view=article&id=285%3Asinal-vermelho-para-o-transito-de-sao-luis-&Itemid=2>. Acesso em: 01 de outubro de 2011.

MENTAWAI 2011, Mentawai *Web* Framework. Disponível em: <http://www.mentaFramework.org>. Acesso em 23 de julho de 2011.

MENTAWAI 2011, Arquitetura do Mentawai *Web* Framework. Disponível em: < http://book.mentaFramework.org/posts/list/10.page >. Acesso em 12 de setembro de 2011.

GOOGLE 2011, Creating a Store Locator with PHP, MySQL & *Google Maps*. Disponível em:
http://code.google.com/intl/pt-BR/apis/maps/articles/phpsqlsearch.html. Acesso em 02 de Setembro de 2011.

SOMMERVILLE, Ian. **Engenharia de Software**. 6 ed. São Paulo: Pearson Education do Brasil, 2003.

BOOCH, Grady. **UML Guia do usuário**. 2 ed. Rio de Janeiro: Elservier, 2005.