A Server-Side GIS Web Service

WARINPORN BOONPRATHAM
TANUNPUCHR CHANSUWAN

BACHELOR OF ENGINEERING PROGRAM IN SOFTWARE ENGINEERING
INTERNATIONAL COLLEGE
KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG
2013

Thesis Academic Year 2013

B.Eng. in Software Engineering

International College

King Mongkut's Institute of Technology Ladkrabang

Title : A Server-Side GIS Web Service.

Authors :

1. Mr. Tanunpuchr Chansuwan Student ID : 53090011
2. Ms. Warinporn Boonpratham Student ID : 53090023

Approved for submission

.....

(Asst.Prof.Dr. Visit Hirankitti)

INTERNATIONAL COLLEGE

Advisor

Date August 28, 2014

A Server-Side GIS Web Service

Mr. Tanunpuchr Chansuwan Student ID: 53090011
Miss Warinporn Boonpratham Student ID: 53090023
Advisor: Asst.Prof.Dr.Visit Hirankitti
Academic Year 2013

Abstract

Location-based services are essential for emergency response as a person requesting help needs to inform his/her location so that a rescue response can be delivered successfully. The aim of this thesis is to develop a server-side GIS (Geographic Information System) web application which provides location services for emergency rescue.

The scope of this work is to develop map and location services to support a client-side web application. These services were deployed on NGINX, a HTTP server, which runs on Ubuntu. The location services are: location search, submission of a user's request, staff's response to the request, and rescue delivery. The services were developed using Django with extensions of GDAL and PROJ4 in order to support working with geospatial data. For the Web Map Service (WMS), map image tiles are sent to a web client using HTTP protocol. GeoServer, a map server, was used to render map images, which can be in different formats, from spatial data in the databases. The server-side application can communicate with a client application via the HTTP using JSON.

In conclusion, GIS is a very useful technology to use for providing location and mapping services. This thesis makes use of basic concepts of GIS which can be used to develop a server-side application for emergency rescue.

Keywords: Geographic Information System

Acknowledgement

Firstly, we would like to express our special thanks of gratitude to our advisor Asst.Prof.Dr.Visit Hirankitti who gave us the golden opportunity to do this wonderful project on the topic "Server-Side GIS Web Service", which also helped us in doing a lot of research and we came to know so many new things. We are very thankful to him. Secondly, we would like to thank our friends who helped us a lot in finishing this project within the limited time.

We are making this project not only for marks but also to increase our knowledge. Thanks again to all those who helped us.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

An accident or a disaster always happens anywhere and anytime, therefore, emergency rescue is also always needed in any location. When a disaster happens, a person suffering that may make a call to a responsible organization to request for help. An inaccurate or incomplete information about the location of disaster can prevent the help to be delivered successfully. To cope with this problem, we can use a location-based service technology.

Geographic Information System (GIS) is a technology which can be used anywhere and anytime on a smartphone. GIS is used in various ways such as searching a location, finding a route and navigation to a location, etc. We found that these functions are useful for alleviating disaster caused by either human or nature. We can use location services to specify exactly where the help is needed.

Therefore, we would like to apply the GIS technology with an emergency rescue to make the rescue more effective, and to make the services available anywhere, we are going to create services on a web site so that anyone can access the services from anywhere in any computer device.

1.2 Objectives

1.2.1. To configure GeoServer in order to deliver map services

1.2.2. To develop a web service for helping rescue disaster victims

1.2.3. To provide a better way of requesting help directly to the Thai Red Cross through location services on a web browser

1.2.4. To provide a manageable request tracking facility where the sender can track requests status and Red Cross can update the requests status

1.2.5. To help make collaboration among Thai Red Cross Society's staffs more effectively

1.2.6. To provide Web Map Service (WMS), Web Feature Service (WFS), searching service, warning service, user service and request service for a web client application to use

1.2.7. To improve efficiency and performance of map and location services

1.3 Contribution

The thesis aims to demonstrate how GIS technology can be used for disaster response. Though, GIS technology is not new, many people still do not know much about it. They only know Google Maps but not the technology behind that. Also, open source software is not just a basic tool, but some are very powerful and many developers are contributing to open source software development. We also, want to help the open source community to expand further by using its software and show how powerful it is.

However, the most important thing is we want to help our society by developing a system which can help other people suffering a disaster. This software can improve a quality of people life.

1.4 Scope of work

In order to achieve the thesis purposes, we need to do:

For server:

- Setting up and installing an Ubuntu server to provide web services
- Setting up and installing NGINX HTTP server and Tomcat7 server
- Setting up and installing a GeoServer to provide map services
- Setting up and installing Django to use with the server

For databases:

- Setting up and installing PostgreSQL with a PostGIS extension for a database to support geospatial data
- Importing geospatial data in shape file format to the database

For map services:

- Importing geospatial data to a database and display it using the GeoServer
- Sharing and managing geospatial data from the database which contains two districts of Bangkok
- Configuring each map layer for easy recognition

- Editing and adjusting each layer style

For location services:

- Developing a searching service by attribute using name of location or house number, and searching service by location, using a geometry location with a specific distance on a layer
- Developing a location search in a selected area
- Developing a user management, registration and login
- Developing a user request service that a user can make a request to the server
- Developing a response to a request service for a staff
- Developing a delivery service for a response to a request

For communication with a web client:

- Design a communication between the server and the client in JSON format

1.5 Structure of the thesis

- Chapter 1 of this thesis explains a motivation, objectives, contribution to readers, and scopes of work to give an overview of why the thesis was developed
- Chapter 2 gives a simple explanation on a problem of the current system that we would like to improve and a review of related work
- Chapter 3 discusses basic knowledge about a Geographic Information System technology relevant to this thesis
- Chapter 4 explains problem description in more detail, functional and non-functional requirements, overview of architecture and a use case diagram
- Chapter 5 focuses on the system design and explains a design of each component including a class diagram
- Chapter 6 explains a development process, tools and techniques which are used during development processes
- Chapter 7 illustrates key functionalities of the system, and explains experiments with results
- Chapter 8 evaluates the system effectiveness
- Chapter 9 summarizes a result of this thesis, problems and obstacles during development, lessons learned and future work

Chapter 2

Problem and Related Work

2.1 Problem

People can tell other people where a disaster is, but it is not good enough because only knowing a location name is insufficient. When they want to request a help, they need to provide a precise position of the place location. Sometimes, they can only receive helps from a certain place. Also, they cannot know the status of their requests. Current emergency rescue is not organized because there is no good communication among the rescue teams. This may cause problems such as the help was delivered to a wrong location or the help was not the kind of help they expected.

GIS can be used to solve these problems. Firstly, GIS can specify an exact location using a map in a way that the user can pin point a location of disaster on the map to warn other users. Secondly, a web service can provide a support of request tracking to provide information of what situation of the user is. Then, a staff knows the situation and takes the right action for the emergency rescue. As a result the emergency rescue will be delivered more effectively and well-organized.

2.2 Review of related works

This thesis makes use of an open-source map server and HTTP server. The most popular open-source map server projects are MapServer and GeoServer. For the HTTP server, Apache and NGINX are the most popular open source software. In this chapter, we will make a survey on those popular map server projects and HTTP servers.

2.2.1 MapServer

MapServer is an open source platform for publishing spatial data and interactive mapping applications to the web. Its main purpose is to display dynamic spatial maps over the Internet. Here's major features of the MapServer:

- It supports display and querying various number of raster, vector, and database formats

- MapServer can be deployed on many operating systems include the most popular ones Windows, Linux and Mac OS X
- It supports popular scripting languages, PHP, Python, Perl, Ruby, Java, .NET
- Map projections can be transformed on-the-fly
- Its map rendering is in a good quality
- Its map services is highly customizable

To summarize, MapServer is a CGI (Common Gateway Interface) program in a Web server. When MapServer receives a request, it uses information passed in the request URL and the Mapfile to create an image of the requested map. It may return images for legends, scale bars, reference maps, and values passed as CGI variables as well. Here is an anatomy of a MapServer Application.

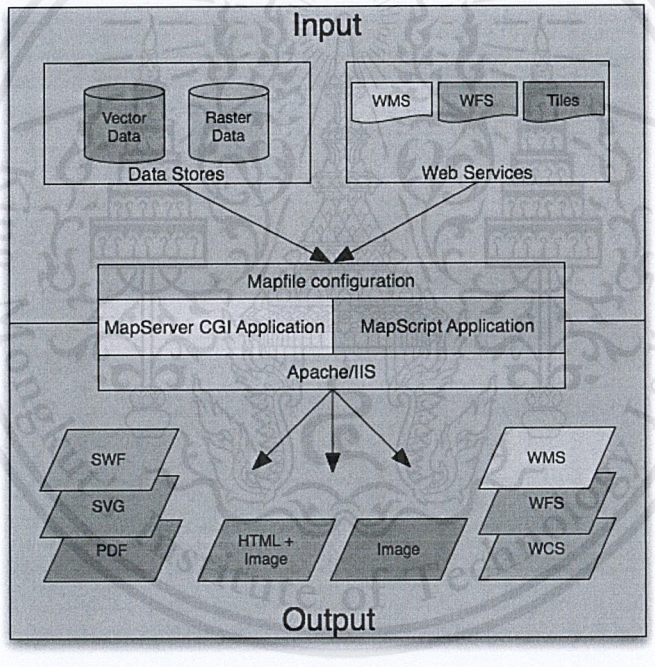

Figure 2.1 Basic architecture of MapServer applications

A simple MapServer application consists of:

- Map file – defines the area of a map, tells where the data is and where to output images
- Geographic Data – default format is the ESRI Shape format
- HTML Pages – is the interface between the user and MapServer
- MapServer CGI – is the executable file that receives and returns images

- Web/HTTP Server – provides the HTML pages in the user's browser

2.2.2 GeoServer

GeoServer is a powerful map and feature server for sharing, analyzing and editing geospatial data from spatial data sources using open standards. GeoServer supports many data formats (ArcSDE, Oracle Spatial, DB2, Microsoft SQL Server, Shapefile, GeoTIFF, and many more). It can output in multiple formats (ESRI Shapefiles, KML, GML, GeoJSON, PNG, JPEG, TIFF, SVG, PDF, and GeoRSS).GeoServer comes with a fully-featured web administration interface and REST API for easy configuration. It is a Java J2EE application which works with Jetty, Tomcat, JBoss, and others.

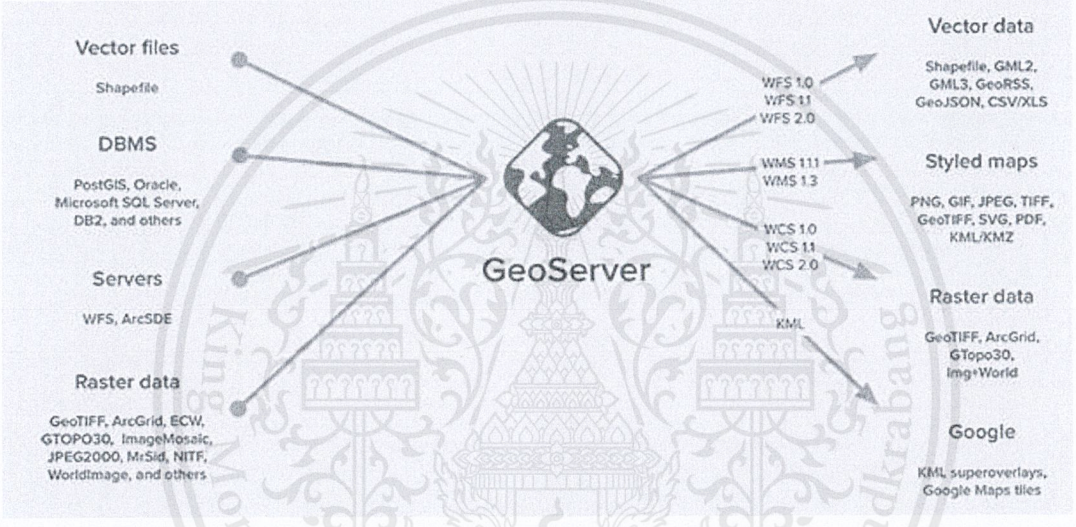

Figure 2.2 GeoServer Open Standard

2.2.3 A Comparison between MapServer and GeoServer

We chose MapServer and GeoServer for comparison because these two map servers are well-known open-source software and very popular among GIS developers. Their documentations are well written and they updates them regularly.

- GeoServer has a friendlier administrator interface than MapServer which all comes to edit the Mapfile
- MapServer is written in C, C++, on the other hand GeoServer is written in Java
- MapServer and GeoServer are both platform independent
- GeoServer provides WMS out of the box, when layers are created, they are automatically set to be provided through the service, whereas MapServer needs manual configuration in the Mapfile
- Both are fast but GeoServer requires more resources because of Java

The reasons GeoServer was selected as our map server are:

- The server provides map image tiles to the client using WMS through HTTP.
- GeoServer is more user-friendly than MapServer.
- GeoServer does not need many configurations before it is ready-to-use. This helps us concentrate on other functionalities.
- GeoServer has built-in core features such as importing spatial database and geospatial data, editing layers with Styled Layer Descriptor (SLD) and GeoWebCache.

HTTP server is the most important part of the server-side application. We looked at two popular open source HTTP servers that are Apache and NGINX.

2.2.4 Apache HTTP Server

The Apache HTTP Server Project aims to develop and maintain an open-source HTTP server for modern operating systems including UNIX and Windows NT. The goal of this project is to provide a secure, efficient and extensible server. Apache http has been the most popular HTTP server on the Internet since April 1996. It is a project of '*The Apache Software Foundation*'. The characteristics of the Apache HTTP Server are:

- a powerful, flexible, HTTP/1.1 compliant web server
- latest protocols including HTTP/1.1 (RFC2616)
- highly configurable and extensible with third-party modules
- customizable modules using the Apache module API
- full source code and unrestrictive license
- runs on Windows 2000, Netware 5.x and above, OS/2, and most versions of Unix
- actively being developed

2.2.5 NGINX HTTP Server

NGINX (pronounced 'engine x') is a lightweight HTTP server originating from Russia. It is free and open source software running under various OS such as Linux-based, Mac OS and Windows OS. NGINX has focused on high performance, high concurrency and low memory usage. A high-level overview of NGINX architecture is presented in figure 2.3.

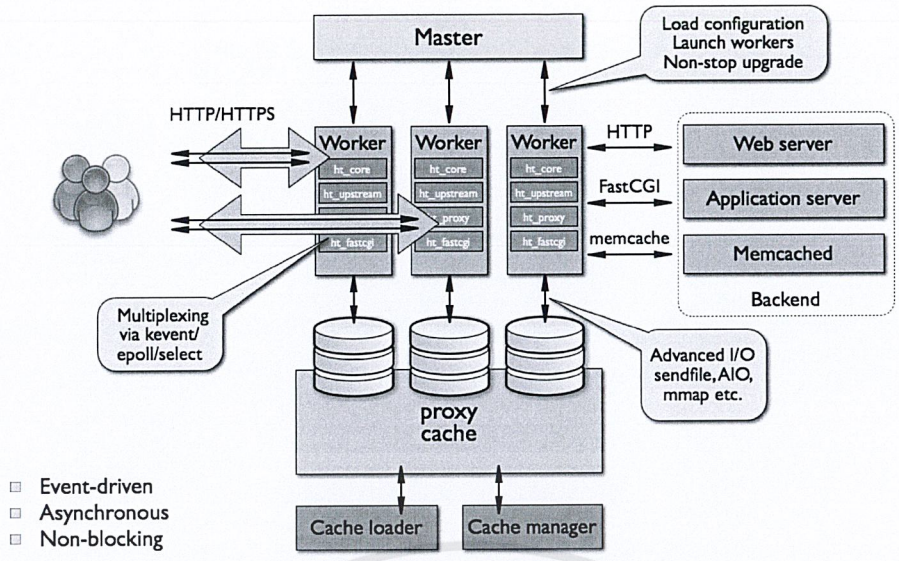

Figure 2.3 Overview of NGINX architecture

2.2.6 Comparison between Apache HTTP Server and NGINX HTTP Server

From studying Apache and NGINX in many tests and researches, we can conclude our finding results into this table.

Table 2.1 Apache vs NGINX

Category	Apache	NGINX
Design	Process-driven[1]	Event-driven[2]
Memory consumption for serving static pages	Higher memory consumption because Apache creates new process for each request	Lower memory consumption
Started development year	1995	2002
Configuration feature	Comes with a lot of configuration features which make it easier to use	Less of configuration features

[1] In process-driven architecture, a new process of thread is created for every connection which requires a lot of overhead

[2] Web Server serves all connections on a few processes to save resources

Documents	Excellent documentation	Average documentation
Components	Has tons of features and provides more functionality	Less components to add more feature
Support OS	More wider range	Less OS supports
First installation	Comes up with many features in the package	Only core features which make it more lightweight
Performance	Depends heavily on hardware resources	Does not completely depend on hardware resources

The reasons we picked NGINX over Apache are:

- NGINX consumes less resources than Apache when serving multiply requests.
- NGINX is a new HTTP server that develops with a fast progress.
- Our server is not powerful, so that NGINX was the first choice for us to use.

Chapter 3

Background Knowledge

3.1 Geographic Information System concepts

Geographic information system (GIS) is an information system which makes use of spatial data and information. To develop a GIS, one needs to have background knowledge of a basic of geography.

3.1.1 Coordinate systems

A coordinate system is a way to represent points in space. Almost everyone is accustomed to two-dimensional (2D) space and three-dimensional (3D) space. Generally, a position in 2D space is assigned by two numbers and a position in 3D space is assigned by three numbers.

a) Cartesian coordinate systems

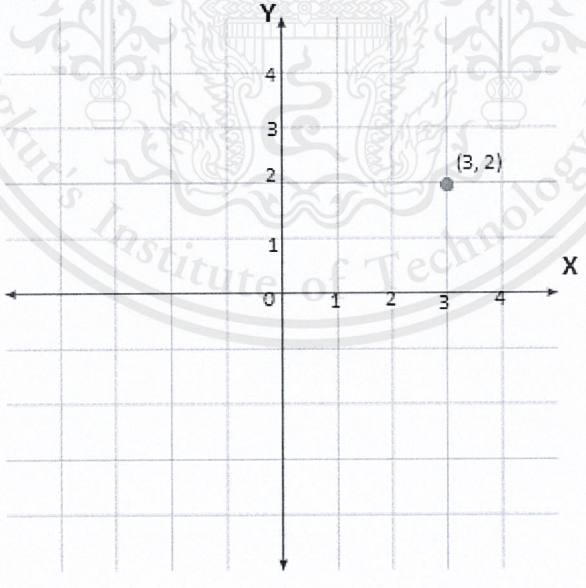

Figure 3.1 2-D Cartesian coordinate system

In 2D Cartesian coordinate system, two axes (lines) cross at right angles. The point at which they cross is called the 'origin'. Horizontal line is called 'x-axis' and the other line is called 'y-axis' as showed in figure 3.1.

Another one is 3D Cartesian coordinate system. By adding a line point toward the plane from 2D Cartesian coordinate system and bend it a little upward. The line is called 'z-axis'. Now, point can be assigned on a 3D plane.

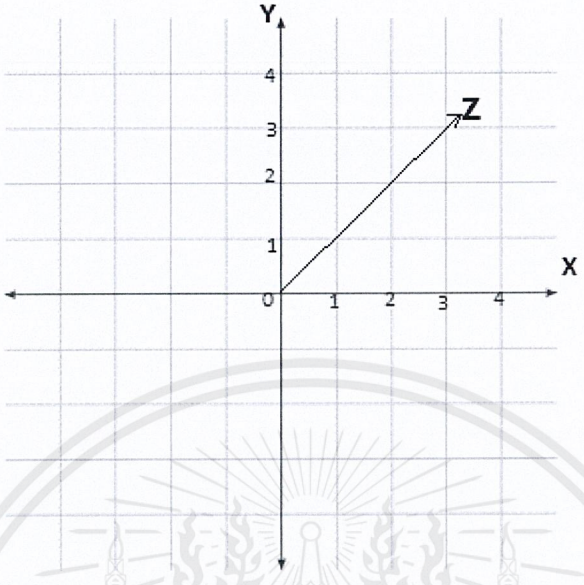

Figure 3.2 3-D Cartesian coordinate system

b) Spherical coordinate systems

A spherical coordinate system is an alternative way to represent point in 3D space. Like a 3D Cartesian coordinate system, it also required three numbers; two of them are angles, and the left is a distance. This system looks like a ray emitting from the origin toward surfaces of the sphere. The latitude-longitude graticule, a gridded reference network of lines encompassing the globe, is based on spherical coordinate system.

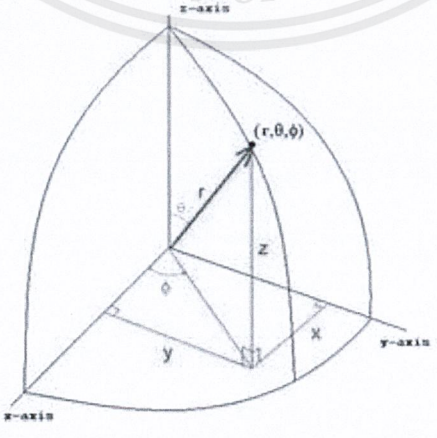

Figure 3.3 Spherical coordinate system

Consider the earth as a hollow sphere consists of three axis passing through the middle as shown in figure 3.4.

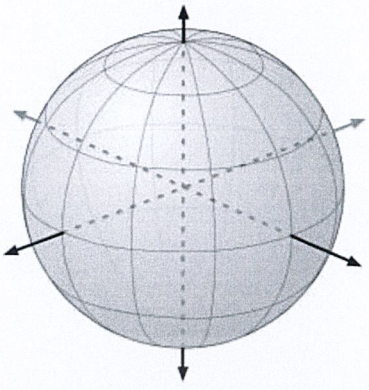

Figure 3.4 The earth with three axis

A line can be drawn from the center of the earth to connect to any point on the earth's surface as shown below in figure 3.5.

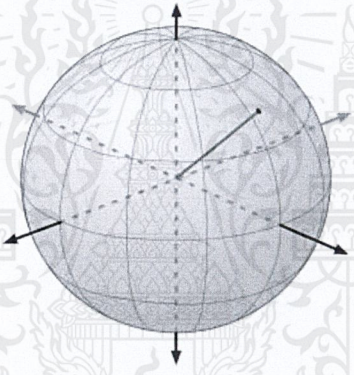

Figure 3.5 A line from the center to the surface

Latitude of the point is the angle that the line makes in the north-south direction. Same as longitude of the point which is the angle that the line makes in the east-west direction. Below is a figure 3-6 to show latitude and longitude of the earth.

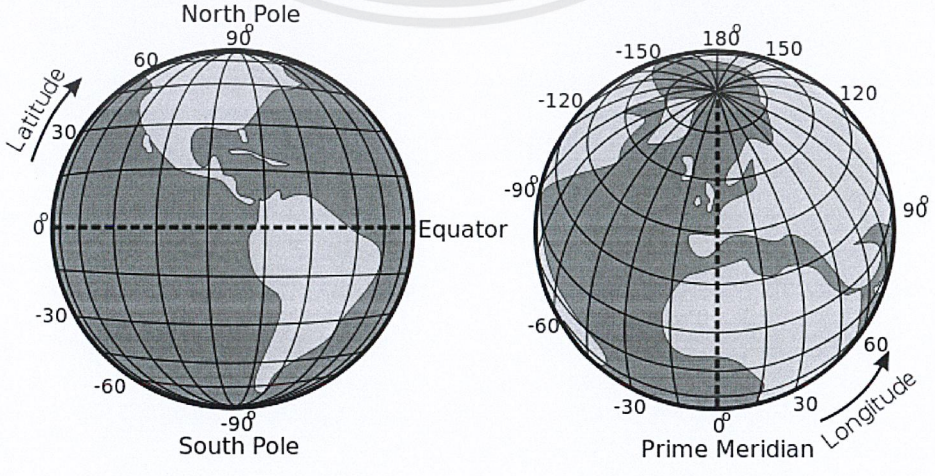

Figure 3.6 Latitude and Longitude

A latitude and longitude value represents a geodetic location. A geodetic location specifies a point on the earth's surface.

3.1.2 Projections

A projection is a mathematical transformation process that turns the three-dimensional shape of the earth into a two-dimensional map. There are three main groups of projections: cylindrical, conical, and azimuthal.

a) Cylindrical projection

Consider a source of light in the center of the earth, it projects the light to the earth's surface to create a shadow on a cylinder paper which is wrapping the earth. As shown in figure 3.7.

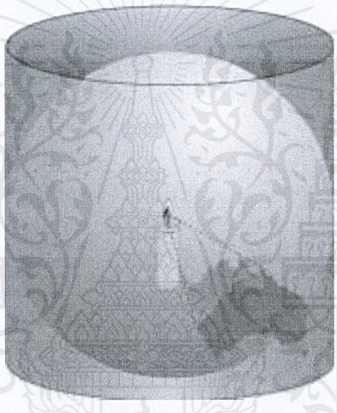

Figure 3.7 The light create the shadow on the surface

Then, unwrapping the paper to get a two-dimensional image of the earth. This is a simple explain. In reality, map projections use sophisticated mathematical transformations to get the same result.

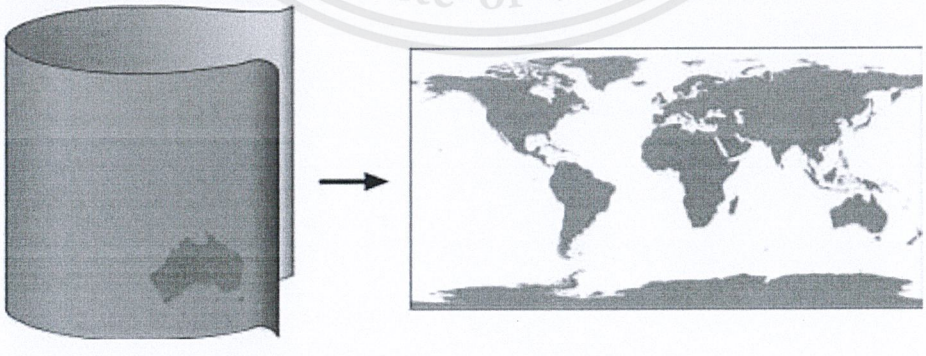

Figure 3.8 Unwrap the paper

Some examples of cylindrical projections are the '*Mercator Projection*', the *'Equal-Area Cylindrical Projection*', and the *'Universal Transverse Mercator Projection'*. Figure 3.9 is an example of a Mercator projection.

Figure 3.9 Mercator projection

b) Conical projection

A conic projection is obtained by projecting the earth's surface onto a cone. By unwrapping the cone, a map is produced. Some examples are the *Albers Equal-Area Projection*, the *Lambert Conformal Conic Projection*, and the *Equidistant Projection*.

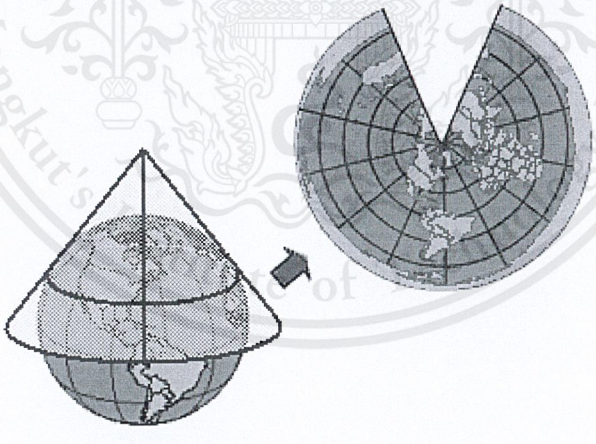

Figure 3.10 Conic projection

c) Azimuthal projection

An azimuthal projection is projecting the earth's surface directly onto a flat surface. Generally, azimuthal projections do not show the entire earth's surface. However, they do show the spherical nature of the earth. Some examples are the *Gnomonic Projection*, the *Lambert Equal-Area Azimuthal Projection*, and the *Orthographic Projection*.

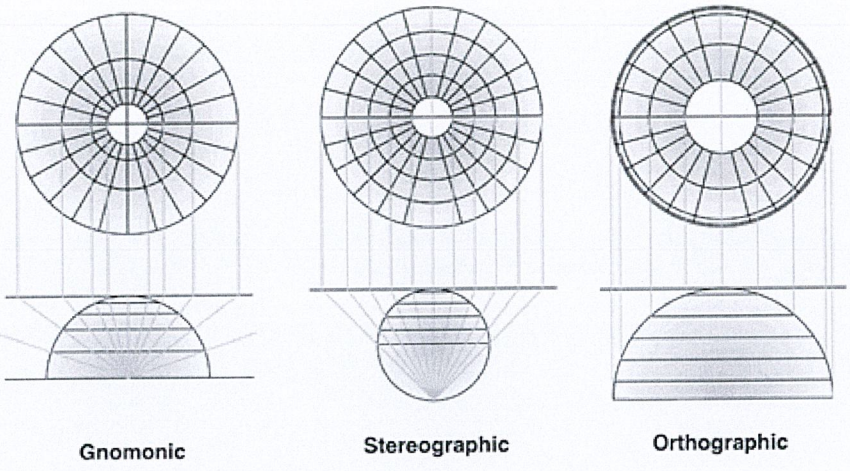

Figure 3.11 Azimuthal projection

3.1.3 Projected Coordinates

Projected coordinates refer to a point on a two-dimensional map that represents the surface of the earth. A projected coordinate system makes use of a map projection. Firstly, converting the earth into a two-dimensional Cartesian coordinate system. Then, placing points onto the plane.

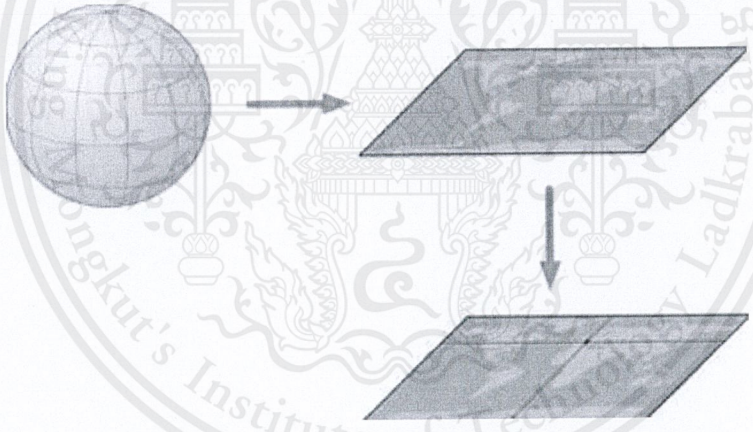

Figure 3.12 Projected coordinate system

As mentioned earlier in coordinate systems. Here is a comparison table between spherical coordinate system and projected coordinate system.

Table 3.1 Coordinate system comparison

	Advantages	**Disadvantages**
Spherical coordinate system	- Points on the earth's surface are accurate - The system does not introduce errors	- Complex calculation is needed to determine the distance or the area - Latitude-longitude plotting is considered distorted

Projected coordinate system	- Distance and area calculations are easy - Graphic representations are realistic	- Almost every point is in the wrong place - All projections introduce errors

Some example of projected coordinate systems: UTM and State Plane.

a) Universal Transverse Mercator (UTM)

UTM was developed based on a series of 60 projections into semi-cylinders that contact the earth along meridians. However, UTM projections are further subdivided into zones covering 6° of longitude and 8° of latitude for most zones.

b) State plane

State plane coordinate systems are designed to have a scale error maximum of about 1 unit in 10,000. The possible error with the UTM coordinate system may be larger: 1 in 2,500.

3.1.4 Datums

A datum is a mathematical model of the earth used to describe locations on the earth's surface. A datum consists of a set of reference points combined with a model of the shape of the earth. The reference points are used to describe location and the model of the earth's shape is used to project the earth onto a two-dimensional plane. There are three main reference datums:

- NAD 27 – is the North American Datum of 1927. NAD 27 is a local datum covering North America.
- NAD 83 – is the North American Datum of 1983. It is considered a local datum covering the United States, Canada, Mexico and Central America.
- WGS 84 – is the World Geodetic System of 1984. This is a global datum covering the entire earth. It is a very popular datum. WGS 84 also has the distinction of being used by Global Positioning System satellites.

3.2 Data types

Geospatial data represents in many forms such as points, line, polygon and raster.

3.2.1 Points

A point is simply a coordinate on a plane. It can be described by two or more numbers within a projected or spherical coordinate system.

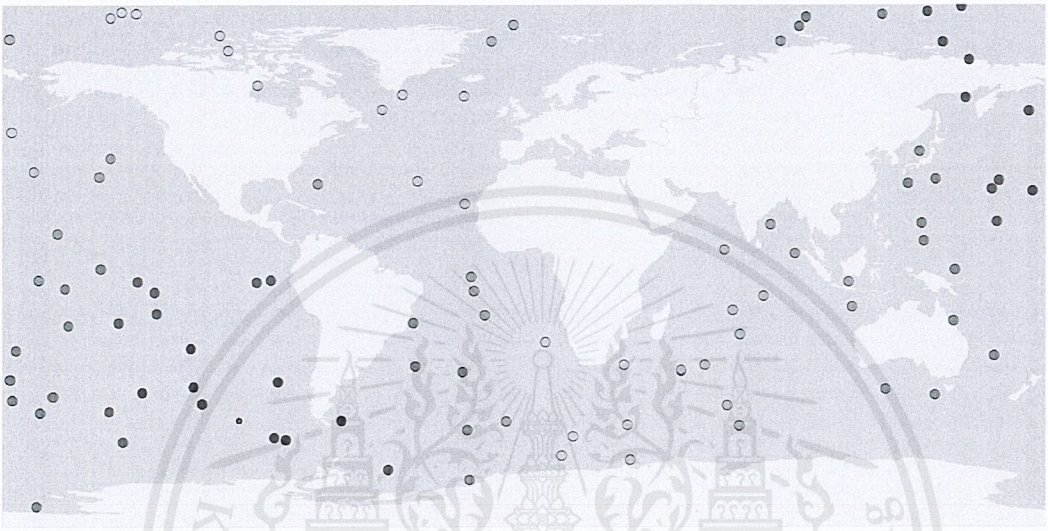

Figure 3.13 Point

3.2.2 Linestrings

A linestring represents a path defied by at least two distinct points. Linestrings are often used in geospatial data to represent roads, rivers, contour lines and so on.

Figure 3.14 Linestrings

3.2.3 Polygons

Polygons are commonly used to represent area. A polygon has an exterior ring which is a linestring to represent its outline. Also, the polygon can have an interior ring to represent a hole in it.

Figure 3.15 Polygons

3.2.4 Raster

Raster organizes information using pixels, sometimes called cells. Each pixel is a space holder for data. Pixels are organized in rows and columns to form tiles. GIS makes frequent use of georeferenced rasters. Pixels in a georeferenced raster corresponds to actual geographical locations and the physical size of the pixels takes on a real unit of measure.

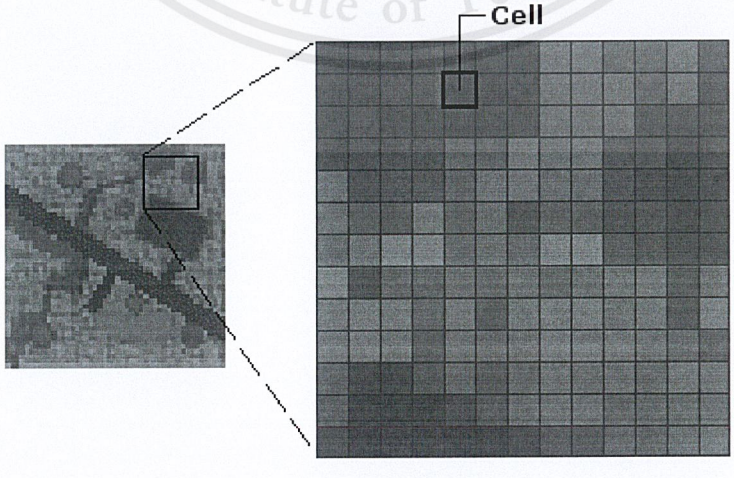

Figure 3.16 Raster

Chapter 4

Requirements and Analysis

4.1 Problem description

Geographic Information System (GIS) is an essential and powerful information system which provides a support for users to access complexity of location data. GIS helps make a decision and solve problems of the location. By applying GIS, we can enhance an information system with location services, in this case, we apply GIS to develop an information system to provide an emergency rescue service.

The problem of current emergency rescue service is that the information of a location given by a person who requests help is not accurate. He/she can only tell rough details of the location, therefore, the help may not be delivered at the correct location. GIS can solve this problem by pin-pointing the exact location by people requesting help. Another problem is the staff who offers a rescue does not know where the help will be delivered, while the person who requested it does not know when the help will arrive. We can also use of GIS to solve this problem.

Our GIS provides map service, location service and application service. For the map service, we import geographic data to our map server, namely 'GeoServer', and the GeoServer will provide a map server to a client GIS web application running on a web browser. Another service, our GIS provides location services such as to search a location by name or by an area of interest, and to provide request and response in order to deliver emergency rescue. The last service of our GIS is to manage user accounts. For example, it provides a registration service for a user to subscribe to the system.

4.2 System Architecture

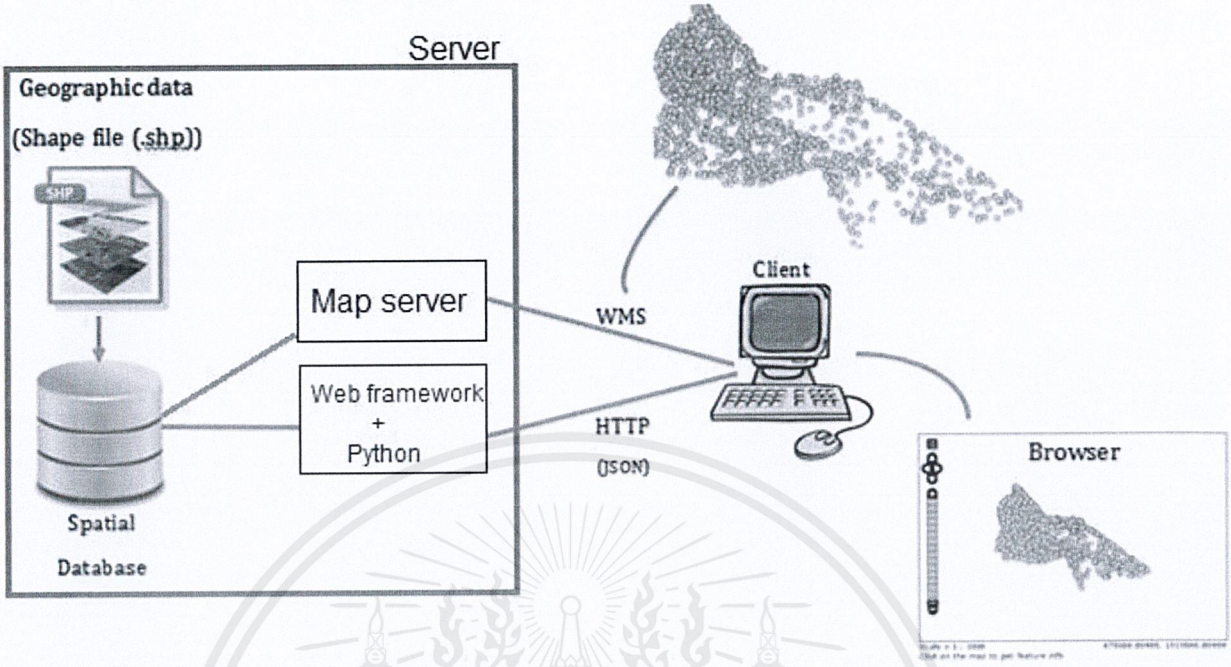

Figure 4.1 System architecture overview

The architecture of our system can be illustrated in figure 4.1 which consists of a database, a web framework, a map server and a client. A geospatial data is retrieved from other resource. To use the geospatial data, the system needs a spatial database for storing. A server provides a Web Map Service (WMS) using a GeoServer, a Java-based map server. GeoServer can provide map images through HTTP for client to display on a browser. Additional services are provided using Django, a python-based web framework, in a JSON format.

4.3 Requirements

4.3.1 Functional requirements

- Map services
  - An admin can import data sources into GeoServer.
  - An admin can select which layer is available for sharing to a web client application.
  - An admin can add/edit layer in a spatial database
  - An admin can add/edit data in database to share using GeoServer

- A web client can request Web Map Server, Web Feature Service and Web Coverage Service though HTTP request form GeoServer.
- An admin can edit layer's style to be displayed in a web client application.
- An admin can group layers into one single layer.
- A user can search locations by name in a specific layer.
- A user can search locations by name in a specific area.
- A user can search locations by name and distance from another location within a specific distance.

- User services
  - A user can register to the system through a client browser
  - A user can login/logout to the system
  - A user can check his/her notification
- Request services
  - A user can make a request to a danger on a specific location with a various kinds of request type
  - A user can track his/her request that has been sent to the system
  - A staff can view incoming requests in their responsibility area
  - A staff can response to a request with a resource and a delivery
  - A user can track the response and the delivery of his/her request
  - A staff can update current position of a delivery
- Services for a supervisor
  - A supervisor can view requests, responses and deliveries
  - A supervisor can generate a report of the helping service
  - A supervisor can evaluate the result of the service

4.3.2 Non-functional requirements

- The system provides and stores a session for the user.
- The data sources are in form of vector data, Shapefile, raster data, MrSID, and spatial databases, PostGIS.
- The encoding of characters stored in the databases, is TIS-620.
- The projection of the geospatial data is EPSG:32647.
- The system must be deployed on Ubuntu 12.04 LTS 64 bit
- The server must have at least this specification: 8 cores CPU with 2.2GHz clock speed, 8GB of memory, 500GB of hard disk
- The system must be available 24 hours and 7 days
- The system must have cache server for map services
- The system must be deployed on NGINX HTTP server
- The system must use GeoServer to serve WMS, WFS and WCS
- The system must use Django 1.5+ and PostGIS 2.0+

4.4 Use case diagram

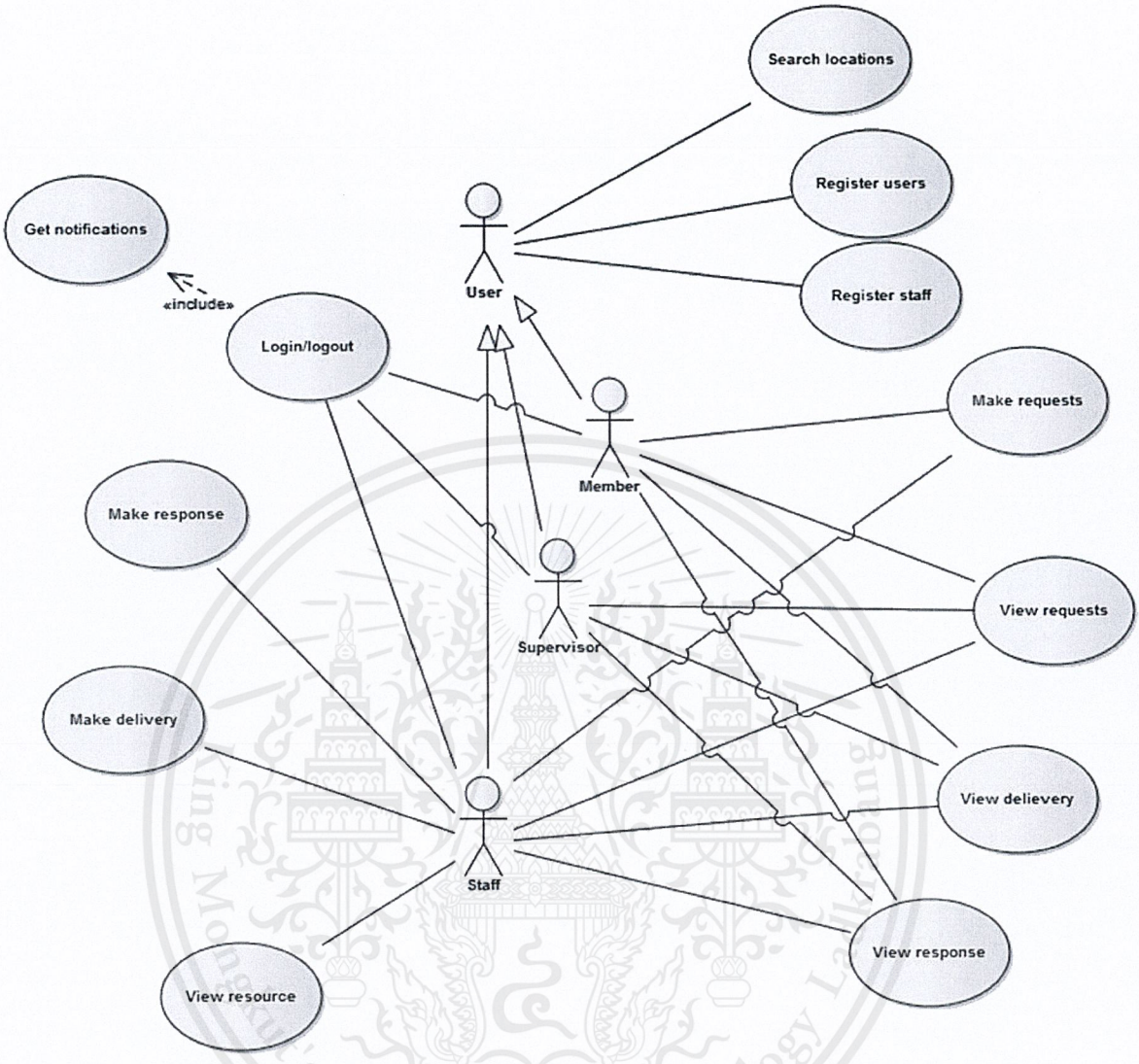

Figure 4.2 Use case diagram

The roles of using the system are divided into three roles: member, staff and supervisor. Every user can search locations. In search locations, a user can specify name and area of searching to find locations in a layer. Staff and member registrations are in a different page. Member and staff can make a request. Member can only view his/her requests. Staff can view all requests in a responsibility area or his/her requests. Supervisor can view any request. Staff can create a response to a request along with a delivery.

4.5 Use case Overview

Table 4.1 Use case overview

Use case	Actors	Overview
Search locations	User	A user can search buildings, roads, or landmark points in a map by entering a name of locations. The user needs to select which layer he/she want to search in. A result of the search is in GeoJSON format.
Register user	User	A user can register as a member of the system. The user needs to input some of his/her information to register such as a username, a password, his/her first name and last name, and citizen id. If the username is not duplicated, the result of registering will be successful.
Register staff	User	A user can register as a staff of the system. Registering as a staff is not different from registering as a member. In addition, the staff needs more information in responsibility area.
Make requests	Member Staff	A member or a staff can make a new request to the system. The member or the staff needs to input information about the request before sending. If the inputs are corrected, the request will be successfully sent.
View requests	Member Staff Supervisor	A member, a staff or a supervisor can view his/her sent requests. He/she can track the request status. The staff can also view requests in his/her responsibility area. The supervisor can view any request on the map.

Login/Logout	Member Staff Supervisor	A registered user can login or logout to his/her registered account. He/she needs to enter a username and password of his/her account to login. He/she can logout by clicking a logout button.
Get notification	Member Staff Supervisor	A registered can receive a notification of his/her request after logon to the system.
Make response	Staff	A staff can response to a request from a user. The staff selects which request is needed to be response. Then, the staff enters information in the response field.
Make delivery	Staff	In addition of the response, a staff can enters information about delivery. The delivery field indicates expected time, method of the delivery, delivery man and current position.
View response	Staff Super Visor	A staff and a super visor can view a created response by a staff. The staff can only view his/her responses. On the other hand, the super visor can view any responses in the map.
View delivery	Staff Super Visor	A staff and a super visor can view a delivery according to the response. The staff can only view the delivery he/she created but the super visor can view any created delivery.
View resource	Staff	When making a response, a staff needs to select which resource is needed for the response.

Chapter 5

Software Design

5.1 System Architecture

This is the system architecture of our server-side application to work with the client-side web application in details. It contains two core components: a database component and a server component. Location service, map server and protocol are in the server component.

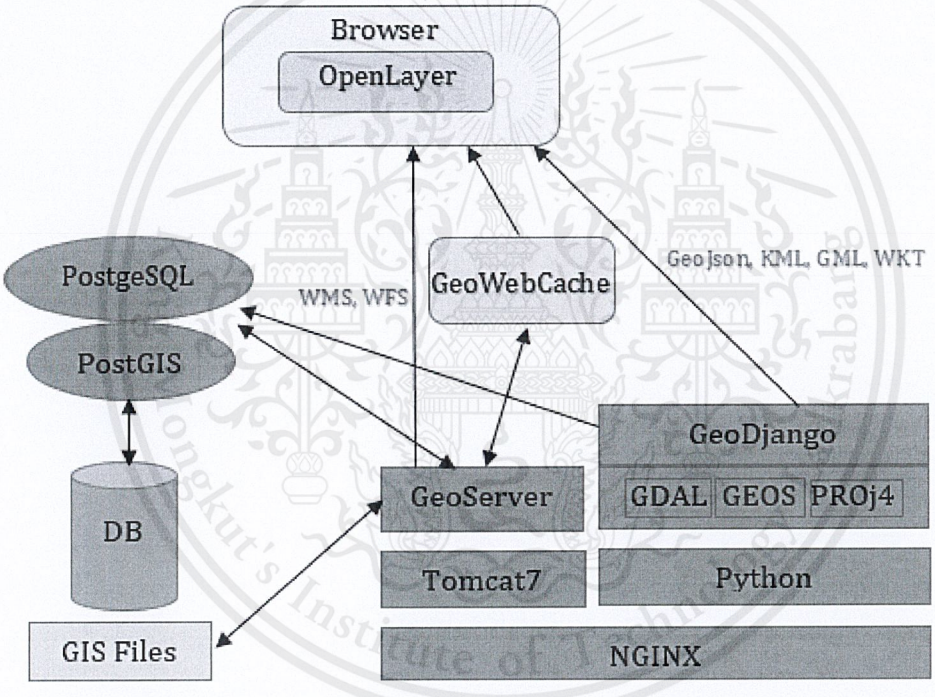

Figure 5.1 Server-Side System Architecture

5.1.1 Server

The entire system resides on Ubuntu Server 12.04 LTS x64. NGINX is a HTTP server for providing services through HTTP to client's browser. NGINX was chosen because of its lightweight and performance. GeoServer, a Java based application, is used to provide Web Map Service, but NGINX cannot communicate to GeoServer directly, so Tomcat7, a Java servlet, is installed to enable the connection. Another service is location service which is deployed using Django, a python-based web framework.

5.1.2 Database

The server is ready to provide services but a database is needed. Since data is geospatial, a spatial database is needed. PostgreSQL with an extension, PostGIS, can store and manage Geospatial data. GeoServer and Django can access the database to provide services.

5.1.3 Map services

GeoServer is responsible for providing Web Map Service to a client. It also has GeoWebCache to support caching of map images. GeoWebCache reduces resource consumption of processing same images.

5.1.4 Location services

Django can provide additional services such as search, request and response. GDAL, GEOS and PROJ4 are extensions for Django to make it supports dealing with geospatial data and spatial query.

5.1.5 Protocol

A browser can send a WMS request for map images of GeoServer through HTTP. Other services are provided through HTTP in JSON format.

5.2 Django

Django is an open source web framework developed in Python language. The main objective of Django is to help developers to create website quickly. An important advantage of Django is developers can manage database easily. Django version 1.6 were used during development.

Django uses the MTV development pattern to encourage loose coupling and strict separation between pieces of an application. By following this pholisophy, you can make changes in the particular piece of the application easily without affecting the other pieces. The three pieces of the philosophy are data access logic, business logic, and presentation logic. Sometimes, it is called Model-View-Controller (MVC) pattern of software architecture. In MVC, "Model" refers to the data access layer, "View" refers to the part of the system that selects what to display and how to display it, and

"Controller" refers to the part of the system that decides which view to use, depending on user input, accessing the model as needed.

Django follows this MVC pattern closely enough that it can be called an MVC framework. Here is roughly details of how the M, V and C break down in Django: M, the data-access portion is handled by Django's database layer. V, the portion that selects which data to display and how to display it, is handled by views and templates. C, the portion that delegates to a view on user input, is handled by the framework itself. Because the "C" is handled by the framework itself and most of the processes in Django happens in models, templates and views. Django then, has been referred to as an MTV framework. In the MTV development pattern:

- M stands for "Model", the data access layer. This layer contains everything about the data.
- T stands for "Template", the presentation layer. This layer contains presentation-related decisions.
- V stands for "View", the business logic layer. This layer contains the logic that accesses the model and defers to the appropriate template.

Figure 5.2 shows the architecture of Django. Here is a description of each part:

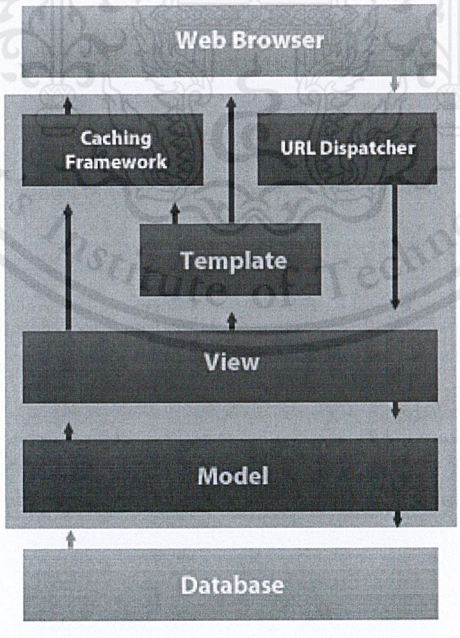

Figure 5.2 Django Architecture

- URL Dispatcher – the URL dispatcher (urls.py) maps requested URL to a view function and calls it. If caching is enabled, the view function can check to see if a cached version of the page exists and bypass all other further steps.
- View – the view function (views.py) performs the requested action.
- Model - the model (models.py) defines the data in Python and interacts with it.
- Caching Framework – caching can be enabled in Django to any performed tasks to be served later.
- Template – templates typically return HTML pages.

Django structure is very well design. Hence, we use Django as our web service framework. In our server-side web application we don't have Template because it is full responsibility for the client-side web application. We only did the back-end of the application. The data of services and geographic data are stored in 'models.py' file which will be described in a class diagram and Django data model later in this chapter. URL Dispatcher, in an 'urls.py' file, receives any incoming request from the client-side application and maps to a specific view function. Here is a sample of 'urls.py' file.

```
urls.py
urlpatterns = patterns('',
    url(r'^search_gis/', search_gis),
    url(r'^admin/', include(admin.site.urls)),
    #user backend
    url(r'^member_register/', member_register),
    url(r'^staff_register/', staff_register),
    url(r'^check_session/', check_session),
    url(r'^update_notification/', update_notification),
    url(r'^login/', user_login),
    url(r'^logout/', user_logout),
)
```

Each URL can pass a request to a specific function. For example, 'url(r'^search_gis/', search_gis),' will send an incoming request from URL that ends with 'search_gis/', to 'search_gis' function. The view funtions are in a 'views.py' file. The functions contain multiple services for our server-side application. These services are grouped into three groups: a GIS service, a User service and a Web service. Each service has various functions to serve the request.

Here is a sample of using functions in View.

```
GeoApp/views.py
def search_gis(request):
    if request.method != 'POST':
        return Http404
    body = request.body
    loaded_json = simplejson.loads(body)
    print loaded_json
    features = loaded_json['features']
    for ff in features:
        geometry = ff['geometry']
        properties = ff['properties']
        if geometry == '':
            return search_name(properties)
        else:
            return search_area(geometry, properties)
```

This function related to the previous code in 'urls.py'. The request will be sent to this 'search_gis' function to perform further actions.

5.3 Class diagram

A class diagram of the server application services, is shown in figure 5.3.

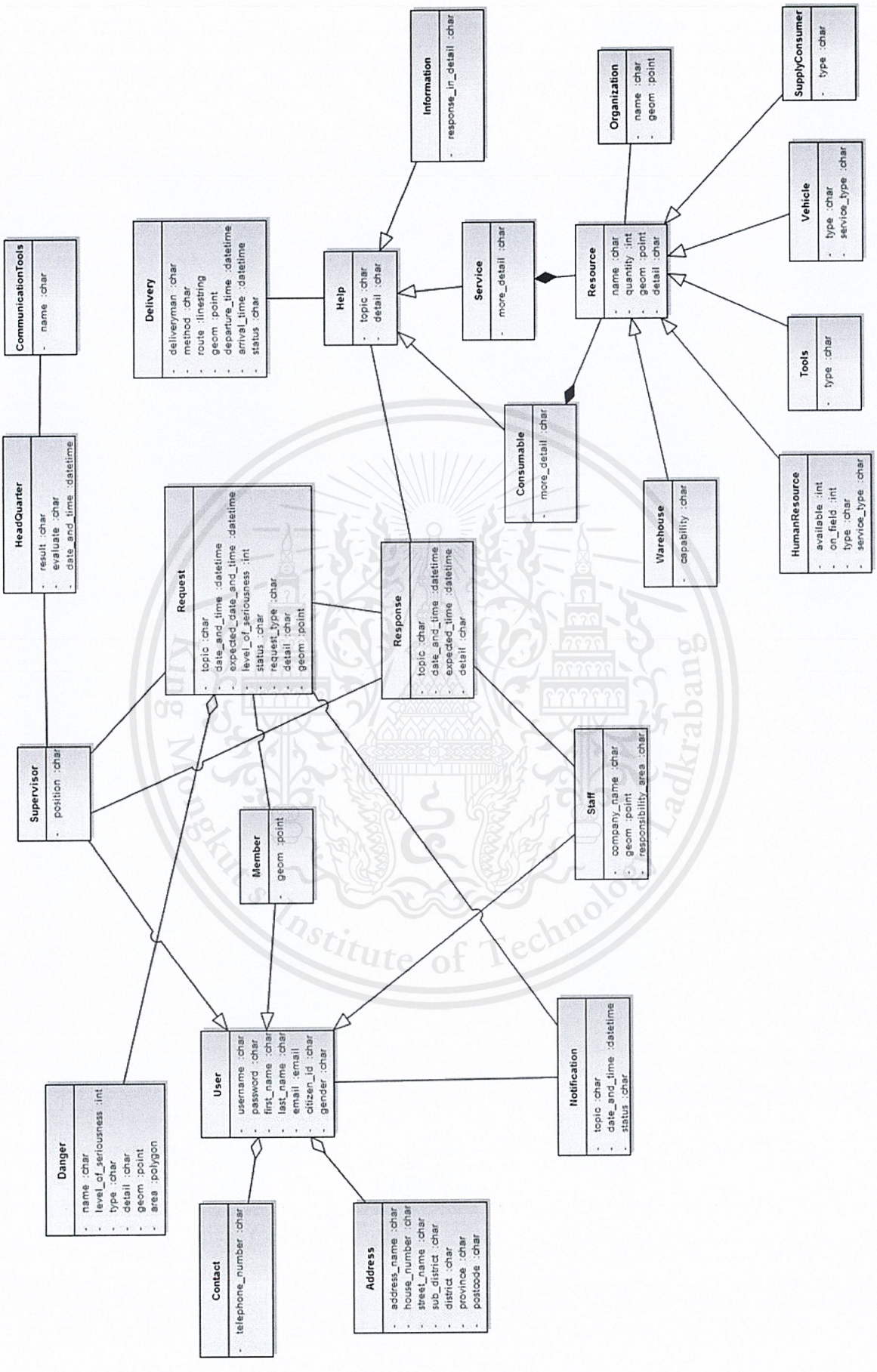

Figure 5.3 Class diagram

5.4 Class description

Table 5.1 Class description overview

Class	Description
User	User class contains information of every users on the system. Username of each user is unique.
Member	Member class inherits the User class and add a variable called 'geom' which is a point of a location of the user.
Staff	Staff class inherits the User class and has some information about responsibility of the staff.
Supervisor	Supervisor class is a user which has an ability to overview the request-response of the system.
Contact	Contact class is a component of the User class. The contact class has an information about a telephone number of the user.
Address	Address class contains information about address of a user. Each user can have more than one address.
Request	A Request class is a class for creating a request object to store in a database. It contains needed information of the request including a location of the request.
Danger	A Danger class provides a danger list to the request. The list is stored in a database.
Notification	A Notification class is used to track and store notifications of a user.
Response	Response class is associated directly to the request class. This class is used to response to a request from a user.
Help	Help class is for selecting a delivery and a resource for providing help according to the response.
Delivery	Delivery class is a class related to the delivery of the resources. It has information of delivery man, packages, status, current position and route of delivery.

Information	Information class keeps only data about information helps
Consumable	Any consumable objects go through this class to the Help class for a response.
Service	Any services from human is passed through this class to keep track of what is going on for a service response.
Resource	A class to check any resources available for a request.
Organization	An organization who is associated to the resources. An owner or donor of the resources.
Warehouse	A warehouse resource can provide a storage.
HumanResource	HumanResource class can provide services such as cooking and medical treatment.
Tools	Tools class can provide a service or a consumable according to which tool is selected.
Vehicle	Vehicle class is a service for transportation.
SupplyConsumer	SupplyConsumer class involves directly to a food, medicine and water for distributing.
HeadQuater	HeadQuater class can generate a report with result of the response.
CommunicationTools	A class which contains information of a collaboration of the organization.

5.5 Django data model

A data model is the single, definite source of information about the data. It contains fields and behavior of the data. Each data model refers to a single database table. To put it simply, each data model is a Python class that subclasses *'django.db.models.Model'*, and each attribute of the data model represents a database field. Figure 5.4 shows a data table of the building data stored in the database.

Field name	Type	Array?	Allow nulls?
gid	int4	No	No
building_n	varchar(200)	No	Yes
bul_tag	varchar(15)	No	Yes
house_id	varchar(11)	No	Yes
house_no	varchar(20)	No	Yes
lbstatus	varchar(1)	No	Yes
tel	varchar(10)	No	Yes
shape_area	numeric	No	Yes
shape_len	numeric	No	Yes
geom	geometry(8357652)	No	Yes

Figure 5.4 Sample data in a table

This is a Building class to store geospatial data of a building. A 'gid' is an id of the building.

```
class Building(models.Model):
    gid = models.IntegerField(primary_key=True)
    building_n = models.CharField(max_length=200, blank=True)
    bul_tag = models.CharField(max_length=15, blank=True)
    house_id = models.CharField(max_length=11, blank=True)
    house_no = models.CharField(max_length=20, blank=True)
    lbstatus = models.CharField(max_length=1, blank=True)
    tel = models.CharField(max_length=10, blank=True)
    shape_area = models.DecimalField(max_digits=65535, decimal_places=65535,
blank=True, null=True)
    shape_len = models.DecimalField(max_digits=65535, decimal_places=65535,
blank=True, null=True)
    geom = models.MultiPolygonField(srid=32647, blank=True, null=True)
    objects = models.GeoManager()
```

A 'building_n' is a name of the building. A 'house_id' and a 'house_no' are different. The 'house_id' is kept for PEA to use, but the 'house_no' is used for general usage. A 'bul_tag' and an 'lbstatus' variables are about electricity uses. A 'tel' variable is a telephone number of the building. A 'shape_area' is an area of the building. A 'shape_len' is a length of a border line of the building. A 'geom' variable, which, in this case, is 'MultiPolygonField', contains each corner position of the building polygon. A 'srid' indicates that the positions are stored using a SRID 32647 projection. An 'objects' variable is for managing this object later. Other data models are not much different. The main difference is which type of the geospatial data it is. So that, the 'geom' could be in 'PointField' and 'LineField'.

5.6 Protocol design for client-server communication

In the old days, web services used XML as their primary data format for transmitting data back and forth. But, an appearance of JSON changed it all. JSON has been a preffered format because it is more lightweight and easy to understand.

JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. It is also a subset of JavaScript's Object Notation, the way objects are built in JavaScript. JSON is built on two structures. It is a collection of name/values pairs and an ordered list of values. Here is an example of JSON data:

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

JSON data format is all well and good but it is not enough to answer our needs. GeoJSON is more suited for our application. GeoJSON is a format for encoding a variety of geographic data structures. A GeoJSON object may represent a geometry, a feature, or a collection of feature. GeoJSON supports these geometry types: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon, and GeometryCollection. Features in GeoJSON contain a geometry object and additional properties, and a feature collection represents a list of features. Here is a sample of our design of a JSON format to communicate with a client-side web application.

```
{
        'status':'view_ok', #or 'view_fail'
        'type':'FeatureCollection',
        'features':[{
                        'geometry':{
                                'type':'Point',
                                'coordinates':[lat,long],
                        },
                        'properties':{
                                'id':1,
                                'name':'',
                                'quantity':'',
                }},{
                        'geometry':{
                                'type':'Point',
                                'coordinates':[lat,long],
                        },
                        'properties':{
                                'id':2,
                                'name':'',
                                'quantity':'',
}}]}
```

This is a sample design of a 'view_resource' service. This is an explanation for each name of the GeoJSON data. In the 'view_resource' service, the client application side does not need to send any information to the server application side so, the client side can send a GET request to the server in a specific URL such as http://example.com/server/view_resource/. The server then send a request to the view functions to perform actions. An example output of GeoJSON objects in as shown above in Figure 5.2. A 'status' name refers to the status of the client request. The query result is a set of objects so the type of the object is a 'FeatureCollection'. A 'features' contains a list of features. In the list, we have many geometry objects. Each object is called feature as already mentioned. Each feature has two main attributes: geometry and properties. Geometry contains a type of the geometry and coordinates of it. Properties contain additional details of the feature such as id, and name.

Chapter 6

Development

The GIS data we obtained from PEA are in two types: raster data type and vector data type. We use these formats to be shared and displayed with GeoServer. The GIS data is only two districts in Bangkok which are Klong Tei and Wang Thonglang.

6.1 GIS Data Formats

There are two main types of GIS data: raster data type and vector data type.

6.1.1 Raster format data

Raster formats are generally used to store bitmapped images such as scanned paper maps or aerial photographs. There are various of raster formats for GIS such as Digital raster graphic (DRG), digital scan of a paper USGS topographic map, GeoTIFF, a TIFF variant enriched with GIS relevant metadata, JPEG2000, an open-source raster format, and MrSID, Multi-Resolution Seamless Image Database. Raster files that we obtained from PEA are in the MrSID format.

The MrSID file format is a wavelet-based image compression technology which can utilize both lossy and lossless encoding. The orginal form of this technology is from Los Alamos National Laboratories (LANL). Now it is developed and distributed by the LizardTech, Inc. GDAL driver supports reading of MrSID image files using LizardTech's decoding software development kit (DSDK). Files in MrSID format contain a set of standard metadata tags such as: IMAGE__WIDTH (width of the image), IMAGE__HEIGHT (height of the image), IMAGE__XY_ORIGIN (the x and y coordinates of the origin), IMAGE__INPUT_NAME (the name or names of the files used to create the MrSID image) etc. MrSID images may contain georeference and coordinate system information in form of GeoTIFF GeoKeys.

Figure 6.1 Sample of raster data image

6.1.2 Vector format data

Geographical features are often expressed as vectors. Vector formats represent spatial data using points, lines, and polygons. Vector data can be in various formats such as Geography Markup Language (GML), an XML based open standard for GIS data exchange, GeoJSON, a lightweight format based on JSON, Keyhole Markup Language (KML), an XML based open standard for GIS data exchange, and Shapefile, a popular vector data GIS format developed by ESRI. The vector data we obtained from PEA is in ESRI Shapefile format. Shapefile is an open specification, developed by ESRI, for storing and exchanging GIS data. Shapefile is an open specification, developed by ESRI, for storing and exchanging GIS data.

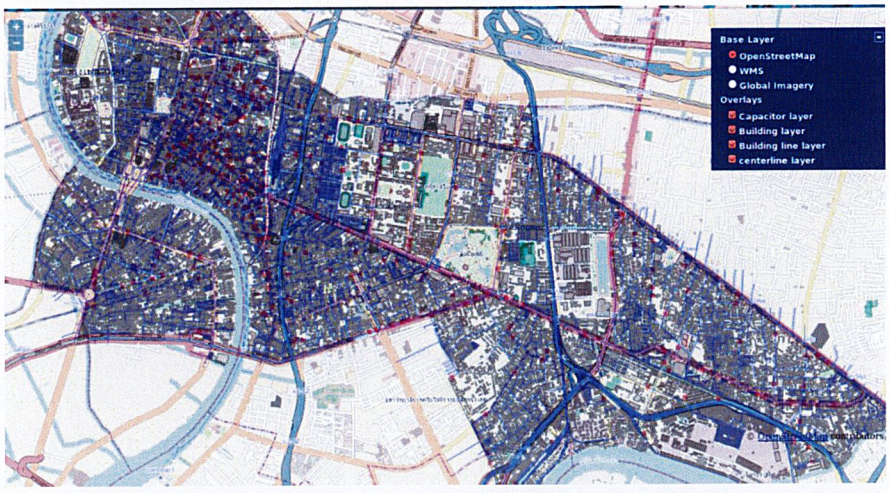

Figure 6.2 Sample of vector data image

Each shapefile holds subfiles with these extensions:

- .shp: It holds the vector data for the geometries.
- .shx: It is a spatial index file for geometries stored in the .shp.
- .dbf: It is a database file for holding non-geometric attribute data such as integer and character fields. Sample data in '.dbf' file.

	A	B	C	D	E
1	LBSTATUS,C,1	LOCATION,C,60	ROAD_NAME,C,60	SUBGROUP,N,4,0	SUBTYPECOD,N,10,0
2	P	วัดคลองเตยนอก	ทางรถไฟ	3	1122
3	P	ที่ทำการ/ด่านเก็บเงินผ่านทาง	ถนนพระราม 3	0	1125
4	P	โรงเรียนนนทรีวิทยา	ถนนพระราม 3	1	1101
5	P	ศาลเจ้าพ่อปากคลอง	ซอยโรงสี	6	1122
6	P	โรงภาพยนตร์/ วอชิงตัน	ซอยสุขุมวิท 22 (สายน้ำทิพย์)	3	1111
7	P	ศาลพระภูมิ	ซอยสุขุมวิท 20 (สายน้ำผึ้ง)	6	1122
8	P	ศาลพระภูมิ	ซอยสุขุมวิท 20 (สายน้ำผึ้ง)	6	1122
9	P	ธนาคารกรุงเทพ	ถนนสุขุมวิท	1	1104

Figure 6.3 Inside of .dbf

- .prj: It contains the spatial reference information for the geographic data stored in the shapefile. Inside '.prj' file, it contains:

```
PROJCS["WGS_1984_UTM_Zone_47N",GEOGCS["GCS_WGS_1984",DATUM["D_WGS_1984",SPHEROID["
WGS_1984",6378137.0,298.257223563]],PRIMEM["Greenwich",0.0],UNIT["Degree",0.017453
2925199433]],PROJECTION["Transverse_Mercator"],PARAMETER["False_Easting",500000.0]
,PARAMETER["False_Northing",0.0],PARAMETER["Central_Meridian",99.0],PARAMETER["Sca
le_Factor",0.9996],PARAMETER["Latitude_Of_Origin",0.0],UNIT["Meter",1.0]]
```

6.2 Database

A database is an organized collection of data used for purpose of modeling some type of organization or organizational process.

6.2.1 Spatial database

A spatial database is a database that defines column data types specifically designed to store objects in space that can be added to database tables. The information stored is usually geographic in nature such as a point location or boundary of a lake. It provides special functions and indexes for querying and manipulating that data callable from a query language such as Structured Query Language (SQL).

A spatial database is often used as just a storage container for spatial data, but it can do much more than that. Although a spatial database need not be relational in nature, most are. PostGIS is used in our project as a spatial database. PostGIS is a spatial database extender for PostgreSQL object-relational database. It adds support for geographic objects allowing location queries to be run in SQL. Here's an example to show a location query of locating name of any landmarkpoint where it is in 'Bang Kapi' district using SQL query.

```
SELECT landmarkpoint.name
FROM district, landmarkpoint
WHERE ST_Contains(district.geom, landmarkpoint.geom)
AND distict.name = 'Bang Kapi';
```

Spatial database can be created the same way as creating a relational database but you need to enable spatial functions to the database. This is an example of creating a PostGIS database. Firstly, we create a database in PostgreSQL using this query.

```
CREATE DATABASE gisdb;
```

You can add any other options as you wish such as giving privileges to someone else. Next, we need to enable spatial features to the database we created with this query. It is necessary to access to the database before enable features.

```
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
```

The first extension is to enable spatial managements. The second extension is used to manage topological objects such as faces, edges and nodes. Topology involved routing nodes to nodes. Now, we have a database that supports spatial queries.

6.3 Importing geospatial to a spatial database

6.3.1 Vector data (Shapefile)

To use geospatial data, shape files, with the system, a conversion from data formats to an SQL format is needed. Geospatial vector data format (.shp) could be obtained from various places. After deciding which shape files to use, calls a command in a terminal:

```
shp2pgsql -W 'TIS620' -s 32647 filename.shp > filename.sql
```

The 'shp2pgsql' is used to convert ESRO Shapefiles into SQL suitable for insertion into a PostGIS/PostgreSQL database. This command has two options '-W' and '-s'. The '-W' is to specify the charater encoding of Shapefile's attributes. In this case, our files are in 'TIS-620' encoding. The other option is '-s' which is used to create and populate the geometry tables with the specified SRID. The files we used is in EPSG: 32647 projection. Lastly, we specify the shapefile and output we want. Since the data is geospatial data, we need a spatial database which was described earlier in this chapter. We import the '*filename.sql*' file to the '*dbname.db*' database using this command.

```
psql -d dbname.db -f filename.sql
```

A '-d' option is to specify database and a '-f' option is to specify the file we want to import. Repeat these steps for all of the shape files. Finally, go to the GeoServer site of the system and add database to its store. Now, the data can be published anywhere in any application using standard protocols such as WMS and WFS. The result of importing Shapfile into the database should look like this.

```
                List of relations
 Schema |       Name        | Type  |  Owner
--------+-------------------+-------+----------
 public | building          | table | adminz
 public | buildingline      | table | adminz
 public | centerline        | table | adminz
 public | districtpolygon   | table | adminz
 public | hydrology         | table | adminz
 public | landmarkline      | table | adminz
 public | landmarkpoint     | table | adminz
 public | roadedge          | table | adminz
 public | spatial_ref_sys   | table | postgres
(9 rows)
```

And inside a table will be like this.

```
gid  | lbstatus |   location   |      road_name       | subgroup | subtypecod |
geom
------+----------+--------------+----------------------+----------+------------
+--------------------------------------------------
  162 | P        | seven eleven | Road1                |        1 |       1113 |
0101000020877F00000715C556547C2441EF7197B11E273741
  164 | P        | seven eleven | Road2                |        1 |       1113 |
0101000020877F00002559D8E1CF302441CC41D850232B3741
  169 | P        | seven eleven | Road4                |        1 |       1113 |
0101000020877F000018086FC1FE6B24411473EEFCD7263741
```

Geometry or geom of the data is in string. It can be transformed in other format that human can read.

6.3.2 Raster data (MrSID)

MrSID is already mentioned previously but in this topic, we are going to explain the support of MrSID in GeoServer. MrSID Encoder is a high-quality, high-performance compression methodology for reducing the size of large raster data. GeoServer does not provide MrSID support as a default installation. We need to install an extension called GDAL to GeoServer. GDAL plugins allow GeoServer to support DTED, EHdr, ERDASImg, MrSID, JP2K and NITF data formats. After GDAL is ready in our server. GeoServer will show available raster supports like this.

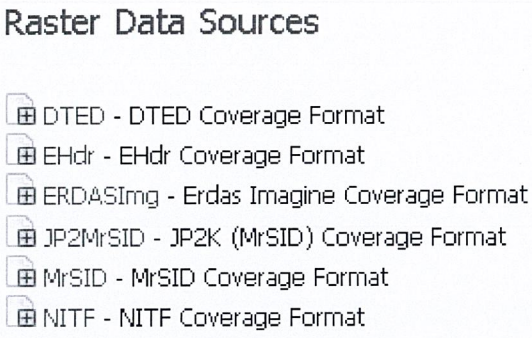

Figure 6.4 MrSID support in GeoServer using GDAL

6.4 Geospatial searching

Django does not support geospatial by default. After creating a spatial database, it is necessary to install additional libraries to enable geospatial support for Django. The libraries are GEOS, PROJ.4 and GDAL. Geometry Engine – Open Source (GEOS) is a C++ port of the Java Topology Suite (JTS). It includes all the OpenGIS Simple Features for SQL spatial predicate functions and spatial operators. PROJ.4 is an open-source Cartographic Projections library. It has almost all projection types and it is very useful to convert coordinates for a series of points. GDAL is a translator library for raster and vector geospatial data formats. Now, a geospatial query can be performed in Django. Since, the database already has spatial data.

```
python manage.py inspectdb --database=dbname.db
```

By running this command, Django will inspect the specific database as in an option '—database=dbname.db' and create a support models.py for the database. Now, we can manage and query the spatial data in Django. To use spatial searching, objects can be called from the Django model. These are some examples of various way to search:

```
qs = Building.objects.filter(<field>__<lookup_type>=<parameter>)
```

Below is an example of spatial query that filters all building objects that contain a point in it.

```
qs = Building.objects.filter(polygon__contains=point)
```

Examples of distance lookups:

'*D*' is a short form for a distance. This query searches for any landmarkpoint object that is greater than or equal to, which is 'point__distance_gte=' in the query, 7 kilometers

from a 'pnt'. The 'pnt' object is a query result of getting an object in a Landmarkpoint table where the name of the object is 'Siam Paragon'.

```
pnt = Landmarkpoint.objects.get(name='Siam Paragon')
qs = Landmarkpoint.objects.filter(point__distance_gte=(pnt, D(km=7))
```

This query result will have any landmarkpoint object that is within 5 kilometers of Siam Paragon.

```
pnt = Landmarkpoint.objects.get(name='Siam Paragon')
qs = Landmarkpoint.objects.filter(point__dwithin=(pnt, D(km=5))
```

The interested object is the same as previous example but the query is different. The 'point__distance_gte' is replaced with 'point__dwithin' which refers to a point that has distance within 5 kilometers, as the query specified, from the 'pnt' object.

6.5 Map services

Services provided by GeoServer through HTTP. Examples of using these services are described in Chapter 7.

6.5.1 Web map service

A Web Map Service (WMS) provides a simple HTTP interface for requesting geo-registered map images from one or more distributed geospatial databases. A WMS request defines the geographic layer and area of interest to be processed. The response of the request is one or more geo-registered map images. Here's a sample request and result of the request.

```
http://.../wms?service=WMS&version=1.1.0&request=GetMap&layers=Proj_GIS:landmar
kpoint&styles=&bbox=660639.6875,1515018.0,672572.6875,1523045.25&width=512&heig
ht=344&srs=EPSG:32647&format=image%2Fpng
```

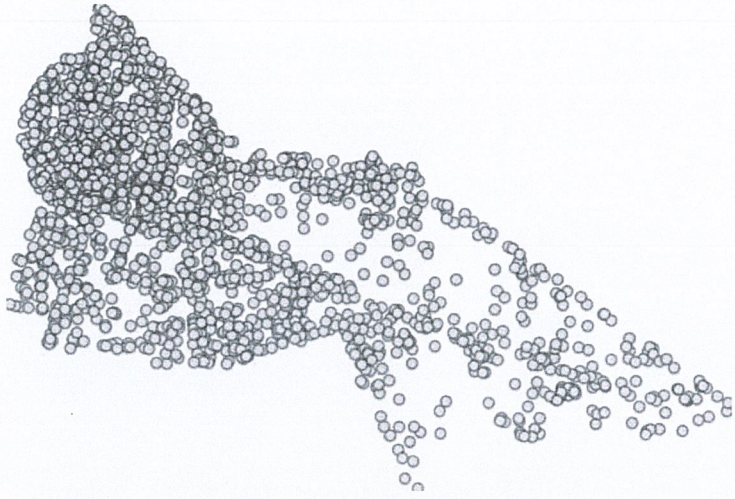

Figure 6.5 Sample image from WMS request

For the request URL, a WMS service is requested with version 1.1.0. We requested to get a map of a 'landmarkpoint' layer which is grouped in Proj_GIS workspace. That's a reason of naming it to 'Proj_GIS:landmarkpoint'. A sytle of the layer is not set which means to get a default one. A 'bbox' parameter means a bounding box of the layer we want to show on a browser. Width and height are the width and height of the shown image. A 'srs' is to select the projection of the image to be shown in EPSG:32647 projection and the format of the image is .png.

6.5.2 Web feature service

A Web Feature Service (WFS) supports requests for geographical feature data along with vector geometry and attributes. Here's a sample HTTP request to the server to request WFS.

```
http://.../ows?service=WFS&version=1.0.0&request=GetFeature&typeName=Proj_GIS:la
ndmarkpoint&maxFeatures=50&outputFormat=json
```

'http://.../' is our domain name of the server. An 'ows' parameter is a standard to request WFS. The service we requested is WFS version 1.0.0. The request wanted to get feature of objects in landmarkpoint of Proj_GIS workspace. A result of the request will only show first 50 items as request in 'maxFeatures.' An output of the request will be in JSON format.

Chapter 7

Results

7.1 Key functionalities

7.1.1 Search GIS

The server-side web application can search a location when the client-side web application sends a request to '/search_gis/' with a POST method in GeoJSON. The client-side web application has to send information that looks like the next sample. In 'search_gis' function, we can search any layer, any type of object and any location that we have in the database. Optional search such as distance and intersection can be defined in properties name/value pairs.

```
{       "type":"FeatureCollection",
        "features":[{
                "type":"Feature",
                "geometry":{
                        "type":"Point",
                        "coordinates":[102.0,0.5]},
                "properties":{
                        "prop0":"value0"}},{
                "type":"Feature",
                "geometry":{
                        "type":"LineString",
                        "coordinates":[[102.0,0.0],[103.0,1.0],[104.0,0.0],[105.0,1.0]]},
                "properties":{
                        "prop0":"value0","prop1":0.0}},{
                "type":"Feature",
                "geometry":{
                        "type":"Polygon",
                        "coordinates":[[[100.0,0.0],[101.0,0.0],[101.0,1.0],
                                [100.0,1.0],[100.0,0.0]]]},
                "properties":{
                        "prop0":"value0",
                        "prop1":{"this":"that"}
}}]}
```

This is served as our communication template for search_gis function. The next code sample is the real GeoJSON data that we receive for search_gis function.

```
{   "type": "FeatureCollection",
    "features": [
        {   "type": "Feature",
            "geometry": {
                "type": "Polygon",
                "coordinates": [
                    [ 665746, 1519165 ],
                    [ 665749, 1518780 ],
                    [ 666539, 1518766 ],
                    [ 666626, 1519319 ],
                    [ 665746, 1519165 ]
                ]},
            "properties": {
                "layer": "Landmarkpoint",
                "name": "ธนาคาร"
}}]}
```

In this sample, we received a request to search for any landmarkpoint that contains 'ธนาคาร' in its name, and the boundary of searching in within a polygon of the coordinates as in the next GeoJSON.The result will look like in this code sample. We only show one of many results here.

```
{   "type": "FeatureCollection",
    "features": [
        {  "geometry": {
                "type": "Point",
                "coordinates": [
                    665909.3243190566,
                    1518884.4186616053]},
            "properties": {
                "layer": "Landmarkpoint",
                "gid": 1137,
                "name": "ธนาคารไทยพาณิชย์"
}}]}
```

The server-side web application will return any found location in GeoJSON to the client-side. We only selected a few necessary properties to return.

7.1.2 GeoServer

GeoServer can be accessed using a browser. GeoServer supports various formats of vector and raster geospatial data and spatial database to be used with. It can display and share the data in images format through WMS or in text formats through WFS. Also, you can edit any imported map layers to be shown as you wanted.

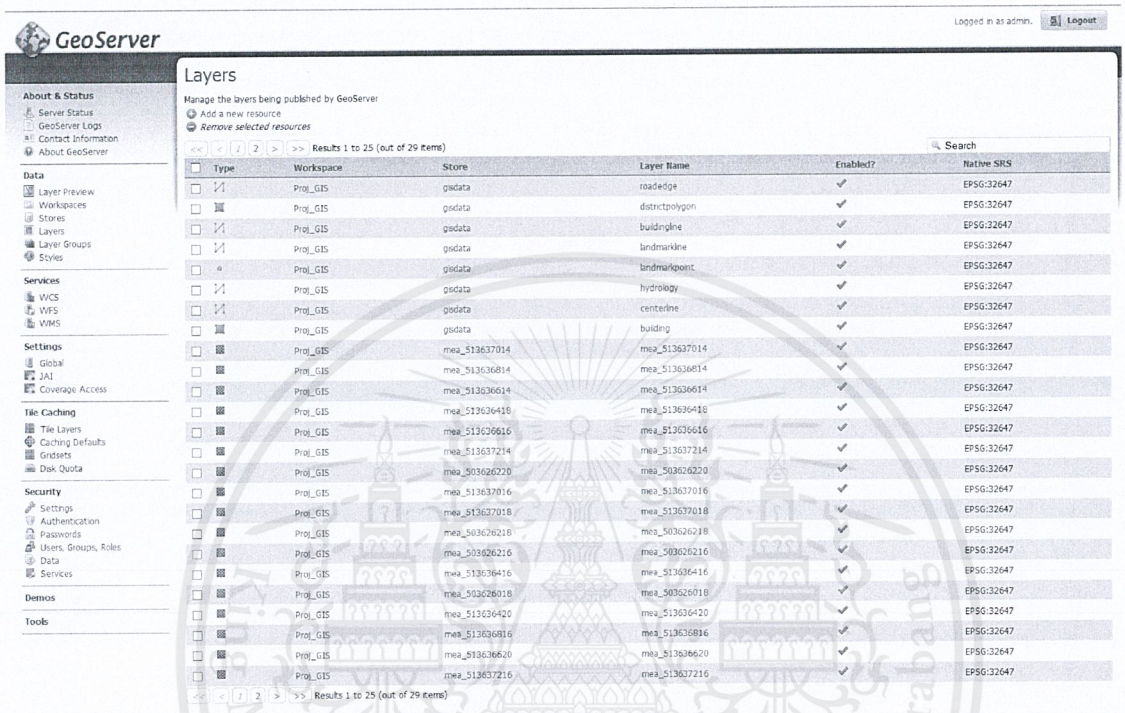

Figure 7.1 Layers page in GeoServer

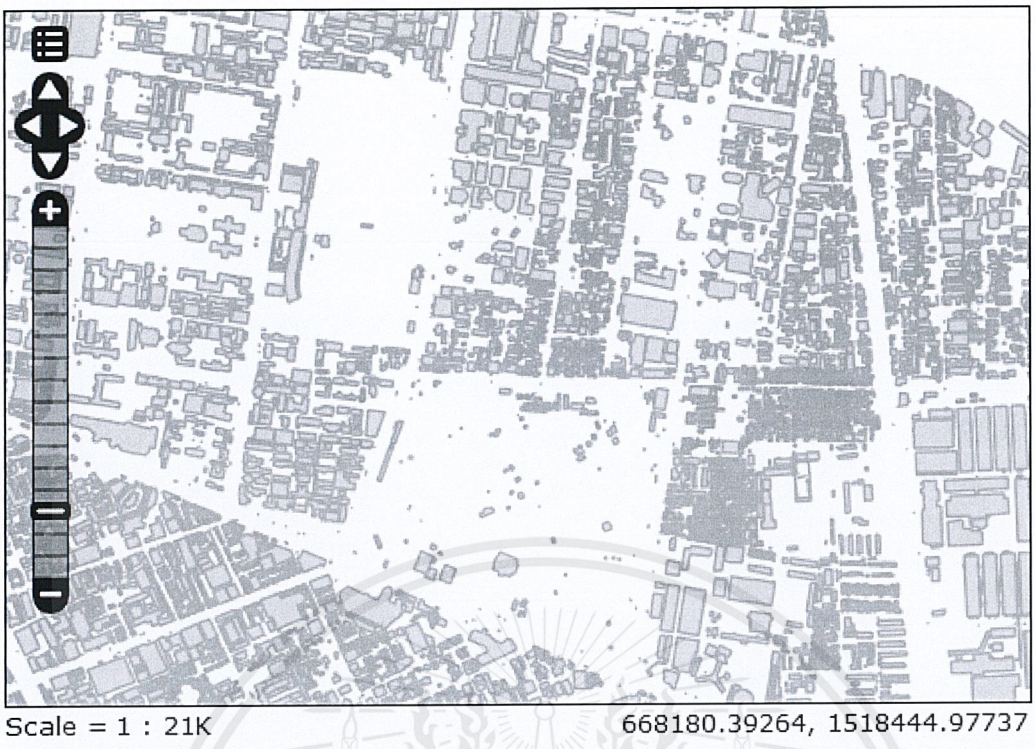

Figure 7.2 An image from WMS with OpenLayer for displaying

GeoServer can serve an image of geographic data through WMS protocol, and geographic features using WFS through JSON. Figure 7.2 is an example of request a building layer using Web Map Service to display using OpenLayer. Figure 7.3 is a sample of a request to the server using Web Feature Service to return the data in JSON format. It request the server in a landmark point layer. This result has Thai characters in it which cause some JSON viewer tools to not be able to display the characters properly. The result contains multiple objects described as a feature. The 'type' refers to which type of it. FeatureColllection means that it is a list of a feature or features. Total features is a number of features in the result. Next is a list of features. The number 0 is an index of the list. Each index has data to describe itself. A 'type' key refers to type of the object which, in this case, is a feature. An 'id' key refers to an id of the feature. A 'geometry' key is a collected of GIS data. It contains a data type and coordinates of the data. A 'properties' key tells additional details of the object. A 'crs' describes the standardize using in the JSON. This result is in EPSG: 32647 projection.

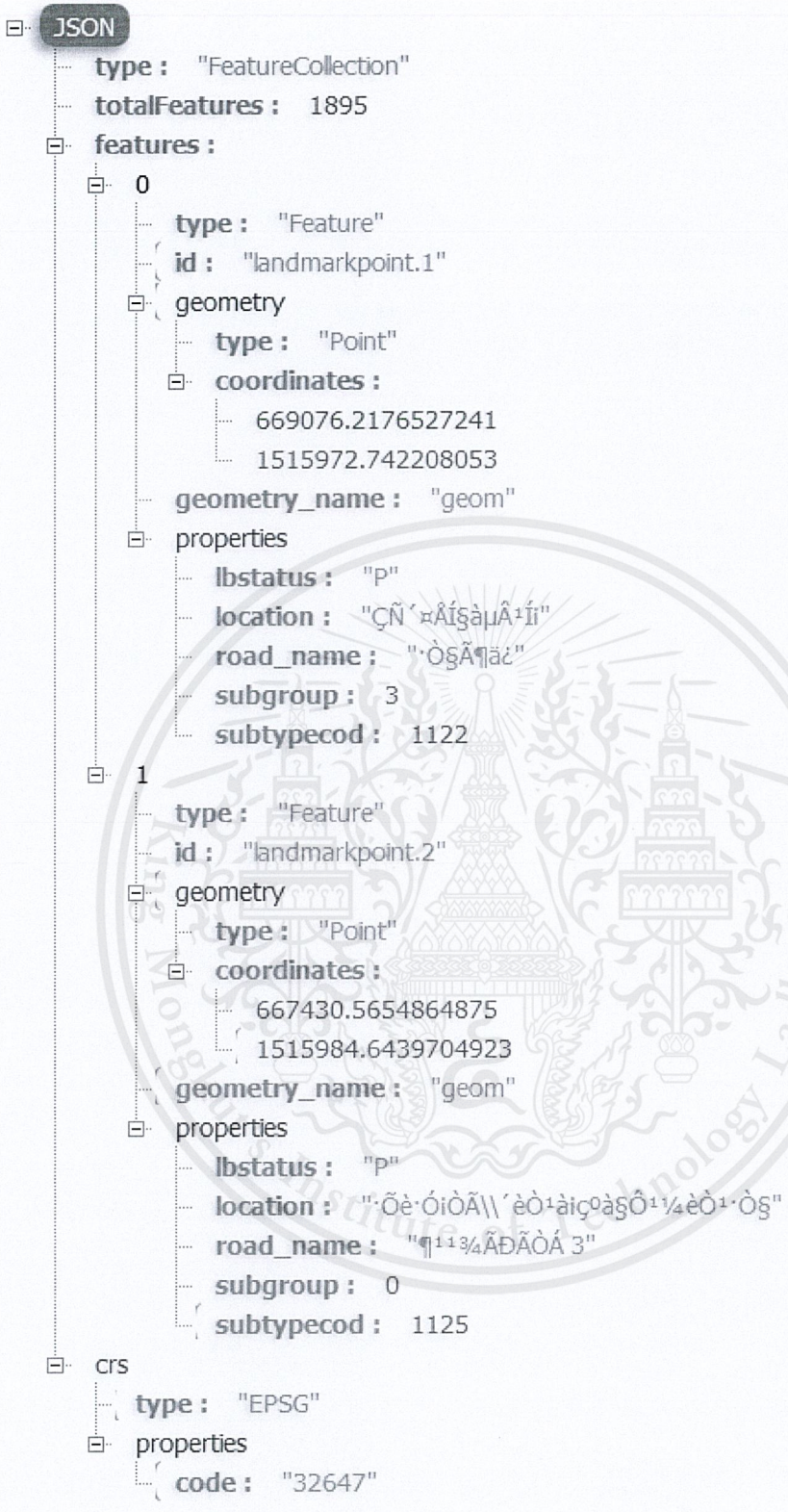

Figure 7.3 WFS sample result (GeoJSON)

7.1.3 Web application services

All of web application services connect to the client through HTTP using JSON format.

```
{
        'username':'user1',
        'password':'password',
        'email','user1@mail.com',
        'first_name':'User1',
        'last_name':'Userlast',
        'gender':'Male',
        'citizen_id':'1234567890123',
        'telephone_number':'1234',
        'address_name':'My office',
        'house_number':'1/22',
        'street_name':'Silom Road',
        'sub_district':'SubD',
        'district':'Dis',
        'province':'BKK',
        'postcode':'10000',
        'geom':[longitude,latitude],
}
```

This is an example of sending JSON to the server to register a user. The client must send username, password and email to the server. Other fields are not necessary. The geom field is to store a location of the user. It can be useful when cooperation with other people so that they can see where another person is. Next is an example of JSON return to client when a user wants to view his requests. Geometry key is a geometry of a location. Coordinate of the location must be in latitude and longitude. Output can be more than one request. Hence, every return requests are in a feature list.

```
{       'status':'search_ok', #or 'search_fail' or 'not_found'
        'type':'FeatureCollection',
        'features':[{
                        'geometry':{
                                'type':'Point',
                                'coordinates':[longitude,latitude],     },
                        'properties':{
                                'id':1,
                                'topic':'',
                                'request_type':'',
                                'level_of_seriousness':'',
                                'date_and_time':'',
                                'expected_date_and_time':'',
                                'status':'',}
                },{
                        'geometry':{
                                'type':'Point',
                                'coordinates':[longitude,latitude],},
                        'properties':{
                                'id':2,
                                'topic':'',
                                'request_type':'',
                                'level_of_seriousness':'',
                                'date_and_time':'',
                                'expected_date_and_time':'',
                                'status':'',
}},]}
```

This is a returned request of searching 'requests' that users made in the server. This is a description of each key parameter:

- 'status' – refers to a completeness of the request. There are three results here: 'search_ok' means the objects are found, 'search_fail' means there is something wrong with the request, and 'not_found' means that there is no object according to the request.
- 'type' – refers to the type of the JSON. In this case, it is a set of objects called features. Therefore, it is called a 'FeatureCollection'.
- 'features' – refers to a list of features. Each feature in the list has many properties in it.
    - 'geometry' – refers to a geometry of the feature. It contains type of the geometry and coordinates.

- 'properties' – refers to additional details of the feature such as name, id etc. It is optional but it can fullfil the object.

7.2 Experiment

7.2.1 GeoServer experiment 1

Any layer deployed in GeoServer can be configured to suit any preference. The first experiment is configuring a layer style. A Styled Layer Descriptor (SLD) is used to setting up any layer style. Here's an example of different SLD for roads. The first one is a simple blue line setting and the second one is a grey line with name property displayed along the line.

```
<NamedLayer>
  <Name>default_line</Name>
  <UserStyle>
  <!-- Styles can have names, titles and abstracts -->
    <Title>Default Line</Title>
    <Abstract>A sample style that draws a line</Abstract>
    <!-- FeatureTypeStyles describe how to render different features -->
    <!-- A FeatureTypeStyle for rendering lines -->
    <FeatureTypeStyle>
      <Rule>
        <Name>rule1</Name>
        <Title>Blue Line</Title>
        <Abstract>A solid blue line with a 1 pixel width</Abstract>
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#0000FF</CssParameter>
          </Stroke>
        </LineSymbolizer>
      </Rule>
    </FeatureTypeStyle>
  </UserStyle>
</NamedLayer>
```

Figure 7.4 SLD basic setting 1

An SLD file is a profile of the OpenGIS WMS encoding standard. Here's a description of the file:

- <NamedLayer> – means that this style has a name and can be called using its name outside.
- <Name> - is to set a name of the style.
- <!-- … --> - is a comment section in SLD.
- <Title> - is a title of the sytle.
- <Abstract> - is used to give a simple explanation of the style.
- <FeatureTypeStyle> - is used to define the symbology for rendering a single feature type.
- <Rule> - contains a styling rule to be evaluated.
- <LineSymbolizer> - specifies styling as lines.
- <Stroke> - specifies the styling for the line.

7.2.2 GeoServer result 1

The map output from the first SLD setting is as shown below. Roads are in blue lines without any text.

Figure 7.5 SLD result 1

7.2.3 GeoServer experiment 2

This experiment is also on configuration of a layer but with different setting.

```
<NamedLayer>
  <Name>_centerline</Name>
  <UserStyle>
    <Title>Center Line</Title>
    <FeatureTypeStyle>
      <Rule>
        <Name>rule1</Name>
        <Title>Grey Line</Title>
        <Abstract>A solid grey line with a 1 pixel width</Abstract>
        <LineSymbolizer>
          <Stroke>
            <CssParameter name="stroke">#B8B8B8</CssParameter>
          </Stroke>
        </LineSymbolizer>
        <TextSymbolizer>
        <Label>
          <ogc:PropertyName>name</ogc:PropertyName>
        </Label>
        <Fill>
          <CssParameter name="fill">#000000</CssParameter>
        </Fill>
        <Font>
          <CssParameter name="font-family">Garuda</CssParameter>
          <CssParameter name="font-size">14</CssParameter>
          <CssParameter name="font-style">normal</CssParameter>
          <CssParameter name="font-weight">normal</CssParameter>
        </Font>
        <VendorOption name="followLine">true</VendorOption>
        <VendorOption name="maxAngleDelta">90</VendorOption>
        <VendorOption name="maxDisplacement">400</VendorOption>
      </TextSymbolizer>
      </Rule>
    </FeatureTypeStyle>
  </UserStyle>
</NamedLayer>
```

Figure 7.6 SLD basic setting 2

This experiment is not significantly different from previous experiment. We expected the image to have text labels on it. Additional description is below:

- <TextSymbolizer> - styles features as text labels.

- <Label> - is the text content for the label.
- <ogc:PropertyName> - specifies the variable we want to display.
- <FIill> - is the fill color of the label text.
- <CssParameter> - is to set how the text should be displayed on the image.
- <Font> - is the font of the text label.
- <VendorOption> is a GeoServer-specific option to manage the look of the text.

7.2.4 GeoServer result 2

The output map from the second SLD setting is as shown in figure 7.7. Roads are grey lines with a road name along the line.

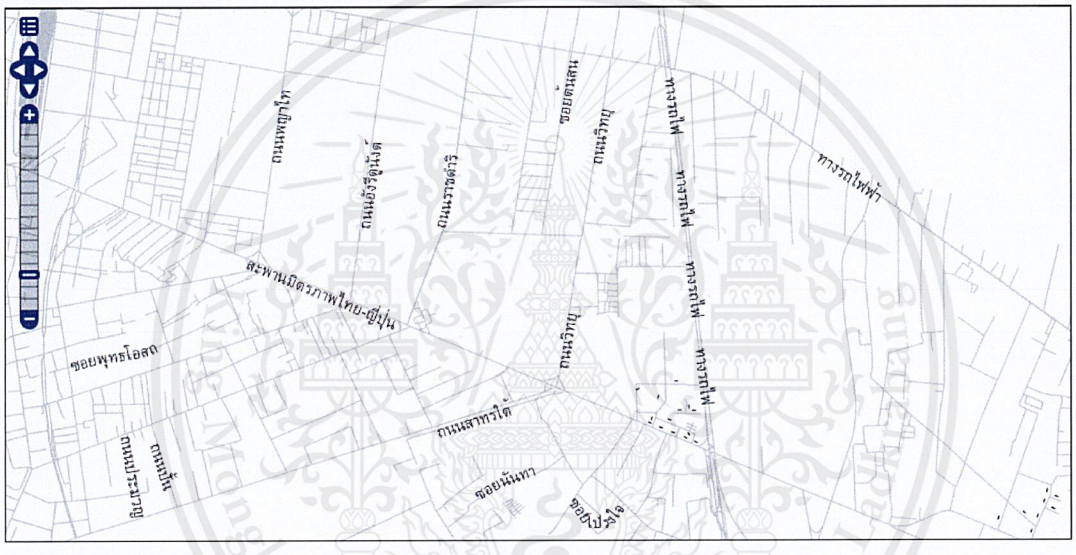

Figure 7.7 SLD result 2

7.2.5 WMS and WFS experiment 1

Web Map Service can be obtained in various format by requesting HTTP to WMS.

```
/geoserver/Proj_GIS/wms?service=WMS&version=1.1.0&request=GetMap&layers=Proj_GI
S:building&styles=&bbox=660628.5,1514829.125,672730.4375,1523045.25&width=400&h
eight=300&srs=EPSG:32647&format=image%2Fpng
```

This request is to get a map of a building layer with a bounding box consists of four numbers, min x, min y, max x and max y. Style tells which SLD is used. Width and height is the size of a result. SRS is a Spatial Reference System for map output. The last one, format, is which data format of the map output. Web Feature Service can be called the same as WMS.

```
/geoserver/Proj_GIS/ows?service=WFS&version=1.0.0&request=GetFeature&typeName=P
roj_GIS:landmarkpoint&maxFeatures=3&sortBy=location&outputFormat=json
```

This HTTP request calls WFS to get feature of 3 objects in a landmarkpoint layer. The output is sorted by location attribute and is in JSON format.

7.2.6 WMS result 1

Web Map Service result from the WMS request is shown in figure 7.8.

Figure 7.8 WMS request result

7.2.7 WFS result 1

Web Feature Service result is shown in figure 7.9 and figure 7.10. The result is sorted using location attribute but GeoServer cannot generate right characters encoding, so Thai characters are not readable.

```
{
    "type": "FeatureCollection",
    "totalFeatures": 1895,
    "features": [
        {
            "type": "Feature",
            "id": "landmarkpoint.1838",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    664412.5684816341,
                    1518740.52335433
                ]
            },
            "geometry_name": "geom",
            "properties": {
                "lbstatus": "P",
                "location": "¾Õ àÍÊ àÎéÒÊì Í¾ÒÃì·àÁ¹·ì",
                "road_name": "¶¹¹¾ÃÐÃÒÁ4",
                "subgroup": 4,
                "subtypecod": 1110
            }
        },
        {
            "type": "Feature",
            "id": "landmarkpoint.724",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    665340.6805174052,
                    1517629.0965784166
                ]
            },
```


Figure 7.9 WFS result (1/2)

```
            "geometry_name": "geom",
            "properties": {
                "lbstatus": "T",
                "location": "´Ô ÍÔ¹¿Ô¹ÔµÕé ¤Í¹â´ÁÔà¹ÕÂÁ",
                "road_name": "¶¹¹¹ÃÒ¸ÔÇÒÊÃÒª¹¤ÃÔ¹·Ãì",
                "subgroup": 3,
                "subtypecod": 1110
            }
        },
        {
            "type": "Feature",
            "id": "landmarkpoint.1873",
            "geometry": {
                "type": "Point",
                "coordinates": [
                    661234.2588447554,
                    1518058.2861132645
                ]
            },
            "geometry_name": "geom",
            "properties": {
                "lbstatus": "T",
                "location": "ÍÁÃ áÁ¹ªÑè¹",
                "road_name": "¶¹¹ÍÔ¹·Ã¾Ô·Ñ¡Éì",
                "subgroup": 4,
                "subtypecod": 1110
            }
        }
    ],
    "crs": {
        "type": "EPSG",
        "properties": {
            "code": "32647"
        }
    }
}
```


Figure 7.10 WFS result (2/2)

Chapter 8

Evaluations and Discussions

8.1 System evaluation

In this chapter, we evaluated the system in categories: HTTP server, GeoServer and Tomcat7, WMS and WFS, web application services, and communication between a server-side application and a client-side application.

8.1.1 HTTP server

Ubuntu has many features for using as a server operating system so that we can set up the server easily and effectively. NGINX HTTP server offers only necessary core features which are enough for us. It is very low memory consumption. It serves HTTP requests perfectly and it is lightweight. The problem is it needs manual configuration. It does not have many extension modules to help. It is not user-friendly. Those who are interesting in NGINX might need some time to learn how to set it properly. Though it introduces some complexity, its efficiency and performance is formidable and it suits the server perfectly.

8.1.2 GeoServer and Tomcat7

GeoServer has a very user-friendly administrator pages for managing the map server. It supports most common vector and raster geographic data, and spatial database. GeoServer provides a ready-to-use Web Map Service which can provides map images in various formats. GeoServer is a Java-based application and platform independent. GeoServer is very easy to use because its pages offer many tools in side bar. A data can be imported in just a few clicks. Any layer can be edited using SLD (Styled Layer Descriptor) which is as same as XML to set the layers style. GeoServer needs to deploy on Tomcat7 server, a Java servlet. Since Tomcat7 and GeoServer are Java-based, it introduces an overhead when using hardware resources.

8.1.3 GeoServer and GeoWebCache

GeoServer consumes some amount of hardware resources so caching is a must. GeoWebCache is an integrated caching server for providing map services. It reduces resource consumption of processing same tiles for many requests. GeoWebCache also reduces response time of loading tiles when display on a browser. GeoWebCache is very important for our system.

8.1.4 WMS and WFS

WMS and WFS is good for simple request but it is not useful for specific spatial query. Its advantage is the output can be in various formats. The performance also relies on GeoServer and Tomcat7. When they are performing normally, everything is very fast. Moreover, they really need GeoWebCache to improve serving performance.

8.1.5 Web application services

GIS technology was proved to be very useful for spatial query and storing location services. The result of searching is always corrected and the location is perfect. There is no false result. Location services perform perfectly and fast. Additional services: request, response, user management and delivery, are fast and correct. GIS technology helps to locate the location of the service to be in exact location which help the rescue team avoid complexity of directions.

8.1.6 Communication between server and client

JSON is good and easy to read data exchange format. It also has a support for geospatial data called GeoJSON. Communication through HTTP in JSON format reduce dependency between a server and a client so that development teams can develop each side independently. Despite of its benefits, using on JSON to communicate introduces some problems when dealing with objects. It has problems when dealing with users because we can exchange data only. Though, communication in JSON format is very fast.

8.2 Effectiveness

8.2.1 Server and NGINX

Ubuntu server is very good as an open-source server. It has many modules to help deploying as a server. NGINX is very effective to be used as a HTTP server because of its lightweight and low memory consumption so that the server does not depend entirely on the hardware. Still, NGINX needs some knowledge to configure it properly. Failing to do that might cause NGINX not able to run at all. Hence, the effective of the server depends heavily on configuration which we haven't study much in details that we couldn't pull the most of the server power to use.

8.2.2 GeoServer and GeoWebCache

GeoServer is very easy, ready-to-use and effective geographic data management and sharing. It helps us reduce time dealing with the data. We can manage and edit the data in GeoServer easily. But it is a Java-based application so it consumes resources that make the server slow down sometimes. GeoWebCache is a very helpful map caching server. When the data is cached, a client can retrieve the same tiles very fast. It also reduces processing time of transforming data into other formats for the browser. GeoServer with GeoWebCache is very effective as a map server.

8.2.3 Web Application Services

GeoDjango, Django with extensions of GDAL, GEOS and PROJ4, is a powerful web framework that supports geospatial data and spatial query. GeoDjango can provide many services we want in many formats. We only used it for search, request, response, user management, and delivery. We can query data from a spatial database and pass it in GeoJSON to return a request from a client's browser. Everything is fast and effective.

8.2.4 Communication

JSON data format is very easy to use. We can apply it with our services easily. Using the format helps us developing the system independently from the client side. Though, it has some problems dealing with users object and session. We had to design another way of dealing with those problems.

Chapter 9

Conclusions

9.1 Project summary

GIS is a powerful technology to be used in many kind of applications. We used it to provide map service and develop location service. To deliver these services, we employ an HTTP server that is NGINX on a server. We selected Ubuntu server and NGINX because NGINX consumes less resources and provides good performance. For geospatial data in this thesis, we used vector data in the format of Shape files and raster data in the format of MrSID. PostgreSQL with a PostGIS extension was used to manage vector data. The raster data was not stored in the database but saved as an image file to be decoded and rendered on the map later.

A map service is provided by GeoServer in a WMS or a WFS form. The data of the map is to be requested from the databases. A location service and application service were developed using Django, a python-based web framework. Location search is available in location service. Search by name on by area on a layer can be used in this service. We also provide request, response and delivery services. Any register user can send a request to the system, then a staff can response to the request in his/her responsible area. Delivery service was to track where the help is and provided an approximate time of the help. Application service is a user service. User service has registration, login, logout, notification and session. A user needs to register to the system before using other services. A notification is to tell the user that his/her request was updated to other status. A session is to check which user is currently using the system.

A HTTP protocol in JSON format was used to communicate between server-side and client-side. There are some problems using only JSON format because we can only pass keys and parameters in JSON. It was hard when dealing with objects.

To summarize, GIS technology is very useful to use with many services. We applied it with an emergency rescue to help people in troubles and to improve the current system of emergency rescue.

9.2 Lessons learned

We learned how to develop GIS application using open source tools which provide us advantages of the price of the system. Open source software is not only free but also is largely supported by people worldwide. We learned how to deploy a web service using Django and NGINX HTTP server. We learned how to manage our time efficiency. Moreover, we learned that caching is very helpful in our situations. Also, we learn to develop the server-side separately form the client-side using only JSON to communicate. Finally, we learned to create a system that helps others.

9.3 Problems and obstacles

We never work with GIS before. We could say that we did not know anything about developing an application with GIS. Also, we have not developed any web service. This project introduced us new knowledge in GIS. Though we did not have an experience with Ubuntu Server, we have been using Linux OS for some time, which helps us to manage Ubuntu and saves us precious time. Since we had to use shared server with other groups, we avoided doing many things that may cause the server malfunctions. Furthermore, connections to the client-side is hard at times because we can send only JSON format to the client. The JSON format cannot contain every detail.

9.4 Future work

- Automatic response could be developed to reduce time to response. When there is an incoming request, the system should be able to decide a simple response to the request.
- Tracking system to see when a delivery was made.
- Developing the system to support many users at a time.
- Optimizing for improving the system's performance of serving map and location services.
- Comparison Apache server and NGINX server on the same system.
- Comparison MapServer and GeoServer on the same system.
- Developing on Cloud.

Bibliography

[1] Erik Westra. 2013. **Python Geospatial Development.** 2nded. Packt Publishing Ltd.

[2] Kyle Rankin and Benjamin Mako Hill. 2010. **The Official Ubuntu Server Book.** Pearson Education, Inc.

[3] Clement Nedelcu. 2010. **Nginx HTTP Server.** Packt Publishing Ltd.

[4] Adrian Holovaty and Jacob Kaplan-Moss. 2009. **The Definitive Guide to django Web Development Done Right.** 2nded. Springer-Verlag New York, Inc.

[5] Juliet Kemp. 2009. Linux System Administration Recipes A Problem-Solution Approach. Springer-Verlag New York, Inc.

[6] Regina O. Obe and Leo S. Hsu. 2013. **PostGIS IN ACTION.** 2nded. Manning Publications Co.