UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ – UTFPR
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

LEANDRO AUGUSTO DE CARVALHO

INTEGRAÇÃO DA API GOOGLE MAPS COM HTML5 E PHP

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2011

LEANDRO AUGUSTO DE CARVALHO

INTEGRAÇÃO DA API GOOGLE MAPS COM HTML5 E PHP

Trabalho de Diplomação apresentado à disciplina de Trabalho de Diplomação, do Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas – CSTADS – da Universidade Tecnológica Federal do Paraná – UTFPR, como requisito parcial para obtenção do título de Tecnólogo.

Orientador: Prof. Me. Fernando Schütz.

MEDIANEIRA

2011

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria de Graduação e Educação Profissional
Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas

TERMO DE APROVAÇÃO

Integração da API Google Maps com HTML5 e PHP

Por

Leandro Augusto de Carvalho

Este Trabalho de Diplomação (TD) foi apresentado às 10:00 h do dia 21 de novembro de 2011 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram argüidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. M. Sc. Fernando Schütz
UTFPR – *Campus* Medianeira
(Orientador)

Prof. Me. Claudio Leones Bazzi
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Me. Pedro Luiz de Paula Filho
UTFPR – *Campus* Medianeira
(Convidado)

Prof. Juliano Rodrigo Lamb
UTFPR – *Campus* Medianeira
(Responsável pelas atividades de TCC)

AGRADECIMENTOS

Dedico esse trabalho aos meus pais e familiares que sempre me incentivaram, deram apoio e ajudaram em minhas decisões.

Ao professor orientador Fernando Schütz que colaborou com minha formação e se mostrou um grande amigo.

Aos meus companheiros de trabalho que serviram como exemplo de dedicação, responsabilidade e colaboraram para minha formação pessoal e profissional.

Aos grandes amigos Régis Eduardo Weizenmann Gregol e Gustavo Grander, que sempre estiveram ao meu lado e me motivaram a alcançar meus objetivos.

“Viemos criativos demais para nos imporem limites”

Robert Nesta Marley

RESUMO

CARVALHO, Leandro Augusto de. **Integração da API Google Maps com HTML5 e PHP**. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

O uso de mapas e informações georreferenciadas para os mais diversos fins tem aumentado nos últimos anos e a procura por soluções computacionais com essas funcionalidades mostra uma nova área para os sistemas de informação. Este trabalho tem como finalidade integrar recursos da API (*Application Programming Interface*) *Maps* disponibilizado pela Google Inc. e dados predefinidos, usando a linguagem HTML5 e PHP para extração de indicadores analíticos e informações georreferenciadas. Diante do exposto foram apresentadas algumas características das linguagens utilizadas e suas principais funcionalidades, além do desenvolvimento de uma aplicação *Web*, como estudo experimental, para demonstrar a interoperabilidade das tecnologias envolvidas.

Palavras-chave: APIs do Google Maps, HTML5, PHP, Geoprocessamento.

ABSTRACT

CARVALHO, Leandro Augusto de. **Integração da API Google Maps com HTML5 e PHP**. 2011. Trabalho de conclusão de curso (Tecnologia em Análise e Desenvolvimento de Sistemas), Universidade Tecnológica Federal do Paraná. Medianeira 2011.

The use of maps and georeferenced information for different purposes has increased in recent years and demand for computing solutions to these features shows a new area for information systems. This work aims to integrate resources (API Application Programming Interface) Maps provided by Google Inc. and default data, using the language PHP and HTML5 for the extraction of analytical indicators. Given the above were presented some characteristics of the languages used and their main features, and developing a web application, as an experimental study to demonstrate the interoperability of the technologies involved.

Palavras-chave: Google Maps API, HTML5, PHP, GIS.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
DQL	*Doctrine Query Language*
HQL	*Hibernate Query Language*
HTML	*HyperText Markup Language*
HTTP	*HyperText Transfer Protocol*
MVC	*Model View Controller*
PHP	*HyperText Preprocessor*
RIA	*Rich Internet Aplication*
RSS	*Really Simple Syndication*
SQL	*Structured Query Language*
SVG	*Scalable Vector Graphics*
UML	*Unified Modeling Language*
XML	*Extensible Markup Language*
W3C	*World Wide Web Consortium*

SUMÁRIO

1 INTRODUÇÃO

Aplicações *Web* são aplicações por natureza distribuída, o que significa que são programas executados em mais de um computador e se comunicam através de uma rede ou servidor (NOURIE, 2006). Geralmente essas aplicações são acessadas através de um navegador de Internet.

A facilidade que a utilização de um navegador *Web* proporciona, assim como a disponibilidade que tais sistemas permitem, são algumas das características que potencializam a popularidade das aplicações *Web*. Nesse mesmo cenário têm-se os navegadores e as tecnologias voltadas para Internet em constante evolução.

O georeferenciamento se tornou um importante aliado a várias áreas comerciais, industriais e de pesquisa. A praticidade de uma pessoa conseguir se localizar, seja através de satélites ou de cruzamento de redes de celular, fez com que tais tecnologias fossem empregadas tanto em aparelhos dedicados para este fim quanto em celulares. A possibilidade de mapeamento em tempo real abre inúmeras possibilidades de aplicações no mercado de *software*.

Empresas como *Google*, *Yahoo*, *Microsoft* e *Amazon* lançaram novas ferramentas *Web* para mapeamento em um passado recente, e coletivamente esses novos desenvolvedores elevaram o nível para o setor de mapeamento através da Internet (PIMPLER, 2009).

Isso mostra uma tendência das grandes empresas e sua preocupação com desenvolvimento de ferramentas de geolocalização. Também pode-se perceber o interesse por setores comerciais em mapear geograficamente as informações de suas bases de dados para auxiliar na tomada de decisões.

Tecnologias voltadas para o desenvolvimento *Web* como PHP e HTML5 estão em constante evolução devido ao aumento dos usuários da Internet. Conseqüentemente, ferramentas cada vez mais produtivas são desenvolvidas e se popularizam entre a comunidade de desenvolvedores.

Uma dessas ferramentas é o *CodeIgniter*, um *framework opensource* de aplicações *Web* para a linguagem PHP (GRIFFITHS, 2010). Para a persistência dos

dados, o *plugin Doctrine* que fazer o mapeamento objeto-relacional, criar as tabelas no banco de dados através dos modelos da aplicação entre outras funcionalidades.

Assim, propõe-se o desenvolvimento de uma ferramenta, como estudo experimental, com uma interface intuitiva usando os recursos do HTML5 e um serviço gratuito de mapeamento geográfico como o *Google Maps*, onde o usuário poderá manipular pontos no mapa, cadastrando as principais características de cada um, para a criação de um banco de dados imobiliário. Além disso, relatórios gerenciais farão com que esses dados tornem-se indicadores para tomadas de decisões.

1.1 OBJETIVO GERAL

Desenvolver um aplicativo para controle imobiliário, como um estudo experimental para integração das tecnologias *Google Maps* API, HTML5 e PHP em um sistema *Web*.

1.2 OBJETIVOS ESPECÍFICOS

Como etapas para concluir o objetivo geral, os seguintes objetivos específicos serão propostos:

- Desenvolver um referencial teórico sobre as ferramentas e tecnologias necessárias para o desenvolvimento do projeto;
- Analisar o projeto do estudo experimental utilizando UML e desenvolvimento baseado no *AgileManifest*;
- Desenvolver um estudo experimental utilizando API do Google Maps, HTML5 e PHP;

1.3 JUSTIFICATIVA

Através de pesquisas na Internet viu-se que há certa carência em sistemas *Web* para mapeamento de informações geográficas voltadas para o ramo imobiliário. Isso fez com que houvesse o interesse em desenvolver uma aplicação que pudesse oferecer funcionalidades produtivas para o setor.

O interesse pela atual especificação do HTML também motivou a execução do projeto, que por representar uma tecnologia com muitas características novas, despertou o interesse em testar a aplicabilidade em um ambiente de desenvolvimento.

Como apresentado na introdução, as áreas de georeferenciamento e desenvolvimento *Web* estão em franca expansão, o que permite a aplicação do trabalho em diversos setores, públicos e privados, utilizando tecnologias de *software* livre, o que diminui (e muito) os custos destas instituições em relação à aquisição de ferramentas.

1.4 ESTRUTURA DO TRABALHO

Este trabalho divide-se em quatro capítulos, sendo que o primeiro trata uma breve explicação sobre o tema abordado, definição dos objetivos e a justificativa para o desenvolvimento do projeto.

O segundo capítulo fornece um breve referencial explicativo das tecnologias e serviços empregados nas etapas de desenvolvimento.

O terceiro capítulo objetiva mostrar de forma prática a utilização e integração das tecnologias e provar a interoperabilidade entre elas através de uma aplicação fictícia.

O quarto capítulo mostra as conclusões do estudo experimental e os resultados obtidos durante a evolução do projeto.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo aborda o estudo dos métodos de desenvolvimento e tecnologias utilizadas no projeto.

2.1 DESENVOLVIMENTO DE SISTEMAS PARA INTERNET

Segundo DEITEL (2008), “quando o navegador Mosaic foi apresentado em 1993, a *Web* teve uma explosão de popularidade. Ela continuou a crescer de maneira vertiginosa durante toda a década de 1990, período conhecido como a ‘bolha ponto-com’. Esta bolha explodiu em 2001”. Na primeira década do século 21, a população conviveu com uma verdadeira mudança de hábitos no acesso a informações, pois diversos novos sistemas surgiram, todos voltados à tecnologias baseadas na Internet e na *Web*.

Uma aplicação *Web* é um sistema do tipo cliente/servidor, que é executada em um ambiente distribuído, onde cada parte que compõe o programa está localizada em uma maquina diferente. O programador, e o programa, nem sempre tem consciência deste fato. (ARAÚJO, 1997)

Sistemas que utilizam o paradigma cliente-servidor já vivenciam hoje uma parte desta experiência, onde a parte relativa à interface com o usuário reside na estação do cliente e a parte de acesso aos dados, no servidor de banco de dados. A lógica da aplicação pode ficar dividida entre o cliente e o servidor. Esta é uma arquitetura em dois níveis. (ARAÚJO, 1997)

As aplicações *Web* utilizam uma arquitetura multi-nível onde as funções executadas pelas aplicações podem estar distribuídas por uma rede de computadores. Elas fazem uso de uma infra-estrutura de rede que é o padrão atual adotado pela Internet. Uma grande revolução na programação para *Web* ocorreu com o advento da RIA – *Rich Internet Application*: aplicações *Web* que oferecem a sensibilidade, recursos e funcionalidade ‘ricos’, que se aproximam das aplicações de desktop. (DEITEL, 2008).

Essa constante busca por renovação na programação para Internet e, principalmente, para *Web* é explicada por DEITEL (2010):

> Atualmente, os usuário estão acostumados a aplicações *desktop* com interfaces gráficas de usuário (GUI) ricas, tais como as utilizadas nos sistemas Mac OS X da Apple, nos sistemas Microsoft Windows, em diversos sistemas Linux e outros. Eles desejam aplicações que possam ser executadas na Internet e na *Web* e se comunicar com outras aplicações. Os usuários querem aplicar as tecnologias de banco de dados para armazenar e manipular seus dados pessoais e de negócios. Eles desejam aplicações que não estejam limitadas à estação de trabalho ou mesmo a alguma rede local de computadores, mas que possam integrar componentes da Internet e da *Web*, e bases de dados remotas. Os programadores querem usar todos esses recursos de maneira realmente portável, para que as aplicações possam ser executadas em várias plataformas sem sofrerem modificações. (DEITEL, 2010, p. 23)

Assim, vê-se que a importância de utilizar técnicas e linguagens de programação *Web* que permitam aplicar funcionalidades de *Web* 2.0, bem como deixar a aplicação mais atraente são fundamentais para o processo.

2.2 PHP

PHP, acrônimo de *HypertextPreprocessor*, é uma linguagem de programação de ampla utilização, interpretada, que é especialmente interessante para desenvolvimento para a *Web* e pode ser mesclada dentro do código HTML. A sintaxe da linguagem lembra C, Java e Perl, e é fácil de aprender. O objetivo principal da linguagem é permitir a desenvolvedores escreverem páginas que serão geradas dinamicamente rapidamente, mas você pode fazer muito mais do que isso com PHP (MANUAL DE PHP, 2011).

O PHP, entre outras características, tem suporte a sessões, permite gerenciar *upload* de arquivos, e tratamento de conexões com bancos de dados.

A linguagem surgiu em 1994 e sofreu várias modificações durante os anos. Também fez com que se popularizasse entre a comunidade de desenvolvedores *Web*. A figura 1 mostra a comparação da popularidade das linguagens de programação mais utilizadas no período de Setembro de 2011. Segundo o gráfico, o PHP encontra-se na 5ª posição atrás somente de Java, C, C# e C++. Isso mostra

que, mesmo sendo uma linguagem relativamente nova, está entre as mais utilizadas para desenvolvimento.

Programming Language	Position Sep 2011	Position Sep 2006	Position Sep 1996	Position Sep 1986
Java	1	1	5	-
C	2	2	1	1
C++	3	3	2	5
C#	4	8	-	-
PHP	5	5	-	-
Objective-C	6	39	-	-
(Visual) Basic	7	4	3	6
Python	8	7	23	-
Perl	9	6	6	-
JavaScript	10	9	21	-
Lisp	14	14	13	3
Ada	18	19	10	2

Figura 1 - Popularidade das linguagens de programação
Fonte: Tiobe (2011)

Além disso, a popularização da linguagem colabora para o incentivo da comunidade de desenvolvedores auxiliarem para correção de problemas através de fóruns. Isso gera segurança para o programador desenvolver um projeto sabendo que terá um certo suporte para as dificuldades que poderão surgir durante a fase de implementação do projeto.

2.3 HTML5

Desde 1999, o desenvolvimento da linguagem HTML (*HyperText Markup Language*) ficou estacionado na versão 4. De lá pra cá, a W3C esteve focada em linguagens como XML e SVG. Enquanto isso, os navegadores estiveram preocupados em desenvolver suas funcionalidades, como exibir páginas em abas e oferecer a integração com leitores de RSS (SARTI, 2009).

Em dezembro de 1999, foi publicada a versão 4.01 do HTML como uma recomendação do W3C. E só em janeiro de 2008 o HTML5 foi publicado como um novo projeto do W3C.

Foram inseridos novos elementos e novas funcionalidades na versão 5 do HTML com o objetivo de padronizar a maneira de se publicar o conteúdo nas páginas.

A tabela 1 mostra alguns dos elementos que foram inseridos ou que sofreram modificações da versão do HTML 4.01 para o HTML5.

Com os novos componentes de estrutura (`section`, `article`, `header` entre outros) é possível organizar o layout das páginas mantendo os elementos separados semanticamente, ou seja, pela função que desempenha e sua importância no contexto.

Além de reduzir o uso de atributos como `id` e `class` para fazer com que elementos como `div` tenham a função de sessões na página, o uso das *tags* de estrutura facilita para o desenvolvedor realizar mudanças no código pois encontra-se mais legível e próximo a linguagem natural.

Os elementos `audio` e `video` permitem que arquivos multimídia sejam executados no navegador sem o uso de complementos externos. Assim, o cliente fica livre de instalar softwares de terceiro para que o conteúdo seja mostrado corretamente.

Tabela 1 - Comparativo de elementos HTML 4.01/HTML 5 - 2011

Nome e descrição do elemento	HTML 4.01	HTML 5
DOCTYPE: Declarar ao navegador qual tipo de documento será mostrado e quais regras deverão ser utilizadas.	`<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01//EN" "http://www.w3.org/TR/html4/strict.dtd >`	`<!DOCTYPE html>`
METATAGS: marcação utilizada para descrever o contexto da página e sua codificação	`<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />`	`<meta charset="UTF-8" />`
SCRIPT: utilizar arquivos de scripts externos	`<script type="text/javascript" src="file.js"></script>`	`<script src="file.js"></script>`
LINK: utilizar arquivos externos	`<link rel="stylesheet" type="text/css" href="file.css" />`	`<link rel="stylesheet" href="file.css" />`
CANVAS: renderizar conteúdo gráfico	Não consta	`<canvas> ... </canvas>`
AUDIO: executar conteúdo de áudio	Não consta	`<audio src="file.mp3" controls></audio>`
VIDEO: executar conteúdo de vídeo	Não consta	`<video src="file.ogg" controls></video>`
HEADER: elemento responsável por definir cabeçalhos de conteúdo no contexto da página	Não consta	`<header> ... </header>`
ARTICLE: elemento responsável por definir artigos de conteúdo no contexto da página	Não consta	`<article> ... </article>`
SECTION: define uma seção de conteúdo	Não consta	`<section> ... </section>`

2.4 GOOGLE MAPS API

Durante muitos anos, usar recursos de mapas pela Internet era uma tarefa difícil devido à baixa qualidade das imagens, lentidão e uma interface não muito intuitiva para o usuário. “Enquanto todo mundo ainda estava fazendo imagens estáticas granuladas, os desenvolvedores do Google desenvolveram em silêncio uma interface limpa desde o *Gmail*. Depois levaram *terabytes* de imagens de satélite e dados de estrada, e deram tudo de graça” (PURVIS; SAMBELLS; TURNER, 2006).

Dessa forma, a API do *Google Maps* ficou disponível para a utilização pela comunidade de desenvolvedores. “Uma API define uma maneira padrão para um programa para chamar o código contido em outro aplicativo ou biblioteca. A API do *Google* define um conjunto de objetos *JavaScript* e métodos que podem ser usados para colocar mapas em páginas *Web*” (ERLE; GIBSON, 2006).

Como a API é baseada em *JavaScript*, os navegadores de Internet possuem compatibilidade com as aplicações que a utilizam. Além disso, a documentação está disponível e pode ser acessada facilmente.

Até a data desse documento, a *Google Maps JavaScript* API Versão 3 era a oficial, com previsão de remoção da versão anterior. Nessa versão existe a possibilidade de integrar funcionalidades com os serviços *Web* implementados para o *Google Maps*, como por exemplo, *Geocoding* API, *Places* API e *Directions* API.

A *Google Maps* API fornece esses serviços da *Web* como uma interface para que serviços externos solicitem dados da Google *Maps* API e usem esses dados em seus aplicativos do *Google Maps* (GOOGLE, 2011). Ou seja, além de ser possível inserir o mapa na página *Web*, com os *webservices* disponíveis, é possível obter informações detalhadas de um determinado endereço, ou até mesmo traçar uma rota do ponto geográfico onde a aplicação esteja sendo acessado até outro ponto qualquer do mapa, por exemplo.

Nesse mesmo exemplo, para determinar o endereço de um ponto, a *Places* API seria utilizada, a *Geocoding* API faria a detecção do ponto onde a aplicação estaria sendo executada e a *Directions* API seria responsável por traçar a rota através das coordenadas geográficas dos pontos.

Para exemplificar o uso da *Google Maps* API, a figura 1 mostra o trecho de código em *JavaScript* que exibe um mapa em uma página HTML.

```
1  <html>
2  <head>
3  <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
4  <script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?sensor=SET_TO_TRUE_OR_FALSE"></script>
5  <script type="text/javascript">
6    function initialize() {
7      var latlng = new google.maps.LatLng(-34.397, 150.644);
8      var myOptions = {
9        zoom: 8,
10       center: latlng,
11       mapTypeId: google.maps.MapTypeId.ROADMAP
12     };
13     var map = new google.maps.Map(document.getElementById("map_canvas"), myOptions);
14   }
15
16 </script>
17 </head>
18 <body onload="initialize()">
19   <div id="map_canvas" style="width:100%; height:100%"></div>
20 </body>
21 </html>
```

Figura 2 - Código HTML com métodos da API GoogleMaps

Fonte: (Google Maps JavaScript API V3, 2011)

Na linha 4, o atributo sensor, aceita dois valores: “`true`” ou “`false`”. Esse atributo define se a aplicação poderá utilizar a localização geográfica do usuário.

O trecho entre as linhas 6 e 14 definem a função que exibe o mapa no contexto da página. Na linha 7 é instanciado um objeto da `LatLng` que recebe dois valores numéricos, onde o primeiro é o valor de latitude e o segundo de longitude.

Esse valor será utilizado para definir o ponto central do mapa. Na linha 9 é definido o nível de *zoom* do mapa, na linha 10 o ponto central que foi definido anteriormente na variável `latLng` e na linha 11 o tipo de mapa que deverá ser renderizado.

Para o atributo `mapTypeId` podem ser usados os seguintes valores:

- ROADMAP: exibe o mapa com blocos mostrando as vias principais.
- SATELLITE: exibe o mapa com blocos de imagens de satélite.
- HYBRID: exibe uma mistura com blocos de imagens de satélite e uma camada de blocos com estradas e nomes de cidade.
- TERRAIN: exibe blocos de relevo físico.

Na linha 13 é instanciado um objeto da classe *google.maps.Map*, que recebe como parâmetro o elemento `div` que será inserido na linha 19, e as opções configuradas anteriormente.

Dessa forma é possível perceber que o desenvolvimento utilizando a API *Google Maps* é bastante intuitivo, pelo fato de ser preciso poucas configurações para ter um mapa sendo mostrado na tela. Esse exemplo objetiva mostrar uma breve introdução de utilização da API. Muito mais pode ser implementado através das várias funções e serviços *Web* que integram com o *Google Maps*.

2.5 CODEIGNITER

CodeIgniter é um *framework* para desenvolvimento de aplicações compatível com PHP5, de modo a ser executado na maioria dos servidores hospedeiros da *Web*. Ele também usa o padrão de projeto MVC (*Model View Controller*), que é uma forma de organizar sua aplicação em três partes distintas: modelos - a camada de abstração de banco de dados, visões – a parte visual, e os controladores - a lógica de negócios (GRIFFITHS, 2010).

Também faz uso do padrão de projeto *Singleton* para carregar as classes que são chamadas várias vezes na aplicação, como por exemplo, conexões de banco de dados.

CodeIgniter também implementa o padrão *ActiveRecord* facilitando para o desenvolvedor escrever consultas SQL (*Structured Query Language*) complexas e

tornando a aplicação mais legível. *ActiveRecord* também permite trocar e alterar os drivers de banco de dados sem precisar reescrever as consultas na aplicação.

A Figura 3 mostra a estrutura de diretório padrão do *CodeIgniter*. O diretório *application* é onde estão os arquivos da aplicação, como por exemplo, configuração de banco de dados, controladores, modelos, *plugins*. No diretório *database* encontram-se os drivers de conexão com banco de dados. No diretório *plugins* estão algumas extensões padrão do *CodeIgniter*.

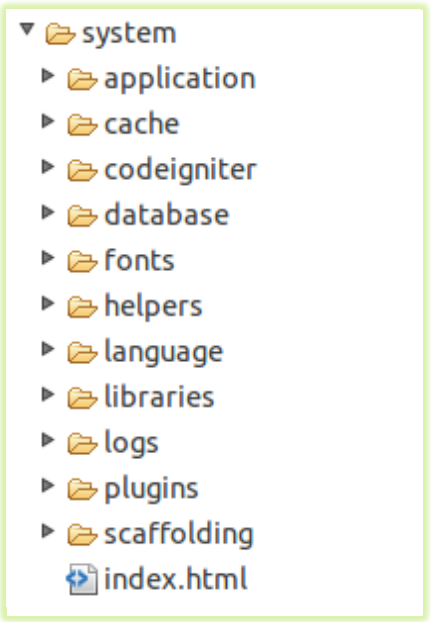

Figura 3 - Estrutura de diretórios CodeIgniter

Um dos *plugins* utilizados para desenvolver essa aplicação foi o *Doctrine* que permite fazer o mapeamento objeto relacional. Uma de suas principais características é a opção de escrever consultas de banco de dados em um dialeto SQL de propriedade objeto orientada chamado *Doctrine Query Language* (DQL), inspirado no Hibernate HQL (DOCTRINE, 2011).

Essa extensão faz com que a aplicação tenha suporte a consultas avançadas de SQL utilizando uma linguagem simples e orientada a objetos. Permite criar as tabelas no banco de dados através do mapeamento das classes do modelo e seus atributos. Tudo isso faz com que a transação de dados entre PHP e banco de dados fique transparente.

3 MATERIAIS E MÉTODOS

Neste capítulo estão descritos os métodos e materiais utilizados para o desenvolvimento da aplicação de exemplo, como estudo experimental, para ilustrar a integração das tecnologias mostradas anteriormente. Esse exemplo consiste de um sistema imobiliário onde o usuário pode marcar pontos em um mapa e cadastrar os dados do imóvel e do proprietário.

No sistema, o usuário também poderá definir áreas no mapa desenhando um polígono sobre a região desejada. Por fim, relatórios poderão ser emitidos com detalhes dos imóveis contidos em determinada área.

A metodologia utilizada para o desenvolvimento do trabalho envolveu primeiramente uma revisão bibliográfica das linguagens e tecnologias aplicadas no desenvolvimento do estudo experimental. Observa-se que para o sucesso do trabalho é uma etapa importante, pois segundo SILVA E MENEZES (2001), “a revisão de literatura é fundamental, porque fornecerá elementos para evitar a duplicação de pesquisas sobre o mesmo enfoque do tema, e favorecerá a definição de contornos mais precisos do problema a ser estudado”.

A pesquisa das tecnologias e linguagens foi de natureza aplicada e com objetivo exploratório, sendo a que mais se aplicava, pois segundo Silva e Menezes (2001), a pesquisa aplicada “objetiva gerar conhecimentos para aplicação prática dirigidos à solução de problemas específicos”, e a pesquisa exploratória “visa proporcionar maior familiaridade com o problema com vistas a torná-lo explícito ou a construir hipóteses, envolve levantamento bibliográfico”.

Para análise e projeto do sistema a UML (*Unified Modeling Language*) foi utilizada, por permitir as definições de características do software, tais como seus requisitos, seu comportamento, sua estrutura lógica, a dinâmica de seus processos e até mesmo suas necessidades físicas em relação ao equipamento sobre o qual o sistema deverá ser implantado (GUEDES, 2005).

O sistema gerenciador de banco de dados relacional utilizado foi o MySQL pois, segundo (MYSQL REFERENCE MANUAL, 2011) “foi desenvolvido

originalmente para lidar com grandes bases de dados, muito mais rápido que as soluções existentes e tem sido utilizado com sucesso em ambientes altamente exigentes. Embora em constante desenvolvimento, MySQL *Server* oferece hoje um rico e proveitoso conjunto de funções. Sua conectividade, velocidade, e segurança fazem MySQL *Server* seja altamente adaptável para acessar bancos de dados na Internet.”

3.1 ANÁLISE E PROJETO

Para a aplicação demonstrativa usando as tecnologias propostas, foram levantados alguns requisitos funcionais como:

- O sistema deve permitir que o utilizador crie pontos no mapa e cadastre as características desse ponto;
- O sistema deve permitir que o utilizador crie polígonos definindo áreas no mapa;
- O sistema deve permitir a emissão de relatórios de uma área, com base nos dados dos pontos contidos no polígono.

3.1.1 Caso de uso: Marcar ponto

Esse caso de uso refere-se à atividade de escolher dentre as ferramentas dispostas, a opção de marcar pontos no mapa. Nesse caso de uso o ator é o utilizador do sistema e a pré-condição para o início do caso de uso, é estar devidamente autenticado na aplicação.

O fluxo de eventos primários respeita a seguinte ordem:

1. Usuário seleciona a ferramenta “Marcar Ponto”;
2. Usuário clica sobre o mapa;

2.1. Com base no ponto clicado, sistema busca o endereço no *webservice Geocoding*;

2.2. Sistema preenche os campos de endereço com base no resultado do *webservice*;

3. Usuário preenche campos de características do ponto;

4. Usuário submete os dados do formulário;

4.1. Sistema valida as informações;

5. Sistema salva os dados no banco;

6. Sistema posiciona a imagem de um ícone sobre o ponto salvo no mapa;

O diagrama de seqüência da figura 4 mostra a interação do usuário no processo de marcar um ponto e cadastrar as informações.

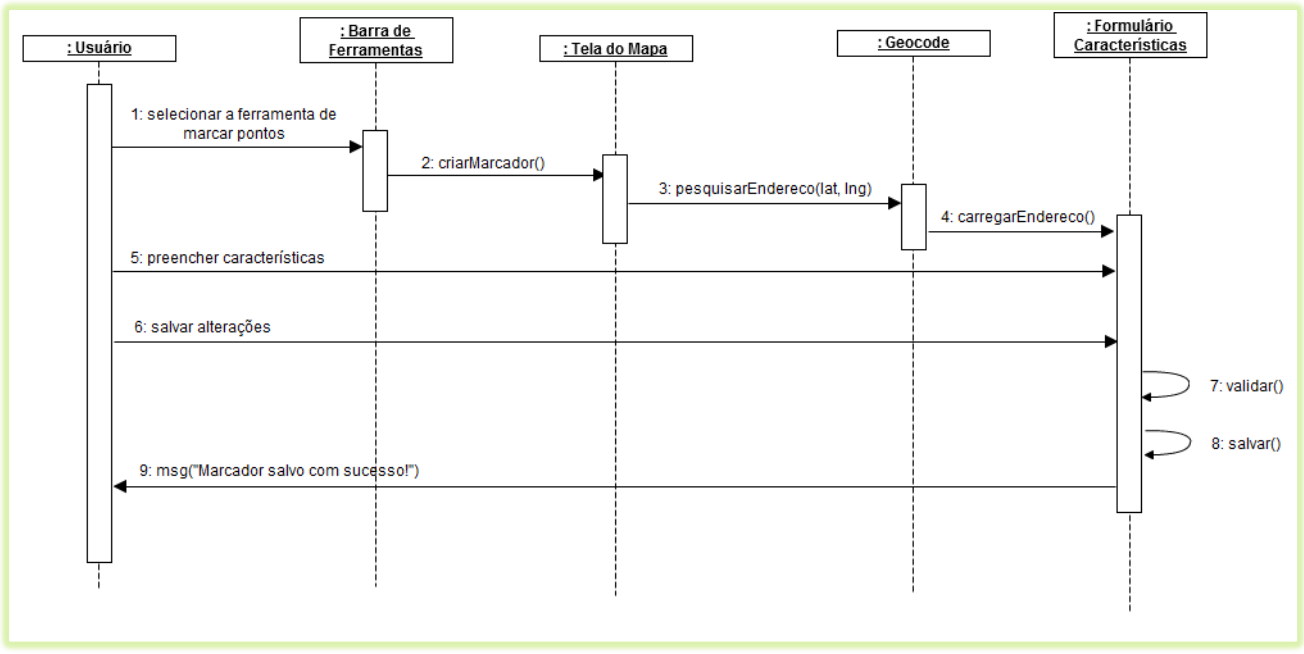

Figura 4 - Diagrama de seqüência

Para a aplicação se tornar modular e fácil de integrar novas funcionalidades, o objetivo foi deixar o banco de dados com tabelas específicas de

geoprocessamento separadas das demais. Isso facilita no momento que forem realizadas as consultas no banco de dados para manipular os pontos no mapa, reduzindo o número de dados para tratar.

Dessa forma, a tabela "*marker*", contém somente dois campos para referenciar a posição geográfica dos pontos. As características referentes a esse ponto ficam na tabela "*feature*", que utiliza um relacionamento um-para-um com a tabela "*marker*".

Para os polígonos que são usados na definição de regiões no mapa, existe uma tabela "*shape*" onde tem um campo para uma breve descrição. Como um polígono é definido por vários pontos no mapa, ele se relaciona com a tabela "*marker*" com uma relação muitos-para-muitos. Assim, verifica-se a necessidade de uma tabela associativa, que nesse caso é chamada de "*shape_marker*".

O Modelo Entidade-Relacionamento da figura 5 mostra as tabelas do banco de dados proposto e suas relações.

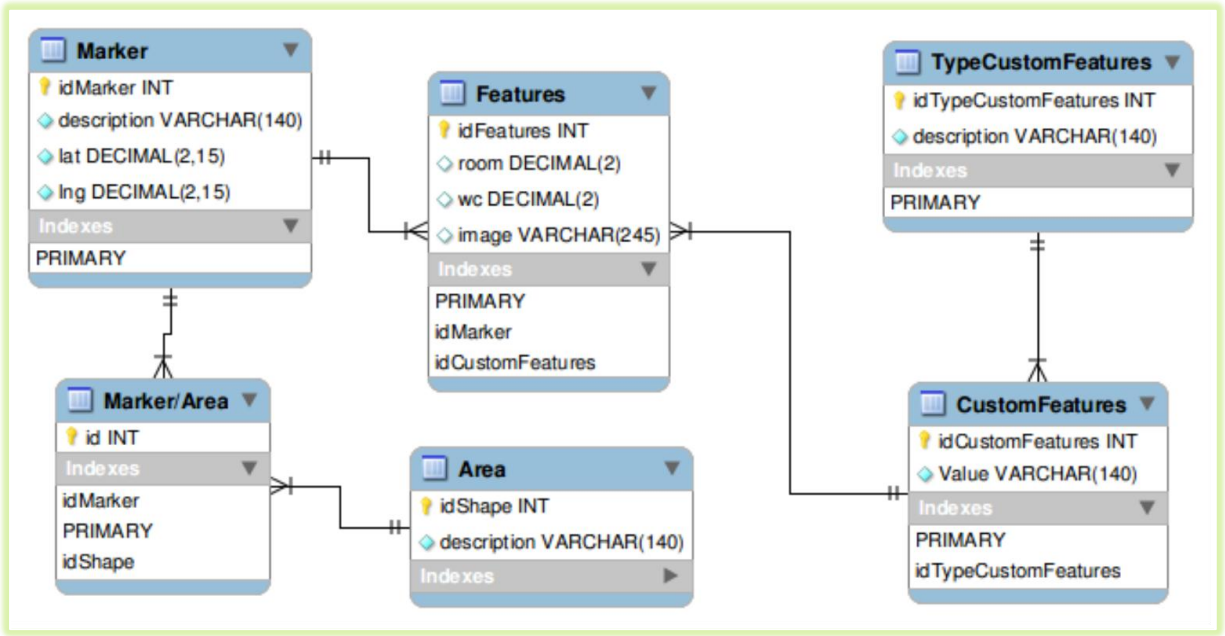

Figura 5 - Modelo Entidade-Relacionamento

3.2 CONFIGURAÇÃO DO AMBIENTE

A primeira etapa do processo de desenvolvimento consistiu em configurar o *framework* para manter a modularidade do projeto. O *CodeIgniter* traz alguns arquivos de configuração que foram editados. São eles o *database.php* e *config.php*. A figura 6 mostra a estrutura dos diretórios e onde estão localizados os arquivos referidos.

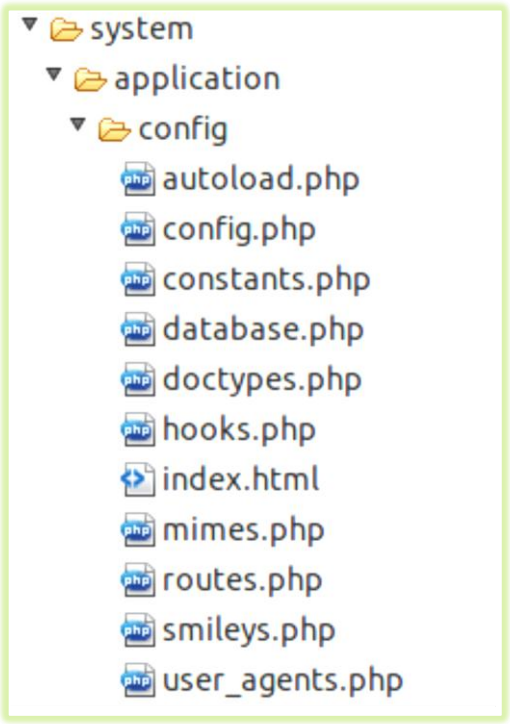

Figura 6 - Arquivos de configuração

O código do arquivo *database.php* é mostrado na figura 7. A linha 40 define o nome do servidor de banco de dados. Nesse caso foi definido como "*localhost*". Na linha 41 é definido o nome de usuário, na linha 42 a senha de acesso e na linha 43 o nome do banco de dados criado para a aplicação. As outras configurações vêm por padrão do *framework* não sendo necessário alterá-las.

```
40  $db['default']['hostname'] = "localhost";
41  $db['default']['username'] = "administrador";
42  $db['default']['password'] = "administrador";
43  $db['default']['database'] = "projeto_tcc";
44  $db['default']['dbdriver'] = "mysql";
45  $db['default']['dbprefix'] = "";
46  $db['default']['pconnect'] = TRUE;
47  $db['default']['db_debug'] = TRUE;
48  $db['default']['cache_on'] = FALSE;
49  $db['default']['cachedir'] = "";
50  $db['default']['char_set'] = "utf8";
51  $db['default']['dbcollat'] = "utf8_general_ci";
52  $db['default']['cachedir'] = "";
```

Figura 7 - Configuração do database.php

No *config.php* foi preciso alterar a linha 14 para que o endereço da aplicação utilizasse o endereço do servidor local. O código dessa alteração está na figura 8.

```
14  $config['base_url'] = "http://localhost/ProjetoTCC/";
```

Figura 8 - Configuração do config.php

4 RESULTADOS E DISCUSSÕES

Este capítulo apresenta a implementação do estudo experimental utilizando o *frameworks CodeIgniter* com *Doctrine* na linguagem PHP, juntamente com a API *Google Maps*.

4.1 PROGRAMAÇÃO DAS CLASSES DO SISTEMA

Como nesse projeto foi utilizado o *framework CodeIgniter* em conjunto com o *plugin Doctrine* para mapeamento objeto-relacional, as classes modelo estendem a classe *Doctrine_Record*. Assim, as classes controladoras da aplicação podem utilizar os métodos implementados por essa classe para fazer a persistência dos dados no banco de dados, por exemplo.

A figura 9 mostra a classe de modelo `Feature`, que representará as características do imóvel e onde há os atributos que darão origem aos campos nas tabelas do banco de dados. Na linha 2 é declarado que a classe estende a classe `Doctrine_Record` que é disponibilizada pelo *plugin Doctrine*.

A função `setTableDefinition` é usada para configurar os atributos que serão mapeados na classe e corresponderão as campos do banco de dados.

A figura 10 mostra a função `setUp` ainda da classe `Feature`. Na linha 33, a função `setTableName` define o nome da tabela no banco de dados. As relações entre outras tabelas são definidas nas linhas 34 e 38.

A linha 34 estabelece uma relação de um-para-um, através da função `hasOne` com a tabela mapeada na classe `Marker`, onde na linha 35 o atributo que representará a chave estrangeira será `marker_id` e, na linha 36, o atributo da tabela estrangeira será `id`. Nesse exemplo fica nítido o mapeamento objeto-relacional onde se usa o nome da classe como parâmetro para realizar o relacionamento, e não o nome da tabela.

```php
<?php
class Feature extends Doctrine_Record
{

    public function setTableDefinition() {

        $this->hasColumn('image', 'string', 255, array('notnull' => false));
        $this->hasColumn('dimension', 'float', 4, array('notnull' => false));
        $this->hasColumn('total_area', 'float', 4, array('notnull' => false));
        $this->hasColumn('build_area', 'float', 4, array('notnull' => false));
        $this->hasColumn('corner', 'boolean');
        $this->hasColumn('illegal_building', 'boolean');
        $this->hasColumn('curb', 'boolean');
        $this->hasColumn('water', 'boolean');
        $this->hasColumn('light', 'boolean');
        $this->hasColumn('sewer', 'boolean');
        $this->hasColumn('wells', 'boolean');
        $this->hasColumn('cesspool', 'boolean');
        $this->hasColumn('lighting', 'boolean');
        $this->hasColumn('years_looking', 'float', 4, array('notnull' => false));
        $this->hasColumn('number_of_floors', 'float', 4, array('notnull' => false));
        $this->hasColumn('edicula', 'boolean');
        $this->hasColumn('garage', 'boolean');
        $this->hasColumn('pool', 'boolean');
        $this->hasColumn('water_points', 'float', 4, array('notnull' => false));
        $this->hasColumn('condominium', 'boolean');
        $this->hasColumn('individual', 'boolean');
        $this->hasColumn('trees', 'float', 4, array('notnull' => false));
        $this->hasColumn('marker_id', 'integer');
    }

```

Figura 9 - Classe Feature

O mesmo ocorre na linha 38 onde existe a função `hasMany`, definindo assim que ocorre uma relação de um-para-muitos com a tabela mapeada na classe `CustomFeature`.

```php
    public function setUp() {
        $this->setTableName('feature');
        $this->hasOne('Marker', array(
            'local' => 'marker_id',
            'foreign' => 'id'
        ));
        $this->hasMany('CustomFeature as CustomFeatures', array(
                'refClass' => 'customFeatureFeature',
                'local' => 'id',
                'foreign' => 'feature_id'
            )
        );
```

Figura 10 - Configuração do mapeamento da classe no banco de dados

O mesmo ocorre para todas as outras classes que funcionam como modelo, alterando somente o valor dos atributos.

4.2 DEFINIÇÃO DOS MODELOS

Depois de definido os modelos, é preciso criar as tabelas no banco de dados. O *Doctrine* implementa uma função chamada `createTableFromModels()` que executa uma *query* de criação das tabelas usando a configuração dos modelos mapeados. A figura 11 mostra o código que executa tal função.

```
8            Doctrine_Core::createTablesFromModels();
```

Figura 11 - Função createTableFromModels()

Para mostrar o formulário de preenchimento das características, foi usado o método `field_data` que retorna uma lista dos campos de determinada tabela no banco de dados.

Esse método é executado na classe controladora chamada `controller/feature_controller.php` no método `index`, responsável por carregar o formulário de cadastro.

A figura 12 mostra o trecho de código referido. A linha 17 contem a execução do método `field_data` buscando os campos da tabela `feature`. O retorno desse método é passado como parâmetro para o formulário de cadastro das características. Essa chamada está na linha 21.

```
13    public function index(){
14
15        $this->CI =& get_instance();
16        $this->current_table = "feature";
17        $fields = $this->CI->db->field_data($this->current_table);
18
19        $data = array('fields' => $fields );
20
21        $this->load->view('features_list', $data);
22    }
```

Figura 12 - Classe feature_controller.php

Com isso é possível gerar o formulário com campos de preenchimento definidos pelo tipo primitivo do campo na tabela. E o HTML5, com os campos `date` e `number` deixam a interface de usuário com o aspecto visual mais agradável.

A figura 13 mostra o código do arquivo `views/feature_form.php` responsável por gerar o formulário.

O código da linha 1 mostra a utilização do módulo de formulários do *CodeIgniter*. A saída do método `form_open` é a marcação do formulário em HTML. O parâmetro representa o valor do atributo `action` do formulário, ou seja, o método que irá tratar esse formulário será o `submit` que está na classe `feature_controller`.

O valor da variável `fields` passado pelo controlador é recuperado na linha 4, onde entra em um laço de repetição para verificar o tipo das variáveis e mostrar os campos do formulário.

Na linha 8 o método `form_input` faz com que apareça na tela um campo de texto. Os parâmetros passados nesse método são atributos para esse campo. Perceba que além dos atributos do HTML 4.01, pode-se adicionar os que surgiram no HTML 5, como o `required`, por exemplo, que faz a validação do campo automaticamente para que seu valor não seja nulo. Isso deixa clara a dinamicidade do framework, e como é simples para programar novos padrões sem precisar desenvolver uma versão mais recente.

Na linha 13 e 18 ocorre o mesmo onde existe os campos do tipo *date* e `number` que foram incluídos na especificação 5 do HTML. Na linha 30 o método `form_submit` cria um botão com função de envio para o controlador e a linha 31 fecha o formulário.

Nesse código fica claro a utilidade da classe `Form` implementada pelo *CodeIgniter* facilitando para o desenvolvedor criar as visões no projeto sem precisar usar o HTML puramente, ou seja, diretamente no código sem usar métodos do PHP.

```php
<?php echo form_open("feature_controller/submit"); ?>
<fieldset>
    <legend>Dados do Imóvel</legend>
    <?php foreach ($fields as $field) {
            if($field->type == "string" && $field->name != "image") { ?>
                <label for="<?php echo $field->name; ?>">
                    <strong><?php echo $this->lang->line($field->name); ?></strong>
                    <?php echo form_input(array('name' => $field->name, 'required' => 'true')); ?>
                </label>
            <?php } elseif($field->type == "date") { ?>
                <label for="<?php echo $field->name; ?>">
                    <strong><?php echo $this->lang->line($field->name); ?></strong>
                    <input type="date" name="<?php echo $field->name; ?>" />
                </label>
            <?php } elseif ($field->type == "real") { ?>
                <label for="<?php echo $field->name; ?>">
                    <strong><?php echo $this->lang->line($field->name); ?></strong>
                    <input type="number" name="<?php echo $field->name; ?>" min="0" />
                </label>
            <?php } elseif($field->name == "image"){ ?>
                <label for="image">
                    <strong>Imagem</strong>
                    <input type="file" name="image" />
                </label>
            <?php }
        } ?>
</fieldset>

<?php
    echo form_submit("submit","Enviar");
    echo form_close();
?>
```

Figura 13 - Formulário feature_form.php

Com esse código, o resultado será a tela mostrada na figura 14. Na figura foi usada uma folha de estilos somente para melhorar a aparência, mas o conteúdo gerado foi exclusivamente do código mostrado anteriormente.

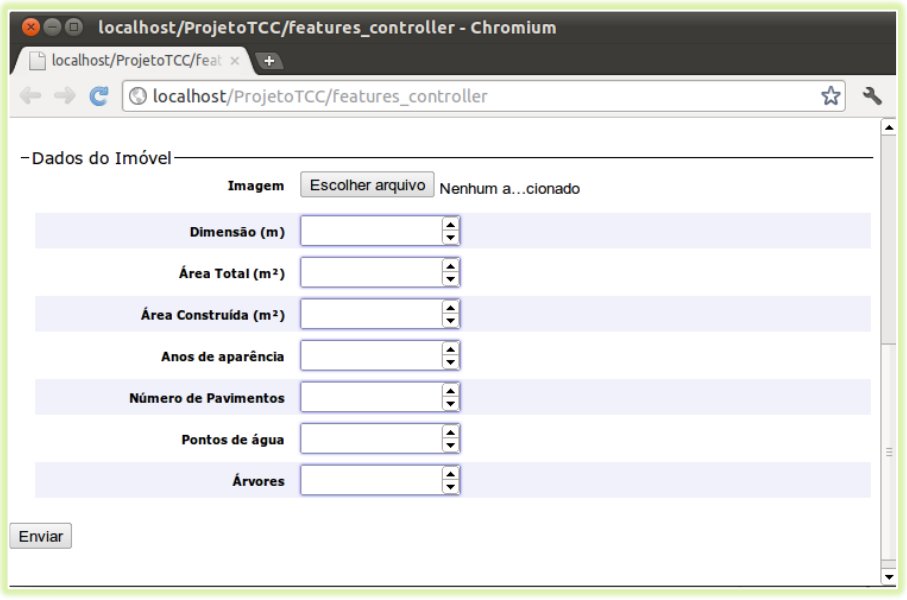

Figura 14 - Formulário de características no navegador

4.3 INTEGRAÇÃO COM A API GOOGLE MAPS

Até então foi mostrado a parte de integração do HTML 5 com o PHP. A API *Google Maps* entra no contexto através do arquivo `tools.js`. Esse arquivo implementa todas as funções que foram utilizadas no desenvolvimento do projeto e é carregado junto com a página principal da aplicação. Para facilitar o entendimento do código e a proposta do trabalho serão mostradas as partes do código que referem-se às funções do *JavaScript* e manipulação dos elementos do HTML 5, envolvendo as funções e serviços do *GoogleMaps*.

A figura 15 mostra um trecho da função `initialize()` que é chamada no momento que a página é carregada e executa as funções para criar o mapa na tela. Na linha 12 cria-se uma variável que recebe um objeto da classe *google.maps.LatLng*, onde os parâmetros são valores de latitude e longitude respectivamente. Essa variável é usada para centralizar o mapa em determinado ponto quando ele for carregado na linha 21.

A linha 14 verifica se existe algum valor no item com a chave "`zoom`" que está no `localStorage` do navegador através do método `getItem()`. Segundo (PILGRIM, 2011) "HTML 5 *Storage* é baseado em pares de chave/valor. Pode-se armazenar dados com base em uma chave e recuperar o valor através da mesma chave. A chave com o nome é uma *string*. Os dados podem ser de qualquer tipo suportado pelo *JavaScript*, incluindo *strings*, booleanos, inteiros ou *floats*". Caso houver algum valor nessa chave, este será atribuído na variável de mesmo nome na linha 15.

A variável `options` da linha 19 representa um conjunto de configurações para configurar a aparência do mapa. A linha 20 define o nível de aproximação do mapa, a linha 21 o centro do mapa, a linha 22 faz com que as opções de tipo de mapa (Satélite, Mapa, Terreno) apareçam.

Na linha 23 é definido o estilo desse controle de opções, através das constantes definidas pela API.

O intervalo entre as linhas 25 e 27 definem atributos de estilo do controle de navegação e posição desse controle.

E na linha 31 é onde define-se o elemento do HTML que receberá as imagens do mapa.

```
10 function initialize() {
11     var zoom = 16;
12     var home = new google.maps.LatLng(-24.62722, -54.225984);
13 
14     if (window.localStorage.getItem("zoom") > 0) {
15         zoom = window.localStorage.getItem("zoom");
16     }
17 
18     /* Opções para o mapa */
19     var options = {
20         zoom: parseInt(zoom),
21         center: home,
22         mapTypeControl: true,
23         mapTypeControlOptions: {style: google.maps.MapTypeControlStyle.DROPDOWN_MENU},
24         navigationControl: true,
25         navigationControlOptions: {
26             style: google.maps.NavigationControlStyle.ANDROID,
27             position: google.maps.ControlPosition.BOTTOM},
28         mapTypeId: google.maps.MapTypeId.SATELLITE
29     };
30 
31     map = new google.maps.Map(document.getElementById("map_canvas"), options);
```

Figura 15 - Trecho de código para inicializar o mapa

Ainda referindo-se ao `localStorage`, a figura 16 mostra o código do intervalo das linhas 123 e 125 onde é definido que o evento "`zoom_changed`", ou seja, o evento de mudança de nível de aproximação, dispara o método da linha 124. Nessa linha a função `window.localStorage.setItem()` faz com que o chave "*zoom*" receba o valor da função `map.getZoom()`, retornando um valor inteiro do nível de aproximação atual do objeto `map`.

Isso faz com que, quando a página for recarregada, o nível de *zoom* será o mesmo da última vez em que o mapa foi acessado.

```
123     google.maps.event.addListener(map, 'zoom_changed', function() {
124         window.localStorage.setItem("zoom", map.getZoom());
125     });
```

Figura 16 - Evento zoom_changed

A função que permite criar pontos no mapa é mostrada na figura 17. Esse método funciona basicamente como o anterior, porém nesse caso o evento é de clique no mapa, como visto no parâmetro “`click`” da linha 177. na linha 178 é chamada a função *placeMarker* onde o parâmetro é a posição `latLng` (latitude e longitude do *GoogleMaps*).

```
177        google.maps.event.addListener(map, 'click', function(event) {
178            placeMarker(event.latLng);
179        });
```

Figura 17 - Evento click para criar o marcador

Um trecho do código da função `placeMarker` é mostrado na figura 18. A linha 133 é exatamente o ponto onde foi clicado no mapa. A linha 134 define em qual mapa será renderizado o ponto e a linha 135 permite que esse ponto não fique fixo no mapa, podendo ser arrastado.

Para gravar esse ponto no banco de dados, primeiro será preciso colocar as informações em um formulário para então enviar para a classe controladora.

O *GoogleMaps* oferece um componente interessante chamado `infowindow`, que simula um balão de diálogo sobre o marcador. Nesse espaço é possível colocar código HTML. Na linha 138 a variável `form` recebe o formulário, onde o atributo `action` irá executar a função `save` da classe `marker_controller`. Na linha 140, o campo de latitude recebe o valor passado na função *placeMarker*. O mesmo ocorre na linha 141, mas para longitude.

Na linha 144 é atribuído o conteúdo do `infowindow`, que nesse caso, é o formulário para confirmar a posição do marcador e salvar no banco de dados.

E por fim, a linha 146 faz com que seja executada a função `infowindow.open`, que expande o balão de diálogo.

```
130 function placeMarker(location) {
131
132     var marker = new google.maps.Marker({
133         position: location,
134         map: map,
135         draggable: true
136     });
137
138     var form =  "<form action='marker_controller/save' method='post' id='form_marker' >" +
139                     "<input type='hidden' value='" + location.lat() + "' name='lat' />" +
140                     "<input type='hidden' value='" + location.lng() + "' name='lng' />" +
141                     "<input type='submit' value='Salvar' />" +
142                 "</form>";
143
144     infowindow.setContent(form);
145
146     infowindow.open(map, marker);
147 };
```

Figura 18 - Função *placeMarker*

O efeito no mapa é mostrado na figura 19. Quando o usuário clicar em Salvar, os dados de latitude e longitude serão mandados através do método `post` do formulário para a classe `marker_controller`.

Figura 19 - Marcador com *infowindow*

A código que implementa o método `save` é mostrado na figura 20. Na linha 31 é instanciado um objeto da classe `Marker`, responsável por tratar os marcadores, ou os pontos no mapa. Na linha 32 atribui-se o valor do parâmetro `lat`,

que vem através no método `post`, para o campo de latitude do marcador. O mesmo ocorre para o campo de longitude na linha 33. Para persistir o objeto no banco de dados, o *CodeIgniter* tem implementado o método `save`, sendo necessário somente fazer a chamada da função como mostrado na linha 34.

Observa-se também como o CodeIgniter faz o tratamento de variáveis de sessão no PHP. Na linha 36, o método `unset_userdata()` limpa todas as variáveis de sessão e na linha 37 é configurado a variável id com o identificador do marcador salvo.

```
29    public function save() {
30
31        $marker = new Marker();
32        $marker->lat = $this->input->post('lat');
33        $marker->lng = $this->input->post('lng');
34        $marker->save();
35
36        $this->session->unset_userdata();
37        $this->session->set_userdata('id', $marker->id);
38
39        redirect("/");
40    }
```

Figura 20 - Função *save*

Para conseguir desenhar polígonos no mapa os procedimentos são semelhantes. Primeiro faz-se com que o evento no mapa crie um polígono e não um marcador como feito anteriormente. A figura 21 ilustra essa mudança no código.

Na linha 183 cria-se uma variável que receberá os vértices do polígono. Na linha 184 é executada a função que elimina os eventos que estão associados ao mapa. Nesse caso, o evento a ser retirado será o de clique no elemento `map`. A linha 185 faz com que a linha seguinte seja executada quando houver o evento de clique no mapa.

```
183         var path = polygon.getPath();
184         google.maps.event.clearListeners(map, 'click');
185         google.maps.event.addListener(map, 'click', function(event) {
186             path.push(event.latLng);
187         });
```

Figura 21 - Evento click para criar polígono

Para carregar os pontos e os polígonos, uma consulta SQL é executada e o retorno gera um arquivo XML. A figura 22 mostra a consulta que busca os pontos referentes aos polígonos e a geração do arquivo XML.

Na linha 60 é executada a consulta que retorna os dados do polígono e os valores de latitude e longitude. Na linha 65 cria-se um `array` que fará a configuração dos nós pai e dos elementos no arquivo XML.

Na linha 67 usa-se o método da classe utilitária de banco de dados do *CodeIgniter* chamada `dbutil`. Nessa classe o método `xml_from_result`, recebe dois parâmetros. O primeiro é o resultado da consulta e o segundo é a configuração do arquivo de destino. Na linha 69 é gerado o arquivo `xml/shapes.xml` usando a saída do método `xml_from_result`.

```
60          $query = $this->db->query(" SELECT s.id, s.color, s.description, m.lat, m.lng
61                                      FROM marker m, shape s, shape_marker sm
62                                      WHERE m.id = sm.marker_id
63                                      AND s.id = sm.shape_id");
64
65          $config = array ('root' => 'shapes', 'element' => 'shape');
66
67          $xml_output = $this->dbutil->xml_from_result($query, $config);
68
69          write_file('xml/shapes.xml', $xml_output);
```

Figura 22 - Consulta dos pontos do polígono

Dessa forma, o método em *JavaScript* chamado `createAreas()` percorre os nós desse arquivo e cria os polígonos na inicialização do mapa. A figura 23 mostra o código do método em questão.

A linha 257 mostra que a variável `xmlhttp` fará uma transferência de dados do servidor para o cliente. Na linha 258, o método open faz a requisição através dos parâmetros `method`, que indica qual método será utilizado para transferir o arquivo, `url`, que indica o caminho do arquivo e `async` mostra se a transferência será assíncrona ou não.

A linha 259 envia a requisição na linha 261 a o retorno da requisição é atribuído em uma variável. Na linha 263 é criado um objeto da classe

`google.maps.Polygon` que representará os polígonos criados. Na linha 265 é instaciado um objeto da classe `google.maps.MVCArray` que funciona semelhante à um objeto `array` do *JavaScript* com algumas funções específicas para manipular os dados.

```
255 function createAreas() {
256
257     var xmlhttp = new XMLHttpRequest();
258     xmlhttp.open("GET","xml/shapes.xml",false);
259     xmlhttp.send();
260
261     var shapes = xmlhttp.responseXML;
262
263     var poly = new google.maps.Polygon();
264
265     var path = new google.maps.MVCArray();
```

Figura 23 - Método createAreas

Feito isso, é possível percorrer os nós usando um laço de repetição e montar o adicionar na variável `path` que definirá os vértices do polígono. Na figura 24, a linha 267 mostra a declaração do laço de repetição. Percebe-se que a condição para sair do *loop* usa o elemento extraído anteriormente do XML.

Na linha 269 é usado é método `push`, que acrescenta um elemento na ultima posição da lista. Nesse caso, o elemento que será adicionado é do tipo `google.maps.LatLng` onde a latitude e a longitude também são elementos provenientes do arquivo XML.

```
267 for (var i = 0; i < shapes.getElementsByTagName("shape").length; i++) {
268
269     path.push(new google.maps.LatLng(
270             shapes.getElementsByTagName("lat")[i].childNodes[0].nodeValue,
271             shapes.getElementsByTagName("lng")[i].childNodes[0].nodeValue));
272
273 }
```

Figura 24 - Percorrer lista de pontos

Ao fim da sequencia de repetição, é executado o código da figura 25, onde na linha 274, o polígono recebe a coleção de coordenadas que servirão de vértices

para o polígono. E por fim, a linha 275 representa a parte do código que faz a área aparecer no mapa.

```
274             poly.setPath(path);
275             poly.setMap(map);
```

Figura 25 - Definir vértices do polígono

A figura 26 é a captura da tela do navegador mostrando a aplicação com o mapa, pontos e marcadores. Nesse exemplo foram alterados os ícones para mostrar a possibilidade de personalizar os marcadores.

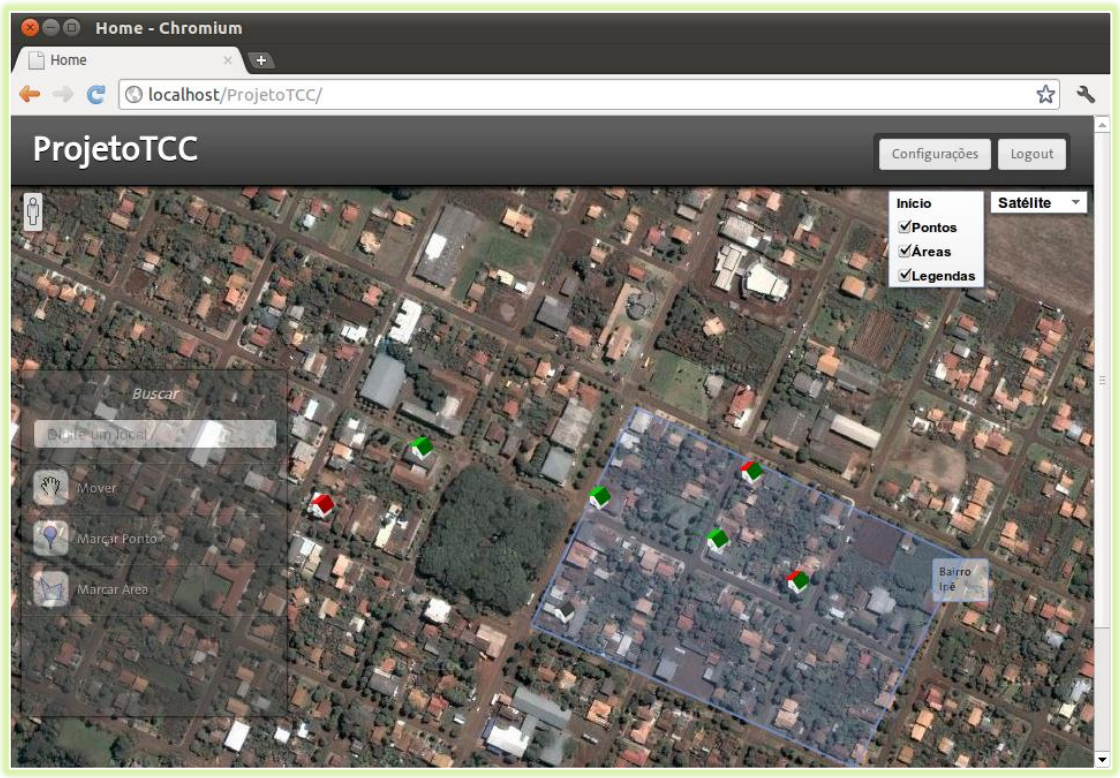

Figura 26 - Captura da tela do mapa

Para extrair informações dos pontos contidos em determinada área, utilizou-se uma função *JavaScript* de uma extensão do *Google Maps* chamada `containsLatLng`. A figura 27 mostra o uso desse método. Na linha 55, `polygon` é um objeto da classe `google.maps.Polygon` já com os vértices definidos e

`point` é um objeto `google.maps.LatLng` referente à um ponto qualquer no mapa.

```
55      if (polygon.containsLatLng(point)) {
56          /* ... */
57      };
```

Figura 27 - Método containsLatLng

Essa função retorna `true` caso o ponto esteja contido na área referenciada. Com isso, é possível obter os dados referentes e mostrar em forma de tabelas, gráficos ou uma simples listagem.

A figura 28 mostra a implementação representando de forma gráfica a área da figura 26 e filtrando por uma característica específica. Nesse caso, a característica escolhida foi o nível de escolaridade do proprietário do imóvel. Isso mostra como o uso da aplicação pode ser amplo através dos dados mapeados, não se restringindo somente às informações do imóvel. Para melhorar a forma de apresentação foi utilizado o *plugin JavaScript* chamado *Highcharts*.

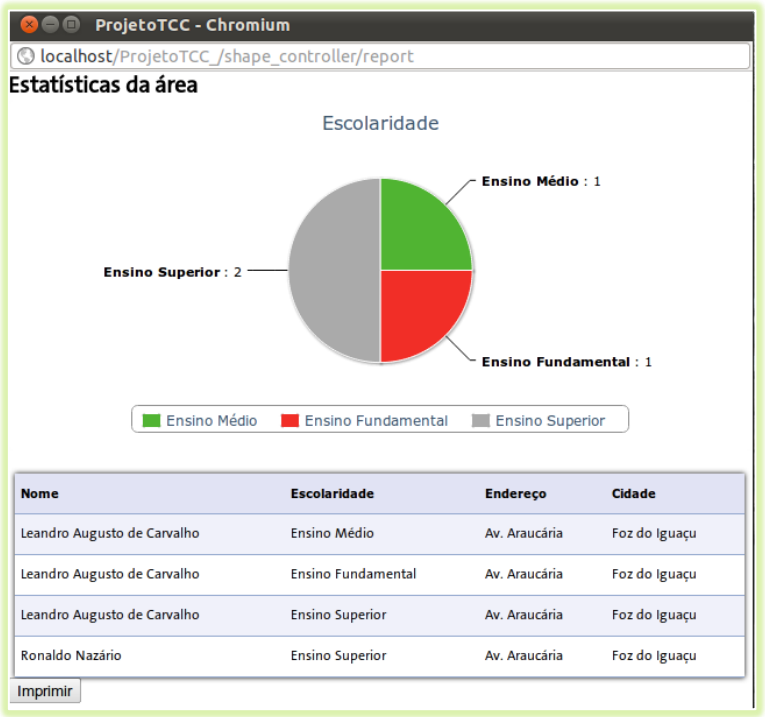

Figura 28 - Representação gráfica dos dados

Esse foi só um exemplo de representação que pode ser feita usando os dados contidos nos pontos. Dependendo a informação que queira adquirir, basta alterar a característica que define o gráfico. Dessa forma, o sistema pode ser aplicada para diversas áreas de conhecimento para gerar indicadores analíticos.

Assim, percebe-se que API do *GoogleMaps* permite ser integrada de diversas formas e possui recursos interessantes, bastando somente realizar algumas configurações para que operem de forma satisfatória com recursos externos como PHP.

5 CONSIDERAÇÕES FINAIS

Os capítulos seguintes mostram as conclusões obtidas com a evolução do projeto e sugestões para trabalhos futuros.

5.1 CONCLUSÃO

A grande quantidade de serviços *Web* e a constante evolução das tecnologias, são fatores que permitem aos desenvolvedores buscarem a interoperabilidade entre os recursos, para assim criar novas aplicações.

Esse trabalho provou que pode ser desenvolvido um sistema que use serviços disponíveis gratuitamente na Internet, deixando a aplicação com bom desempenho e fácil manutenção através das melhores práticas de programação.

Também demostrou a eficiência de usar um *framework* de desenvolvimento pela agilidade permitida no processo de programação, visto que o cronograma para o projeto era relativamente curto perante o escopo e as funcionalidades implementadas, além das muitas características que propiciam facilidade para o programador.

A especificação do HTML5 ainda não se encontra em uma versão definitiva, o que pode inibir os desenvolvedores a manter os projetos na nova versão da linguagem. Porém, com o grande número de elementos já documentados e disponibilizados, pode-se trabalhar em aplicativos para *Web* com essa nova tecnologia sem grandes problemas.

É interessante observar a experiência que o desenvolvedor adquire ao trabalhar em projetos que utilizem serviços estáveis e popularizados como o *Google Maps*, pois o programador conta com fóruns de ajuda e uma comunidade de desenvolvedores que contribuem para auxiliar nos problemas que poderão serem encontrados.

5.2 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Com esse trabalho, surgiu a ideia de implementar uma aplicação mais robusta e com mais funcionalidades para um determinado ramo de negócio.

Outro fator observado durante o projeto foi que, existem poucos *frameworks* para desenvolvimento de ferramentas geoespaciais voltadas para o PHP. Ou seja, uma iniciativa no sentido de usar algum *framework* de PHP e modificá-lo para facilitar a integração com o *Google Maps* API, por exemplo, poderia deixar a etapa de programação mais ágil e menos suscetível a erros.

6 REFERENCIAS BIBLIOGRÁFICAS

ARAUJO, João Gualberto R. **O Desenvolvimento de Aplicações *WEB***. Boletim bimestral sobre tecnologia de redes produzido e publicado pela RNP – Rede Nacional de Ensino e Pesquisa. ISSN 1518-5974 em 3 de Outubro de 1997. Disponível em < http://www.rnp.br/newsgen/9710/n5-3.html>. Acessado em 28 de Outubro de 2011.

DEITEL, Paul J; DEITEL, Harvey M. **Ajax, Rich Internet Applications e desenvolvimento *Web* para programadores**. 1ª Edição. Tradução Célia Taniwaki e Daniel Vieira. São Paulo: Pearson Prentice Hall, 2008.

DOCTRINE. Doctrine – Object Relational Mapper. Disponível em <http://www.doctrine-project.org/projects/orm>. Acessado em 27 de Outubro de 2011.

ERLE, Schuyler; GIBSON, Rich. **Google Maps Hacks**. 2006. Editora O'Reilly

GOOGLE. Google Maps Api Webservices. Disponível em <http://code.google.com/intl/pt-BR/apis/maps/documentation/*Web*services/>. Acessado em 25 de Outubro de 2011.

GRIFFITHS, Adam. 2010. **CODEIGNITER 1.7 PROFESSIONAL DEVELOPMENT**. EditoraPackt Publishing.

GUEDES, Gilleanes T. A. **UML 2 – GUIA DE CONSULTA RÁPIDA.** 2ª Edição. São Paulo: Novatec Editora, 2005.

MANUAL DE PHP. Disponível em <http://www.php.net/manual/en/preface.php>. Acessado em 25 de Outubro de 2011.

MYSQL REFERENCE MANUAL. What is MySQL? Disponível em <http://dev.mysql.com/doc/refman/5.1/en/what-is-mysql.html>. Acessado em 31 de Outubro de 2011.

NOURIE, Dana. 2006. **JAVA TECHNOLOGIES FOR *WEB* APPLICATIONS.** Disponível em <http://www.oracle.com/technetwork/articles/javase/*Web*apps-1-138794.html#*Web*app>. Acessado em 11 de Julho de 2011.

PILGRIM, Mark. **DIVE INTO HTML5**. Disponível em http://mislav.uniqpath.com/diveintohtml5/. Acessado em 19 de Setembro de 2011.

PIMPLER, Eric. 2009. **MASHUP MANIA WITH GOOGLE MAPS**. Disponível em < http://geochalkboard.files.wordpress.com/2009/01/google-maps-pdf-article-v51.pdf>. Acessado em 17 de Agosto de 2011.

PURVIS, Michael; SAMBELLS, Jeffrey; TURNER, Cameron. **Beginning Google Maps Applications with PHP and Ajax: From Novice to Professional**. 2006. Editora Appress.

SARTI, Erika. 2009. **Introdução ao HTML 5**. Disponível em < http://www.infowester.com/introhtml5.php>. Acessado em 03 de Agosto de 2010.

SILVA, Edna Lúcia da; MENEZES, Estera Muszkat. **METODOLOGIA DA PESQUISA E ELABORAÇÃO DE DISSERTAÇÃO**. 3ª edição. Laboratório de Ensino a Distância da UFSC. 2001.

TIOBE. TIOBE Programming Community Index for September 2011. Disponível em <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>. Acessado em 18 de Setembro de 2011.