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

BERNARDO SILVEIRA ESTEVES VALE

AMBIENTE COMPUTACIONAL PARA ANÁLISE DE DADOS ESPACIAIS

TRABALHO DE DIPLOMAÇÃO

MEDIANEIRA
2013

BERNARDO SILVEIRA ESTEVES VALE

AMBIENTE COMPUTACIONAL PARA ANÁLISE DE DADOS ESPACIAIS

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

Orientador: Prof Dr Claudio Leones Bazzi

Co-orientador: Prof Me Everton Coimbra de Araújo

MEDIANEIRA

2013

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

Ambiente Computacional para Análise de Dados Espaciais

Por

Bernardo Silveira Esteves Vale

Este Trabalho de Diplomação (TD) foi apresentado às 14:00 h do dia 21 de março de 2013 como requisito parcial para a obtenção do título de Tecnólogo no Curso Superior de Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná, *Campus* Medianeira. Os acadêmicos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado com louvor e mérito.

Prof. Dr Claudio Leones Bazzi UTFPR – *Campus* Medianeira (Orientador)	Prof. Me Alessandra B. G. Hoffmann UTFPR – *Campus* Medianeira (Convidado)
Prof. Me Fernando Schutz UTFPR – *Campus* Medianeira (Convidado)	Prof. Msc. Juliano Rodrigo Lamb UTFPR – *Campus* Medianeira (Responsável pelas atividades de TCC)

AGRADECIMENTOS

Agradeço primeiramente a Deus pela oportunidade de aprender neste tempo de provações, algumas lições aprendidas neste período de graduação serão levadas por toda a vida, e tenho plena certeza que estas serão fundamentais para o meu crescimento profissional e pessoal.

Agradeço aos meus familiares pelo apoio incondicional nas horas fáceis e difíceis, em especial a minha mãe Elisabeth que fez o possível para que eu pudesse me preocupar apenas com os estudos.

Aos amigos e colegas que me apoiaram e foram minha família no período de graduação, creio não ser possível atingir meus objetivos acadêmicos sem eles, em especial aos meus grandes amigos Alessandro Minkowski e Marcelo Oliveira.

“Agradeço todas as dificuldades que enfrentei; não fosse por elas, eu não teria saído do lugar. As facilidades nos impedem de caminhar. Mesmo as críticas nos auxiliam muito.”
(Chico Xavier)

RESUMO

VALE, Bernardo S. E. Ambiente computacional para análise de dados espaciais. 2013. Trabalho de Diplomação, Universidade Tecnológica Federal do Paraná. Medianeira 2013.

Os Sistemas de Informação Geográfica (SIG) são soluções que dão suporte aos profissionais da área do geoprocessamento, tais soluções proporcionam auxilio nas tomadas de decisão e na melhoria da produção cartográfica.

A construção de SIG's utilizando a plataforma Web pretende trazer vantagens encontradas somente neste modelo de desenvolvimento, como, facilidade de acesso, manutenção e escalabilidade simplificada, porém, existem também limitações ao usar esta plataforma, pois, não se encontra um leque tão grande de recursos como no *Desktop*.

O presente trabalho constitui-se na criação de uma aplicação *Web* baseada na tecnologia *Google Web Toolkit* (GWT), utilizando banco de dados espaciais para armazenamento e controle dos recursos espaciais, além de utilizar o banco PostgreSQL para trabalhar cruzando dados de diversas fontes. Esta aplicação tem como objetivo trazer aos seus usuários uma solução em análise de dados espacial e estatística.

Palavras-chave: GWT, PostGIS, OpenLayers, SIG, Geoprocessamento, Estatísticas

RESUMO EM LINGUA ESTRANGEIRA

VALE, Bernardo S. E. Ambiente computacional para análise de dados espaciais. 2013. Trabalho de Diplomação, Universidade Tecnológica Federal do Paraná. Medianeira 2013.

The Geographic Information's Systems (GIS) are solutions that give support to geoprocessing area professionals, such solutions provide help on decision making and on cartography production improve.

The built of GIS using Web platform aspire to bring advantages founded only on this development model, like, access easiness, maintenance and simplified scalability, on the other hand, some limitations exists using this platform, because Web doesn't have a large group of resources like Desktop platform.

The present work its establish the creation of a Web application based of Google Web Toolkit (GWT) technology using spatial data base to store and control spatial resources, further use PostgreSQL database to work with different sources data. This application has the objective of bring to their users a solution on spatial data analysis and statistics.

Palavras-chave: GWT, PostGIS, OpenLayers, SIG, Geoprocessament, Statistics

LISTA DE SIGLAS

AJAX	*Asynchronous JavaScript*
BLOB	*Binary Large Object*
CSS	*Cascading Style Sheet*
GPS	*Global Positioning Systems*
IDE	*Integrated Development Environment*
JRE	*Java Runtime Library*
MVC	*Model-View-Controller*
MVP	*Model-View-Presenter*
RIA	*Rich Internet Applications*
RPC	*Remote Procedure Call*
SGBD	Sistema Gerenciador de Banco de Dados
SGBDOR	SGBD Objeto-relacional
SIG	Sistema de Informação Geográfico
WKT	*Well-know-text*
XML	*Extensible Markup Language*

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

A necessidade de analisar mapas em conjunto, aliado a carência de apoio intelectual para auxílio em questões relacionadas ao georreferenciamento é suprida através de sistemas computacionais modernos chamados de Sistemas de Informações Geográficas (SIG). O apoio destes sistemas para análise de dados espaciais vem se tornando popular e tornam-se cada vez mais poderosos na medida em que a computação e as metodologias de análise deste tipo de dado evoluem.

Grande parte destas ferramentas foram construídas para o ambiente *Desktop* impossibilitando que estes facilitadores estejam presentes na *internet*, porém, a necessidade de que estas ferramentas sejam encontradas na *Web* viabiliza a construção de novas gerações de SIG's focados nesta plataforma.

O presente trabalho teve como objetivo, desenvolver um Sistema de Informações Geográficas (SIG), utilizando a plataforma *Web*, e que permita realizar a análise espacial de dados, além de possibilitar a análise estatística tradicional.

Utilizando a aplicação o usuário terá em mãos uma forma de manipular mapas diferentes simultaneamente, guardar na nuvem dados espaciais e também com valores numéricos para aplicar estatísticas espaciais, criar novas abordagens em mapas e controlar todas as informações que o mesmo, criar e enviar ao sistema.

1.1 OBJETIVO GERAL

Desenvolver uma solução computacional que permita a manipulação e análise de dados espaciais por meio de um ambiente *Web*.

1.2 OBJETIVOS ESPECÍFICOS

- Realizar estudo referente ao funcionamento prático do banco de dados espacial PostGIS;
- Realizar o estudo referente ao desenvolvimento WEB com GWT e seu *framework* para desenvolvimento *Model-View-Presenter*, *GWT-Platform*;
- Desenvolver a solução computacional para a manipulação dos dados espaciais utilizando as tecnologias estudadas nas etapas anteriores;
- Estudar as principais funções estatísticas, e implementá-las à solução computacional;
- Testar a solução computacional e analizar os resultados obtidos.

1.3 JUSTIFICATIVA

As primeiras soluções em geoprocessamento começaram a surgir na década de 50 com o intuito de aperfeiçoar a produção de mapas, porém, a tecnologia na época ainda era muito limitada, e a solução tornava-se inviável, só na década de 70 essas soluções começaram a se tornar viáveis devido ao avanço na área de hardware, nesse período surge a sigla GIS (*Geography Information System*), ou SIG.

Os SIG's são soluções que se popularizaram devido a aplicações como Google Maps, Google Earth, porém estas aplicações já existem a bastante tempo, e são imprescindíveis ao geoprocessamento, ao estudo da cartografia,geografia,botânica, etc.

Os bancos de dados espaciais surgem então para fornecer apoio aos sistemas de informação geográficos como forma de armazenamento e manipulação de dados espaciais.

Análise é o exame minucioso de uma coisa em cada uma das suas partes.. A análise é essencial para tomadas de decisão em qualquer área, não sendo diferente para os SIG, logo, importante em uma aplicação de manipulação de dados espaciais.

O Projeto visa dar continuidade aos estudos iniciados junto ao grupo de pesquisar em bancos de dados da UTFPR – Medianeira, onde foram estudados conceitos em bancos de dados espaciais, linguagem procedural de bancos, e iniciado o projeto proposto para o Trabalho de Diplomação. A proposta é de construir uma aplicação funcional, que seja leve, capaz de analisar, guardar e manipular dados espaciais. Acessíveis por meio da internet por usuários com conhecimento básico em cartografia, para o sistema conseguir orientar o banco (O usuário final não sabe o funcionamento de um banco de dados espacial, sua necessidade e de apenas manipular esses dados).

1.4 ESTRUTURA DO TRABALHO

O trabalho foi dividido em três capítulos principais, Referencial Teórico; Materiais e Métodos; Resultados.

O Referencial teórico aborda os conceitos necessários para que todos os elementos da aplicação sejam compreendidos, neste capítulo são abordadas todas as tecnologias usadas tais como suas finalidades.

Materiais e métodos mostram o que foi usado na construção da aplicação, a arquitetura geral e também a análise do projeto de forma simplificada, mostrando os requisitos e a visão geral da aplicação.

O Resultado é o capítulo que descreve o sistema construído na sua forma final, a descrição de todas suas funcionalidades e como o usuário irá interagir com a mesma.

2 REFERENCIAL TEÓRICO

2.1 GEOPROCESSAMENTO

Geoprocessamento é o conjunto de técnicas de coleta, tratamento, manipulação e apresentação de informações espaciais (FURTADO, Vasco, 2002). Este conjunto de técnicas veem sendo explorado desde a metade do século passado quando, pode ser somado a informática para analisar grupos de mapas e dados.

O principal objetivo do geoprocessamento é fornecer ferramentas para que os diferentes usuários determinem as evoluções espacial e temporal de um fenômeno geográfico e as inter-relações entre diferentes fenômenos (ASSAD & SANO, 1993).

Para atingir estes objetivos, o Geoprocessamento usa como base de estudo a informação geográfica, ou seja, uma informação de um determinado local (Latitude, Longitude, Altitude), estes dados podem ser coletados de diversas formas e o avanço destas formas, vem acarretando na evolução do Geoprocessamento em si. Entre as principais formas de aquisição, está o Sensoriamento Remoto (SR), o *Global Positioning Systems* (GPS) e a Aerofotogrametria

2.1.1 Sensoriamento Remoto

O SR é a ciência e a arte de obter informações sobre objetos, áreas ou fenômeno a partir da análise de dados adquiridos por um dispositivo sem entrar em contato com o objeto, area ou fênomeno investigado (KUMAR, 2005).

A condição principal imposta por essa definição clássica, que é o sensor estar a uma distância remota do objeto, estabelece a base para definir o sensoriamento remoto numa concepção um pouco mais científica, que é regida segundo os seguintes preceitos:

i) exigência: ausência de matéria no espaço entre o objeto e o sensor;

ii) consequência: a informação do objeto é possível de ser transportada pelo espaço vazio;

iii) processo: o elo de comunicação entre o objeto e o sensor é a radiação eletromagnética, a única forma de energia capaz de se transportar pelo espaço

Com base nesses preceitos, uma definição mais científica que se pode dar ao SR seria: SR é uma ciência que visa o desenvolvimento da obtenção de imagens da superfície terrestre por meio da detecção e medição quantitativa das respostas das interações da radiação eletromagnética com os materiais terrestres (MENESES & ALMEIDA, 2012).

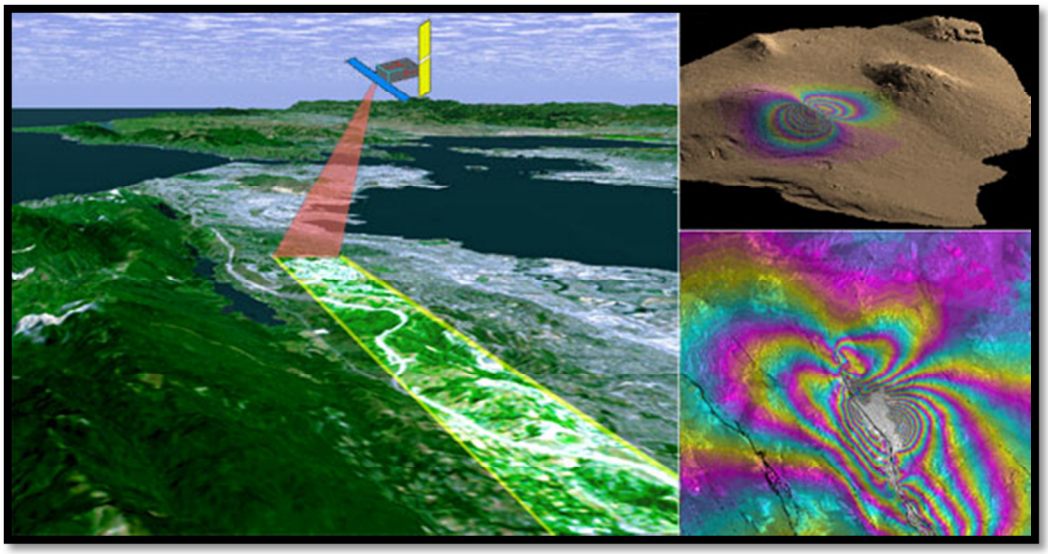


Figura 1 - Exemplo de sensoriamento remoto através do satélite como sensor
Fonte: ZEBKER (2013)

2.1.2 Global Positioning Systems

O GPS é um sistema de navegação baseado em satélites construído pelo departamento de defesa dos EUA no início da década de 70 e que permite prover informações de posicionamento e tempo continuamente em qualquer parte do globo terrestre e sobre qualquer condição climática (O GPS divide-se em três componentes, referentes a constelação de satélites, estações de controle e sistema de recepção). Os satélites são responsáveis em emitir informações sobre seu posicionamento, as estações de controle avaliam as rotas dos satélites e efetuam as devidas correções nas informações, e o receptor obtêm as informações dos satélites, identifica os satélites em orbita e triangula sua posição para formar sua coordenada 3D (Latitude, Longitude e Altitude) (Adaptado EL-RABBANY, 2002).

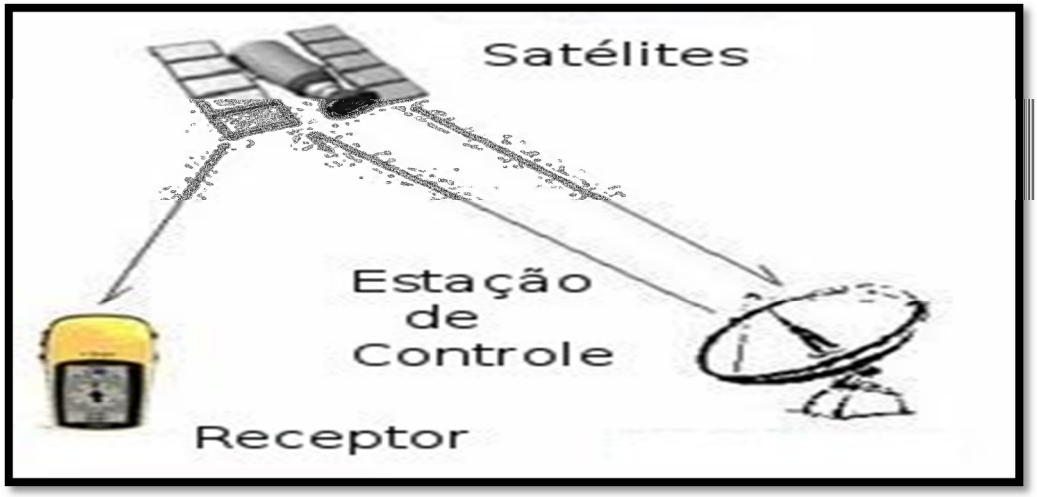


Figura 2 - Componentes do GPS
Fonte: Autoria Própria

2.1.3 Aerofotogrametria

Aerofotogrametria é uma tecnologia para obtenção de dados topográficos através de processos de gravação, medição e interpretação de imagens fotográficas e padrões de energia eletromagnética radiante e outras fontes. Utiliza-se de câmeras especializadas atreladas a aeronaves para obtenção das imagens e interpretes capazes de reconhecer e diferenciar os objetos contidos na imagem (BRITO & COELHO, 2002).

A técnica de aerofotogrametria é a mais utilizada para a elaboração de mapeamento de áreas de grandes dimensões principalmente em razão de apresentar produtos precisos a custos relativamente baixos (RIBEIRO, 1995). Essa técnica faz uso de fotografias aéreas tomadas com câmara aerotransportada rigorosamente calibrada (com distância focal, parâmetros de distorção de lentes e tamanho de quadro do negativo conhecidos), montada com o eixo ótico próximo da vertical em uma aeronave devidamente preparada e homologada para receber o sistema (MATOS, 2005).

A Figura 3 exibe uma ilustração exemplo da técnica de Aerofotogrametria.

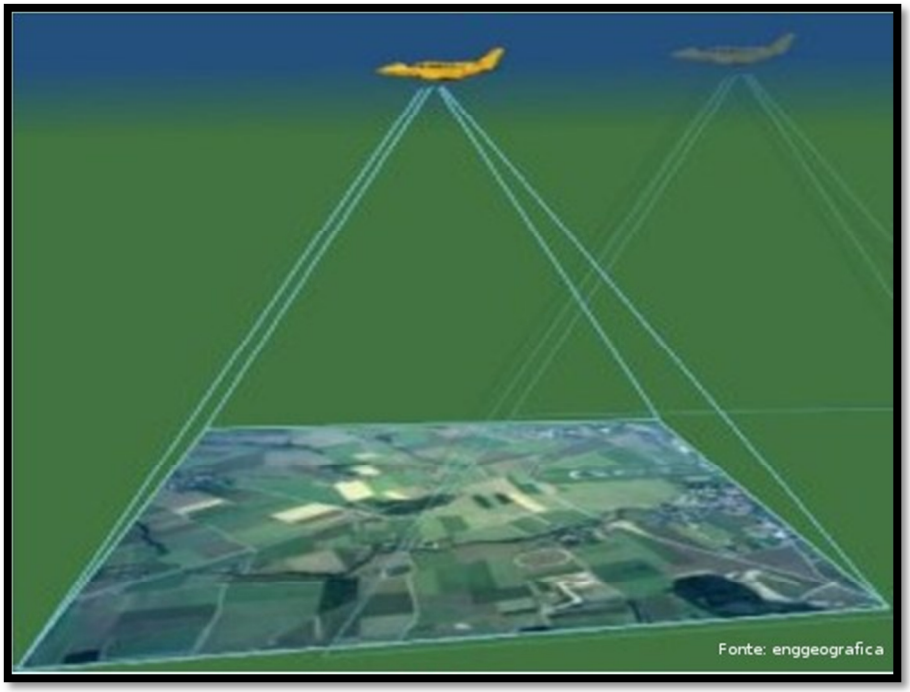


Figura 3 – Mapeamento através da Aerofogrametria
Fonte: SECRETARIA DE EDUCAÇÃO DO PARANÁ (2013)

2.2 SISTEMAS DE INFORMAÇÃO GEOGRÁFICOS

"Qualquer conjunto de procedimentos manuais ou baseados em computador destinados a armazenar e manipular dados referenciados geograficamente" (ARONOFF, 1989)

"Uma tecnologia de informação que armazena, analisa e exibe dados espaciais ou não - SIG é de fato uma tecnologia e necessariamente não é limitada a um simples e bem definido sistema de computador" (PARKER, 1988).

"Uma entidade institucional, refletindo uma estrutura organizacional que integra tecnologia com um banco de dados, expertise e continuado apoio financeiro" (CARTER, 1994)

"Um sistema de informações baseado em computador que permite a captura, modelagem, manipulação, recuperação, análise e apresentação de dados georreferenciados" (WORBOIS, 1995).

SIGs são ferramentas que permitem manipular dados espaciais, produzindo informações úteis para tomada de decisão, visando colocar em prática as ações necessárias aos resultados obtidos. Esses sistemas se aplicam a qualquer tema que possua informações vinculadas a um

determinado lugar no espaço e os seus dados possam ser representados através de mapas (SILVA, 2007).

As múltiplas operações apresentadas por um SIG podem ser classificadas em três grupos, de acordo com o fim a que se destinam (adaptado INPE, 2004):

- Gerenciamento de banco de dados geográficos – armazenamento, integração e recuperação de dados de diferentes fontes, formatos e temas dispostos em um único banco de dados;

- Análises espaciais – a partir de um banco de dados geográficos, são efetuadas combinações e cruzamentos de dados por meio de operações geométricas e topológicas cujo resultado é a geração de novos dados;

- Produção cartográfica – operação de edição e configuração da representação gráfica dos dados visando a visualização através da tela ou na forma impressa.

2.2.1 Estrutura e Arquitetura de um SIG

Segundo (CÂMARA & QUEIROZ,1998), pode-se afirmar que um SIG têm os seguintes componentes em sua estrutura:

- Interface com usuário;
- Entrada e integração de dados;
- Funções de consulta e análise espacial;
- Visualização e plotagem;
- Armazenamento e recuperação de dados (organizados sob a forma de um banco de dados geográficos).

Estes componentes se relacionam de forma hierárquica (Figura 4). No nível mais próximo ao usuário, a interface homem-máquina define como o sistema é operado e controlado. No nível intermediário, um SIG deve ter mecanismos de processamento de dados espaciais (entrada, edição, análise, visualização e saída). No nível mais interno do sistema, um sistema de gerência de bancos de dados geográficos oferece armazenamento e recuperação dos dados espaciais e seus atributos (CÂMARA & QUEIROZ, 1998).

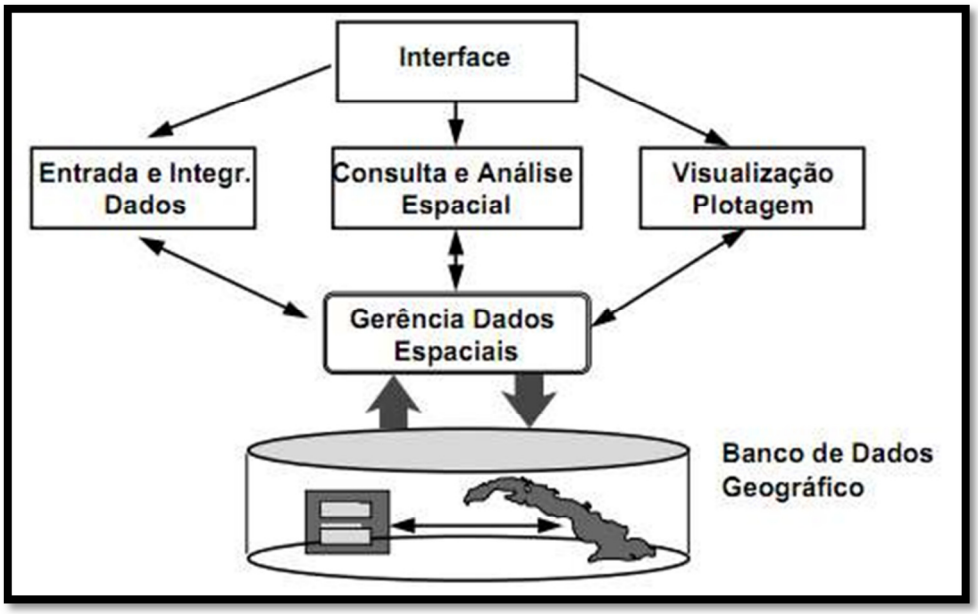


Figura 4 - Elementos que compõe um SIG
Fonte: CÂMARA & QUEIROZ (1998)

Nota-se que cada subsistema presente na Figura 4 têm funções e objetivos diferentes, porém, todos devem estar contidos num SIG.

Sobre a arquitetura de um SIG, pode-se destacar que a principal diferença está na forma como os dados são gerenciados no SGBD. Atualmente, dispõe-se de três formas de gerenciar os dados georreferenciados.

2.2.1.1 Arquitetura Dual

Nesta arquitetura os dados georreferenciados não ficam no SGBD e são referenciados através de dados alfanuméricos únicos que fazem a ligação lógica com o dado georreferenciado presente no SIG.

Principal Vantagem: Como nesta arquitetura o SGBD só gerencia dados alfanuméricos pode-se trabalhar com qualquer SGBD relacional do mercado.

Principal Desvantagem: Controlar as representações dos objetos espaciais já que estes dados no SGBD são apenas uma ligação lógica com o SIG, isto gera uma implementação complexa e dificulta operações como consulta, gerência de transações, integridade, etc.

2.2.1.2 Arquitetura Integrada Baseada em SGBD

Nas arquiteturas integradas os dados georreferenciados são inseridos e controlados pelo SGBD. Os campos georreferenciados são armazenados em forma de *Binary Large Object* (BLOB)[1] junto com quais queres informações alfanuméricas.

Principal Vantagem: O SGBD têm controle dos objetos, sendo assim, a gerência de transações, integridade e concorrência é mais fácil, pois utilizará recursos do SGBD.

Principal Desvantagem: Impossibilidade de captura da semântica devido os dados serem binários e limitações para manipular dados binários.

2.2.1.3 Arquitetura Integrada Baseada em Extensões

Este tipo de arquitetura limita-se somente a SGBD's objeto-relacionais (SGBDOR). As extensões contêm funcionalidades e procedimentos que permitem armazenar, acessar e analisar dados espaciais de formato vetorial (CÂMARA & QUEIROZ, 1998).

Principal Vantagem: Diversidade de funções, procedimentos e tipos de dados para armazenar, acessar e analisar dados espaciais, facilitando o controle destes dados.

Principal Desvantagem: Falta de padronização das extensões quanto a SQL, cada extensão criou seu modelo dificultando integrações entre os mesmos.

A Figura 5 exibe graficamente a diferença entre as arquiteturas Dual e integradas.

---

[1] Estrutura de dados para armazenar informações (dados) binários de qualquer formato. Tipo conhecido para quase a totalidade dos SGBD's

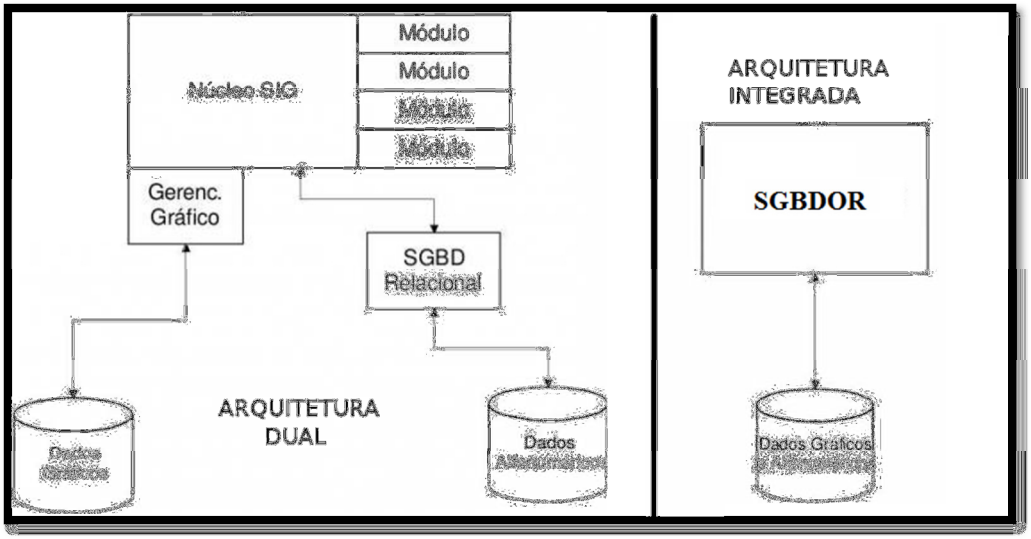


Figura 5 - Arquitetura de um SIG, comparação entre Dual e Integrada
Fonte: Autoria Própria

2.2.2 Dados Georreferenciados

Consistem em informações referentes a uma posição no globo terrestre (Latitude, Longitude e Altitude), estes dados são referenciados graficamente pelos SIG e utilizados como objeto de estudo nestes sistemas.

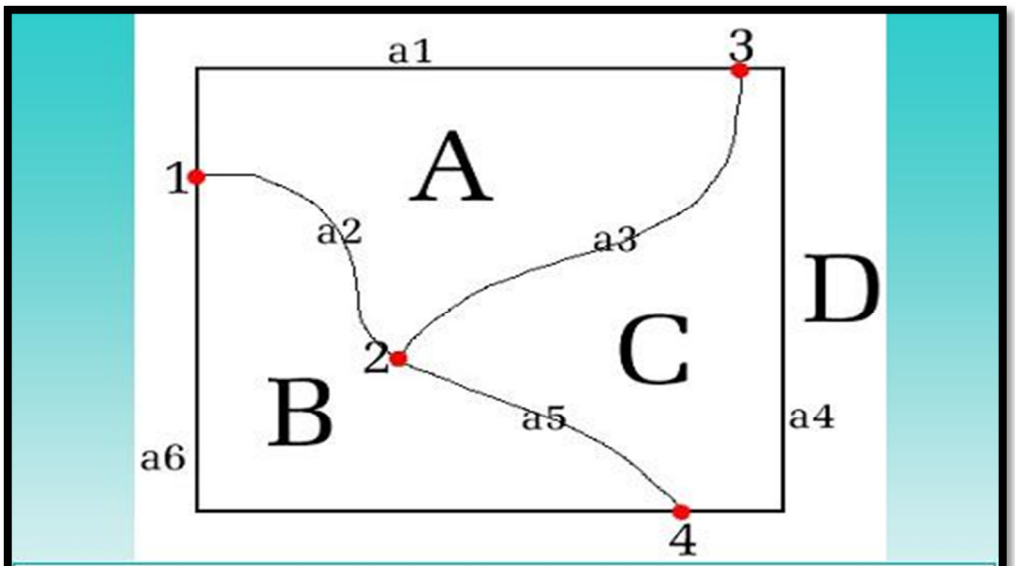
2.2.3 Estrutura de Dados Espaciais

O armazenamento e a manipulação de dados georreferenciados não é uma tarefa comum. Várias técnicas para projetar um banco de dados para SIG já foram desenvolvidas e aplicadas com êxito. Por muitos anos, os pesquisadores desta área focaram em encontrar soluções para estruturação de dados para SIG, e após várias experiências chegaram a dois tipos de estruturas de dados: vetoriais e matriciais (SILVA, 2004).

2.2.4 Dados Vetoriais

Também chamada de *Vector* utiliza entidades gráficas como pontos, linhas, e polígonos para representar os elementos de um mapa. Esta estrutura referencia suas entidades através de um par de coordenadas absolutas (X, Y).

A estrutura de armazenamento dos dados vetoriais pode ser topológica ou do tipo *spaghetti*. Na topológica os relacionamentos espaciais estão contidos em tabelas, e podem ser representados como nós, arcos ou polígonos. Os nós são uma entidade unidimensional que representam os vértices inicial e final dos arcos, além das feições pontuais (DORNELES & ANDRADE ,2012) (Figura 6).



Topologia					
Polígono: área		Nó: conectividade		Arco: contigüidade	
Polígono	Arcos	Nó	Arcos	Arco	Polígonos Esquerda & Direita
A	a1, a2, a3	1	a1, a2, a6	a1	A D
B	a2, a5, a6	2	a2, a3, a5	a2	A B
C	a3, a4, a5	3	a1, a3, a4	a3	A C
D	a1, a4, a6	4	a4, a5, a6	a4	C D
				a5	B C
				a6	B D

Figura 6 - Estrutura Topológica e sua forma de armazenamento.
Fonte: UNBC GIS LAB (2005)

Na estrutura *spaghetti*, as coordenadas das feições são armazenadas linha a linha, resultando em arquivos contendo uma lista de coordenadas. A simplicidade desta estrutura limita a sua utilização em análises espaciais, já que pode gerar incongruências (DORNELES & ANDRADE,2012).

A Figura 7 exibe uma comparação entre as duas estruturas e aponta a lista de incongruências que podem ser geradas na estrutura *spaghetti*.

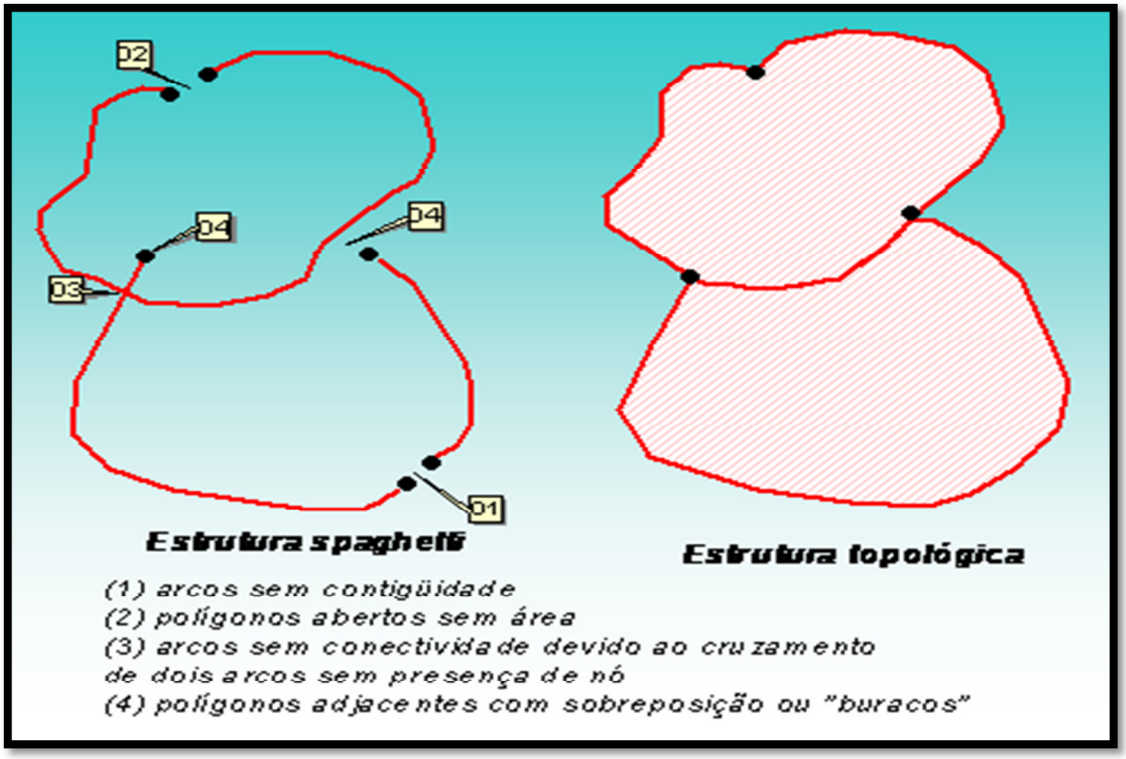


Figura 7 - Comparação entre a estrutura Spaghetti e Topológica
Fonte: FRANCISCO (2006)

A Figura 8 mostra a diferença de uma imagem do mundo real em comparação com a estrutura vetorial, observa-se que alguns dados são ignorados, e só alguns elementos são catalogados, e em forma de pontos, linhas ou polígonos.

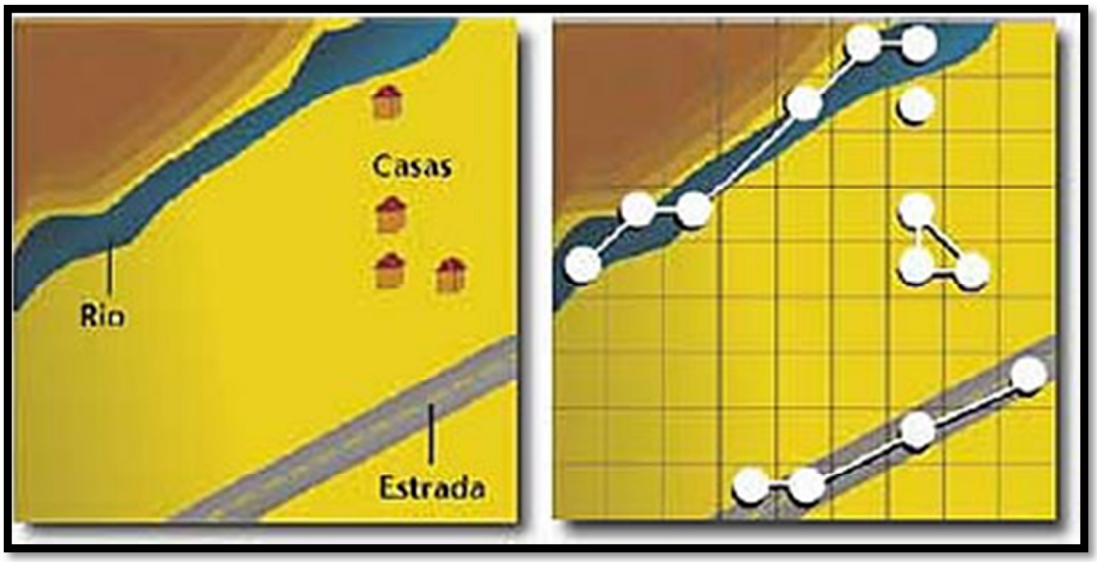


Figura 8 - Comparação mundo real com a Estrutura Vetorial
Fonte: SQL Magazine (2004)

2.2.4.1 Ponto

Em estrutura vetorial um ponto é uma única coordenada (X, Y) de um mapa, observe um exemplo na Figura 8, a casa em frente ao Rio.

2.2.4.2 Linha

Uma linha é uma junção de no mínimo dois pontos (vértices) ligados constituindo vetores. Na Figura 8, o Rio e a Estrada são exemplos de Linha.

2.2.4.3 Polígono

Um polígono é a junção de no mínimo três vértices ligados, sendo que o último obrigatoriamente deve ser igual ao primeiro para criar uma geometria fechada. O conjunto de casas na Figura 8 é um exemplo de polígono.

2.2.5 Estrutura Matricial

Também chamada de estrutura *Raster* ou *Grid Data* é mais uma fotografia do que um mapa, representando o mundo real como uma superfície continua. Os dados representam a paisagem como uma grelha(matriz) regular de células quadradas – pixels (HURVITZ, P, 1998).

Essa grelha(I,J) é composta por I colunas e J linhas que referenciam as células nela contidas. Desta maneira, cada elemento do mapa, é referenciado através da sua posição na grelha e também poderá estar associado a um atributo ou dado descritivo. Quanto a qualidade da imagem, esta, dependerá do tamanho da célula onde quanto menor, maior a qualidade da imagem.

Quaisquer dados espaciais podem ser armazenados no formato *Raster,* sendo este muito utilizado para a representação de dados contínuos (por ex. altitude, relevo, pH dos solos, salinidade da água, etc.) (HURVITZ, P, 1998).

A Figura 9 mostra a mesma comparação da Figura 8, porém, comparando o mundo real com a imagem na estrutura Matricial.

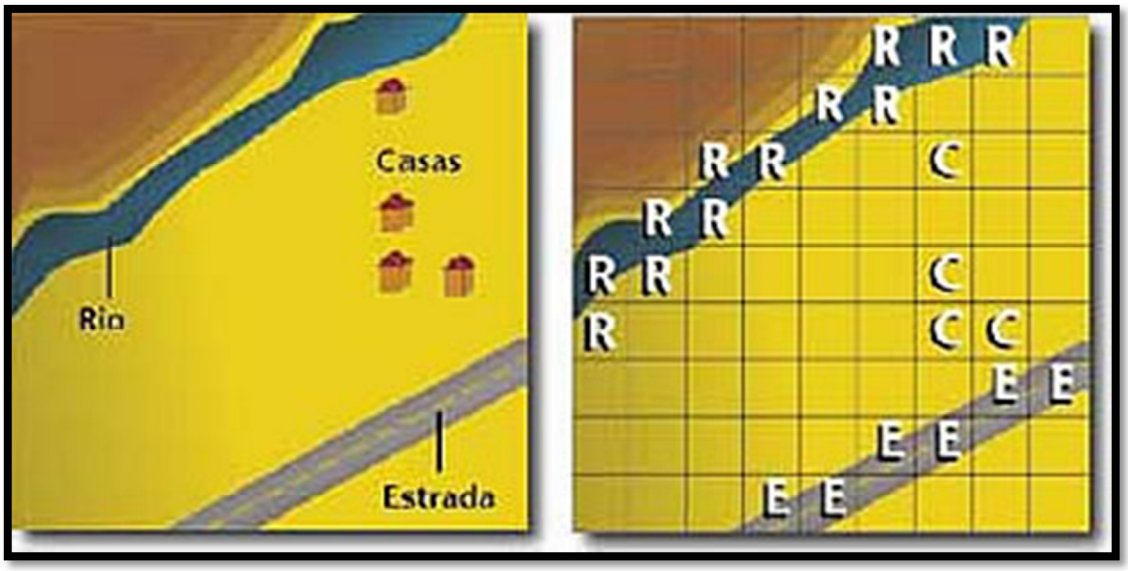


Figura 9 - Comparação do mundo real com Estrutura Matricial
Fonte: SQL Magazine (2004)

2.2.6 Comparando as Estruturas

A maioria dos SIG atuais suporta tanto a estrutura matricial como a vetorial, permitindo transformações entre elas. É importante ressaltar que nenhuma das estruturas é a ideal em todas as ocasiões e os critérios de escolha baseiam-se fundamentalmente nos objetivos do projeto. De um modo geral quando o objetivo do estudo é a distribuição espacial de objetos, o desenvolvimento de análises de rede, e o conhecimento acerca dos relacionamentos espaciais entre os objetos, então a estrutura vetorial é amais adequada. Por outro lado se o objetivo do estudo é a variabilidade espacial de um fenômeno (como é o caso em estudos ambientais, por exemplo), então a estrutura matricial pode ser mais adequada (FÁTIMA & SANTOS, 2010).

Tabela 1- Vantagens e Desvantagens dos modelos de dados espaciais

Modelo	Vantagens	Desvantagens
VETORIAL	• Estrutura Compacta • Eficiência na análise de relacionamentos espaciais. • Feições são representadas precisamente, por pontos, linhas e polígonos	• Estrutura complexa exigindo programas sofisticados e caros • Operações de superposição de níveis de informação, mais complexas
MATRICIAL	• Simplicidade de implementação das operações de superposição	• Dificuldade de representação de relacionamentos topológicos

	• Programas mais baratos computacionalmente e simples de usar	• Dificuldade na associação de atributos e feições • Arquivos muito grandes

Fonte: Comparação entre os modelos vetorial e matricial Adapt. de (FÁTIMA & SANTOS, 2000)

2.3 O BANCO DE DADOS ESPACIAL

Segundo (GÜTING, Ralf, 1994) podemos definir um banco de dados espacial se ele seguir três pontos principais:

1. Um banco de dados espacial é um Sistema Gerenciador de Banco de Dados (SGBD).
2. Oferece tipos de dado espacial em seu modelo de dados e em sua linguagem *Query*.
3. Oferece no mínimo indexação espacial e eficientes algoritmos para Spatial Join (Junção Espacial).

Podemos interpretar a definição da seguinte forma. Um banco de dados oferece os tipo de dados geográficos mais comuns como pontos, linhas e polígonos e consegue manipular esses dados através de linguagens *Query* como se estivesse manipulando qualquer outro tipo de dado comum em um SGBD. Por fim, esse sistema deve ser capaz de retornar grandes coleções de dados com diversos relacionamentos sem percorrer todos os dados envolvidos, ou seja, indexação para dados espaciais.

2.3.1 PostgreSQL

O PostgreSQL é um sistema gerenciador de banco de dados objeto relacional, gratuito e de código fonte aberto, desenvolvido a partir do projeto Postgres, iniciado em 1986, na Universidade da Califórnia em *Berkeley*, sob a liderança do professor Michael Stonebraker. Em 1995, quando o suporte a SQL foi incorporado, o código fonte foi disponibilizado na *Web* (http://www.postgresql.org). Desde então, um grupo de desenvolvedores vem mantendo e aperfeiçoando o código fonte sob o nome de PostgreSQL (PostgreSQL, 2012).

Segundo DOUGLAS K. e DOUGLAS S. (2003) hoje em dia, PostgreSQL é um dos SGBD mais avançados disponíveis. Entre suas principais características podemos citar:

- Objeto Relacional: No PostgreSQL cada tabela define uma classe. PostgreSQL implementa herança entre tabelas;
- Funções e operadores são polimórficos[2];
- Compilação padronizada – A sintaxe do PostegreSQL implementa grande parte da SQL92[3] e muito das características da SQL99[4]. Quando diferenças na sintaxe ocorrem, na maior parte e devido a características únicas do PostgreSQL;
- Diversidade de linguagens procedurais – *Triggers* e outros procedimentos podem ser escritos em diversas linguagens. Como o PLPG/SQL,PL/Java,PL/Perl,PL/Python entre outro;
- Tipos de dados únicos – PostgreSQL fornece uma grande variedade de tipos de dados além dos comuns *Numeric,String* e tipos de Data. Também encontrará booleanos, tipos específicos para lidar com conexão, entre outros;
- Extensibilidade – Uma das mais importantes características. No PostgreSQL tudo pode ser estendido, por exemplo pode-se criar novos tipos de dados, operadores, pacotes de funções, até novas linguagens procedurais. Como PostgreSQL é codigo aberto e possui uma grande comunidade de admiradores, é comum encontrar extensões novas na *internet* para lidar com diversos problemas.

2.3.2 PLPG/SQL

SQL é uma linguagem declarativa que permite os programadores dar instruções de execução ao banco de dados. Porém, SQL não pode ser usada para executar códigos procedurais, com condições de caso, iteração. Para suprir essa limitação surgiu se o *Procedural Language extensions to SQL*, popular PL/SQL (ORACLE, 2012).

Com o surgimento desse novo paradigma criado pela ORACLE, outras linguagens procedurais surgiram para dar suporte ao seu banco de dados.

---

[2] Em engenharia de software, polimorfismo significa permite que referências de tipos de classes mais abstratas representem o comportamento das classes concretas que referenciam.
[3] Terceira divisão da linguagem SQL, nela foram aprimorados seus recursos e sua sintaxe. Definida no ano de 1992.
[4] Também chamada de SQL3. Quarta divisão da linguagem SQL, definida em 1999.

O PLPGSQL é o PL/SQL para o PostgreSQL. Segundo o manual de uso do PostgreSQL PLPGSQL foi criada com o objetivo de ser uma linguagem procedural que pudesse:

- Ser utilizada para criar procedimentos de funções e de gatilhos;
- Adicionar estruturas de controle à linguagem SQL;
- Realizar processamentos complexos;
- Herdar todos os tipos de dado, funções e operadores definidos pelo usuário;
- Ser definida como confiável pelo servidor;
- Ser fácil de utilizar.

A Figura 10 mostra uma função simples em PLPGSQL para buscar um determinado texto de acordo com um parâmetro especificado.

```
1  create or replace function
2  returna_geometria (nome varchar)
3  return text as $$
4  declare
5  wnome varchar;
6  geometria text;
7  begin
8          wnome := upper(nome);
9                  select ST_astext(the_geom) into geometria
10                 from tb_municipios where nome_1=wnome;
11                 end;
12 $$ LANGUAGE plpgsql;
```

Figura 10 - Exemplo de função na linguagem procedura PLPGSQL
Fonte: Autoria Própria

2.3.3 PostGIS

Desenvolvido pela *Refractions Research* com intuito de trazer a possibilidade de lidar com dados espaciais ao sistema de banco de dados objeto-relacional PostgreSQL, PostGIS é uma extensão que permite que objetos SIG (Sistemas de Informação Geográfica) sejam armazenados em banco de dados.(RAMSEY, 2002). PostGIS inclui suporte para índices espaciais baseados em Árvore de busca genérica (*GiST-based*)[5] , *R-Tree*[6] e funções para análise e processamento de objetos GIS (PostGIS,2013).

[5] Estrutura de dados usada para construir uma variedade de árvores de pesquisa.

PostGIS é uma solução gratuita e código-aberto, facilita a manipulação de dados espaciais dispondo de uma grande gama de funções, índices espaciais e tipos específicos de dados, facilitando a manipulação dos dados, segundo o apontamento de OBE & HSU (2009), PostGIS possui uma gama de funções não disponíveis em soluções espaciais no mercado como, ORACLE *Spatial*, IBM *Spatial DataBlade* entre outros. Também aponta que sua agilidade está em quesito de igualdade, e certas vezes melhor que seus concorrentes.

PostGIS oferece os seguintes tipos de dados (Figura 11)[7]:

- ***Point*:** Uma única medição no plano cartesiano (X,Y). Também pode ser criada em três dimensões adicionando "Z" como valor para altitude e também pode ser adicionado mais um valor alfa-numérico "M" chamado de medida.

Exemplos: POINT(0 1),POINT(0 2 4) POINT (0 -22 4 3.33)

- ***LineString*:** Junção de dois pontos no plano cartesiano. Podendo ser representados com apenas X,Y ou até com X,Y,Z,M.

Exemplos: LINESTRING(0 0, 1 1) , LINESTRING (0 0 3, 1 4 2) , LINESTRING(0 0 3 24.33, 1 2 3 99.22).

- ***Polygon*:** Junção de no mínimo três linhas. Podendo ter anéis exteriores e interiores (Figura 10).

Exemplo: POLYGON(0 0, 1 1, 1 -1, 0 0) POLYGON((-0.25 -1.25,-0.25 1.25,2.5 1.25), (2.25 0,1.25 1,1.25 -1,2.25 0),(1 -1,1 1,0 0,1 -1))[8]

- ***CircularString:*** Junção de arcos onde o fim de um arco é o início de outro.

Exemplo: CIRCULARSTRING(0 0,2 0, 2 2, 0 2, 0 0)

- **CompoundCurves:** Coleção de CircularString's com LineStrings.

Exemplo: COMPOUNDCURVE((2 2, 2.5 2.5), CIRCULARSTRING(2.5 2.5,4.5 2.5, 3.5 3.5), (3.5 3.5, 2.5 4.5, 3 5))

- ***CurvePolygon:*** Representa um polígono que têm ou nos anéis exteriores ou nos interiores, um objeto do tipo *CircularString*.

Exemplo: CURVEPOLYGON(COMPOUNDCURVE(CIRCULARSTRING(0 0,2 0, 2 1, 2 3, 4 3),(4 3, 4 5, 1 4, 0 0)), CIRCULARSTRING(1.7 1, 1.7 0.9, 1.6 0.5, 1.4 0.6, 1.7 1))

---

[6] Estrutura de dados em árvore usados para métodos de acesso espacial.
[7] A Figura 11 não representa os tipos de dados com curvatura, como, *CircularString, ConpoundCircle* e *CurvePolygon*, porém estes dados podem ser manuseados a partir da versão 1.3 do PostGIS lançada no dia 13 de Agosto de 2007.
[8] O Primeiro grupo de dados antes da primeira vírgula correspode aos anéis exteriores, os demais, aos interiores.

- **Coleções de Geometrias:** São usados para agrupar várias geometrias num só objeto. Como *MultiPoint*, *MultiLineString*, *MultiPolygon* e *GeometryCollection* para agrupar objetos de tipos diferente.

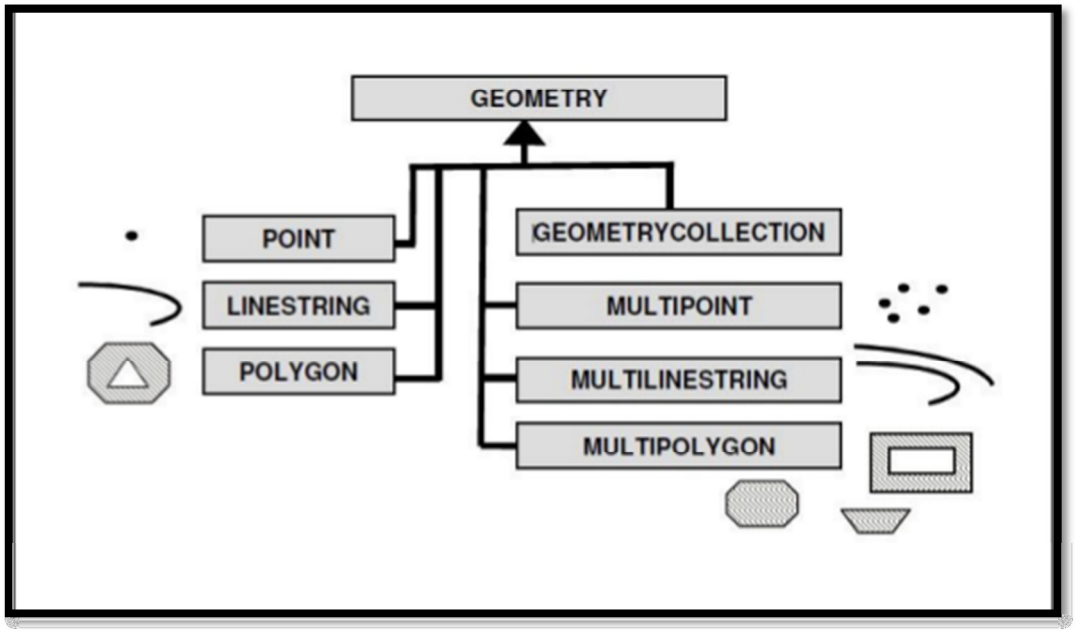


Figura 11 – Tipos de objetos suportados no PostGIS para a classe Geometry
Fonte: BATISTA (2006)

PostGIS dispõe de duas classes de dados espaciais para manipulação, *Geometry* e *Geographic*, segundo OBE & HSU (2009) para melhor escolher que classe de dados usar deve-se considerar a real necessidade da aplicação. Se for necessário apenas pequenos cálculos, operações de checagem de relacionamento entre os dados e se os dados não cobrerem uma larga área, então, a escolha deve ser a classe *Geographic*. Porém *Geometry* possui um conjunto de funções bem mais rico, a checagem de relacionamento normalmente é mais rápida e esta classe de dados possui suporte maior nas ferramentas de mapeamento espacial *Web* e *Desktop*.

2.3.3.1 Geometry

É a classe de dados espaciais baseada em um plano cartesiano, portanto, a menor distância entre um ponto e outro é uma reta. A terra é um esferoide, sendo assim, traz grandes problemas na precisão quando os valores ou o resultado de operações extrapolam a grandes distâncias territoriais (SP PERL MONGERS, 2011).

2.3.3.2 Geographic

Esta classe de dados têm funções para inserção de dados similares, as mesmas representações em texto, a única diferença é que as coordenadas são recebidas e expressadas em WGS 84 Lon Lat[9], e suas medidas são sempre representadas nas unidades de metro/metro quadrado. Comparado a classe *Geometry*, esta classe possui bem menos funções (OBE & HSU, 2009).

2.4 GOOGLE WEB TOOLKIT

Lançado no dia 16 maio de 2006 *Google Web Toolkit* popular GWT, é um *toolkit Web* para Java criado pela Google Inc que visa produtividade e performance em aplicações web de grande complexidade.

GWT provê uma plataforma para criação de *Rich Internet Applications* (RIA) no sentido de permitir ao cliente a manutenção do estado e até executar cálculos localmente com um completo modelo de dados sem necessariamente precisar chamar o servidor a cada atualização da interface (COOPER & COLLINS, 2008).

A grande vantagem de se usar GWT é que o desenvolvedor irá escrever puramente código Java e esse código será transformado em *JavaScript*,Ajax e HTML. Diversos framework's e toolkit's para *Web* esbarram em um grande problema, a dificuldade de corrigir erros devido ao não encapsulamento dessas tecnologias (*JavaScript*,Ajax) que são mais morosas de encontrar falhas. Já desenvolvendo Java, torna-se mais fácil corrigir problemas, devido a grande variedade de ferramentas para inspeção do código.

A Figura 12 mostra o esquema do resultado da compilação do GWT

[9] Elipsóide de referência de origem geocêntrica utilizado pelo Sistema de Posicionamento Global.

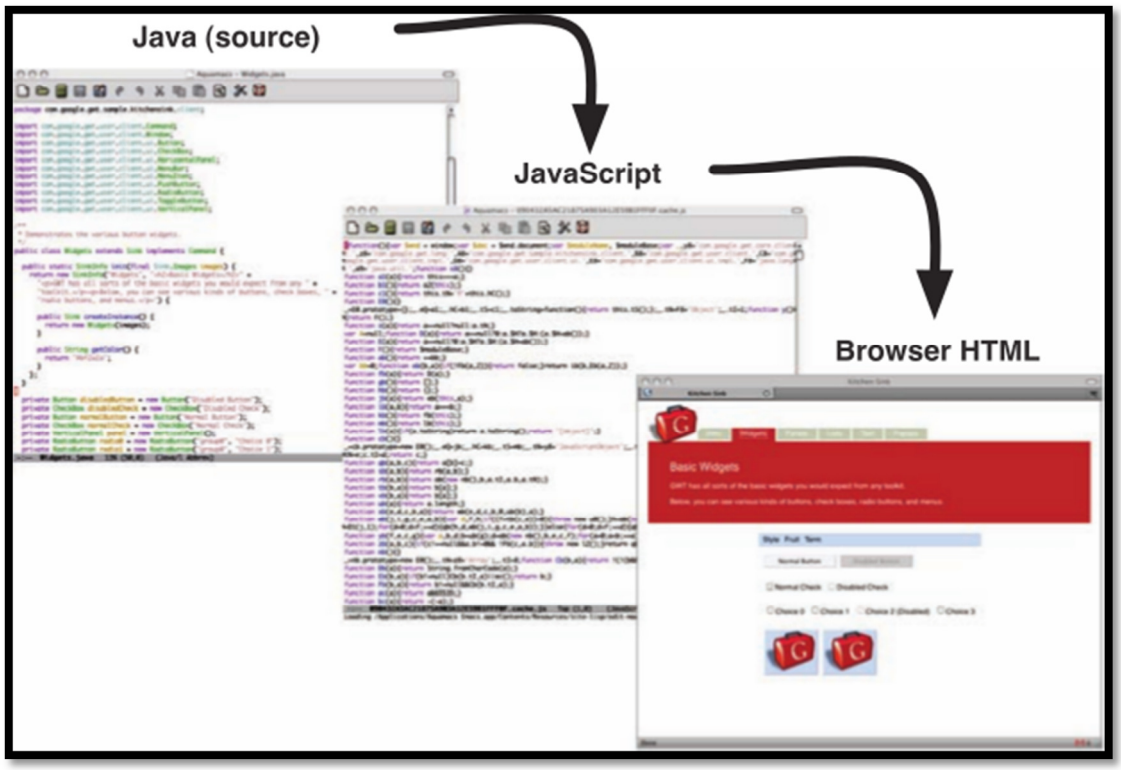


Figura 12 - Visão Geral da compilação do GWT
Fonte: COOPER & COOLINS (2008)

O principal objetivo do GWT é a criação de aplicações *cross-browser*[10], assim o desenvolvedor cria apenas uma versão, sem precisar adaptar-se a um determinado Browser.

2.4.1 GWT para o Desenvolvedor

GWT possui uma grande quantidade de recursos para *User Interface*(UI) *cross-browser* permitindo aos desenvolvedores construir com facilidade uma grande diversidade de *layouts*, nesse sentido, GWT mistura aspectos do desenvolvimento Desktop e *Web*, pois, oferece recursos para trabalhar com Cascading Style Sheet (CSS) juntamente com seus objetos gráficos. Obviamente, optar por recursos como CSS pode acarretar na perda da principal característica do GWT, a aplicação *cross-browser*.

A Figura 13 mostra a implementação de um componente UI em uma aplicação GWT.

[10] Aplicações que não dependem de um Browser expecifico para sua visualização.

```
1  /**
2   * @author Bernardo Vale
3   * Exemplo GWT, iniciando um projeto
4   */
5   public class GWTInicio implements EntryPoint{
6    /**
7     * Método da interface EntryPoint
8     * Nele podemos chamar RootPanel(Painel Raiz)
9     * onde podemos inserir Botões,Labels,TextBox
10    * entre outros
11    */
12    public void onModuleLoad(){
13      //Objeto botão comum
14      Button b = new Button("Ola GWT");
15      RootPanel.get().add(b);
16    }
17  }
```

Figura 13 - Exemplo de como inserir um Elemento simples num projeto GWT
Fonte: Autoria Própria

Na Figura 13 é possível observar umas das formas de criação de Layout com o GWT, porém através de uma de suas tecnologias, o *UiBinder*, podemos criar *Layouts* de maneira declarativa utilizando *Extensible Markup Language* (XML) para definir os elementos, chamados no GWT de *Widgets*, e Java para orientá-los. Essa maneira foi introduzida na versão 2.0 do GWT, e praticamente substituiu a maneira programática, pois, *UiBinder* nos permite separar o Layout da camada de negócios, onde fazemos validações, interações com o servidor, etc.

A Figura 14 demostra o mesmo exemplo da Figura 13, mudando da maneira programática para declarativa.

```
<ui:UiBinder xmlns:ui='urn:ui:com.google.gwt.uibinder'
   xmlns:g='urn:import:com.google.gwt.user.client.ui'>
   <g:HTMLPanel>
     <g:Button text="Ola GWT" ui:field="botaoIniciar"/>
   </g:HTMLPanel>
</ui:UiBinder>
```

```
public class UiBinderView extends Composite{

  @UiField
  Button botaoIniciar;
}
```

Figura 14 - Criação de Layout da maneira declarativa através da tecnologia UiBinder
Fonte: (Autoria Própria)

2.4.2 Arquitetura do GWT

Um projeto GWT é dividido em duas partes. *Client-Side* e Server-Side, mas normalmente também cria-se um lado que seja conhecido pelos dois, o *Shared*.

2.4.2.1 Client Side

Na arquitetura GWT usa-se o Client-Side para fazer o código responsável pela visualização do usuário, além da manipulação do mesmo. Podemos dizer que no *Client-Side* estão armazenados os códigos Java que se transformarão em HTML,CSS,*JavaScript* e Ajax. Em GWT, pode-se afirmar também que uma aplicação que não utiliza um Banco de Dados, pode ser composta de apenas código no Client-Side, porém isto geralmente não acontece, pois, no *Client-Side* se dispõe apenas de parte da enorme gama de classes do Java, trata-se na verdade, de uma emulação das bibliotecas do *Java Runtime Library* (JRE), sendo assim, geralmente precisamos contar com *Server-Side* para resolver outros problemas além da comunicação com o Banco de Dados.

Nem todas as classes do JRE podem ser usadas no *Client-Side* de uma aplicação GWT, apenas certos pacotes que, ao ver dos criadores, eram imprecindiveis para os desenvolvedores, dentre os disponíveis[11]:

- java.lang
- java.lang.annotation
- java.math
- java.io
- java.sql
- java.util
- java.util.logging

2.4.2.2 Server Side

Tudo que não será usado pelo navegador está no *Server-Side*, basicamente usa-se está parte da aplicação para atender funções como, acesso ao banco de dados, conexões com *WebService*[12], entre outros recursos que não podem ser feitos no *Client-Side,* como por exemplo a manipulação de arquivos, operações complexas e/ou demoradas, entre outras.

No *Server-Side* compila-se o codigo Java puramente, ou seja, como qualquer outra aplicação Java tudo se transformará em arquivos ".class"[13].

O *Server-Side* é a parte da aplicação responsável por atender as requisições do *Client-Side*, esta é sua única função, e a faz seguindo os padrões e conceitos da computação distribuida.

2.4.3 Comunicação entre Client-Side e Server-Side

A diferença principal entre aplicações AJAX e as tradicionais aplicações HTML é que em AJAX não precisa-se recarregar dados HTML enquanto se está executando. Isso se da porque páginas AJAX na verdade são executadas como aplicações dentro do navegador,

[11] A lista completa de todas as classes dentre os pacotes disponíveis pode ser acessada através do endereço https://developers.google.com/web-toolkit/doc/latest/RefJreEmulation
[12] Solução utilizada na integração de sistemas e na comunicação entre aplicações diferentes.
[13] Extensão do código compilado Java.

então, não há necessidade de requisitar uma nova página HTML do servidor para atualizar a interface. Porém estas aplicações como qualquer outra baseada em Cliente/Servidor, necessitam de um mecanismo para interagir com o servidor através da internet, para carregar os dados requisitados, este mecanismo é o *Remote Procedure Call* (RPC) (GOOGLE,2012).

RPC é uma técnica para permitir que um cliente execute procedimentos em outro computador ou servidor na rede (BLOOMER, John, 1992)

O GWT utiliza RPC baseado em Java *Servlets*[14], este método permite que objetos sejam serializados/deserializados e transmitidos através da rede.

Além disso, o GWT permite a integração de outros mecanismos[15] baseados em RPC para a comunicação com seu *Server-Side*, porém o método RPC baseado em Java *Servlets* é bem mais simples de ser usado, mas o desenvolvedor tem a liberdade de integrar outro mecanismo.

2.5 GOOGLE WEB TOOLKIT PLATFORM

GWT torna aplicações *Web* parecerem muito simples. No entando, criar uma aplicação eficiente, que possa se expandir rapidamente com GWT, não é uma tarefa simples. Uma boa forma de começar um projeto GWT com esta eficiencia, é aderir-se a uma arquitetura bem estabelecidada e moldada levando em conta todas as melhores práticas do GWT (BEAUDOIN,2011).

GWTP é uma coleção de componentes construídos como uma arquitetura pode-se fazer uso destes componentes, mas se necessário pode-se também criar um projeto desde o começo utilizando todo o pacote. Qualquer que seja a escolha, GWTP irá otimizar a compilação garantindo que somente as opções escolhidas farão parte do código compilado (BEAUDOIN, 2011).

Durante uma conferência[16], fora discutido as melhores práticas para projetos GWT, e um dos principais pontos tocados nesta conferência foi que, a arquitetura que melhor atende

[14] Classe na linguagem de programação Java que dinamicamente processa requisições e respostas,
[15] A lista de mecanismos e como usa-los pode ser encontrada na página do GWT <*https://developers.google.com/web-toolkit/doc/1.6/DevGuideServerCommunication#DevGuideServerSide*>
[16] Google I/O,2009 Melhores Práticas para GWT. O vídeo da conferência pode ser acessado através do link : http://www.google.com/intl/pt-BR/events/io/2009/sessions/GoogleWebToolkitBestPractices.html

os requisitos e a estrutura do GWT era o modelo *Model-View-Presenter* (MVP), desbancando a atual arquitetura usada para tais projetos, o *Model-View-Controller* (MVC), após esta conferência uma tentativa bem sucedida de adaptação ao MVP fora concluída, e o *framework* GWTP, trás consigo esta adaptação.

2.5.1 Arquitetura MVP

Publicada em 1996 por Mike Potel para se tornar o modelo de programação da próxima geração de aplicações Java e C++ da época. Potel descreveu o MVP como um modelo unificado que era adaptável a aplicações multicamadas e a múltiplas arquiteturas cliente-servidor (EZRA, 2007).

O Modelo MVP é dividido em seis abstrações:

- *Model*: Interagi com o banco de dados
- *Selection*: Define o subconjunto de dados selecionados
- *Command*: Executa operações dentro da Selection.
- *View*: Responsável pelo desenho das telas da aplicação.
- *Interactor*: Trata os eventos do usuário e invoca os *Command* apropriados
- *Presenter*: criar o *Model*, a *View*, as *Selections*, os *Commands* e *Interactors*, e gerenciar as interações entre eles (MIRANDA, 2009).

Na atualidade, o modelo MVP é comumente dividido em apenas três abstrações, pois Presenter têm a responsabilidade de englobar *Command, Selection* e *Interaction* (MIRANDA, 2009). O *framework* GWTP também utiliza esta abstração.

A Figura 15 mostra a abstração utilizada em projetos GWTP utilizando o modelo MVP.

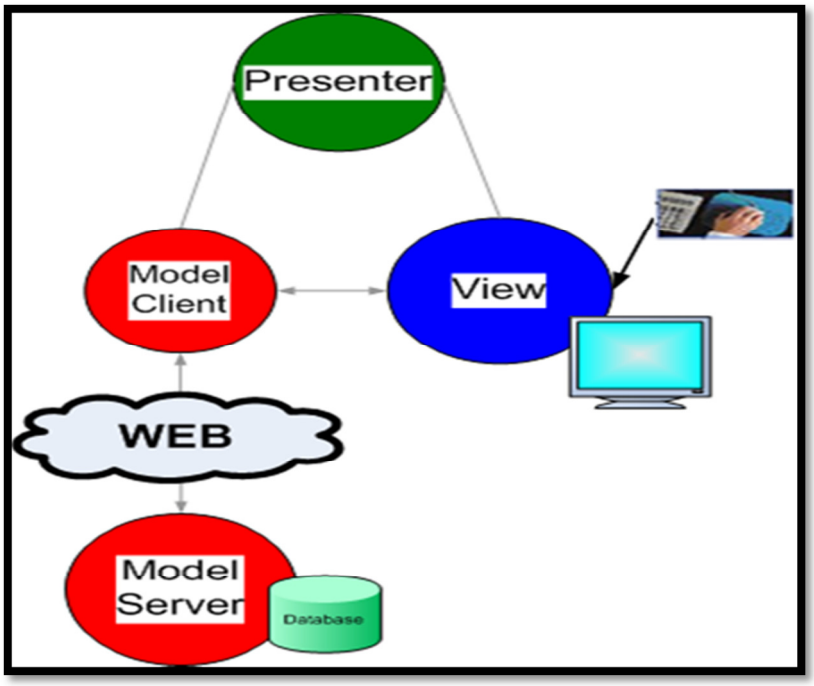


Figura 15 - Abstração Cliente/Servidor para o modelo MVP
Fonte: (MIRANDA, 2009)

Como GWT é uma aplicação divida entre Cliente e Servidor (*Client-Side* e *Server-Side)* algumas medidas devem ser tomadas para à adaptação do projeto, uma delas e que devemos manter uma cópia do Modelo na parte do cliente para que ambos lados consigam se comunicar. No GWTP está parte do Cliente fica na parte específica na aplicação chamada *Shared* onde ambos lados têm conhecimento, está parte, funciona como um adaptador do modelo para que a comunicação seja efetuada, já que o modelo de dados do servidor muitas vezes é diferente devido a sua função de englobar a camada de persistência.

2.5.2 O Tripé GWTP, Presenter,View e Proxy

Nas aplicações GWTP toda tela corresponde a um tripé, *Presenter*, *View* e *Proxy*. Este tripé foi criado para cobrir com eficiência algumas habilidades criadas pelo GWTP e GWT com a arquitetura de desenvolvimento MVP. Em nível de código, este tripé esta contido em uma única classe (Figura 16), sendo *View* e *Proxy* interfaces internas da classe *Presenter*.

```
1  public class MainPresenter extends
2      Presenter<MainPresenter.MainView,MainPresenter.MainProxy> {
3      @Inject
4      public MainPresenter(EventBus eventBus, MainView view,
5                           MainProxy proxy) {
6          super(eventBus, view, proxy);
7      }
8
9      @Override
10     protected void revealInParent() {
11         revealPlace( new PlaceRequest("main"), false );
12     }
13
14     public interface MainView extends View {}
15     @ProxyCodeSplit
16     @NameToken("main")
17     public interface MainProxy extends ProxyPlace<MainPresenter>{}
18 }
```

Figura 16 - Exemplo de tripé Presenter,View e Proxy
Fonte: Autoria Própria

Atenta-se a partir da Figura 16, algumas observações e características.

- Toda tela em GWTP possui injeção de dependência, que são geridas pelo *framework* Google Gin. Obrigatoriamente, inserindo a anotação *Inject* no construtor da *Presenter*.
- A interface *View*, define todos os componentes que serão utilizados para gerir a tela dentro da *Presenter*. Outra classe implementará esta *View*, chamada *ViewImpl*(Figura 16), sendo esta, responsável pela declaração dos objetos visuais da tela.
- O método *RevealInParent* herdado da classe pai *Presenter*, é responsável por mostrar como a tela será inserida no *browser*, em síntese, existem várias maneiras. No (Figura 16) foi feito uma requisição de exibir a tela com o codinome Main. O parâmetro falso significa que a aplicação garante que se pode utilizar o botão Voltar dos navegadores para acessar o histórico de navegação.
- A anotação *ProxyCodeSplit* é usada para conseguir carregar uma página por partes do código, sendo carregado apenas uma vez durante a sessão, aumentando drasticamente a velocidade.

- A anotação NameToken define o nome da Presenter perante ao navegador, podendo ser acessada através do nome da aplicação, mais o seu chamado *Name Token*.
- A implementação do *Proxy* é criada automaticamente pelo GWT.

Em GWTP uma tela corresponde a um único *Widget* painel, onde tudo que se deseja exibir é inserido neste *Widget*, observa-se na Figura 16 a classe de implementação da *View*.

```
1  public class MainView extends ViewImpl
2  implements MainPresenter.MainView {
3
4    private static String html =
5      "<h1>Exemplo de ViewImpl</h1>\n" +
6      "<table align=\"center\">\n" +
7      "  <tr>\n" +
8      "    <td colspan=\"2\" style=\"font-weight:bold;\">Hello GWTP</td>\n" +
9      "  </tr>\n" + "  <tr>\n" + "</table>\n";
10   HTMLPanel panel = new HTMLPanel(html);
11   @Inject
12   public MainView() {
13     final Label exemploLabel = new Label();
14     panel.add(exemploLabel);
15   }
16   @Override
17   public Widget asWidget() {
18     return panel;
19   }
20 }
```

Figura 17 - Exemplo de ViewImpl no GWTP
Fonte: Autoria Própria

O método *AsWidget* consiste num método obrigatório da super calsse *ViewImpl* onde se têm como resultado o Widget painel que representa a tela. Observa-se na Figura 17 que Esta classe implementa a interface declarada na Figura 17. Atenta-se também a injeção de dependência presente na *ViewImpl.*

2.6 OPEN LAYERS

OpenLayers é uma biblioteca *JavaScript* código-aberto para criação de mapas interativos na WEB visualizados em quase todos *browser* atuais. OpenLayers permite a criação de mapas totalmente customizados, em cada aspecto de um mapa, seja suas camadas, controles, eventos, etc. Pode-se usar diferentes servidores de mapa como o *GoogleMaps*, *OpenMaps,* Yahoo! Maps, Bing, entre outros (HAZZARD, 2011).

Basicamente, OpenLayers é um visualizador de mapas que consegue trabalhar simultaneamente com diversas camadas, seu trabalho é a partir de um servidor de mapas como fonte de dados, representar visualmente o aspecto de um mapa na *Web*.

Esta biblioteca consegue executar esta tarefa através da tecnologia AJAX, enviando uma requisição ao servidor de mapas, a cada iteração do usuário com o mapa, como por exemplo, efetuar uma operação de *Zoom*[17], então OpenLayers recebe todas as imagens de mapas do servidor e as encaixa novamente. Como trabalha com AJAX, que utiliza chamadas assíncronas, ou seja, a tela não precisará ser atualizada, a iteração com usuário é mais dinâmica e intuitiva (HAZZARD, 2011).

2.6.1 Layers

Como OpenLayers consegue trabalhar com diversos servidores de mapas ao mesmo tempo, criou-se o conceito de Camadas ou *Layers*. Na prática isto quer dizer que pode-se buscar dados em diferentes fontes em um único mapa. Para isto, cria-se um objeto *Layer* e atrela-o ao mapa.

Vale salientar, que neste conceito de camadas é ordem é um fator importante, visto que, estas camadas se sobrepõe umas as outras, formando, uma espécie de cebola no mapa, ou seja, a camada de cima cobrirá a debaixo. Felizmente o OpenLayers traz a possibilidade de controlar esta sopreposição dinamicamente.

2.6.1.1 Base Layer

É denomidada Base Layer, a camada que sera sobreposta por qualquer outra camada que seguir, ou seja, a camada mais profunda do mapa. A primeira camada inserida no mapa é denomidada a *Base Layer*, pode-se também, mudar a *Base Layer* dinamicamente, porém, outra camada deve ser "promovida" a *Base Layer* para garantir a integridade do mapa

2.6.1.2 Overlay Layer

[17] Aumentar ou diminuir o nível de detalhamento do mapa.

Apenas e somente uma camada é denomida *Base Layer*, todas as outras são chamadas de *Overlay Layer*. Do inglês, a palavra *Overlay* significa “Sopreposição”. Estas camadas estão sobrepostas a *Base Layer*, e cada camada que se seguir estará sobreposta a anterior.

A Figura 18 demonstra com maior clareza os conceitos de *Base* e *Overlay Layer*.

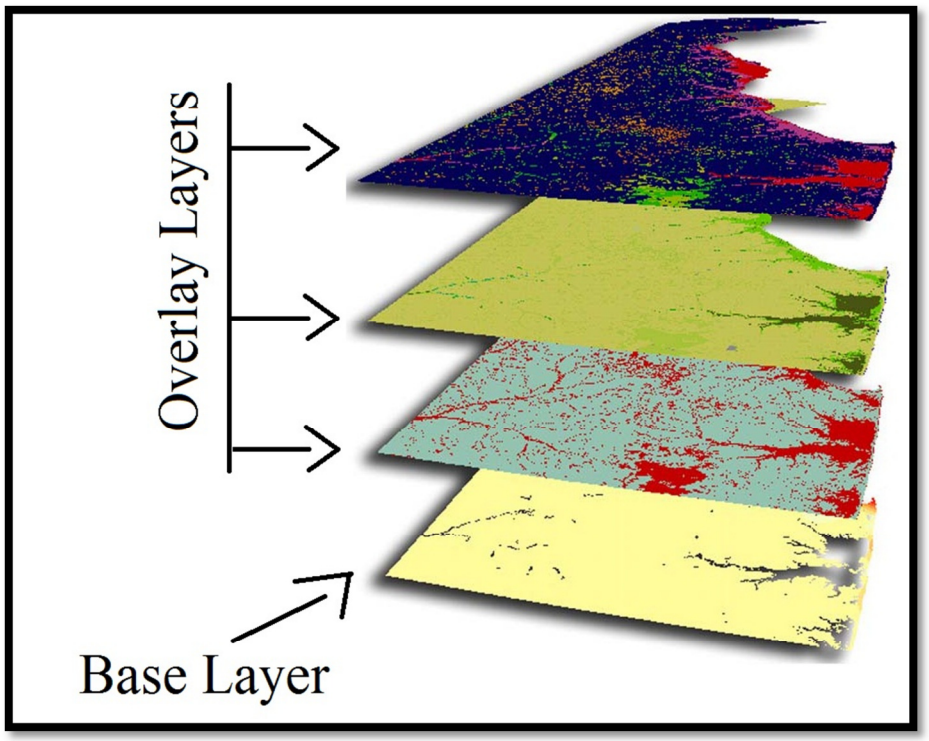


Figura 18 - Sobreposição das camadas no Openlayers
Fonte: OCEAN EXPLORER (2013)

2.7 GWT-OPENLAYERS

GWT-OpenLayers é um *wrapper*[18] Java para a API de JavaScript OpenLayers. Permite que projetos GWT usem o OpenLayers atráves de sua tecnologia (GWT-OpenLayers,2012).

2.8 CORRELAÇÃO ESPACIAL CRUZADA

A função de correlação mede o grau de correlação de uma variável, em um dado instante, consigo mesma, em um instante de tempo posterior. Ela permite que se analise o

[18] Na computação, *Wrapper* é um empacotador. Algo que funciona como adaptador para utilizar alguma coisa que seria impossível sem tal adaptador.

grau de irregularidade de um sinal. Pode ser definida como a razão entre a auto covariância e a variância para um conjunto de dados (SILVA&LIMA,2013)

A correlação cruzada de uma dada variável se define pela distância, ou atraso com que se deseja medi-la. Quando essa distância é zero, tem-se o valor máximo 1, pois trata-se da variável correlacionada com ela mesma.

A fórmula para o cálculo da correlação espacial cruzada é a seguinte:

$$I_{YZ} = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} * Y_i * Z_j}{W\sqrt{m_Y{}^2 * m_Z{}^2}}$$

em que:

I_{YZ} - Corresponde ao nível de associação entre a variável Y e Z, variando de -1 a 1, sendo: correlação positiva $I_{YZ} > 0$ e correlação negativa $I_{YZ} < 0$; W_{ij} - corresponde a matriz de associação espacial, sendo calculado por $W_{ij} = (1/(1 + D_{ij}))$, sendo D_{ij} a distância entre os pontos *i* e *j*; Y_i - Corresponde ao valor da variável *Y* transformada no ponto *i*. A transformação se dá para se ter uma média zero, pela equação: $Y_i = (Y_i - \overline{Y})$, em que $\overline{Y}$ é a média amostral da variável Y. Z_j - Corresponde ao valor da variável *Z* transformada no ponto *j*. A transformação se dá para se ter uma média zero, pela equação: $Z_j = (Z_j - \overline{Z})$, em que $\overline{Z}$ é a média amostral da variável Z. W – Corresponde a soma dos graus de associação espacial, obtidos através da Matriz W_{ij}, para $i \neq j$; $m_Y{}^2$ - Corresponde a variância amostral da variável Y; $m_Z{}^2$ - Corresponde a variância amostral da variável Z.

3 MATERIAIS E MÉTODOS

Esta sessão apresenta os recursos utilizados na criação do projeto além de demostrar a análise prevista para a construção da aplicação.

3.1 TECNOLOGIAS UTILIZADAS

Para o desenvolvimento da aplicação foram utilizados os seguintes recursos:

Para codificar a parte WEB do projeto, fez-se uso da *Integrated Development Environment* (IDE) IntelliJ IDEA Community Edition. PostgreSQL na versão 9.1 e PostGIS 2.0 para extensão espacial. Para codificação dos *Stored Procedure's* e demais orientações ao banco de dados foi utilizado a ferramenta de administração PgAdmin III. A Linguagem de programação utilizada foi Java. Para os *Stored Procedure's* no banco de dados, PLpg/SQL.

3.1.1 Frameworks e Facilitadores utilizados no Projeto

A lista completa de todos os *frameworks* utilizados no projeto e uma breve descrição de sua utilidade no projeto:

- GWT 2.4: Principal *toolkit* de desenvolvimento Web;
- GWT-Platform: Facilitador e guia para melhores práticas em projetos GWT;
- GWT-Bootstrap: Principal framework Front-end para fornecer interfaces ricas;
- Google Guice: Framework para injeção de dependência no *Server-side;*
- Google Gin: Framework para injeção de dependência no *Client-side;*
- GWT-Upload: Framework para fornecer melhor alternativa no *upload* de arquivos;

- GWT-Chosen: Back-end para fornecer componentes de multipla seleção mais ricos;
- GWT-OpenLayers: Framework para visualização de mapas;
- GWT-HighCharts: Framework para criação de gráficos;
- GWT-Query: Framework para melhor manipulação de arquivos CSS dinamicamente;
- Maven: Framework para gerenciar o projeto;
- Tomcat7: Servidor de aplicação;
- EclipseLink: Framework para comunicação com o banco de dados e orientação da camada de persistência;

3.2 ANÁLISE DO PROJETO

Com intuito de prever melhores formas para a criação dos recursos da aplicação e definir os seus objetivos de maneira clara, aborda-se a visão geral do sistema e os requisitos do mesmo.

3.2.1 Visão Geral do Sistema

A aplicação experimental proposta compreende em um objeto de estudo e análise de dados espaciais, capaz de converter arquivos texto (.txt) contendo amostras de dados como localização espacial e medidas em geometrias no banco de dados. A partir desta definição, os dados podem ser utilizados como amostras de estudo e análise em mapas e operações de estatística.

3.2.2 Requisitos do Sistema

Esta sessão apresenta os requisitos do sistemas, como funcionalidades, restrições e regras de negócio para a aplicação que foram implementadas.

F1 Cadastrar Usuários	Oculto()
Descrição: O sistema deve fornecer mecanismo para criação de novos usuários de acesso à	

aplicação				
Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 Formulário	A tela para cadastros deve conter as seguintes informações para preencher, sendo todas obrigatórias: Nome,Senha,Confirmar Senha, Email e nome do usuário.	Interface	()	(x)

Quadro 1 - Requisito Cadastrar Usuário
Fonte: Autoria Própria

F2 Inserir conjunto de Dados	Oculto()
Descrição: O sistema deve prover a funcionalidade de armazenar geometrias a partir de arquivos texto cedidos pelo usuário e inseridos junto a sua base no banco de dados.	

Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 Formulário	A tela de inserção deve obrigatoriamente conter campo para *upload* do arquivo texto.	Interface	()	(x)
N1.2 *Login* Obrigatório	A tela deve ser exibida apenas para usuários cadastrado.	Segurança	()	(x)
N1.3 Forma de armazenamento	Todos os dados devem ser inseridos em tabelas no banco. Cada arquivo texto dará origem a apenas uma tabela.	Usabilidade	()	(x)

Quadro 2 - Requisito Inserir Conjunto de Dados
Fonte: Autoria Própria

F3 Manter Base de dados do usuário	Oculto(x)
Descrição: Internamente os dados criados pelos usuários conforme o Requisito R2, devem estar armazenados separadamente por usuário.	

Quadro 3 - Manter Base de Dados do Usuário
Fonte: Autoria Própria

F4 Inserir Dados Manualmente	Oculto()
Descrição: O sistema deve prover uma forma de armazenar mais tuplas em uma tabela já criada na base de dados do usuário, conforme o Requisito F2. Adicionando dados espaciais manualmente, ou seja, digitando valores de Latitude, Longitude.	

Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 Login Obrigatório	A tela deve ser exibida apenas para usuários cadastrado.	Segurança	()	(x)
N1.2 Formulário	A tela de inserção deve obrigatoriamente conter mapa de todas as tabelas do usuário do sistema, onde, o usuário selecionará a tabela onde irá adicionar dados.	Interface	()	(x)
N1.3 Pré-Visualização	A tela deve conter uma forma de exibição dos dados que forem sendo inseridos durante a transação.	Usabilidade	(x)	()

Quadro 4 - Requisito Inserir Dados Manualmente
Fonte: Autoria Própria

F5 Efetuar *Login*	Oculto()
Descrição: O sistema deve ter uma forma de verificação do usuário que deseja entrar no sistema, através de um formulário.	

Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.2 Formulário	A tela deve conter os campos nome do usuário e senha. E devem ser obrigatoriamente validados para entrada no sistema	Interface	()	(x)

Quadro 5 – Requisito Efetuar Login
Fonte: Autoria Própria

F6 Visualizar Dados	Oculto()
Descrição: O sistema deve prover uma forma de exibir todas as tabelas criadas pelo usuário do sistema, também os valores contidos nestas tabelas.	

Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 Login Obrigatório	A tela deve ser exibida apenas para usuários cadastrado.	Segurança	()	(x)
N1.2 Valores Geométricos	Todos os campos que contiverem dados do tipo *Geometry*, devem mostrar seus dados na forma *Well-know-text* (WKT)[19]	Usabilidade	(x)	()

Quadro 6 - Requisito Visualizar Dados
Fonte: Autoria Própria

F7 Visualizar Dados em Mapa	Oculto()
Descrição: O sistema deve prover uma forma de exibir os dados inseridos em sua	

[19] Formato para exibir dados espaciais em forma de texto.

representação geométrica em mapas georreferenciados.				
Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.2 Login Obrigatório	A tela deve ser exibida apenas para usuários cadastrado.	Segurança	()	(x)
N1.2 Camadas de Exibição	Poderá se exibir diversas tabelas no mapa de uma vez.	Usabilidade	()	(x)
N1.3 Cores na camada	Cada tabela inserida deve possuir uma representação gráfica em cor diferente.	Interface	(x)	()
N1.4 Legenda da Camda	Uma amostra da cor da camada deve existir para cada camada que estiver no mapa	Interface	(x)	()

Quadro 7 - Requisito Visualizar Dados em Mapa
Fonte: Autoria Própria

F8 Aplicar Correlação Espacial	Oculto()			
Descrição: O sistema deve oferecer o cálculo da correlação espacial entre duas tabelas que contenham valores amostrais.				
Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.2 Login Obrigatório	A tela deve ser exibida apenas para usuários cadastrados.	Segurança	()	(x)
N1.2 Formulário	A tela deve ter campos para informar duas tabelas para efetuar o cálculo.	Interface	()	(x)

Quadro 8 - Requisito Aplicar Correlação Espacial
Fonte: Autoria Própria

F9 Gráfico de Amostras	Oculto()			
Descrição: O sistema deve oferecer a visualização dos valores de medida das tabelas que possuem dados amostrais.				
Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 *Login* Obrigatório	A tela deve ser exibida apenas para usuários cadastrados.	Segurança	()	(x)
N1.2 Formulário	A tela deve oferecer a lista das tabelas que o usuário possui	Interface	()	(x)
N1.3 Opções do Gráfico	O usuário deve ser capaz de mudar algumas opções do gráfico como: Nome do gráfico, Nome do Eixo X e Y.	Interface	(x)	()

Quadro 9 - Requisito Gráfico de Amostras
Fonte: Autoria Própria

F10 Criar Geometrias em Mapa	Oculto()
Descrição: O sistema deve prover a capacidade do usuário de criar geometrias desenhando no mapa.	

Requisitos Não Funcionais				
Nome	**Restrição**	**Categoria**	**Desejável**	**Permanente**
N1.1 Login Obrigatório	A tela deve ser exibida apenas para usuários cadastrado.	Segurança	()	(x)
N1.2 Formulário	O tela deve oferecer a opção de criar linhas, polígonos e pontos.	Interface	()	(x)
N1.3 Pontos com Medida	Os pontos devem receber os dados de medida antes de serem inseridos no banco.	Usabilidade	()	(x)
N 1.3 Importação ao banco de novas tabelas	O usuário poderá enviar seus desenhos ao banco devendo preencher dados como o valor do Srid e o nome da tabela.	Usabilidade	()	(x)
N 1.4 Importação ao a partir das tabelas	O usuário poderá também, inserir novas geometrias em tabelas previamente criadas.	Usabilidade	(x)	()

Quadro 10 - Requisito Criar Geometrias em Mapa
Fonte: Autoria Própria

3.3 ARQUITETURA DO PROJETO

O projeto pode ser divido em duas partes, a aplicação *Web* feita com Java e GWT e a segunda sendo o pacote de funções, procedimentos e gatilhos no PostgreSQL com PostGIS.

3.3.1 ARQUITETURA DA APLICAÇÃO WEB

Sobre a aplicação *Web*, pode-se observar abaixo a lista de pacotes e subdivisões do projeto, tendo está lista uma breve descrição da responsabilidade das partes da aplicação, dividas nas três partes de projetos GWT (*Client,Server,Shared*):

Pacotes no *Client-side:*

- **View:** Este pacote armazena todas as classes responsáveis pela parte visual, o *back-end* da aplicação. Tudo que está relacionado a visualização está contido neste pacote.
- **Presenter:** Toda a lógica e regras de negócio que interessam a View estão neste pacote. Também é responsável por comunicações com o *Server-side*.

- **Inject:** Pacote responsável por gerir toda a injeção de dependência no *Client-side*. Também é usado para armazenar classes com responsabilidade de mapear as telas da aplicação.
- **Events:** Este pacote armazena classes responsáveis pela comunicação entre telas.
- **Gatekeeper:** Este pacote armazena classes responsáveis por regras de negócio relacionadas à exibição ou não de uma determinada tela.
- **Commum:** Este pacote armazena todas as classes que podem servir de modelo para componentes das telas. Também armazena outras classes que servem de alguma forma, de modelo para algo na aplicação.

Pacotes no *Server-side*:

- **ActionHandler:** Este pacote têm como responsabilidade receber as requisições do *Client-Side* e retornar as mesmas. Também faz a comunicação com a camada de acesso ao banco.
- **Connection:** Pacote responsável por criar conexões com o banco de dados na duração da aplicação.
- **Controller:** Responsável por ser o intermediador entre *ActionHandlers* e a camada de acesso ao banco.
- **Converter:** Este pacote tem a responsabilidade de converter alguns objetos em formatos diferentes.
- **Dao:** Este pacote armazena classes que têm a responsabilidade de comunicar com o banco de dados.
- **HttpSession:** Este pacote armazena classes responsáveis por armazenar dados da sessão do usuário.
- **Inject:** Pacote responsável pela gestão da injeção de dependência no *Server-side.*
- **Model:** Armazena classes de modelo da aplicação.
- **Servlet:** Armazena classes que fornecem serviços WEB no *Server-side*.
- **Util:** Armazena classes que tem utilidades diversas na aplicação, porém, não se encaixam em nenhum dos pacotes anteriores.

Pacotes no *Shared-side:*

- **Command:** Armazena classes responsáveis por especificar as requisições e respostas das comunicações entre *Client* e *Server*.

- **Model:** Armazena todas as classes de modelo que precisam ser conhecidas em ambos lados (*Client* e *Server*).
- **Util:** Armazena classes que tem utilidades diversas em ambos lados da aplicação, tais como, enumeradores, constantes, etc.

3.3.2 Arquitetura da Aplicação no Banco de Dados

No banco de dados, temos apenas um subdivisão, uma parte armazena tabelas gerais do banco, e tabelas pertinentes ao PostGIS, na segunda parte temos a subdivisão por usuário, que armazenam tabelas criadas pelos usuários da aplicação. A Figura 19 exibe o banco de dados visualizado no *software* PgAdmin III.

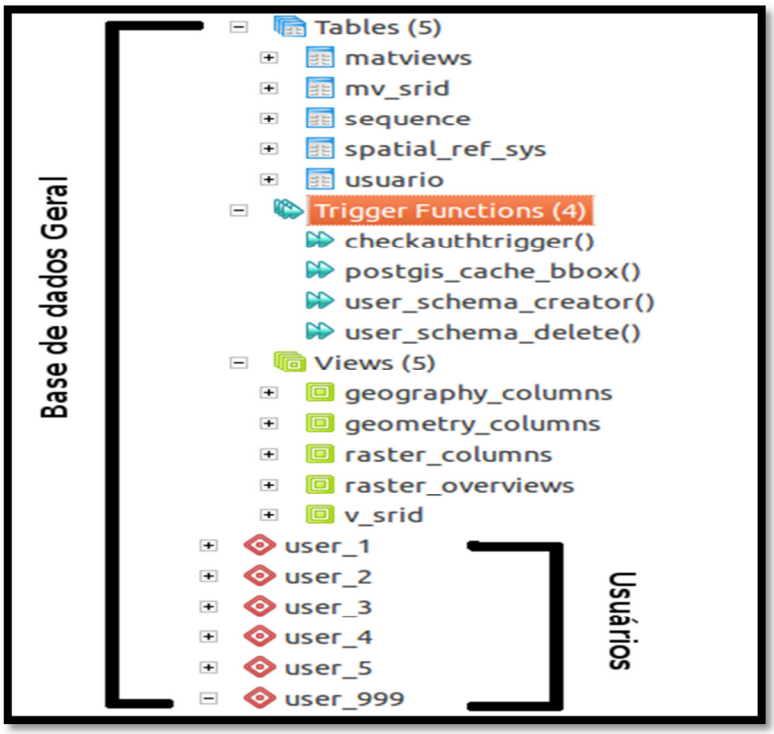


Figura 19 - Visão Geral do Banco de Dados
Fonte: Autoria Própria

As tabelas exibidas na Figura 19 têm as seguintes aplicações:

- **Matviews:** Armazena referências para todas as *Views* materializadas[20]

[20] View materializada é um tipo de tabela que fica fica armazenada no disco, porém funciona como uma view.

- **Mv_srid:** Esta *View* materializada armazena alguns campos da tabela *Spatial_ref_sys* que serão exibidos aos usuários na inserção de dados via arquivo texto.
- **Sequence:** Tabela que mapea *Sequences* do banco.
- **Spatial_ref_sys:** Tabela específica do PostGIS, nela estão armazenados todas os SRID existentes no PostGIS.
- **Usuario:** Tabela responsável por armazenar dados dos usuários do sistema.

Os gatilhos (*Triggers)* **Checkauthtrigger** e **Postgis_cache_bbox** são gatilhos específicos do PostGIS, quanto aos outros:

- **User_schema_creator:** Trigger usada para criação de *Schema*. Chamada a cada inserção na tabela Usuario.

User_schema_delete: Trigger usada para deleção de *Schema*. Chamada a cada remoção na tabela Usuario.Todas as *Views* são específicas do PostGIS com exceção da *View* abaixo*:*

- **V_srid:** View onde se armazena um processamento de remoção de texto da coluna de descrição do *Srid* da tabela Spatial_ref_sys

4 DESENVOLVIMENTO

A presente sessão tem como finalidade apresentar os resultados obtidos na construção da aplicação proposta, além de mostrar a aplicabilidade de todos recursos estudados.

4.1 CHAMADAS AO SERVIDOR

A aplicação trabalha com a gestão de dados espaciais cedidos pelo usuário. Todos estes dados serão guardados no PostgreSQL. Este é o ciclo da aplicação. Dentre os recursos da aplicação e as iterações com o usuário, existe uma sequência lógica para a reposta do recurso solicitado pelo usuário: Preparar a requisição, enviar a requisição ao servidor, enviar estas informações a uma *Stored Procedure* no banco, receber e exibir o resultado ao usuário.

A preparação da requisição consiste em receber as informações necessárias do usuário para os passos seguintes serem concluídos com a exibição do resultado. O envio da requisição é a etapa onde todas as informações passadas serão remanejadas as respectivas classes responsáveis pela comunicação com o servidor. Já no servidor, classes responsáveis por gerir a determinada requisição, farão uma chamada ao servidor de banco de dados e aguardará sua resposta, que quando recebida será retornada ao cliente e exibido ao usuário.

O esquema de chamadas ao Servidor pode ser melhor visualizada através da Figura 20. Nota-se que as setas verdes representam à chamada e as vermelhas a reposta.

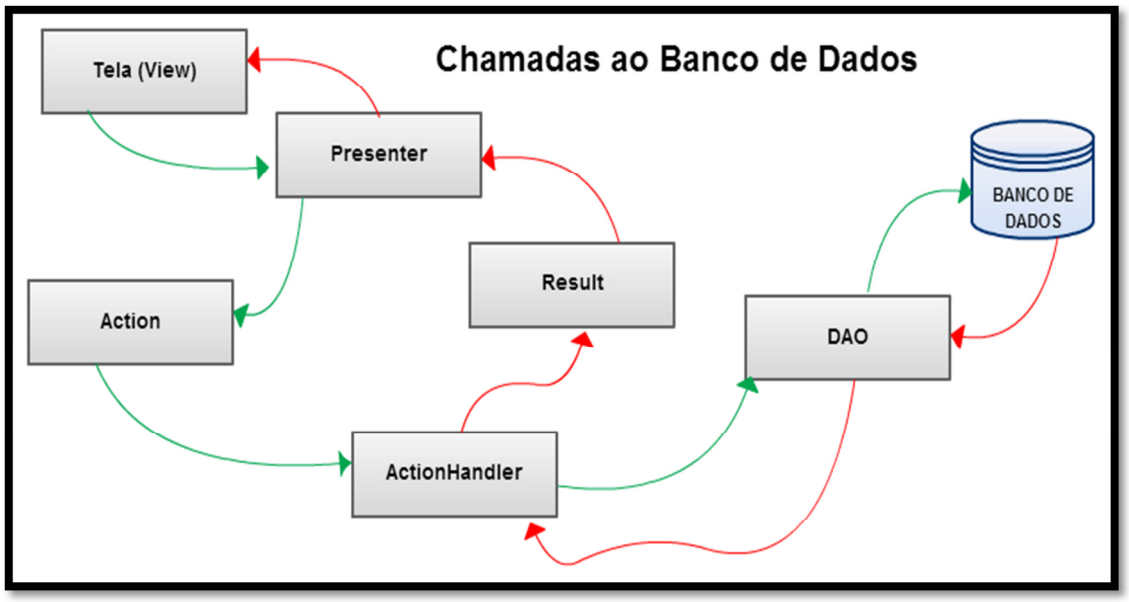


Figura 20 – Ciclo das requisições do projeto
Fonte: Autoria Própria

A preparação das requisições é feita na *Presenter*, recebendo os dados da *View*, logo mais, a etapa de envio de requisição consta no relacionamento entre *Action* e *ActionHandler*. Já no servidor *ActionHandler* envia e recebe dados do banco de dados intermediados pelo *Data Access Object* (DAO). Por fim repassadas ao *Result* no ciclo de resposta.

4.2 ACESSO À APLICAÇÃO

Um dos requisitos não funcional da aplicação é que alguns recursos da aplicação não possam ser acessados sem que um usuário previamente cadastrado esteja autenticado na aplicação. Para que este requisito seja validado precisou-se de três etapas:

1. Cadastrar um usuário
2. Autenticação dos usuários
3. Barrar recursos para usuários não autenticados

4.2.1 Cadastrar Usuários

Um dos requisitos funcionais do sistema é o cadastro de usuários, parte importante da aplicação, pois, os recursos precisam ser guiados por estes usuários para que o banco de dados saiba onde buscar, armazenar e apagar dados.

A Figura 21 exibe a tela responsável pelo cadastro dos usuários.

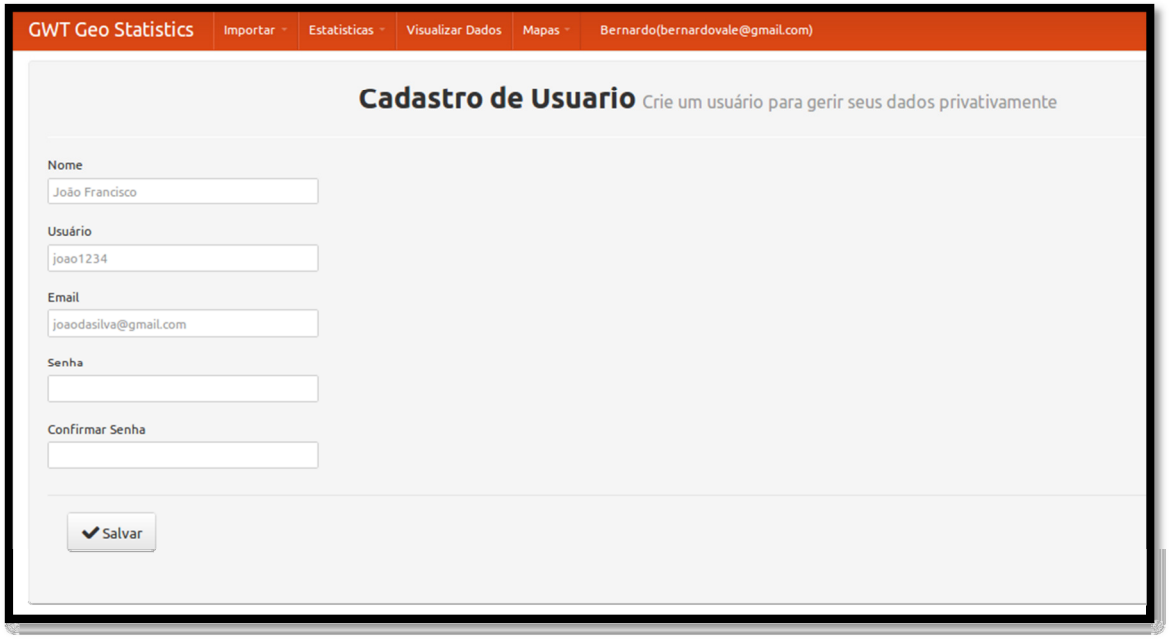


Figura 21 - Cadastro de Usuários
Fonte: Autoria Própria

Observa-se no requisito funcional R1 que todas as informações devem ser preenchidas antes do envio dos dados. Após o preenchimento dos dados e clicar no botão cadastrar ou pressionar a tecla *Enter*, os dados serão enviados ao Servidor e devolvidos com sucesso se os dados forem corretamente preenchidos.

O trecho de código responsável pelo envio dos dados ao servidor pode ser observado na Figura 22.

```
1   /**
2    * Popula o usuario DTO a parti da tela.
3    */
4   private void populaUsuario() {
5       ClearMsgsEvent.fire(this);
6       CadastrarUsuarioView v = getView();
7       if (usuario == null) {
8           usuario = new UsuarioDTO();
9       }
10      usuario.setNome(v.nome().getValue());
11      usuario.setUsuario(v.usuario().getValue());
12      usuario.setEmail(v.email().getValue());
13      usuario.setSenha(v.senha().getValue());
14  }
15  private void doSalvar() {
16      populaUsuario();
17      new AbstractCallback<SalvarUsuarioResult>(this) {
18          @Override
19          protected void callService(AsyncCallback<SalvarUsuarioResult> asyncCallback) {
20              dispatcher.execute(new SalvarUsuarioAction(usuario), asyncCallback);
21          }
22          @Override
23          public void onSuccess(SalvarUsuarioResult result) {
24              if(result.isOk()){
25                  String s = StringUtil.arrayToString(result.getMsgs());
26                  ShowMsgEvent.fire(CadastrarUsuarioPresenter.this,
27                          "Usuário cadastrado",AlertType.SUCCESS);
28                  doNovo();
29              }else{
30                  ShowMsgEvent.fire
31                  (CadastrarUsuarioPresenter.this, StringUtil.arrayToString(result.getMsgs())
32                  ,AlertType.ERROR);
33              }
34          }
35      }.go();
36  }
```

Figura 22 - Trecho de código do envio do usuário ao servidor
Fonte: Autoria Própria

O método especificado na linha 16 (Figura 22) é responsável por obter da *View* os valores dos campos preenchidos e inseri-los no objeto “usuário”. O método “doSalvar”, acionado ao clicar no botão “Salvar”, envia ao servidor o objeto, e aguarda seu resultado, exibindo uma mensagem de confirmação caso obtenha problemas e outra se tudo ocorreu bem. Nota-se que a propriedade booleana “isOk” distingue as duas repostas.

Ao serem capturados pelo *ActionHandler*, ocorre a comunicação com o banco de dados, e mediante ao resultado, respostas diferentes (Figura 23) são enviadas novamente ao cliente.

```
@Override
   public SalvarUsuarioResult execute(SalvarUsuarioAction action, ExecutionContext context)
   throws ActionException {
       List<String> msgs;
       try {
          //Verifica no banco se um nome já foi usado
           if(dao.existeNome(action.getUsuario().getUsuario())){
               return new SalvarUsuarioResult(false, Arrays.asList("Usuário já existe, tente outro nome!"));
           }else{
               dao.save(converter.toBean(action.getUsuario()));
           }
           //Problemas com os dados, enviar falha ao usuário
       } catch (ConstraintViolationException cve) {
           msgs = new ArrayList<String>();
           for (ConstraintViolation<?> violation : cve.getConstraintViolations()) {
               msgs.add(violation.getMessage());
           }
           return new SalvarUsuarioResult(false, msgs);
           //Problemas diversos
       } catch (Exception e) {
           e.printStackTrace();
           return new SalvarUsuarioResult(false, Arrays.asList("Erro fatal: " + e.getMessage()));
       }
        return new SalvarUsuarioResult(true, Arrays.asList("Usuario salvo"));
   }
```

Figura 23 - ActionHandler responsável pelo cadastro de usuários
Fonte: Autoria Própria

O método “execute” das classes de *ActionHandler* recebem os dados cedidos pelo cliente em sua *Action* e repassam um *Result* ao cliente. Observa-se na primeira operação de controle que uma função “existeNome” do DAO é chamada, esta função verifica se o nome escolhido pelo usuário já existe e tenta salvar o usuário.

Atenta-se também as exceções da cláusula *Catch,* a classe *ConstraintViolationException* captura deformidades quanto a um padrão previamente estabelecido na classe modelo de um determinado objeto (Figura 24). Estas deformidades são capturadas e pela exceção e adicionadas a um *ArrayList* que conterá todas as mensagens de erro.

```java
@Entity
public class Usuario implements Bean {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "seq_usuario")
    @SequenceGenerator(name = "seq_usuario",sequenceName = "seq_usuario", allocationSize = 1)
    private Long id;
    @NotNull(message = "Campo Nome vazio")
    @Size(min = 6, max = 90, message = "Nome deve ter entre 6 e 90 caracteres.")
    private String nome;
    @NotNull(message = "Campo Usuario vazio")
    @Size(min = 5, max = 25, message = "Usuario deve conter entre 5 e 25 caracteres.")
    private String usuario;
    @NotNull(message = "Campo Senha vazio")
    @Size(min = 6, max = 20, message = "Senha deve conter entre 6 e 20 caracteres.")
    private String senha;
    @Email(message = "Email invalido.")
    @NotNull(message = "Campo email vazio")
    private String email;
```

Figura 24 - Classe modelo Usuário
Fonte: Autoria Própria

Se a cláusula *Catch* não for acionada no banco de dados, um novo registro de usuário será criado. Neste banco, um gatilho está previsto para ser acionado a cada inserção na tabela Usuario. A sua função é criar um *Schema* (Figura 25) que armazenará todas as tabelas que este usuário irá criar, deletar, modificar enquanto utilizar esta aplicação.

Um Schema é uma forma de trabalhar com diversos banco de dados em um único banco. Previamente criado, existe o *Schema* public que armazena tudo, quando não for especificado um *Schema*. Porém pode-se criar diversos *Schemas* para facilitar a organização e também a segurança. Na aplicação a maior importância é que o usuário consiga enchergar apenas o que criou, logo, todas as suas operações devem ir de encontro com seu próprio *Schema*.

Outra Trigger, encarrega-se de deletar o *Schema* quando o mesmo registro de usuário for apagado (Figura 25). Observa-se que é utilizada a denotação “user_” mais o Identificador único (ID) daquele usuário, assim, um padrão único para chamar recursos de um *Schema* é definido.

```
CREATE OR REPLACE FUNCTION user_schema_creator()
RETURNS trigger LANGUAGE plpgsql
AS
$$
begin
        EXECUTE 'CREATE SCHEMA user_'||new.id||' AUTHORIZATION postgres';
        return new;
end;
$$
CREATE OR REPLACE FUNCTION user_schema_delete()
RETURNS trigger LANGUAGE plpgsql
AS
$$
begin
        EXECUTE 'DROP SCHEMA user_'||old.id||' cascade';
        return new;
end;
$$
create trigger usuarios_schema after insert
on usuario for each row
EXECUTE PROCEDURE user_schema_creator();

create trigger usuarios_delete_schema after delete
on usuario for each row
execute procedure user_schema_delete();
```

Figura 25 - Trigger de inserção e deleção de usuários
Fonte: Autoria Própria

4.2.2 Autenticação dos Usuários

Como observado sobre os *Schemas*, precisa-se que um usuário esteja autenticado na aplicação para obter seu ID e assim conseguir encontrar os seus recursos contidos no seu *Schema*, logo, precisa-se que usuário forneça suas credenciais antes de utilizar os diversos recursos que à aplicação oferece.

A Tela de Autenticação pode ser visualizada a partir da Figura 26

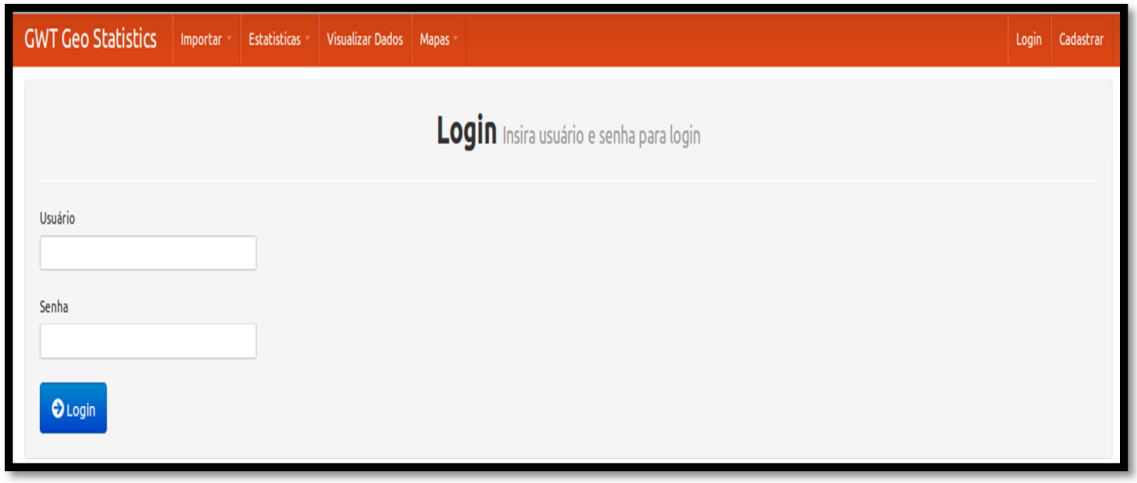


Figura 26 - Tela de Autenticação do sistema
Fonte: Autoria Própria

Quando preenchidos, os dados serão enviados ao servidor, onde a verificação da idoneidade será feita, caso verdadeira, o objeto do usuário será gravado na sessão de seu navegador, para que assim, outras telas possam verificar se o usuário existe e se continua no sistema.

O Figura 27 exibe o trecho de código responsável pela chamada ao servidor.

```
private void doLogin() {
    ShowLoadingEvent.fire(LoginPresenter.this, "");
    dispatcher.execute(new BuscarUsuarioAction(getView().usuario().getValue(),
    getView().senha().getValue()),
            new AsyncCallback<BuscarUsuarioResult>(){
        @Override
        public void onFailure(Throwable throwable) {
            HideLoadingEvent.fire(LoginPresenter.this);
            ShowMsgEvent.fire(LoginPresenter.this,throwable.getMessage(), AlertType.ERROR);
        }

        @Override
        public void onSuccess(BuscarUsuarioResult r) {
            HideLoadingEvent.fire(LoginPresenter.this);
            if(r.isOk()){
                ShowMsgEvent.fire(LoginPresenter.this, AuthStatus.LOGIN.msg, AlertType.SUCCESS);
                keeper.setUsuario(r.getUsuario(),true);
            }
            else{
                ShowMsgEvent.fire(LoginPresenter.this,
                        Erros.CREDENCIAIS_INVALIDAS.getNomeErro(),AlertType.ERROR);
            }
        }
    });
}
```

Figura 27 - Código da autenticação dos usuários
Fonte: Autoria Própria

Observando o código pode-se notar a chamada de duas classes “ShowLoadingEvent” e “HideLoadingEvent”, estas, são responsáveis respectivamente por exibir uma janela *Popup* informativa sobre a chamada ao servidor e remover esta janela assim que a chamada estiver concluída. Os recursos usados nesta chamada são similares a chamada para cadastro de usuários. Uma verificação booleana e um informativo a respeito do resultado da operação.

No servidor, são feitas duas operações, primeiramente, a verificação se as credencias enviadas correspondem a algum registro na tabela Usuario. Depois, o objeto é armazenado na sessão do navegador (Figura 28).

```
@Override
public BuscarUsuarioResult execute(BuscarUsuarioAction a, ExecutionContext context)
throws ActionException {
    try {
        Usuario u = control.autenticar(a.getLogin(),a.getSenha());
        if(u != null){
            requestProvider.get().getSession()
                    .setAttribute(SessionAtributtes.USUARIO_LOGADO, u);
            return new BuscarUsuarioResult(true,cv.toDTO(u));
        }else{
            return new BuscarUsuarioResult(false,null);
        }
    } catch (Exception e) {
        e.printStackTrace();
        return new BuscarUsuarioResult(false,null);
    }
}
```

Figura 28 - ActionHandler da autenticação
Fonte: Autoria Própria

Neste trecho faz-se uso de novos recursos, como uma controladora, para intermediar o DAO, também, é usado uma classe chamada UsuarioConverter(“cv”) que é responsável por converter um DTO para um *Bean* que somente o servidor conhece, e vice-versa. Por fim a classe *RequestProvider*, um *Servlet* do GWT para lidar com sessão, ela é usada para adicionar o objeto enviado pelo cliente.

4.2.3 Barrar Recursos

Previamente especificado em todos os requisitos funcionais exceto F1 e F5, um usuário não autenticado deve ser incapaz de acessar todas as outras telas. Para prover este recurso de segurança, fez-se uso de um recurso do GWTP chamado *Gatekeeper*.

Do inglês *Gatekeeper* significa Porteiro. Exatamente esta é sua função, um porteiro de telas. A implementação desta classe consiste em apenas um método chamado “canReveal()” que retorna um booleano. Quando um usuário pedir a exibição de uma classe que possui este “porteiro”, o método *canReveal()* é chamado, e retorna verdadeiro se puder exibir a tela requisitada.

Mediante esta habilidade do GWTP, criou-se um *Gatekeeper* para proteger as telas que não podem ser acessadas sem autenticação do usuário. A Figura 29 exibe os três métodos chaves do Gatekeeper criado na aplicação para barrar os usuários não autenticados.

```
    @Override
    public boolean canReveal() {
        return getUsuario() != null;
    }
public UsuarioDTO getUsuario()
    {
        if(usuario == null)
        {
            loadUsuarioFromServer();
        }
        return usuario;
    }
    private void loadUsuarioFromServer() {
        dispatchAsync.execute(new BuscarUsuarioLogadoAction(),
              new AsyncCallback<BuscarUsuarioLogadoResult>() {
            @Override
            public void onFailure(Throwable caught) {

            }
            @Override
            public void onSuccess(BuscarUsuarioLogadoResult result) {
                if(!(result.getDto().equals(usuario))){
                    setUsuario(result.getDto());}
                else
                {setUsuario(result.getDto());}
            }
        });
    }
```

Figura 29 - Classe LoginGateKeeper
Fonte: Autoria Própria

Pode-se observar o fluxo da chamada ao Gatekeeper a partir da Figura 29. O método "canReveal" é chamado, o mesmo faz uma comparação booleana com o método "getUsuario", responsável por guardar os dados do usuário presente no sistema. Se não existir, o método ainda tenta buscar na sessão, no lado do servidor, algum objeto pertencente a algum usuário autenticado.

Com a criação do *Gatekeeper*, se uma tela precisar de autenticação basta adicionar a anotação "UseGatekeeper" e o nome do *Gatekeeper* em seu *Proxy* (Figura 30).

```
1  @UseGatekeeper(LoginGateKeeper.class)
2  public interface ImportarManualProxy
3                  extends ProxyPlace<ImportarManualPresenter>{}
```

Figura 30 - Uso do Gatekeeper na tela de importação de dados manual
Fonte: Autoria Própria

4.3 IMPORTAR DADOS POR ARQUIVO TEXTO

Uma das fontes de obtenção de dados para à aplicação é a criação de geometrias a partir de um arquivo texto contendo valores de latitude, longitude, altitude e medida. Um usuário faz *upload* de um arquivo texto contendo tais valores e o resultado final é uma tabela com valores geométricos em seu *Schema* de acordo com os dados do arquivo.

A Figura 31 exibe a tela para importação de dados por arquivo texto.

Figura 31 - Tela de Importação de dados geográficos
Fonte: Autoria Própria

Dentre os campos contidos na tela (Figura 31) alguns são usados para orientação do PostGIS, outros para orientação das classes que faram a varredura do arquivo.

O *Spatial Reference System Identifier* (SRID) informa ao banco em que projeção encontra-se os dados geométricos. Em aplicações GIS, SRID é um valor único que representa uma projeção geográfica. No PostGIS quando precisa-se criar uma coluna geográfica, esse atributo precisa ser especificado. A função do PostGIS *AddGeometryColumn* (Figura 32), responsável por criar uma coluna geográfica numa determinada tabela, requer obrigatoriamente que especifique um SRID.

```
text AddGeometryColumn(
    varchar catalog_name,
    --Opcional: Catálogo de dados
    varchar schema_name,
    --Opcional: Nome do Schema
    varchar table_name,
    --Nome da tabela
    varchar column_name,
    --Nome da coluna geométrica
    varchar srid,
    --Identificador único da projeção
    varchar type,
    --Tipo(Point,Linestring,Polygon...)
    integer dimension
    --Dimensão 2=2D,3=3D
);
```

Figura 32 - Função AddGeometryColumn
Fonte: Autoria Própria

Observa-se na Figura 32 que alguns dos dados que se encontram no formulário de inserção, são obrigatórios para a criação de uma coluna geométrica.

O caractere delimitador é uma informação de necessidade dos algoritmos de varredura do arquivo texto. Esse caractere será aquele em que no arquivo texto, separa as colunas e as linhas.

O Tipo Geométrico é utilizado na função de inserção do PostGIS. Nela o usuário deverá escolher que tipo de geometria estará inserindo no banco. *Point*, *Linestring* ou *Polygon*.

O Nome da tabela refere-se à tabela resultante após a inserção dos dados.

O botão de Mais Opções, quando acionado, exibe um *Popup* perguntando sobre o desejo de buscar mais SRID no banco de dados. A utilização deste recurso se da a partir do seguinte problema, existem mais de cinco mil registros de SRID, logo, o processo de obtenção dos mesmos é demorado. Como o público alvo desta aplicação utilizará em grande parte a SRID brasileira, a mesma fica fixada no campo de SRID, permitindo, a partir deste botão, a obtenção da lista completa no banco.

A dimensão geométrica é necessária tanto para os algoritmos de varredura, quanto para a função do banco, pois, se a escolha for três dimensões, deverá obrigatoriamente existir uma coluna com os dados da altitude.

O campo “Tem Cabeçalho?” informa aos algoritmos de varredura se a primeira linha do arquivo texto será usada para denominar a ordem das colunas do arquivo texto. Opcional, somente escolhida quando a orientação do arquivo texto foge do padrão de ordenação das colunas: Latitude, Longitude, Altitude, Medida (Figura 33).

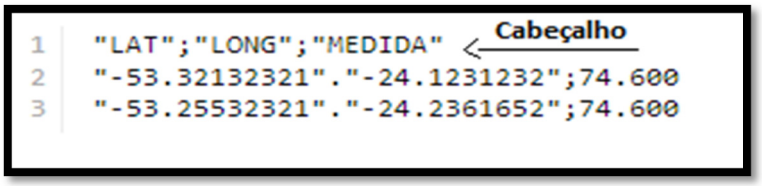


Figura 33 - Cabeçalho dos arquivos texto.
Fonte: Autoria Própria

As colunas geométricas Latitude, Longitude, Medida, Altitude. São campos que aparecem para preencher caso o usuário informar que seu arquivo texto possui um cabeçalho,tal qual no Figura 33, se o fizer, deverá escrever o nome da coluna exatamente como está em seu cabeçalho.

Para criar uma geometria de duas dimensões, por exemplo, precisa-se de dois dados geográficos, latitude e longitude, porém, ambos são números flutuantes e seria impossível para os algoritmos de varredura apontar quem é quem, mediante a esse empecilho, surgiu a necessidade de preencher esses dados, o usuário indica que sua coluna de latitude tem o nome “LAT” como no Figura 33, por exemplo, dessa maneira, o algoritmo de varredura saberá que os dados de latitude estão na coluna do texto “LAT”. No caso de dados com três dimensões surge a coluna de Altitude, e caso esteja criando um *Point*, ainda preencherá a coluna de Medida, que seria um valor de amostra daquele ponto geográfico.

Caso não exista cabeçalho, os algoritmos de varredura seguem a sequência lógica das colunas, a primeira coluna será de latitude, a segunda de longitude, caso exista terceira, será altitude, caso exista quarta, será a medida.

Dentre as informações importantes a serem citadas sobre esta habilidade da aplicação:

- As classes de varredura tem como ponto de referência o caractere de quebra de linha, portanto, uma linha é tratada como um ponto na geometria resultante;

- Um arquivo texto gera apenas uma geometria do tipo *Linestring* ou *Polygon*, já quando a escolha for pontos, cada linha do arquivo texto será um registro na tabela resultante;
- Após o envio ao servidor, *Stored Procedures* farão a criação da tabela e também a inserção dos dados naquela tabela.

4.4 IMPORTAÇÃO MANUAL

Conforme o requisito funcional F4 sobre a necessidade desta característica, a importação manual é uma forma de armazenar dados geométricos em tabelas já criadas sem a necessidade de arquivos texto, apenas digitando os valores geométricos correspondentes.

A Figura 34 exibe o resultado final da tela de Importação Manual.

Figura 34 - Tela Importar Dados Manualmente
Fonte: Autoria Própria

Observa-se na Figura 34 a obrigatoriedade de seleção de uma tabela através do primeiro campo do formulário.

Nesta tela o usuário seleciona a tabela específica que irá receber os dados, e insere os valores das colunas nos respectivos campos.

Como algumas tabelas possuem campos a mais ou a menos, existem algumas regras de negócio para que o resultado seja coerente. Ao inserir a primeira linha de dados, o campo da tabela fica bloqueado (Figura 34), sendo desbloqueado somente quando a transação dos dados for completa ou quando o mesmo acionar o botão “Cancelar”. Desta maneira a aplicação garante que dados errôneos não sejam inseridos.

O botão “Visualizar”, conforme o requisito não-funcional 1.3, é usado para que se exiba em uma tabela, todos os dados que já foram inseridos (Figura 35) e que serão enviados ao servidor assim que o usuário concluir a transação acionando o botão “Confirmar”.

Figura 35 - *Popup* de pré-visualização das tuplas
Fonte: Autoria Própria

Após o envio ao banco de dados, uma *Stored Procedure* responsável pelo tipo de geometria presente na tabela será acionada para armazenar os dados.

4.5 VISUALIZAR DADOS

A tela de Visualização de Dados (Figura 36) é responsável por exibir em completo, todas as tabelas e registro criadas pelo usuário. Nela pode-se também excluir uma tabela da base de dados do usuário (*Schema*).

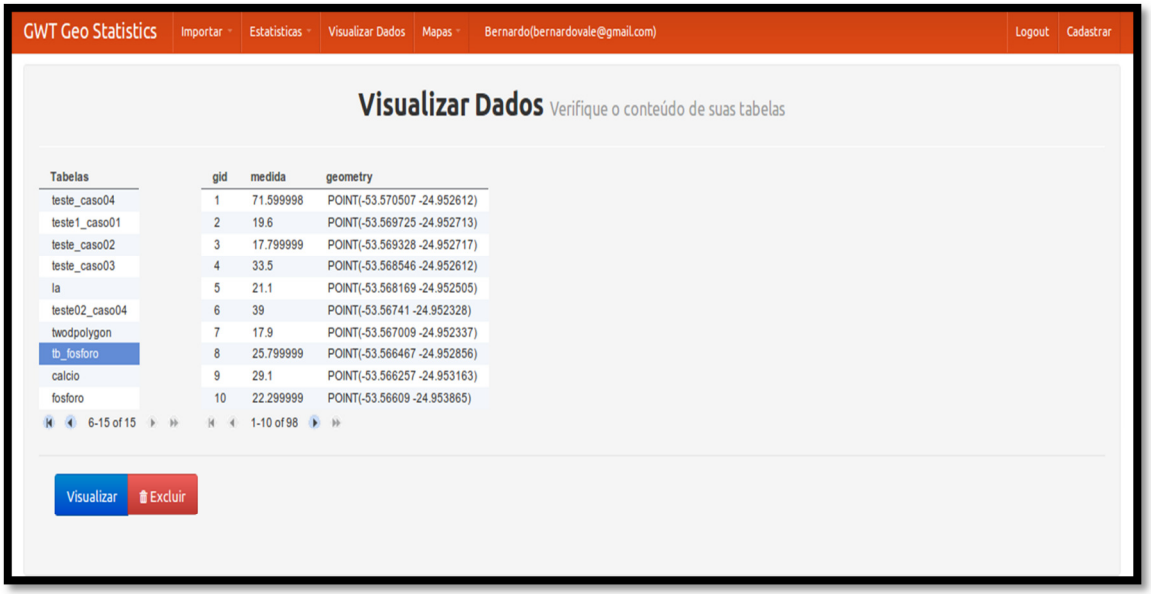


Figura 36 - Tela Visualizar Dados
Fonte: Autoria Própria

Nesta tela (Figura 36) podemos observar quatro elementos. Um painel que exibe todas as tabelas do banco de dados do usuário, outro painel para exibir os registros e dois botões, Visualizar e Excluir.

Ao iniciar esta tela, apenas serão carregados os nomes de todas as tabelas presentes no *Schema* do usuário. Caso um usuário selecione um item e acione o botão Visualizar, este, irá chamar o servidor, pedindo todos os valores daquela tabela (Figura 35), ou seja, nome de todas as colunas e todos os registros da mesma. Os componentes da tela são dinâmicos, e irão criar colunas e linhas exatamente igual a estrutura da tabela no banco de dados.

Outra opção nesta tela é de excluir uma tabela presente do banco de dados, a partir da ação executada pelo botão “Excluir” (Figura 36).

4.6 GRÁFICO DE AMOSTRAS

Na tela do gráfico de amostras (Figura 37) , o usuário tem a habilidade de criar um gráfico dinâmico de acordo com os valores amostrais de sua tabela. Atenta-se que apenas tabelas que armazenam pontos podem gerar tal gráfico, pois, são as únicas que possuem valores de amostra.

Figura 37 - Tela do Gráfico de Amostras
Fonte: Autoria Própria

Observa-se na tela um botão chamado Opções, este quando acionado exibe (Figura 38) algumas opções de edição do gráfico gerado. Estas opções são usadas para modificar o gráfico resultante da operação São elas: Nome do Gráfico, Eixo X e Eixo Y.

Figura 38 - Opções do Gráfico de Amostras
Fonte: Autoria Própria

Já o botão “Gerar”, quando acionado, buscará os valores da tabela no banco de dados e em seguida, chamará as funções de geração do gráfico, resultando um gráfico com os valores amostrais na forma de *Scatterplot* com as opções específicas criadas (Figura 39).

Figura 39 - Gráfico gerado a partir da tabela especificada
Fonte: Autoria Própria

4.7 CÁLCULO DE CORRELAÇÃO ESPACIAL

O cálculo de correlação espacial, trabalha analisando os valores de duas tabelas. Tendo em vista os valores de medida das duas, ou seja, somente tabelas que armazenam pontos podem usufruir desta característica da aplicação.

A Figura 40 exibe a tela responsável pelo cálculo.

Figura 40 - Tela da Correlação
Fonte: Autoria Própria

Após selecionar as tabelas e clicar no botão “Gerar”, no banco de dados, uma *Stored Procedure* fará os cálculos e retornará uma lista com quatro números. Estes números

correspondem o cálculo referente as comparações de X com Y, X com X, Y com X e Y com Y, sendo X a primeira tabela selecionada e Y a segunda.

Os resultados podem ser observados na Figura 41

Figura 41 - Resultados da correlação
Fonte: Autoria Própria

4.8 VISUALIZAR DADOS NO MAPA

A tela para visualização de dados no mapa compreende-se em um visualizador de mapas que exibe o globo terrestre, podendo o usuário, mudar sua orientação conforme queira, por exemplo, pode-se visualizar o globo conforme os dados do satélite, mapas de ruas, de forma híbrida ou a forma padrão do servidor de mapas Google Maps V3.

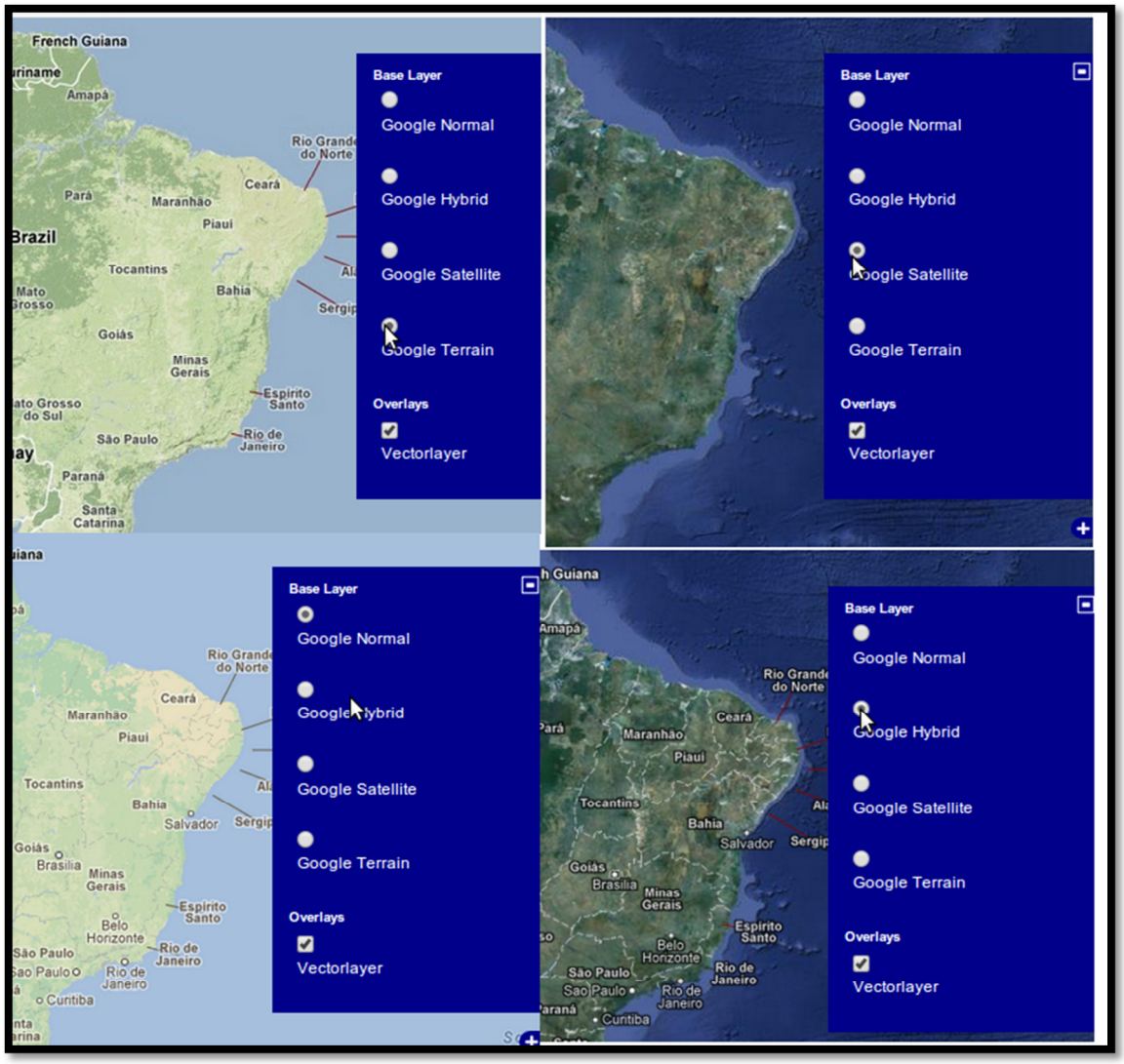


Figura 42 - Base Layers dos Mapas
Fonte: Autoria Própria

As Geometrias da base de dados do usuário também podem ser inseridas e visualizadas no mapa, conforme o requisito não funcional N1.2 do requisito funcional R7. Cada tabela do usuário pode ser inserida e removida conforme solicitado, sendo visualizada como vetor gráfico no mapa (Figura 43).

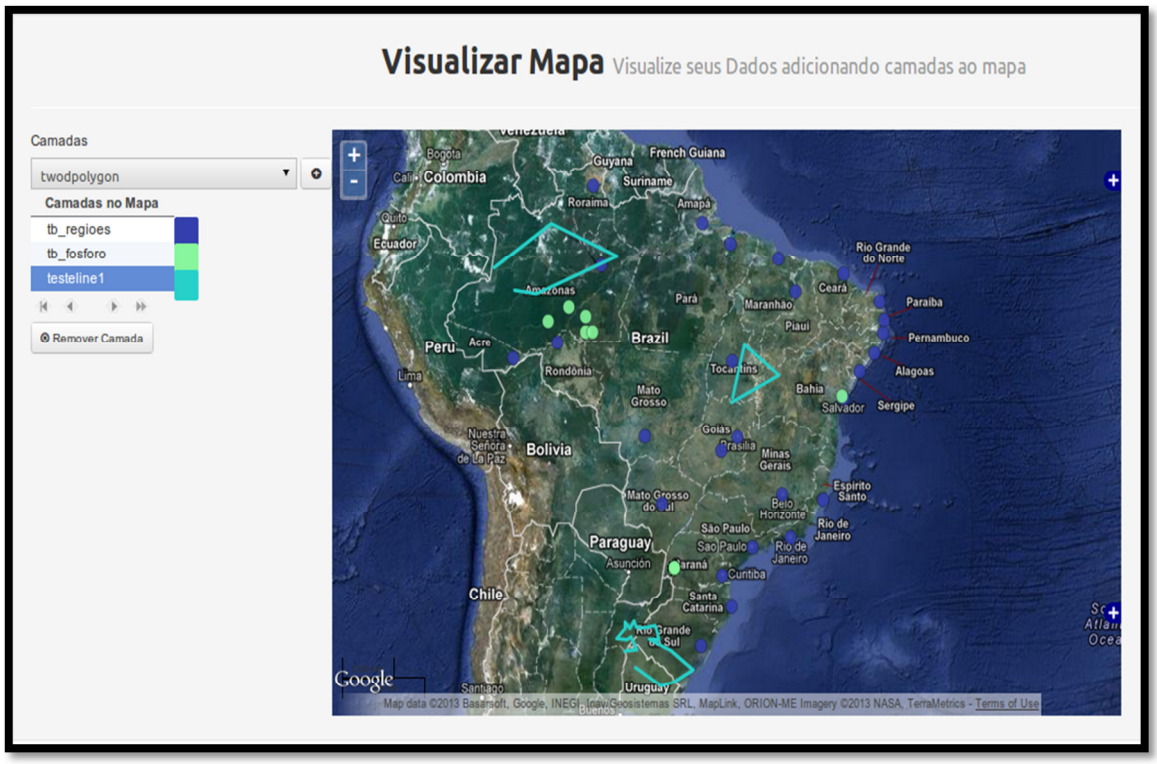


Figura 43 - Inserção de Geometrias no Mapa
Fonte: Autoria Própria

Pode-se observar a partir da Figura 43 que a cada inserção no mapa, uma cor diferente é relacionada à tabela para que se visualizem diversas camadas sem confundir umas as outras. Ainda como auxílio, uma amostra da cor é adicionada ao lado da tabela que se armazenam os nomes das camadas presentes no mapa.

A Figura 44 exibe o trecho de código para montar o objeto que representa o mapa na tela.

```
public static MapWidget buildGoogleMap(TipoGeografico tipo){
    MapOptions defaultMapOptions = new MapOptions();
    defaultMapOptions.setNumZoomLevels(16);
    defaultMapOptions.setMaxExtent(new Bounds(
            -20037508.34, -20037508.34, 20037508.34, 20037508.34
    ));
    //Cria o Objeto MAPA
    MapWidget mapWidget = new MapWidget("800px", "500px", defaultMapOptions);
    //Base Layers do Google Maps

    GoogleV3Options gHybridOptions = new GoogleV3Options();
    gHybridOptions.setIsBaseLayer(true);
    gHybridOptions.setType(GoogleV3MapType.G_HYBRID_MAP);
    GoogleV3 gHybrid = new GoogleV3("Google Hybrid", gHybridOptions);

    //Objeto onde se adicionam todas as Camadas
    final Map map = mapWidget.getMap();

    map.addLayer(gHybrid);


   //Controles do mapa, como, mudar camadas,zoom,Visualizador Rapido.
    map.addControl(new OverviewMap());
    map.addControl(new ScaleLine()); //Display the scaleline
    map.addControl(new LayerSwitcher());
    //Metodo para adicionar a opção de desenhar, presente em algumas telas.
    addDrawFeature(vectorLayer,map,tipo);

    //Põe o zoom no centro do Brasil
    LonLat lonLat = new LonLat(-48.90414886474542,
            -14.760339037323494);
    lonLat.transform(DEFAULT_PROJECTION.getProjectionCode(),
            map.getProjection()); //transform lonlat para OSM
    map.setCenter(lonLat, 4);
    mapWidget.getElement().getFirstChildElement().getStyle().setZIndex(0);
    return mapWidget;
}
```

Figura 44 - Método de construção do mapa
Fonte: Autoria Própria

Todas as telas que precisam de mapas utilizam este método (Figura 44), todas as opções básicas para montar um mapa no GWT-OpenLayers estão presentes. Este método retorna um objeto do tipo *MapWidget* que representa o visualizador de mapas.

4.9 CRIAR GEOMETRIAS NO MAPA

Como outra forma de importação de dados à aplicação, dispõe-se da importação a partir do desenho das geometrias no mapa. Esta importação pode ser feita de duas maneiras, a partir de uma tabela existente na base de dados ou criando uma nova tabela para importação. Para fins de organização, estas duas maneiras foram separadas em telas diferentes.

4.9.1 Importação a partir de Tabelas

A tela para importação a partir de uma tabela requer apenas a pré-seleção de uma tabela. A partir desta seleção a aplicação verifica que tipo de geometria se encontra naquela tabela e disponibiliza um mapa para desenhar aquele tipo geométrico.

A Figura 45 exibe a tela de importação a partir de Tabela

Figura 45 - Tela de Importação a partir de Tabela
Fonte: Autoria Própria

A partir da seleção, também são criados does botões: “Cancelar” e “Salvar”. Ao acionar o botão “Cancelar”, volta-se a tela de seleção da tabela e todos os desenhos são descartados, ao passo que se acionar o botão “Salvar”, os desenhos serão convertidos em valores de Latitude e longitude e inseridos na respectiva tabela.

No caso da tabela ser do tipo “Ponto”, uma grade também será exibida na tela, esta, (Figura 46) deve ser preenchida com os valores da coluna de medida de cada ponto inserido na tela. Quando adicionados no mapa uma função tem a responsabilidade de adicionar mais

uma linha na grade dos valores da medida, bastando o usuário alterar o valor da medida para o valor que desejar.

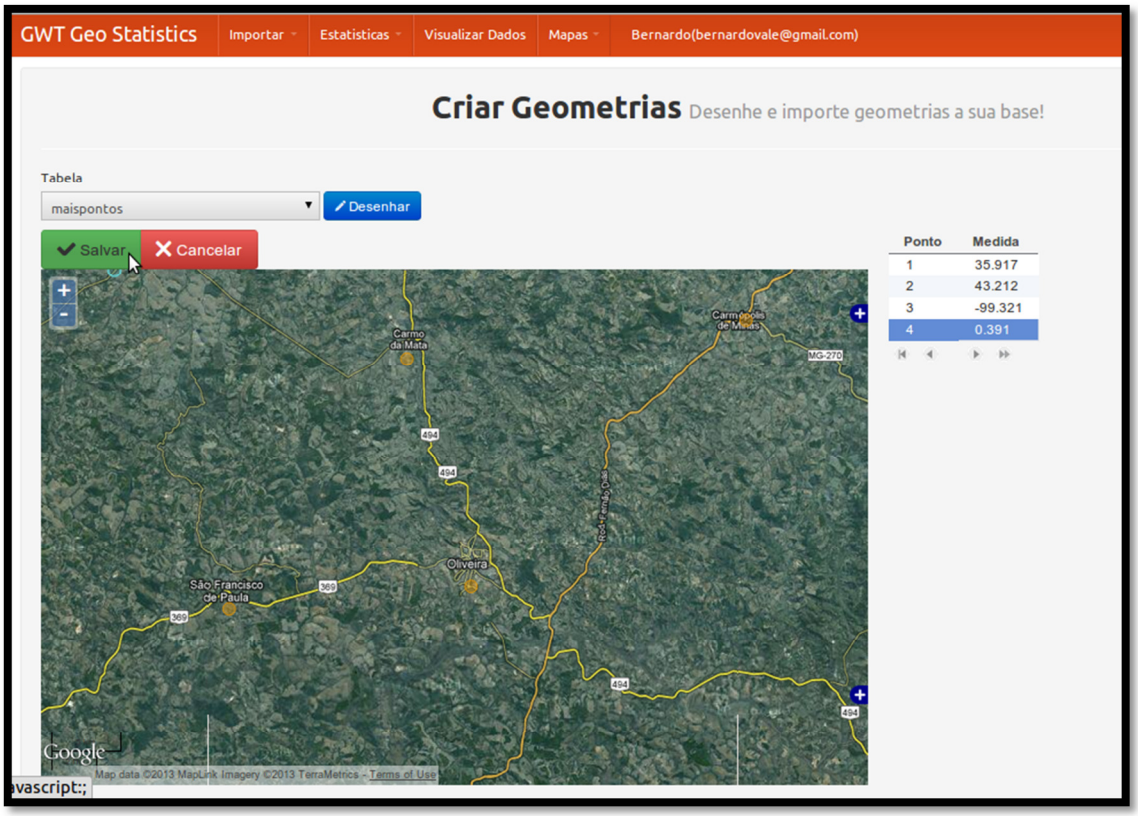


Figura 46 - Criando pontos em tabela existente
Fonte: Autoria Própria

Nota-se que as *Stored Procedures* responsáveis pela adição dos valores as tabelas são as mesmas que são utilizadas na tela de importação manual, já que, utilizam os mesmos mecanismos de importação, pois, estes desenhos, são transformados em valores de Latitude e Longitude.

4.9.2 Importação para novas Tabelas

Esta tela difere-se um pouco da anterior, pois, necessita de algumas informações a mais, pois, ira criar uma nova tabela. Todas estas informações são obrigatórias, sem elas, o PostGIS não conseguira construir a coluna geométrica, tão pouco a *Stored* Procedure conseguira criar a tabela.

Primeiramente, o usuário deve, obrigatoriamente, preencher o formulário (Figura 47) com os dados da nova tabela, são eles: Nome da tabela, valor do SRID e tipo geométrico.

Figura 47 - Tela para Criar novas Tabelas de Geometria
Fonte: Autoria Própria

Acionando o botão “Desenhar” e tendo preenchido corretamente os campos, a tela mudara de forma escondendo o formulário e exibindo o mapa para desenho das geometrias (Figura 48), exatamente como, a tela de Importação a partir de tabelas. Pode-se notar também que a mesma tabela dos valores da medida será exibida caso a tipo geométrico for “Ponto”.

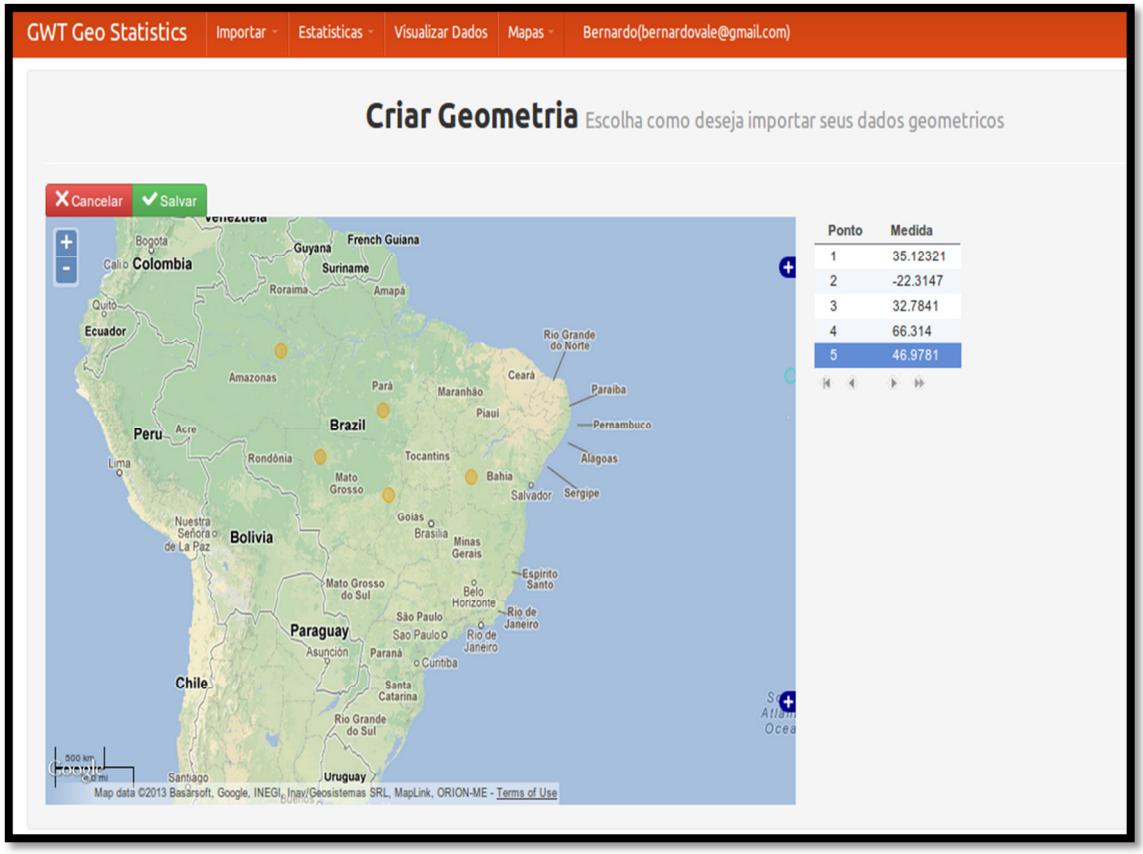


Figura 48 - Adição de pontos no mapa
Fonte: Autoria Própria

Os dois botões “Cancelar” e “Salvar” também são exibidos nesta tela, e exercem a mesma função.

Pode-se notar também que as *Stored Procedures* usadas para salvar as geometrias e a nova tabela são as mesmas usadas na importação de dados a partir de arquivo texto.

5 CONCLUSÃO

Conclui-se após a pesquisa, a imensa aplicabilidade dos bancos de dados espaciais, beneficiando-se do esforço de terceiros, grandes projetos e pesquisas foram desenvolvidas, sendo estas, de grande valia para produzir um SIG.

Projetos na área de geoprocessamento oferecem as organizações ferramentas poderosas para análise e tomadas de decisão, com o surgimento de ferramentas como OpenLayers, este trabalho tende a ficar cada vez mais simples, em consequência, projetos mais complexos surgirão.

Pôde-se constatar como as tecnologias usadas auxiliam no processo de desenvolvimento, sem elas teria sido feito muito pouco, este é o grande diferencial num projeto de *software*, a produtividade.

Com o estudo na área de geoprocessamento e desenvolvimento *Web*, abriu-se um leque de novas oportunidades, pois são áreas que podem ser exploradas em uma pós-graduação para fins acadêmicos ou com fins comerciais através da criação de projetos assim como o proposto no presente trabalho.

5.1 TRABALHOS FUTUROS/CONTINUAÇÃO DO TRABALHO

Aplicações SIG será um novo foco depois do estudo realizado, de fato, existem muitas tecnologias para serem estudadas a fundo. Neste projeto muito ainda pode se fazer, a fim de amostragem, cito uma lista de novas funcionalidades que poderão ser implementadas como melhorias:

- Integração com arquivos *Keyhole Markup Language* (KML) seja para importação no projeto, ou até para exportação das tabelas de dados no formato KML, assim, o usuário poderia visualizar estes dados numa aplicação como o Google Earth.
- Integração com *Shape File*. Estes arquivos são muito utilizados e uma integração seria viável, o governo federal disponibiliza diversas análises para este formato como, focos de queimada, áreas de estrada, áreas de terra indígena, etc. Esta integração facilitaria o trabalho, pois, o usuário não precisaria criar no sistema o que já foi criado por equipes especializadas em geoprocessamento.

- Aplicação de outras funções de estatísticas como, *Boxplot,* Correlação espacial por porção de terra, entre outras. Estas novas funções seriam úteis para dar outras opções na análise dos valores de medida na aplicação.

De fato, existem muitas aplicabilidades para o projeto, à continuação deste poderia resultar inclusive, numa aplicação real para o mercado de trabalho. Concluindo, o interesse em aplicações SIG é grande, e pretende-se ampliar os conhecimentos nesta área.

REFERÊNCIAS

ARONOFF, Stan **Introduction to GIS**, 1989

ASSAD & SANO **Sistema de Informações Geográficas: Aplicações na Agricultura**, Ministério da Agricultura, do Abastecimento e da Reforma Agrária, 1993

.

BATISTA, Claudio. **Banco de Dados Espaciais**. 2006. Disponível em: < http://www.dsc.ufcg.edu.br/~baptista/cursos/SIG/bdespacial.ppt>. Acesso em: 04 Fev.2013.

BEAUDOIN, Philippe, **Google Web Toolkit Platform**, 2011 Disponível em: <https://code.google.com/p/gwt-platform>. Acesso em: 28 de Jan 2013

BLOOMER, John Power, **Programming with RPC**, EUA, 1992

BRITO, J. N.; COELHO, L; **Fotogrametria Digital**. 1. ed. Rio de Janeiro, Brasil: Instituto Militar de Engenharia, 2002. Disponível em: <http://e-foto.sourceforge.net/e-book-pt.html >. Acesso em 21 DE Mar. 2013

CÂMARA, Gilberto; QUEIROZ, Gilberto R., **Arquitetura de Sistemas de Informação Geográfica**. Salvador, 1998. Disponível em: <http://www.dpi.inpe.br/gilberto/livro/introd/cap3-arquitetura.pdf>. Acesso em: 30Jan.2013.

CARTER, G. F. Bonham, Geographic Information Systems for Geoscientists, 1994

COOPER, Robert & COLLINS, Charles, **GWT in Practice**, Greenwich, CT, EUA, 2008

DORNELES S. ; ANDRADE J., **Sistemas de Informação Geográficas e uma proposta para a cidade de Teresópolis,** Rio de Janeiro, 2012. Disponível em: <http://ssdorneles.com/tcc/downloads/artigo_sig.pdf>. Acesso em 26 de Mar. 2013

DOUGLAS, Korry & DOUGLAS, Suzan – **PostgreSQL a comprehensive guide to building, programming, and administering PostgreSQL databases**. 2003, EUA.

EL-RABBANY, Ahmed, **Introduction to GPS: Global Positioning System**, 2002

EZRA, Aviad **Twisting the MVC Triad - Model View Presenter (MVP) Design Pattern**, 2007, Disponível em: <http://aviadezra.blogspot.com.br/2007/07/twisting-mvp-triad-say-hello-to-mvpc.html>. Acesso em 28 de Jan. 2013

FÁTIMA, Maria & SANTOS, Simone, **Conceitos básicos em Sistemas de Informação Geográficos e Cartografia aplicados a Saúde**, Brasília, 2000.

FECOMÉRCIO, **Acesso a internet quase dobra em quatro anos**, Disponível em<http://www.fecomerciorj.org.br/publique/cgi/cgilua.exe/sys/start.htm?infoid=11475&tpl=printerview&sid=90>. Acesso em 18 de Nov. 2012

FURTADO, Vasco, **Tecnologia e Gestão da Informação na Segurança Pública**, Rio de Janeiro, 2002.

GOOGLE, **Communicate with a Server**, 2012, Disponível em: <https://developers.google.com/webtoolkit/doc/latest/DevGuideServerCommunication#DevGuideRemoteProcedureCalls>. Acesso em: 26 de Jan. 2012

GÜTING, Ralf, **An Indroduction to Spatial Database Systems**, Hagen, Alemanha, 1994 Disponivel em: <http://bscw.rediris.es/pub/bscw.cgi/d673129/Introducci%C3%B3n%20a%20Bases%20de%20Datos%20Espaciales.pdf>. Acesso em: 22 de Jan 2013

INPE, **SPRING Versão 4.1**. São Paulo: INPE, 2004. Disponível em: <www.inpe.br>. Acesso em 11 de Mar. 2013

HAZZARD, Erik, **OpenLayers 2.10 Beginner's Guide**, 2011, Birmingham, UK

HURVITZ, P; **The GIS Spatial Data Model**, Washington D.C, EUA, 1998

KUMAR, S, **Basics of Remote Sensing and GIS**, 2005

MATOS, Ana Cristina, **Implementação de Modelos Digitais de Terro para aplicações na Área de Geodésia e Geofísica na América do Sul**, 2005, Disponível em: <http://www.teses.usp.br/teses/disponiveis/3/3138/tde-10102005-104155/pt-br.php>. Acesso em: 11 Mar. 2013

MENESES, Paulo & ALMEIDA, Tati, **Introdução ao Processamento de Imagens e Sensoriamento Remoto**, Brasília, 2012, Disponível em: <http://www.cnpq.br/documents/10157/56b578c4-0fd5-4b9f-b82a-e9693e4f69d8>. Acesso em: 11 de Mar. 2013.

MIRANDA, Fábio, **Padrão MVP para Aplicações Web Cliente-Servidor**, 2009, Disponível em: <http://fabiolnm.blogspot.com.br/2009/12/padrao-mvp-para-aplicacoes-web-cliente.html> Acesso em 28 de Jan 2013

OBE & HSU, **PostGIS in Action**, 2009

OCEAN EXPLORER, **National Oceanic and Atmospheric Administration**. 2013, Disponível em:<http://oceanexplorer.noaa.gov/technology/tools/mapping/media/gis_layers.html> Acesso em 30 de Mar de 2013

ORACLE, **Sobre o PL/SQL**, Disponível em: <http://www.oracle.com/technetwork/database/features/plsql/index.html>. Acesso em: 26 de Jan 2013

PARKER, H. D. **The unique qualities of a geographic information system**, 1988

RAMSEY, P. **PostGis Manual**, 2002 Disponível em <http://postgis.refractions.net>. Acesso em: 26 de Jan 2013.

RIBEIRO, S. C. L. **Automação Fotogramétrica e Geração de Modelos Digitais do Terreno** (MDTs). Dissertação (Mestrado) - Escola Politécnica, Universidade de São Paulo, São Paulo, 1995

SECRETARIA DE EDUCAÇÃO DO PARANÁ, **Aerofotogrametria**, Disponível em: <http://www.geografia.seed.pr.gov.br/modules/galeria/listaEventos.php> Acesso em 01 de Abr 2013

SILVA, Evaldo, **Introdução aos Sistemas de Informação Geográfica**, 2004, Disponível em: <http://www.devmedia.com.br/introducao-a-sistemas-de-informacoes-geograficas/1595> Acesso em: 31 de Jan.2013

SILVA & LIMA, **A Função de Autocorrelação e a Escolha do Passo da Reconstrução**, 2013, Disponível em: < http://legacy.unifacef.com.br/novo/publicacoes/IIforum/Textos%20EP/Antonio%20Carlos%20e%20Fabiano.pdf> Acesso em 03 Abr. 2013

SQL MAGAZINE, **Extensões Espacias em MySQL**, Disponível em: <http://www.sqlmagazine.com.br/resumo_sql14.asp> Acesso em: 26 de Jan 2004

SP PERL MONGERS – **Postgis**, 2011 Disponível em: <http://sao-paulo.pm.org/artigo/2011/PostGIS>. Acesso em 29 de Outubro 2012

POSTGRESQL, **Sobre PostgreSQL,** Disponível em : <http://www.postgresql.org/about/>. Acesso em: 22 de Jan 2013

UFF – Universidade Federal Fluminense Estudo Dirigido em SIG, **Sistemas de Informação Geográfica e Geoprocessamento**, 2005, Disponível em: < http://www.professores.uff.br/cristiane/Estudodirigido/SIG.htm> Acesso em 04 Fev 2013.

UNBC GIS LAB. **Introduction to Geographic Information System**. Lecture 3b. Canada: University of Northern British Columbia. Disponível em: http://www.gis.unbc.ca/courses/geog300/lectures/lect6/index.php. Acesso em 04 Fev 2013.

WORBOYS, M., **GIS: A computing perspective**, 1995

ZEBKER, Howard, **Radar remote sensing at Stanford**, 2013, Disponível em:<http://ee.stanford.edu/~zebker> Acesso em 01 de Abr 2013