Route Finding Using Geographic Information System

Abstract

The main goal of this work is to find shortest route between one building to another first at the desktop environment and then using the Web map service. The research part of this work will comprise of a look at Geographic Information systems (GIS) technologies including advantage of using GIS. GIS Web services, and how these interact with each other will also be discussed. Following this, the GIS standards will be discussed with a more detailed discussion on OGC simple feature specification ,WMS and WFS

The practical part of this work will comprise of setting up a hardware and software environment. Components integrated in the system include a spatial database PostGIS, OpenJump, a GIS server GeoServer and front end technology OpenLayer. Following this, an exploration of the environment will take place in the form of practical hands on use of the software to build an understanding of how the research can be applied in a practical way. The streets use OSM data from CloudMade and the buildings data are a subset of the Geodirectory from An Post originally.

Keywords: Open Source, pgRouting, GeoServer, OpenLayers, route finding

List of Figures

List of Tables

Listings

Chapter 1. Introduction

1.1 Aim

The aim of this project is to research and develop an open source geographical information system that allows to find the shortest route between two points .i.e buildings. Find route from one building to another building using PostgreSQL/PostGIS (initally display result in OpenJump). Key to achieving this aim is the successful integration of database technology, a GIS server and front-end web technology to display information to the end user through a browser.

1.2 Objectives

By choosing my main objective was to:

- Research, install and configure a spatial database.
- Research, install and configure client that act as gis viewer to perform analysis of datasets, query the database.

- Research, install and configure software to act as the client interface in conjuction with the user's browser
- Research, install and configure coding language that interact with spatial database.
- Acquire competency using the tool OpenJump, PostgreSql and GeoServer.
- To acquire the data which will be used for the route finding
- To load the database with valid data
- To design the systems that integrates all the components
- To create a queries that will allow the user to initially interact with the data in OpenJump
- To create an interface that allows the user to interact with the system .

1.3 Glossary

AJAX	Asynchronous JavaScript and XML
Bounding Box	This is a rectangle that is used to identify the area we are interested in, whether that is finding geometries in a database or telling a map server the area of the map we wish to see
Clientside	Client side referrs to the user's computer, specifically their web browser.
EPSG	European Petroleum Survey Group. This is the standards group for the naming of projections. The two we are interested in are 4326 (WGS84) and 29900 (Irish Grid)
DIV	This is a HTML instruction for dividing the layout of a web page
GIS	Geography Information System

GML	Geographic Markup Language. This is based on XML and is used to describe geographic features in respect to their location, shape and height
OGC	Open Geospatial Consortium, which is the standards body for geospatial information like web based services, spatial databases
OpenJump	This is a Java based product for exploring spatial information
OWS	OGC web service
URL	Uniform Resource Locator
WFS	Web Feature Service. This is a service given by a Map Server to return features in GML and also allow transactions (WFS-T) on the feature.
WKB	Well Known Binary. This is how a database often stores a geographic feature in the database
WMS	Web Map Service. This is a service provided by the Map Server to return map images to the client for a specified area and sometimes conditions using filtering

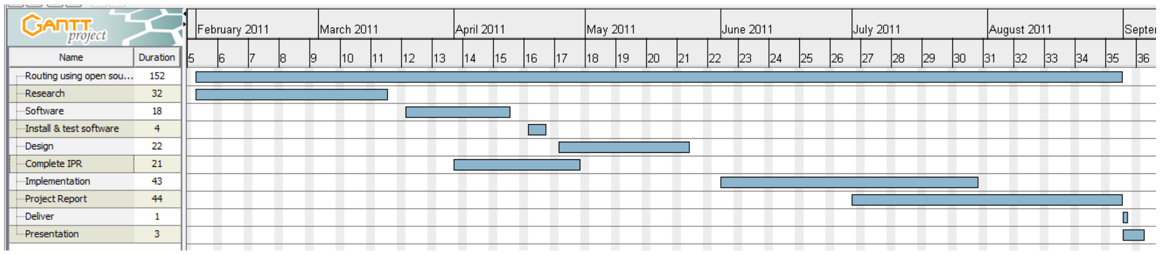
Table 1. 1:List of Glossary items

1.4 Methodology

This project involves only one resource (me) however traditional life cycle models requires many specialist So iterative style approach i.e. prototype method would be suitable for this project where all the requirements are defined then design the system and then implement rather then developed the whole product and move to the another area that is not well known.

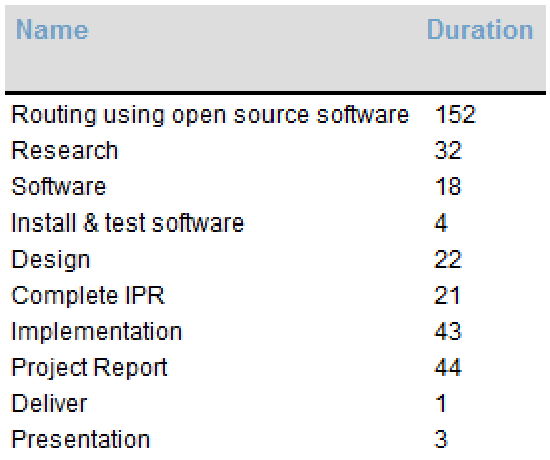
1.5 Project Management

Project management is very important activity that overlaps many phases of system methodology. According to Whitten et al (2001) project management is the process of defining, planning, directing, monitoring and controlling a project to develop a system within allocated time and budget . Figure 1.1 is the gantt chart showing the project plan to be followed.



Name	Duration
Routing using open sou...	152
Research	32
Software	18
Install & test software	4
Design	22
Complete IPR	21
Implementation	43
Project Report	44
Deliver	1
Presentation	3

Figure 1. 1: Gantt chart showing the project plan



Name	Duration
Routing using open source software	152
Research	32
Software	18
Install & test software	4
Design	22
Complete IPR	21
Implementation	43
Project Report	44
Deliver	1
Presentation	3

Figure 1.2:Listing of the task name and duration from Gantt Chart

1.6 Overview of the architecture used

The diagram below illustrates the different components needs to build the system and the interaction between them. Each of these and their installation and usage will be discussed later in document.

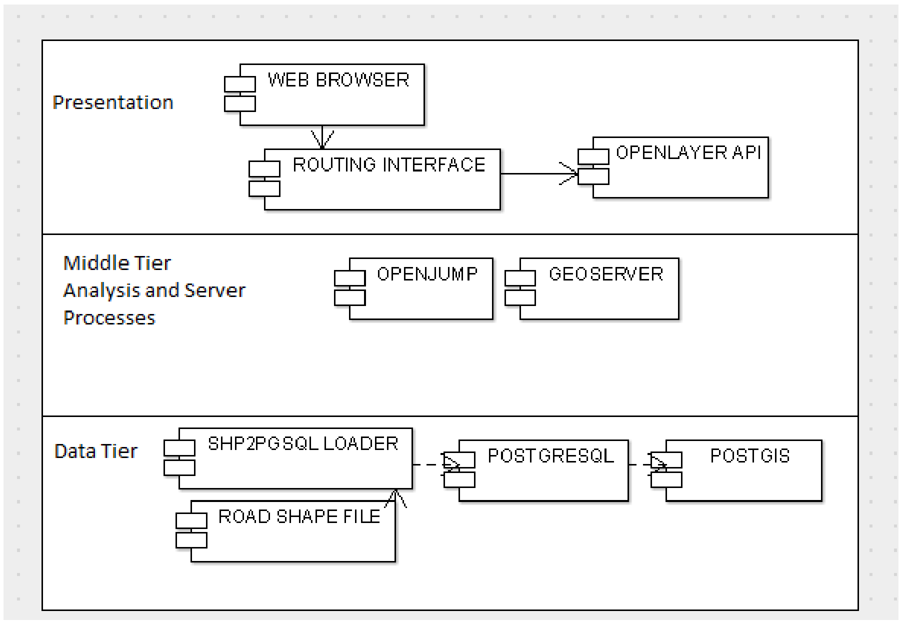


Figure 1.3: Three Tier Architecture

1.7 Report Overview

Chapter 1: Introduction

Chapter 1 provides the aim and objectives of the project. It also list the glossary terms used in this project and describes the chosen development method.

Chapter 2: Background

Chapter 2 provides background information on GIS technology ,its advantages in our daily life. Also provides information on the specific aspects of GIS technology and standard related to the project.

Chapter3: Technology requirement

Chapter 3 discusses and evaluate the different technologies and application needs, and underlines the reason for the choosing tool to support the implementation of this project.

Chapter4: Shortest path algorithm

Chapter 4 explains the Dijkstra shortest path routing algorithm which was chosen for this project.

Chapter 5: Loading data, setting for pgrouting and putting postgis data into geoserver

Chapter 5 deals with procedures how the data was loaded and store in the database and outlines the integration between PostGIS and Geoserver. This chapter focuses on how the individual components are brought together.

Chapter6: Code

Chapter 6 Explains the code written in PL/pgSQL language and OpenLayer format.

Chapter7: Conclusion

Chapter 7 Summarizes the whole project and outlines the learning outcomes, the benefit the application and further work.

2. Research Areas

This chapter presents the research areas that were crucial for this project. It begins by describing what geographic information system is and those elements that makes the GIS possible. It also point outs the various standards of Open Geospatial Consortium and their relevance to the web mapping services.

2.1 Geographic Information Systems

Geographic Information Systems is neither a single thing nor a single analysis(ISS, 2006), the primary thing that makes GIS difference is location, the place where almost everything that happens, happens somewhere (Longley et al. 2010).Whether it's the regular delivery of morning newspaper, the synchronization of traffic lights on way to work, or the convenient location of favorite park, GIS make these things happen (ESRI,2011a), so “GIS is a computerized tool for solving geographic problems” (Longley et al. 2010, p.16). In today’s 21st century organizations all over the world are using GIS to manage the environment, work more efficiently, provide better customer service and save money. A geographic information system (GIS) is the integration of hardware, software, and data for capturing, managing, analyzing, and displaying all forms of geographically referenced information (GIS.com, 2011).

Many historians and historical geographers regard GIS as primarily being concerned with mapping. Although mapping is one of the key abilities of GIS is a specialized form of database because each item of data, be it a row of statistics, a string of text, an image,or a movie, is linked to a coordinate-based representation of the location that the data refer to (Geogray & Healy ,2007).Thus GIS combines spatial data in the form of points, lines, polygons, or grid cells, with the attribute data held in conventional database form. This provides a structure that is able to answer queries not only about what features are in the database, but also about where they are located. This is what makes GIS unique

GIS maps are interactive. On the computer screen, map users can scan a GIS map in any direction, zoom in and out to see different areas with more or less detail, they can decide what features they want to see and how they are symbolized, and, most importantly, they can access a database of information about all the features shown on the map (GIS, 2008).

2.2 Components of GIS

The following diagram illustrates the six component parts the geographic information systems

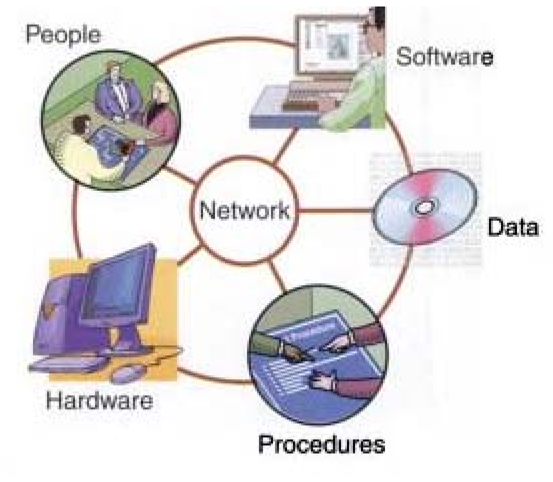


Figure 2. 1: Parts Of Gis(Longely 2007)

2.3 Why use GIS?

GIS provides numerous benefits and advantages in our daily life. According Longley et al. (2010) GIS

- Affects each of us, every day for example the energy to power the alaram comes from the local energy company which makes the use of GIS to manage all its assets.
- Can be used to make effective decision making
- Has great practical importance
- Encourage public participation in decision making
- Supports mapping, measurement, management, monitoring and modeling operations for example one can combine the location of mobile workers, located in real-time by GPS devices, in relation to customers' homes, located by address and derived from your customer database. GIS maps this data, giving dispatchers a visual tool to plan the best routes for mobile staff or send the closest worker to a customer. This saves tremendous time and money(GIS.com)
- Provides a challenging and stimulating educational experience for students..

2.4 Spatial Databases

Some sort of data is needed to be GIS useful. Spatial databases are such systems designed specifically to include data with spatial attributes, such as geographical location, distance, and extent (Winstanley, 2009). Güting (1994) defines spatial database system as:

1. A spatial database system is a database system
2. It offers spatial data types in its data model and query language
3. It supports spatial data types in its implementation, providing at least spatial indexing and efficient algorithms for spatial join2.

2.5 Spatial Data

The fundamental ways of representing geography data in digital computers are Raster and Vector.

2.5.1 Raster data

Raster data as shown in the figure geography information is represented as an array of square cells. Remote sensing satellite is one of most common form of raster which capture information in raster form (Longley et al. 2010).A raster data associates attributes with grid cells.

2.5.2 Vector data

Vector data as shown in the figure is a coordinate-based data model which represents geographic features as points, lines, and polygons. Each point feature is represented as a single coordinate pair, while line and polygon features are represented as ordered lists of vertices (ESRI, 2011b). This project makes a use of vector data.

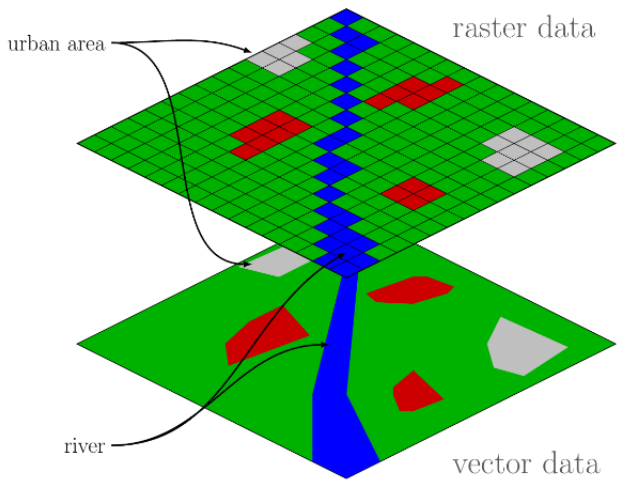


Figure 2. 2: Raster and vector

2.6 Spatial join

A spatial join is done between road and buildings table dataset used in this project with respect to a spatial predicate. A spatial join links two or more tables based on a spatial relationship, rather than the classic non-spatial relational attribute (Lecture2, 2011). Predicates can be a combination of directional, distance, and topological spatial relations (e.g. overlap, contains). In case of non-spatial join, the joining attributes must of the same type, but for spatial join they can be of different types (Lecture2, 2011).

Example:

Query: For all the rivers listed in the River table, find the counties through which they pass.

```
SELECT r.name, r.name
FROM river AS r, county AS c
WHERE crosses(r.the_geom,c.the_geom)=True
```

The spatial predicate "Cross" is used to join River and Country tables

2.7 OpenStreetMap

OpenStreetMap data was used for the road data required for this project so it was necessary to understand about OpenStreetmap data. OpenStreetMap is a free editable map of the world (OSM). Its aim is to create a set of map data that's free to use, editable, and licensed under new copyright schemes (Haklay, & Weber, 2008). First this started with mapping streets, now it has already gone far beyond which include footpaths, buildings, waterways, pipelines, woodland, beaches, postboxes, and even individual trees. The project also includes administrative boundaries, details of land use, bus routes, and other abstract ideas that aren't visible from the landscape itself.

2.7.1 Data Format

OpenStreetMap uses three basic primitive data models: nodes, ways and relations.In mathematical terms openstreetmap data model is mixed graphs which consits of vertices and edges (Bennett,2010).

The default format for representing the data model is XML.

There are several attributes common to every primitive type. Each has a numerical ID, but these are only unique within each type, so there could be a node, way, and relation all with the same ID number(Bennett,2010).

2.7.1.1 Nodes

Nodes represents points in space which provides position information and all other primitives rely on nodes for their location. In the figure the The Blue Blazer is represented as a node which is a pub in Edinburg (Bennett,2010).

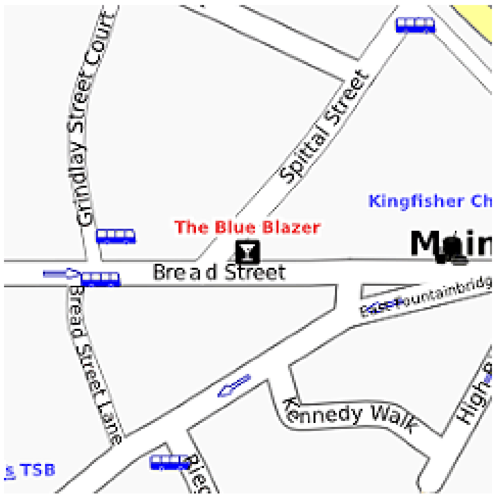


Figure 2. 3:The Blue Blazer represented as node(Bennett, 2010).

2.7.1.2 Ways

Ways are ordered list of nodes which represents a polyline or polygon such as roads, paths and waterways (Bennett,2010). They can also be closed to form areas. Where they're used to describe linear features, the way should normally be placed down the center line of the physical feature, and at the perimeter for an area. Figure 2.5 shows the way which is the southern end of Parliament Street in London.

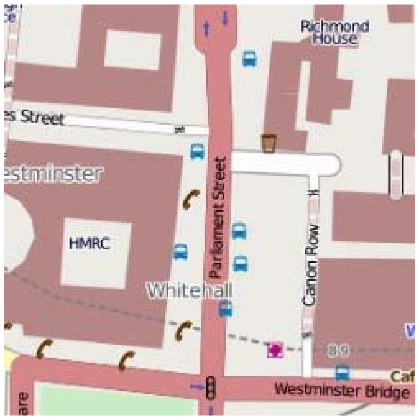


Figure 2. 4: Parliament Street in London represented as way(Bennett, 2010).

2.7.1.3 Relations

Relations are groups of nodes, ways and other relations which can be assigned certain properties (Bennett, 2010). Relations allow mappers to model features that can't be described using a single node or way, or where two of the same type of feature overlap. Examples include complex, branching streets, long distance routes, or the turn restrictions at junctions.

2.8 Web services

W3C Web Services architecture(WSA) (2004)specification defines a web service technically as follows:

"A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a

manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards".

Web services are frequently used by web application programming interfaces (API's) that can be accessed over a network and executed on a remote system hosting the requested

GIS web services are one of the invention of the web service. Google Maps and Yahoo Maps are the examples of the GIS web services. GIS web services provides geospatial data on the web and as well as allow people to ask question based on the location (Deoliveira).

This project is also based on GIS web services which is concerned with bringing World Wide Web Consortisum and Open Geospatial Consortium (OGC) together to provide GIS web service i.e to find the shortest route between two houses.

2.9 Industry Standards

The Open Geospatial Consortium was founded by small group of member in 1994 (OGC History) with a vision to "Achieve the full societal, economic and scientific benefits of integrating location resources into commercial and institutional processes worldwide" (Reed, 2011).
This project adherence to the OpenGIS Simple Features Specification For SQL to support storage and query and OGC compliant GeoServer web map service(WMS) and web feature service (WFS).

2.9.1 OGC Simple Features Specification

A simple feature is defined by the OpenGIS as an abstract specification which have both spatial used for shape file and non-spatial attributes suitable to use when manipulating row data. Spatial attributes are geometry valued, and simple features are based on 2D geometry with linear interpolation between vertices [OGCSFS1.1,].

Feature specifications schema was appropriate for used in order to return the list of feature tables from a database, the list of geometry columns for any feature table in the database and the spatial reference system for any geometry column in the database

The following basic functions were used to manipulate geometry object:

- SRID ( ): Integer—used to manipulate the Spatial Reference System ID of the geometric object
- AsBinary( ):Binary-allow well-known binary representation of Geometry to their boundaries.
- AsText( ):String-allow well-known text representation of Geometry

The following methods were used for testing Spatial Relations between geometric objects and to support spatial analysis respectively.

- Contains(anotherGeometry:Geometry):Integer- Returns (True) if this Geometry ' spatially contains' another Geometry.
- Buffer(distance:Double):Geometry-Returns a geometry that represents all pointswhose distance from this Geometry is less than or equal to distance.Calculations are based in the Spatial Reference System of the Geometry.

The following diagram illustrates the representation of geographic object and how they are connected on the space representing as point, polygon or multipoint.

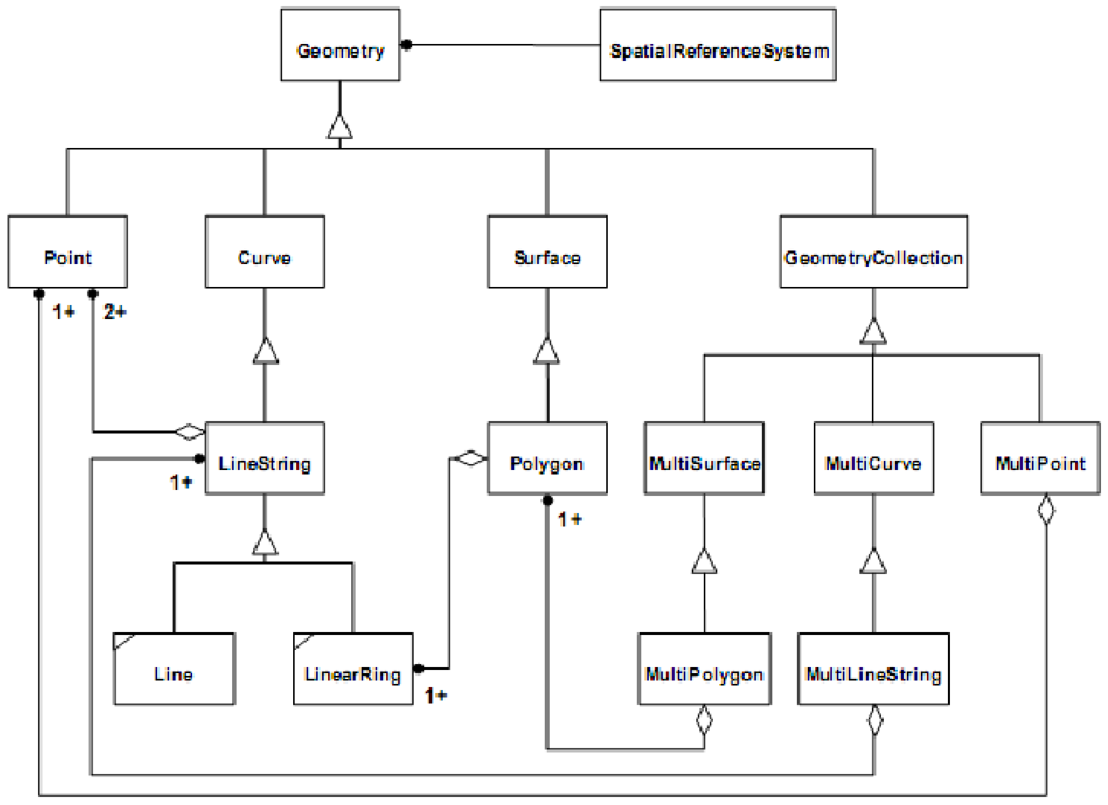


Figure 2. 5:OpenGIS Geometry Class Hierarchies (OGCSFS 1.1, p2-2)

As it can be seen from figure that there are many different types of spatial information that can be stored in a compliant system, we will be mainly concerned with points and lines. Points have only an X and Y value in respect to a spatial object, for buildings Whereas lines are a set of points that are grouped together in a particular sequence to give a line, these would be the routes from one building to another.

2.9.2 Features Table Architectures

As this project data are taken in PostGIS it was necessary to understand the table structure of OpenGIS Simple Feature Specification for SQL how the geographic object are presented.

The figure below describes the database schema necessary to support the OpenGIS simple feature data model. A feature table or view corresponds to an OpenGIS feature class. Each feature view contains some number of features represented as rows in the view. Each feature contains some number of geometric attribute values represented as columns in the feature view. Each geometric column in a feature view is associated with a particular geometric view or table that contains geometry instances in a single spatial reference system. The correspondence between the feature instances and the geometry instances was accomplished through a foreign key that is stored in the geometry column of the feature table.

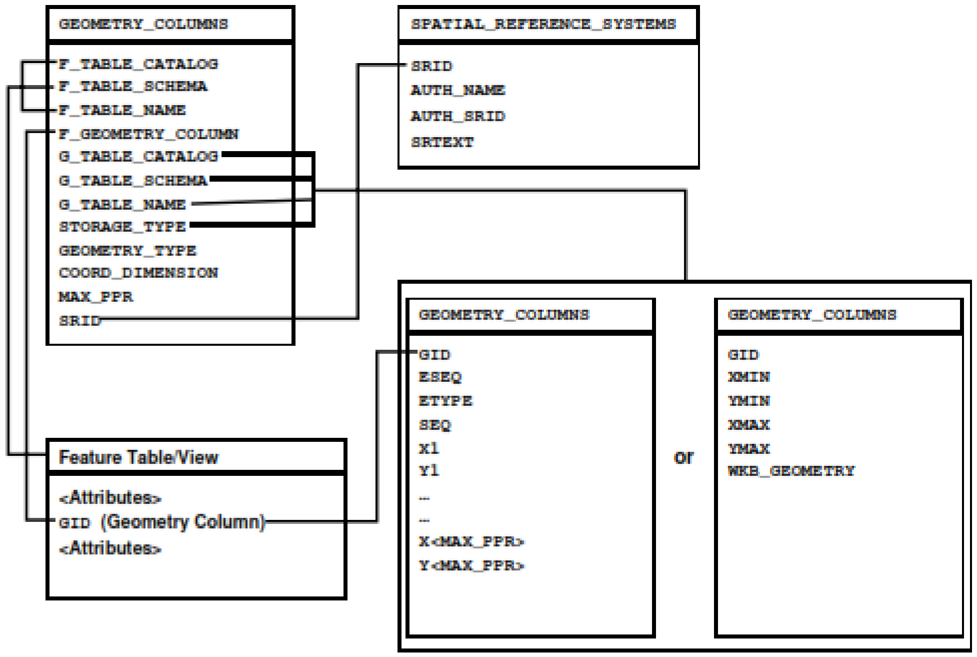


Figure 2. 6: Schema for feature tables under SQL92 (OGCSFS1.1, p2-20)

2.9.3 OGC web map service and web feature services

GeoServer forms a core component of the Geospatial Web by providing the reference implementation of the Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS)(GeoServer, 2011). However for this project we are only concerned with the Web Map Service and Web Feature Service. Figure 2.10.3 below shows the OGC Web Service Architecture which displays how the specifications are related to each other and also the operations that each one defines.

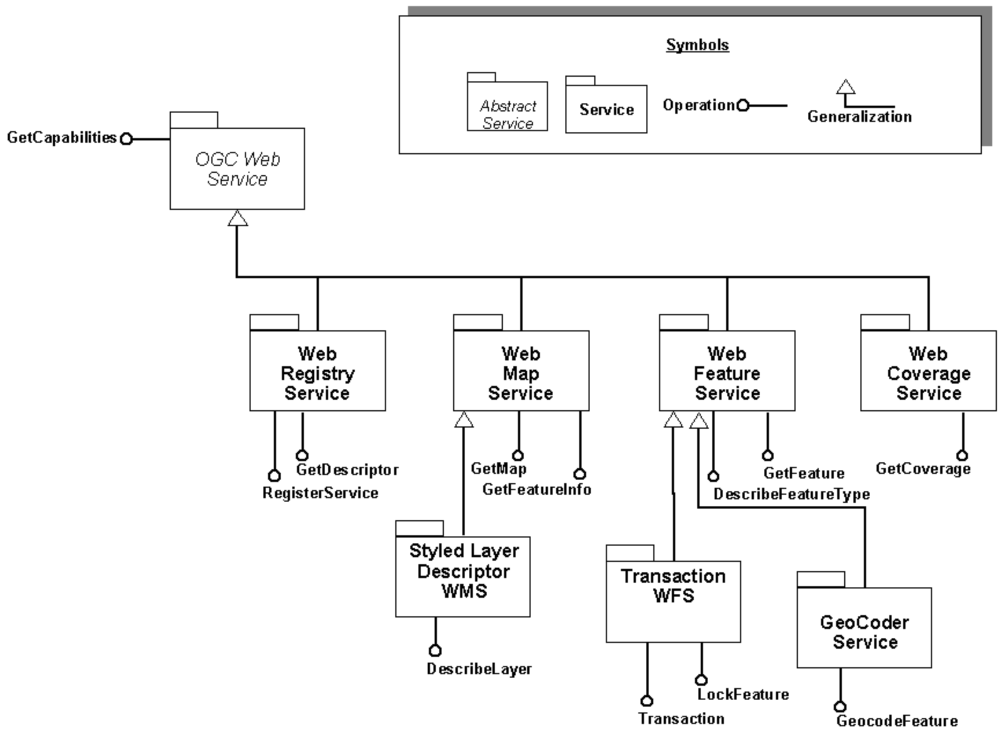


Figure 2. 7 The OGC web service architecture (OWS, 2002)

Generally, the interaction between the Client and Server in both the Web Feature Service and Web Map Service as defined in their respective specification documents function in similar ways. Both services provide public interfaces through which clients can request information about the feature types and the operations on those feature types that it stores and both services respond to those requests with XML documents.

The communication between client and server is achieved via the HTTP protocol over a heterogenous network, such as the World Wide Web. HTTP supports both GET or POST methods (OGC, 2002). Both services have a GetCapabilities request that provides service level information available about the WMS/WFS. In a GET request, the service that the client requires is specified through key value pairs (KVPs) in the URL of the request. In a POST request, the content of the request is encoded in XML and passed to the server through the POST-BODY of the request header.

2.9.3.1 Web map Service (WMS)

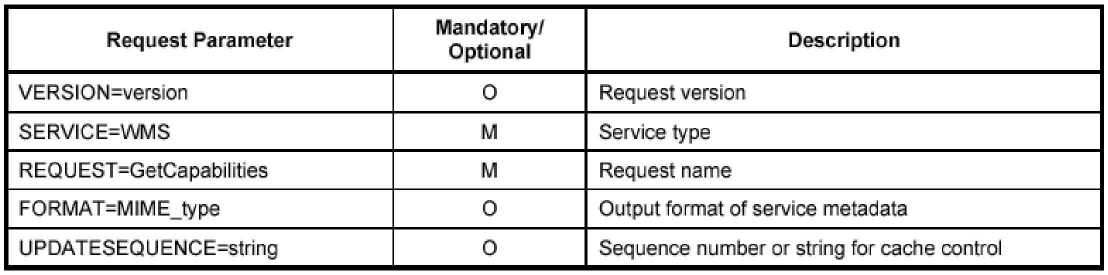
WMS is a standard for displaying map images. WMS can register and overlay maps from multiple remote sources. A map is not the data itself. WMS-produced maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats (OGCWMS 1.3.0, p5).

WMS is fine for presentation and a delivery mechanism, but not good for user interaction

The OGC Web Map Service Standard supports the 3 following basic interfaces: GetCapabilities,
GetMap and GetFeatureInfo.

2.9.3.1.1 GetCapabilities

GetCapabilities is a request from a client to a service that gives back the capabilities or services that this service provides for maps (OGCWMS1.3, p21). This is an XML document that gives the metadata of the information available. The metadata is readable by a machine and the human eye and contains a description of the server's information content and the acceptable request parameter values. Listed below are the possible parameters of a `GetCapabilties` request.



Request Parameter	Mandatory/ Optional	Description
VERSION=version	O	Request version
SERVICE=WMS	M	Service type
REQUEST=GetCapabilities	M	Request name
FORMAT=MIME_type	O	Output format of service metadata
UPDATESEQUENCE=string	O	Sequence number or string for cache control

Table 2. 1: The parameters of a GetCapabilities request(OGCWMS1.3, p21)

The GetCapabilities response is an XML document which contains the service metadata. The metadata will appear in a human readable format in the client application. In the case of this work that is a web browser (OGCWMS1.3, p22).

The contents of the metadata that is returned can include:

- Service information such as a name, title, URL and other optional information such as contact information, fees, access constraints.

- Capability information which lists the operations supported by the server, their output formats and also the URL prefix for each of these operations.

- Layers and styles metadata which outlines the layers and styles that are available from the server.

- Format specifiers which include valid output formats for an operation, supported exception formats and the format of context at URLs .

2.9.3.1.2 GetMap

The `GetMap` operation allows to return a map to the user's client application. The `GetMap` operation provides the parameters which are outlined in table 2.2 to allow the

user to send a `GetMap` request to a web map server to retrieve a map (OGCWMS1.3, p32).
The response of which is a map of spatially referenced information based on the parameters specified in the request.

Request Parameter	Mandatory/ Optional	Description
VERSION=1.3.0	M	Request version.
REQUEST=GetMap	M	Request name.
LAYERS=layer_list	M	Comma-separated list of one or more map layers.
STYLES=style_list	M	Comma-separated list of one rendering style per requested layer.
CRS=namespace:identifier	M	Coordinate reference system.
BBOX=minx,miny,maxx,maxy	M	Bounding box corners (lower left, upper right) in CRS units.
WIDTH=output_width	M	Width in pixels of map picture.
HEIGHT=output_height	M	Height in pixels of map picture.
FORMAT=output_format	M	Output format of map.
TRANSPARENT=TRUE|FALSE	O	Background transparency of map (default=FALSE).
BGCOLOR=color_value	O	Hexadecimal red-green-blue color value for the background color (default=0xFFFFFF).
EXCEPTIONS=exception_format	O	The format in which exceptions are to be reported by the WMS (default=XML).
TIME=time	O	Time value of layer desired.
ELEVATION=elevation	O	Elevation of layer desired.
Other sample dimension(s)	O	Value of other dimensions as appropriate.

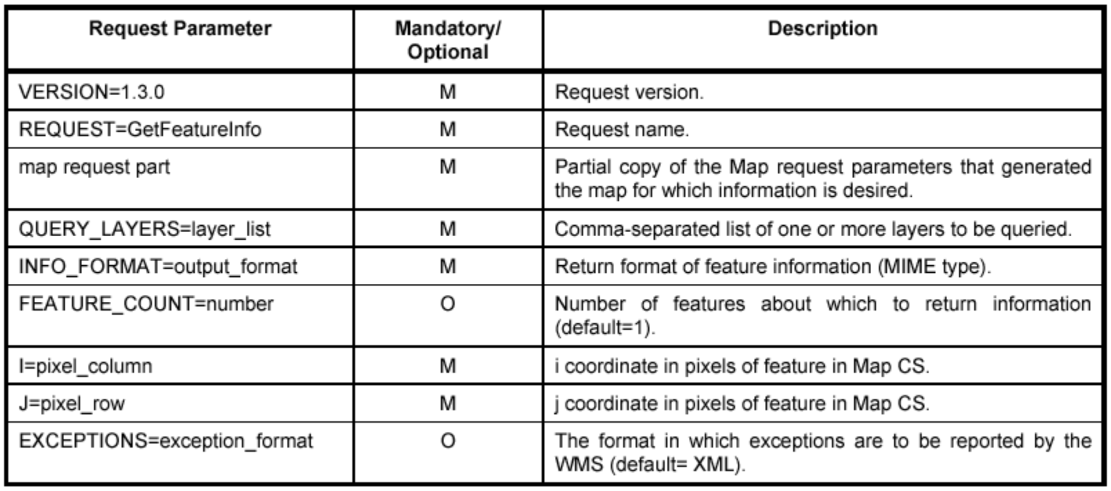
Table 2. 2 The Parameters of a GetMap request (OGCWMS1.3, p33)

The response to a valid GetMap request is a map which corresponds to spatially referenced information layer requested, in the desired style, and having the specified coordinate reference system, bounding box, size, format and transparency (OGCWMS1.3, p37).
An invalid GetMap request shall yield an error output in the requested Exceptions format (or a network protocol error response in extreme cases).

2.9.3.1.3 GetFeatureInfo

`GetFeatureInfo` allows to get information about previous map requests. An example of this is that when a user sees a map, they can click on a point on this map to obtain more information (OGCWMS1.3, p38). It provides the possibility of the client to request

the details of geometry in the map through a mouse click at an XY coordinate on the active map layer on screen.
The parameters of a GetFeatureInfo request are listed in Table 2.3.



Request Parameter	Mandatory/ Optional	Description
VERSION=1.3.0	M	Request version.
REQUEST=GetFeatureInfo	M	Request name.
map request part	M	Partial copy of the Map request parameters that generated the map for which information is desired.
QUERY_LAYERS=layer_list	M	Comma-separated list of one or more layers to be queried.
INFO_FORMAT=output_format	M	Return format of feature information (MIME type).
FEATURE_COUNT=number	O	Number of features about which to return information (default=1).
I=pixel_column	M	i coordinate in pixels of feature in Map CS.
J=pixel_row	M	j coordinate in pixels of feature in Map CS.
EXCEPTIONS=exception_format	O	The format in which exceptions are to be reported by the WMS (default= XML).

Table 2. 3: The parameters of a GetFeatureInfo request. (OGCWMS1.3, p39)

2.9.3.2 WFS

The OGC Web Map Service allows a client to overlay map images for display served from multiple Web Map Services on the whereas the OGC Web Feature Service allows a client to retrieve and update geospatial data encoded in Geography Markup Language (GML) across the Web using platform independent calls (OGC 1.1.0, p12).
The WFS standard defines interfaces and operations for data access and manipulation on a set of geographic features, including (OGC 1.1.0, p7):

Get or Query features based on spatial and non-spatial constraints
Create a new feature instance
Get a description of the properties of features
Delete a feature instance (WFS-T)
Update a feature instance (WFS-T)
Lock a feature instance (WFS-T)

By default, the specified feature encoding for input and output is the Geography Markup Language (GML) which in turn is written in XML.

A WFS specification provides 3 basic interfaces that allow it to service the requests that it receives from clients. These are: GetCapabilities, DescribeFeatureType, and GetFeature. These are considered as a READ-ONLY web feature service (OGCWFS 1.1, p17).

2.9.3.2.1 GetCapabilities

A web feature service must be able to describe its capabilities. Specifically, it must indicate which feature types it can service and what operations are supported on each feature type(p16). The request can be sent as either GET or POST request. The GET request URL looks like this:

```
http://hostname[:port]/path/ows?service=WFS&request=GetCapabilities
```

The same service can also be requested by sending the request encoded in XML in the POST body:

2.9.3.2.2 DescribeFeatureType

A web feature service must be able, upon request, to describe the structure of any feature type it can service (OGCWFS 1.1 ,p16).The response schema describes the feature types available and the operations allowed on those feature types.

2.9.3.2.3 GetFeature

A web feature service must be able to service a request to retrieve feature instances. In addition, the client should be able to specify which feature properties to fetch and should be able to constrain the query spatially and non-spatially.

2.10 Summary

This project researches and integrates the components of GIS i.e hardware, software,data, procedures and people.

Gis data can be stored as a point , lines and polygon in the spatial database. Some sort of data is needed for the GIS to be useful ,this project uses open street map data (which is free editable data) for the street data which are stored as a lines buildings data which is stored as points.

Processing geographic data was originally only available through desktop applications but now is available over the Internet either via client-server interaction . This project is based on both desktop and client server model.

3 System setup

There is no definitive set of components to use when creating a GIS. This chapter details the system that has been chosen for this project.

3.1 Hardware

All applications are installed on a single machine, in this case an HP laptop running Windows 7 which has the following specifications (Windows 7).

CPU	2.20 GHz
RAM	4.00 GB

Operating System	Windows 7

Table 3. 1:Hardware

3.2 Software

Table 3.2 shows the main list of the software chosen for this project.

	Version	Function
PostgreSQL	8.4	Database
Postgis	1.4	Database
pgRouting	1.0.3	Routing
OpenJump	1.3.1	Client,desktop gis
Geoserver	2.1.0	Mapserver
Openlayers	2.8	Web client,front end
Firefox	5.0	Web browser
PL/pgSQL		Procedural language

Table 3. 2:Software Applications used in system

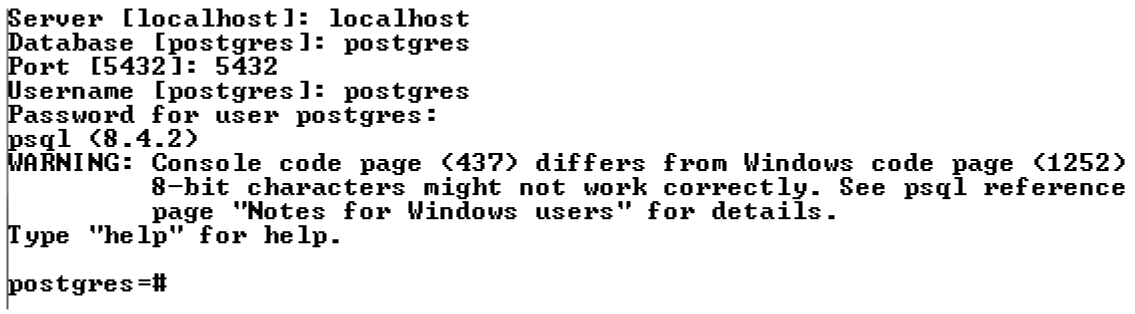
3.2.1 PostgreSQL

PostgreSQL is a powerful, open source object-relational database system which has more than 15 years of active development and a proven architecture because of its strong reputation for reliability, data integrity, and correctness (PostgreSQL, 1996-2010).

PostgreSQL with the spatial addon PostGIS was chosen as the author has previous experience with the software and it integrates easily with other software used in the project. It was the ideal to for dataset storage, as is a free and open source application and it easy to install and use.

PostgreSQL is a database server product when request are made to the database the server processes the request, prepares the data and returns result to the application (Mitchell, 2005).

Listing below shows the database service is running.



```
Server [localhost]: localhost
Database [postgres]: postgres
Port [5432]: 5432
Username [postgres]: postgres
Password for user postgres:
psql (8.4.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
Type "help" for help.

postgres=#
```

Listing 3. 1:Verifying PostgreSQL database is working properly

3.2.1.1 Standard Compliance

PostgreSQL includes most SQL which strongly conforms to ANSI-SQL:2008 standard and has full support for subqueries (including subselects in the FROM clause), read committed and serializable transaction isolation levels (PostgreSQL, 1996-2010).

3.2.1.2 Features

PostgreSQL runs on all major platforms. Its data integrity feature provides support for foreign keys, joins, views, triggers, and stored procedures (in multiple languages). It includes most SQL: 2008 data types, it also supports storage of binary large objects.

It supports compound, unique, partial, and functional indexes which can use any of its B-tree, R-tree, hash, or GiST storage methods.

This project makes use of procedural language feature of PostgreSQL and PL/pgSQL language was selected which installs by default within PostgreSQL

3.3 PostGIS

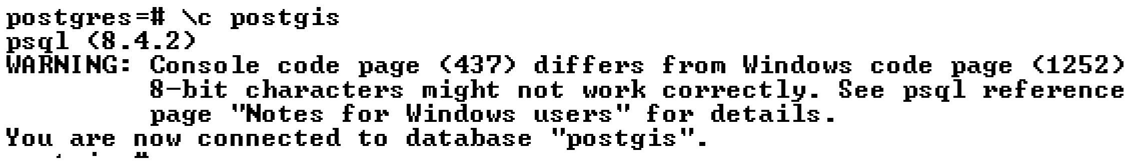
PostGIS is the most powerful open source spatial database which spatially enables the PostgreSQL open source relational database management system (Obe & Hsu, 2010). PostGIS is an extension to the PostgreSQL object-relational database system which allows Geographic Information Systems objects to be stored in the database.
PostGIS includes support for GiST-based R-Tree spatial indexes, and functions for analysis and processing of GIS objects (PostGIS).
It adds to PostgreSQL several spatial data types and over 300 functions for working with these spatial types. geometry/geography types packaged in Microsoft SQL Server 2008+ do for SQL Server.

PostGis was installed for this project to stored the spatial data such that it was possible to do interaction from openjump, openlayer and server .i.e geoserver which was used through test face to implementation phase of this project.

3.3.1 Testing PostGIS functionality

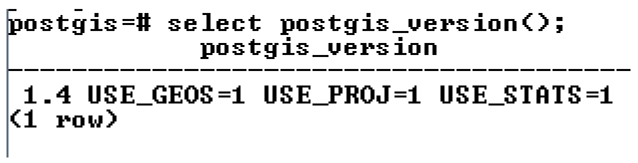
Some simple tests can be done to connect to the postgis database using psql command. The psql command lists the databases and runs scripts and also provides an interface for typing in SQL commands and shows query results. This is done by starting psql followed by the name of a database to connect to (Mitchell, 2005).

When psql start up tells the version of the program, gives few lines of helpful tips to start the program and then leaves with the prompt the prompt is the name of the database followed by =#



```
postgres=# \c postgis
psql (8.4.2)
WARNING: Console code page (437) differs from Windows code page (1252)
         8-bit characters might not work correctly. See psql reference
         page "Notes for Windows users" for details.
You are now connected to database "postgis".
```

Listing 3. 2: Verfying PostGIS functionality



```
postgis=# select postgis_version();
            postgis_version
---------------------------------------
 1.4 USE_GEOS=1 USE_PROJ=1 USE_STATS=1
(1 row)
```

Listing 3. 3: Showing the version of PostGIS

The output tells the version of PostGIS being used is 1.4 and also shows that it can use GEOS and PROJ libraries as part of PostGIS. Proj libraries are used to reproject or transform coordinates and GEOS is an advanced geometry engine that allows a whole suit of manipulations and analysis of geometry data types (Mitchell, 2005).

3.3.1 Standard Compliance

PostGIS/PostgreSQL was the first open source database to support OGC compliant spatial SQL, PostGIS supports many of the OGC/ISO SQL/MM compliant spatial functions you will find in these other OGC compliant databases as well as numerous additional ones that are unique to PostGIS.

3.3 OpenJump

OpenJUMP is an open source Geographic Information System (GIS) written in the Java programming language (openJUMP). It is a vector GIS that can read raster as well(SourceForge).

SQL queries can be issued and view directly from OpenJump which act as a client for PostgreSQL/PostGIS (Lecture 4, 2010). However PostgreSQL/PostGIS queries bring a layer into OpenJump, they do not update PostgreSQL/PostGIS (Lecture 4 ,2010).

Openjump was used through this project to perform analysis of datasets, query the database, add layer and attribute.

3.3.1 Standard Compliance

openJUMP supports OGC standard like simple feature interface standards(SFS) geographic markup language (GML), web map service(WMS), and web feature service(WFS) .The openjump interface can be seen in figure 3.1. The data that can be seen in figure 3.1 is dublin_highway1 and buildings_geodir.

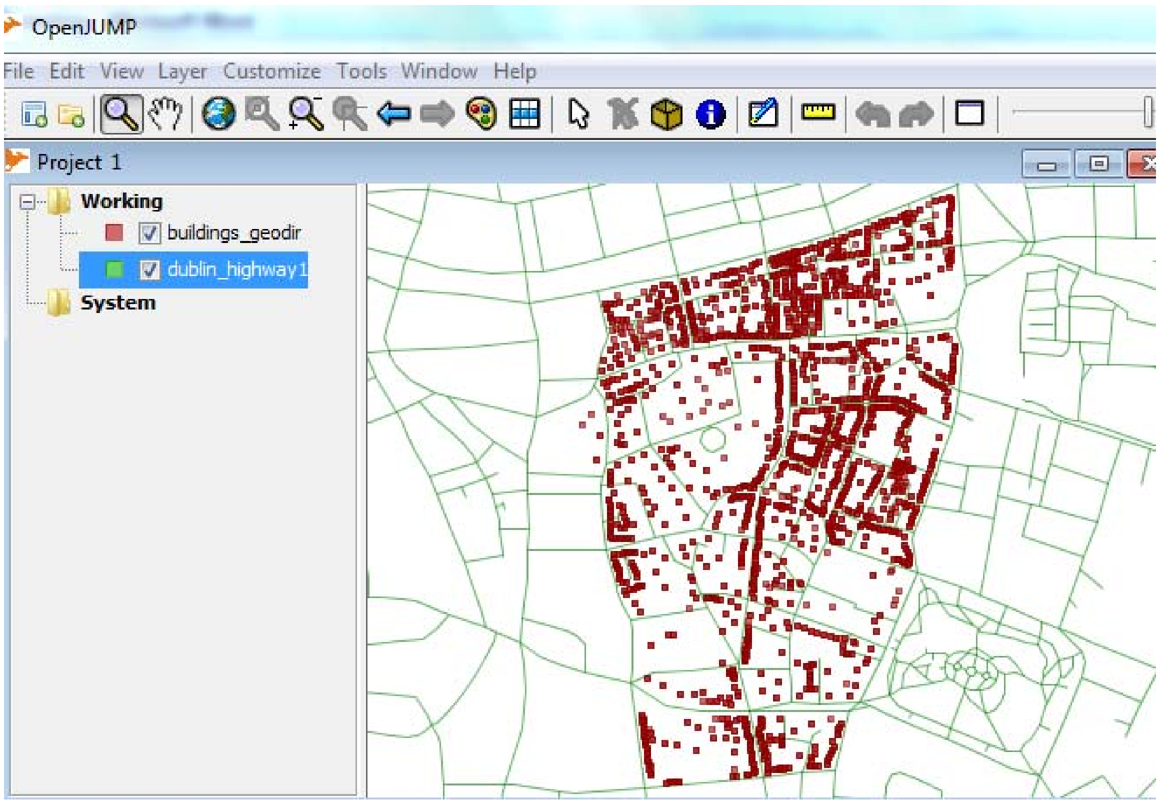


Figure 3. 1: Openjump user interface

The installation process was very easy to perform and it is supported by windows, Mac OS and Linux platform,it requires java 1.5 or later there is a great documentation on this website and growing user's community.

It can be use as GIS data viewer but has limits in reading very large data files and has limited support for cartographic projections [Informer] .

3.4 pgrouting

This project is based on route finding, PostgreSQL/PostGIS doesn't automatically provides the routing functionality so pgRouting was installed for that. pgRouting is an extension of PostgreSQL and PostGIS. It adds a geospatial routing functionality to PostGIS (FOSS4G, 2011). pgRouting is open source available under GPLv2 license.

pgRouting includes three types of shortest path search algorithm(FOSS4G,2011):

- Dijkstra,
- A-Star,
- Shooting Star

As author had previous knowledge of shortest path queries using Dijkstra shortest path so it was chosen among those three algorithms. Dijkstra algorithm is explained in chapter 4.

3.5 PL/PgSQL

PL/pgsSQL is a extension to SQL (Geschwinde & Schönig, 2001). This language initially written by Jan Wieck offers the programmer far more execution potential than `SELECT`, `INSERT`, or `UPDATE` commands.
It is a loadable, procedural language. A procedural language is a programming language where sequence of steps are specified and followed to produce an intended programmatic result (CMD, 1997).
PL/pgSQL is the only "PL" language installed by default for PostgreSQL so no extra effort was required to install this language as a part of this project, but many others are available, including PL/Java, PL/Perl, plPHP, PL/Python, PL/R, PL/Ruby, PL/sh, and PL/Tcl(PostgreSQL, 2011).

3.5.1 Why use PL/pgSQL

PostgreSQL is compatible with all data types, operators, and functions within PL/pgSQL code. The "SQL" in PL/pgSQL is indicates of the fact that allows to directly use the SQL language from within PL/pgSQL code (PostgreSQL, 2011).

PL/pgSQL was chosen as because multiple SQL statements can be executed from a PL/pgSQL code block, the statements are processed at one time, instead of the normal behavior of processing a single statement at a time as a result, this increases the power, flexibility and performance of the program

Another important aspect of using PL/pgSQL is its portability; it is platform independent its functions are compatible with all platforms that can operate the PostgreSQL database system.
Disadvantage is that inserting function in the database doesn't guarantee that function can be called (Geschwinde & Schönig, 2001).

3.6 Firefox

Mozilla Firefox is the open source web browser developed by Mozilla Corporation and was the browser of choice during development of the prototype. Firefox, was downloaded from http://getfirefox.com. uses the Gecko layout engine to render XHTML markup and CSS, is currently the most W3C open standards compliant browser (Mozilla.com).
Many extensions are available in firefox to enhance its capabilities firebug version 1.7.3 was downloaded from http://getfirebug.com . Firebug as an addon on firefox makes the web development process much easier and quicker. With these tools, anything on the site can be change on the fly, without editing or saving any files. So firebug provides an invaluable aid by providing developer level debug tools such as the JavaScript and CSS live edit feature and the ability to troubleshoot the HTTP request and response headers when working with Geoserver for this project.

3.7 GeoServer 2.1.0

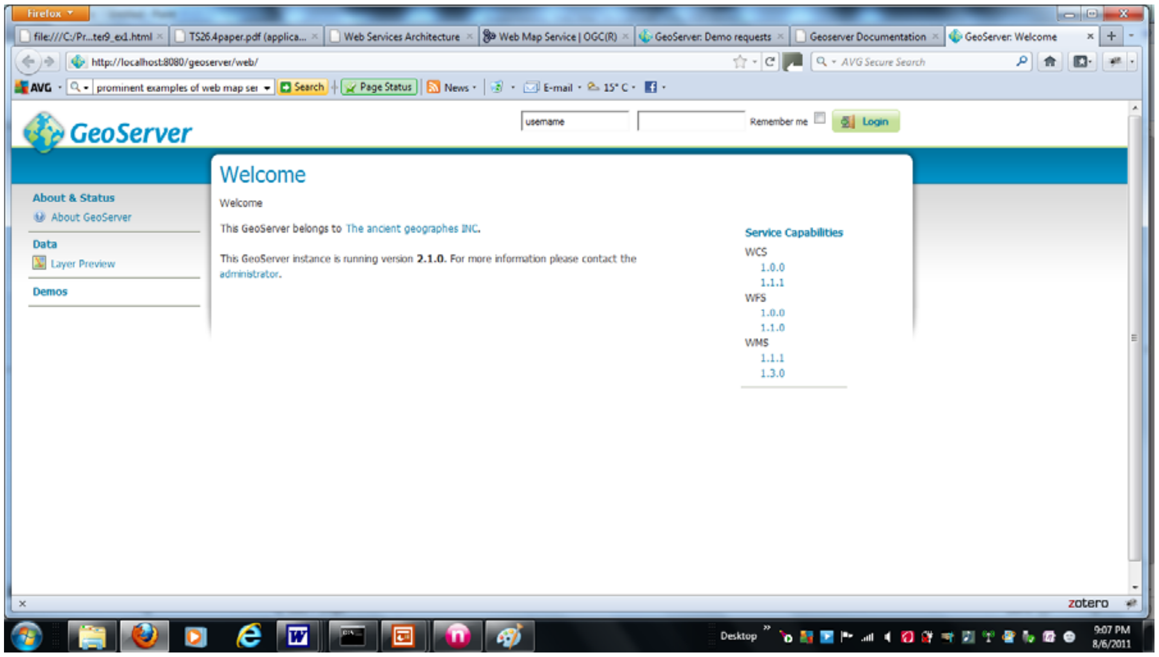
GeoServer is an Java Enterprise Edition open source software server Java that allows users to share and edit geospatial data (Geoserver 2.1.0).
GeoServer is Open Geospatial Consortium (OGC) Web Feature Service (WFS) and Web Coverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS).
Windows Installer version of GeoServer: `geoserver-2.1.0-ng.exe` was downloaded from http://geoserver.org/display/GEOS/GeoServer+2.1.0 and set up manually.
Users can create maps in a variety of output formats, as OpenLayers which is an integrated part of its structure, map generator will be quick and easy, and will be easy to establish connection with traditional GIS architectures.
It easy to install and to use, it supported by many OS platform and have great document

support on his website, make it the perfect open source web map sever to use.
It also has a constraint for implementing extra feature on a standalone earth client; as user will have to know java scripting language to programmatically control it.

3.6.1 Testing Geoserver

At first need to start GeoServer by going to the Start Menu, and clicking Start GeoServer in the GeoServer folder. Then, to load the Geoserver web admin tool, the following URL is used:

```
http://localhost:8080/geoserver/web/
```



Welcome

Welcome

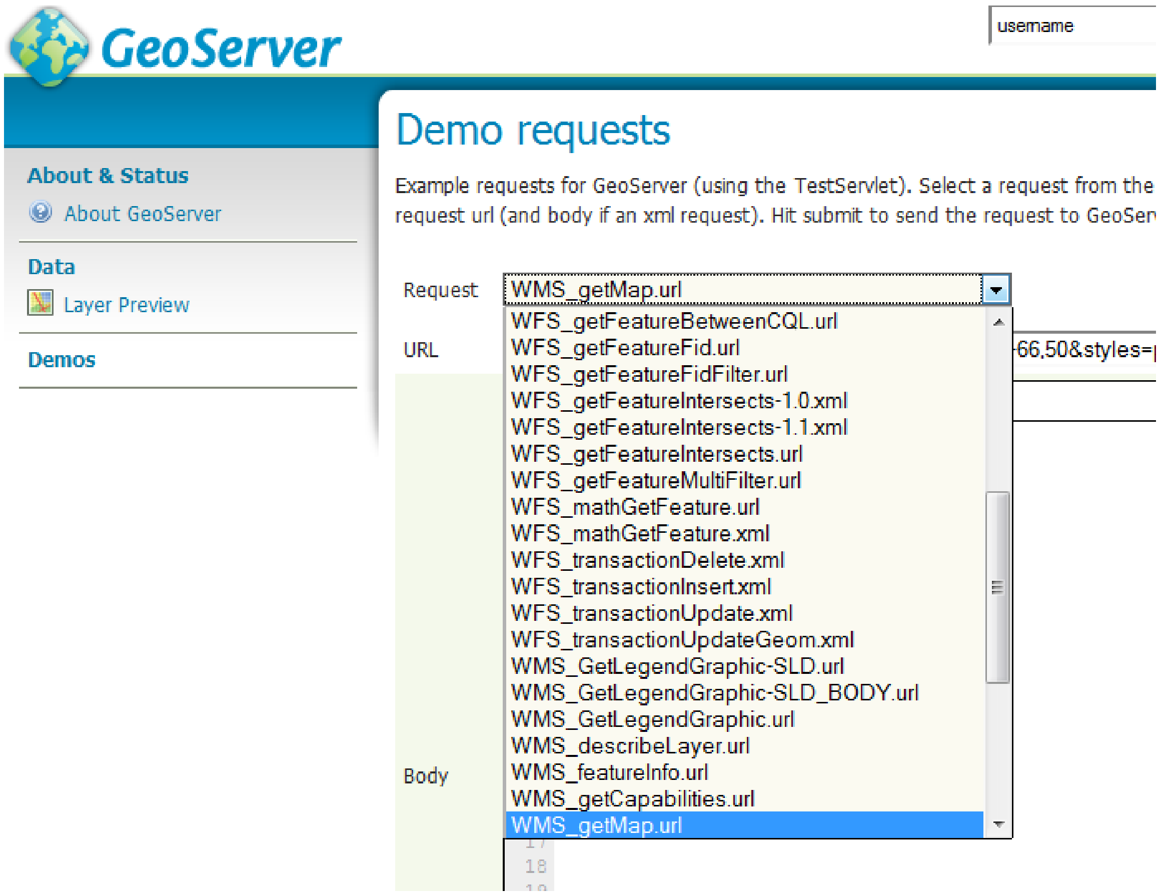
This GeoServer belongs to The ancient geographes INC.

This GeoServer instance is running version **2.1.0.** For more information please contact the administrator.

Figure 3. 2: The Geoserver Web Admin Tool

To run some of the sample requests that can be used to test the installation, the Geoserver demo page was loaded :

http://localhost:8080/geoserver/web/?wicket:bookmarkablePage=:org.geoserver.web.DemoPage



Demo requests

Example requests for GeoServer (using the TestServlet). Select a request from the
request url (and body if an xml request). Hit submit to send the request to GeoSer

WMS_getMap.url

66,50&styles=

WFS_getFeatureBetweenCQL.url
WFS_getFeatureFid.url
WFS_getFeatureFidFilter.url
WFS_getFeatureIntersects-1.0.xml
WFS_getFeatureIntersects-1.1.xml
WFS_getFeatureIntersects.url
WFS_getFeatureMultiFilter.url
WFS_mathGetFeature.url
WFS_mathGetFeature.xml
WFS_transactionDelete.xml
WFS_transactionInsert.xml
WFS_transactionUpdate.xml
WFS_transactionUpdateGeom.xml
WMS_GetLegendGraphic-SLD.url
WMS_GetLegendGraphic-SLD_BODY.url
WMS_GetLegendGraphic.url
WMS_describeLayer.url
WMS_featureInfo.url
WMS_getCapabilities.url
WMS_getMap.url

Figure 3. 3: The GeoServer Demo page

.

The three tests to run are the tests that correspond with the three web map operations that were researched and discussed in chapter 2. The tests are:

WMS_getMap.url
WMS_getCapabilities.url
WMS_featureinfo.url

Test *WMS_getMap.url*
After choosing the `WMS_getMap.url` test and hitting submit, the following result displayed.

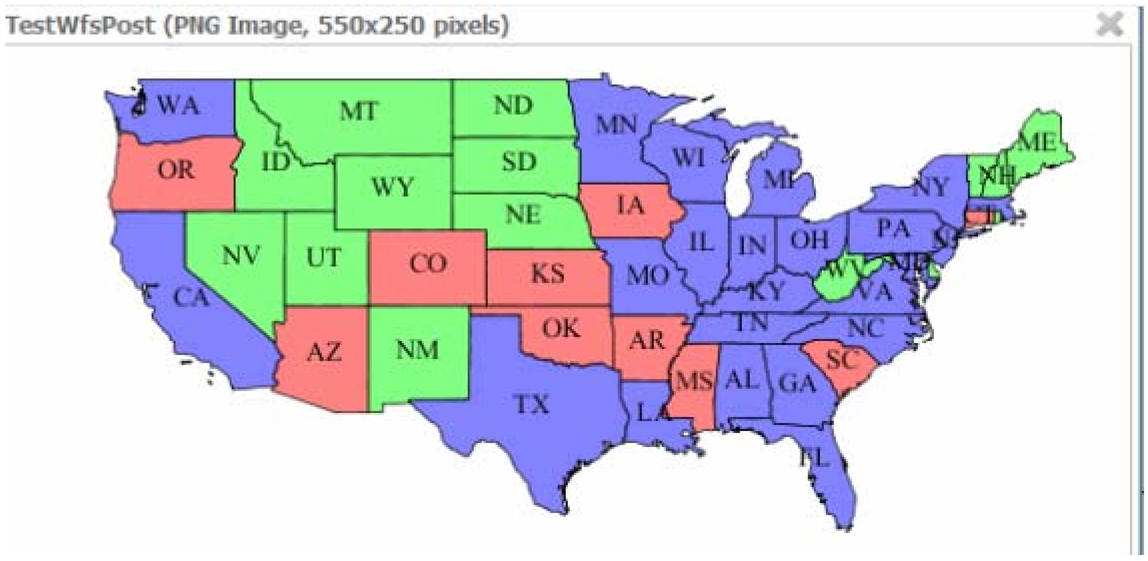
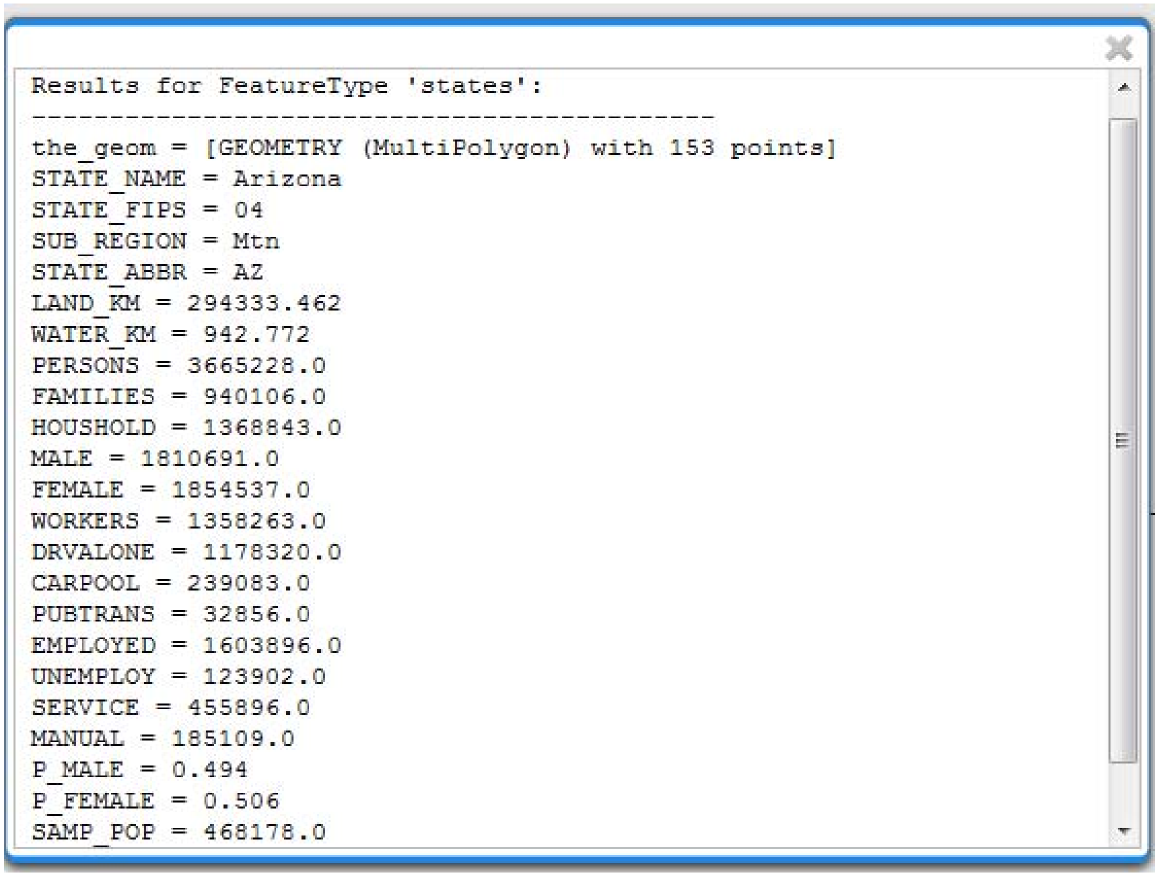

TestWfsPost (PNG Image, 550x250 pixels)

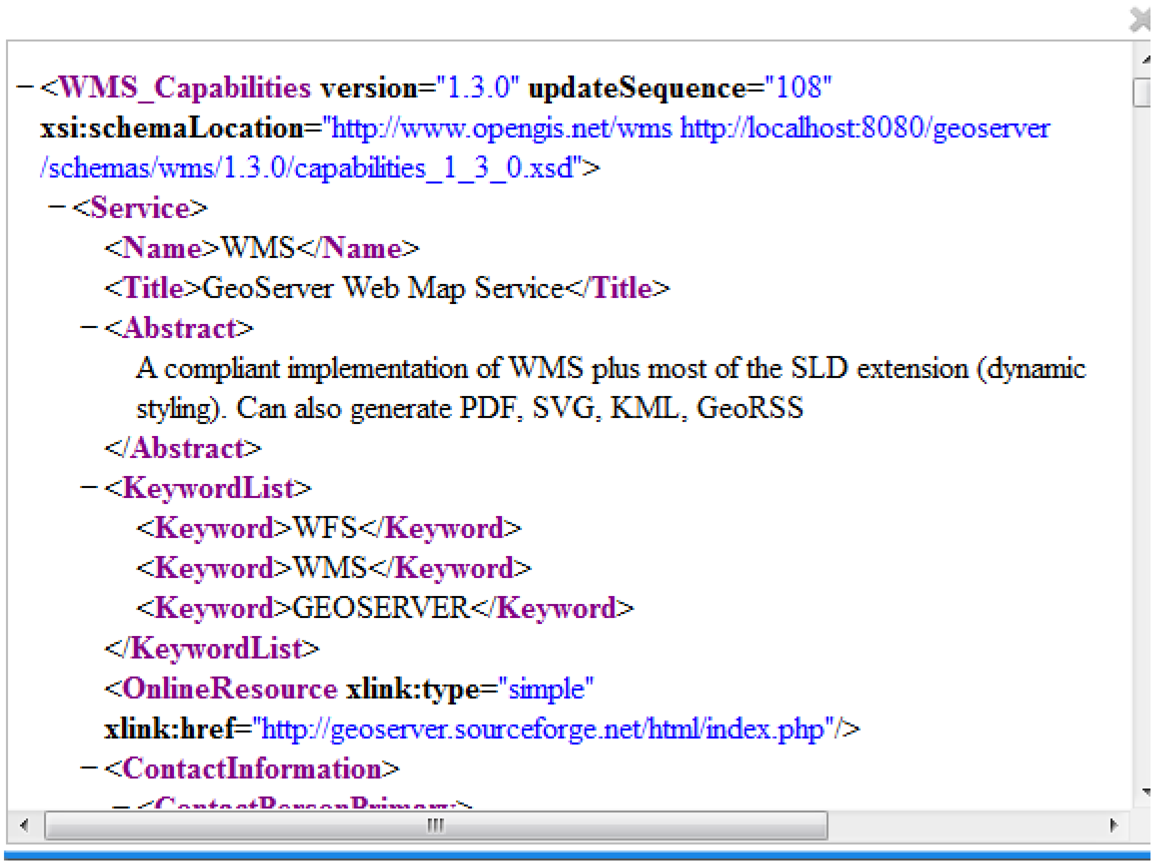
Figure 3. 4: Result of the WMS_getMap.url test

Test WMS_featureinfo.url



```
Results for FeatureType 'states':
--------------------------------------------
the_geom = [GEOMETRY (MultiPolygon) with 153 points]
STATE_NAME = Arizona
STATE_FIPS = 04
SUB_REGION = Mtn
STATE_ABBR = AZ
LAND_KM = 294333.462
WATER_KM = 942.772
PERSONS = 3665228.0
FAMILIES = 940106.0
HOUSHOLD = 1368843.0
MALE = 1810691.0
FEMALE = 1854537.0
WORKERS = 1358263.0
DRVALONE = 1178320.0
CARPOOL = 239083.0
PUBTRANS = 32856.0
EMPLOYED = 1603896.0
UNEMPLOY = 123902.0
SERVICE = 455896.0
MANUAL = 185109.0
P_MALE = 0.494
P_FEMALE = 0.506
SAMP_POP = 468178.0
```

Figure 3. 5: Result of the WMS_getCapabilities.url test



```
- <WMS_Capabilities version="1.3.0" updateSequence="108"
  xsi:schemaLocation="http://www.opengis.net/wms http://localhost:8080/geoserver
  /schemas/wms/1.3.0/capabilities_1_3_0.xsd">
  - <Service>
      <Name>WMS</Name>
      <Title>GeoServer Web Map Service</Title>
    - <Abstract>
        A compliant implementation of WMS plus most of the SLD extension (dynamic
        styling). Can also generate PDF, SVG, KML, GeoRSS
      </Abstract>
    - <KeywordList>
        <Keyword>WFS</Keyword>
        <Keyword>WMS</Keyword>
        <Keyword>GEOSERVER</Keyword>
      </KeywordList>
      <OnlineResource xlink:type="simple"
      xlink:href="http://geoserver.sourceforge.net/html/index.php"/>
    - <ContactInformation>
```

Figure 3. 6: Result of the WMS_getCapabilities.url test

3.7 OpenLayers

The traditional front-end technologies of HTML and CSS are used to generate content along with OpenLayers. OpenLayers is an open source, clientside JavaScript project that allows the connection to any OGC, WMS or WFS web compliant service, such as GeoServer (OpenLayers). According to the Hazzard (2010) openlayer allows to make interactive web maps, viewable in nearly any web browser.Since it is a client side library, it requires no special server side software or settings it can be used without even download anything.

3.7.1 Openlayers and web mapping

OpenLayers is client side mapping library. The figure below is Client/Server model which is the core of how all web applications operate. In the case of a web map application, some sort of map client (e.g.,OpenLayers) communicates with some sort of web map server (e.g., geoserver)

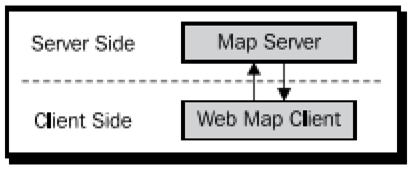


Figure 3.7: Client/Server Model

3.7.1.1 Web map Client

OpenLayer as a client handles to ask a mapserver what a user wants to look at and all this is possible through asynchronous JavaScript (**AJAX**) calls to a map server. OpenLayers sends requests to a map server for map images every time a user interact with the map, then OpenLayers pieces together all the returned map images so it looks like one big, consistent map.

3.7.1.2 Web map Server

A map server (or map service) provides the map itself. There are many different mapserver backends. For example WMS, Google Maps, Yahoo! Maps, ESRI ArcGIS, WFS, and OpenStreet Maps. With OpenLayers, user can use as many different backends for map server. For this project I am using OGC, WMS server. However the basic principle behind all those map service is that they allow to specify the area of the map the user is interested in by sending a request, and then the map servers response by sending the map image. OpenLayer is not a web map server; it only consumes data from them. To work with map server using OpenLayers just supplying a URL in it is enough.

3.8 Summary

There are many different options that can be used for each component of the GIS. However a GIS will always contain an integration of hardware, software, data and procedures.

Chapter 4 Routing Algorithm

There are many options for routing algorithm. This chapter explains the routing algorithm chosen for this project.

4.1 Shortest Path Dijkstra

Dijkstra's algorithm, developed by Dutch computer scientist Edsger Dijkstra in 1959 is often used in routing. It is a graph search algorithm that solves the single-source shortest path problem for a graph with non negative edge path costs, outputting a shortest path tree (Sutskever, 2008).

A path from a source vertex v to a target vertex u is said to be the shortest path if its total cost is minimum among all v-to-u paths. Dijkstra's algorithm is based on the following assumptions (Chen,2003):

- All edge costs are non-negative.
- The number of vertices is finite.
- The source is a single vertex, but the target may be all other vertices.

4.2 Principle of the algorithm

The graph is made of two entities *vertices* or nodes, and *edges* which link vertices together. Edges are directed and have an associated *distance*, sometimes called the weight or the cost. The distance between the vertex *u* and the vertex *v* is noted [*u*, *v*] which is shown in figure 3.1and is always positive (Renaud Waldura,2007) .

Dijkstra's algorithm defines vertices in two distinct sets, the set of *unsettled* vertices and the set of *settled* vertices. A vertex is considered settled, and moved from the unsettled set to the settled set, once its shortest distance from the source has been found. Initially all vertices are unsettled, and the algorithm ends once all vertices are in the settled set.

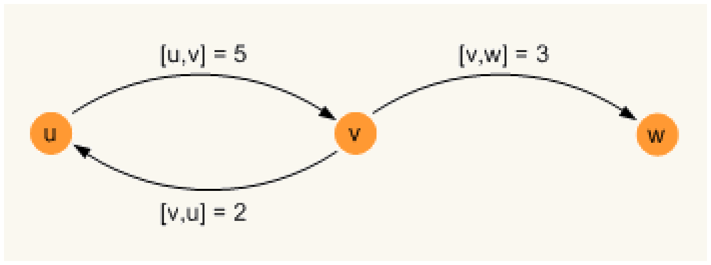


Figure 4. 1: Dijkstra shortest path algorithm example

4.3 Example

According to Wladura (2007) the following data structures are used for this algorithm.

d stores the best estimate of the shortest distance from the source to each vertex.

π stores the predecessor of each vertex on the shortest path from the source.

S the set of settled vertices, the vertices whose shortest distances from the source have been found

Q the set of unsettled vertices

Figure 4.2 shows the graph of Dijkstra shortest path algorithm starting at the source vertex a.

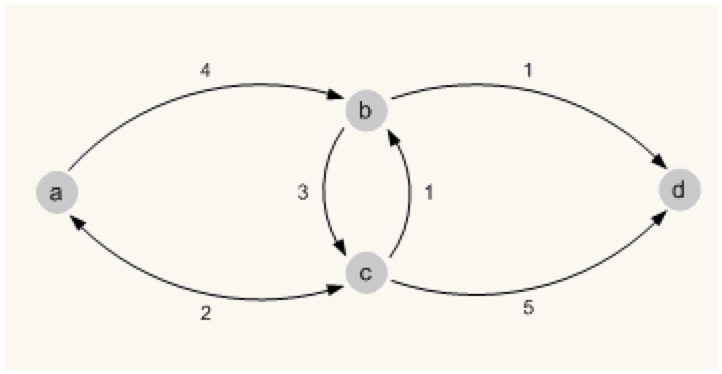


Figure 4. 2 Initialisation in Dijkstra shortest path algorithm with source vertex a

First iteration is done by adding source vertex *a* to the set *Q*. *Q* isn't empty, its minimum is extracted, *a* again. Then *a* is added to *S* and its neighbour is relaxed.

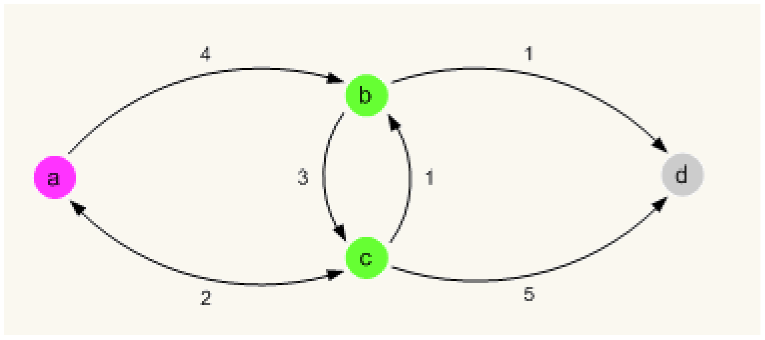


Figure 4. 3: First Iteration

Vertices adjacent to *a*, are *b* and *c* (in green in figure 3.3). At first the best distance estimate from *a* to *b* is computed. *d*(*b*) was initialized to infinity, therefore the calculation will be :

```
d(b) = d(a) + [a,b] = 0 + 4 = 4
```

$\pi(b)$ is set to *a*, and *b* is added to *Q*. Similarily for *c*, *d*(*c*) is assigned to 2, and $\pi(c)$ to *a*. Now *Q* contains *b* and *c*. As seen in figure 3.3, *c* is the vertex with the current shortest distance of 2. It is extracted from the queue and added to *S*, the set of settled nodes. Then the neighbours of the c, a, b and d are relaxed.

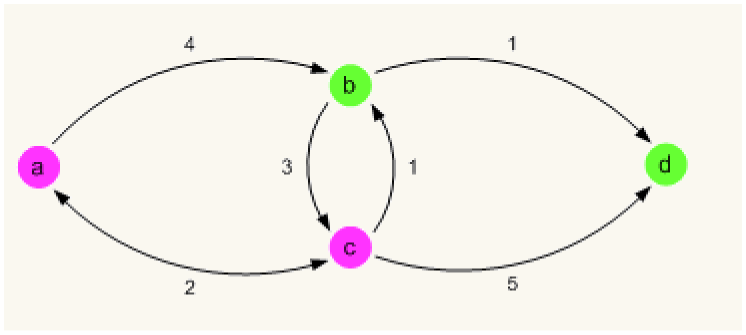


Figure 4. 4: Second Iteration

a is ignored because it is found in the settled set.The first pass of the algorithm had concluded that the shortest path from *a* to *b* was direct. But when looking at *c*'s neighbor *b*, shorter path going through c exits between a and b i.e.

```
d(b) = 4 > d(c) + [c,b] = 2 + 1 = 3
```

d(*b*) is updated to 3, and $\pi(b)$ updated to *c*. *b* is added again to *Q*. The next adjacent vertex is *d*, which haven't covered yet. *d*(*d*) is set to 7 and $\pi(d)$ to *c*.

The unsettled vertex with the shortest distance is extracted from the queue, it is now *b*. It is added to the settled set and its neighbors c and d are relaxed.

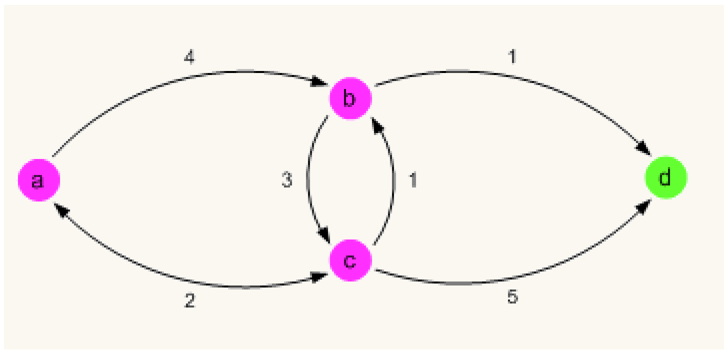


Figure 4. 5: Third Iteration

c is skipped because it has already been settled. But a shorter path is found for *d*:

```
d(d) = 7 > d(b) + [b,d] = 3 + 1 = 4
```

Therefore *d*(*d*) is updated to 4 and *π*(*d*) to *b*. Then *d* is added to the Q set.

At this point the only vertex left in the unsettled set is *d*, and all its neighbors are settled. The algorithm ends. The final results are displayed in red below in figure 3.6:

- *π* - the shortest path, in predecessor fashion
- *d* - the shortest distance from the source for each vertex

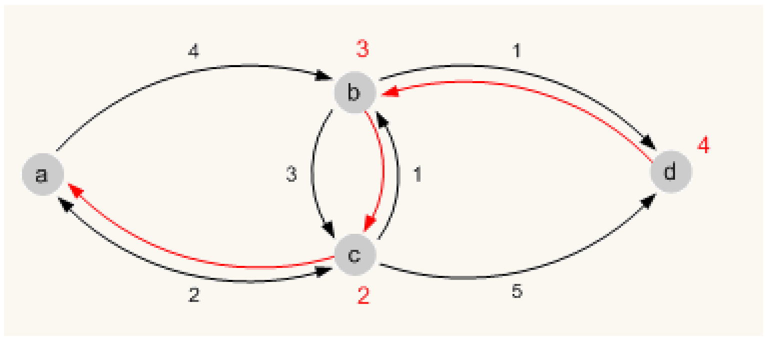


Figure 4. 6: Fourth Iteration

4.4 UML for Dijkstra’s shortest path algorithm

Figure 4.7(a) and 4.7(b) corresponds the UML class diagrm for dijkstra’s shortest path algorithm. UML class diagrams are used to describe the static view of an application (Rumbaugh et al. 1999): the main constituents are classes and their relationships. A class is a description of a concept, and may have attributes and operations associated with it (Purchase et al. 2001).

Here the class diagram are represented with boxes which contain three parts

- The upper part holds the name of the class.
- The middle part contains the attributes of the class.
- The bottom part gives the methods or operations the class can take or undertake.

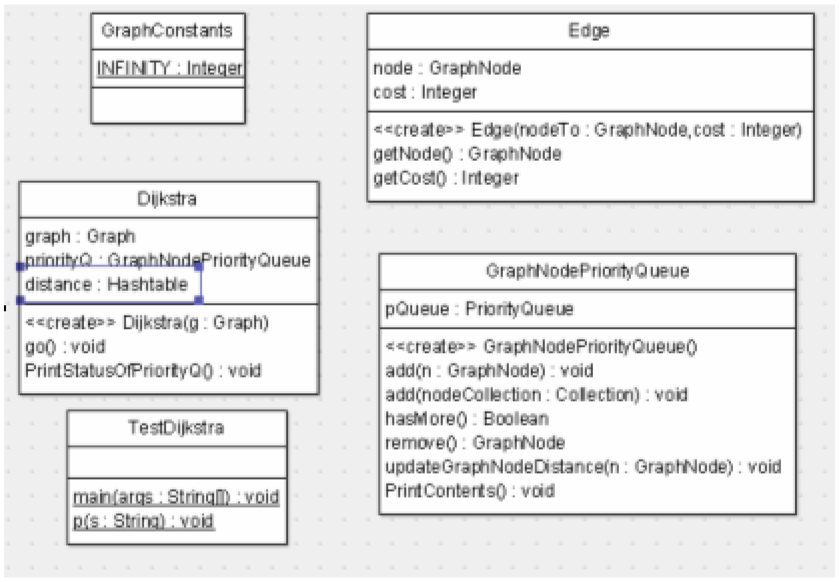

GraphConstants
INFINITY : Integer
Edge
node : GraphNode
cost : Integer
<<create>> Edge(nodeTo : GraphNode,cost : Integer)
getNode() : GraphNode
getCost() : Integer
Dijkstra
graph : Graph
priorityQ : GraphNodePriorityQueue
distance : Hashtable
<<create>> Dijkstra(g : Graph)
go() : void
PrintStatusOfPriorityQ() : void
GraphNodePriorityQueue
pQueue : PriorityQueue
<<create>> GraphNodePriorityQueue()
add(n : GraphNode) : void
add(nodeCollection : Collection) : void
hasMore() : Boolean
remove() : GraphNode
updateGraphNodeDistance(n : GraphNode) : void
PrintContents() : void
TestDijkstra
main(args : String[]) : void
p(s : String) : void

Figure 4. 7(a) UML for Dijkstra’s shortest path algorithm(Lecture 6,2011)

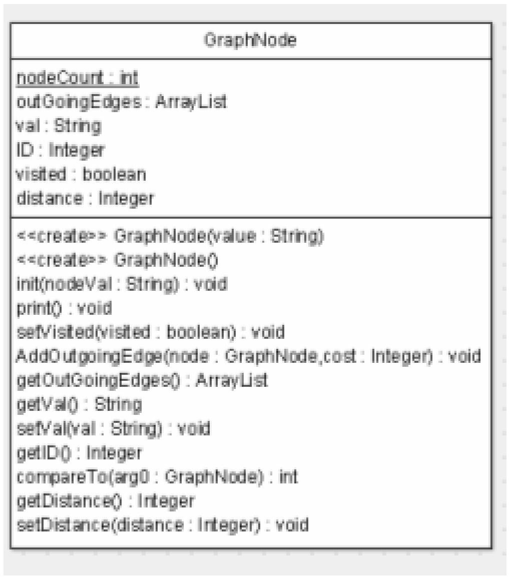

GraphNode
nodeCount : int
outGoingEdges : ArrayList
val : String
ID : Integer
visited : boolean
distance : Integer
<<create>> GraphNode(value : String)
<<create>> GraphNode()
init(nodeVal : String) : void
print() : void
setVisited(visited : boolean) : void
AddOutgoingEdge(node : GraphNode,cost : Integer) : void
getOutGoingEdges() : ArrayList
getVal() : String
setVal(val : String) : void
getID() : Integer
compareTo(arg0 : GraphNode) : int
getDistance() : Integer
setDistance(distance : Integer) : void

Figure 4. 7(b) UML for Dijkstra's shortest path algorithm(Lecture 6,2011).

4.5 Function

The Dijkstra shortest path algorithm is implemented as shown in Listing 3.1.The shortest path function has the following declaration (pgRouting, 2011).

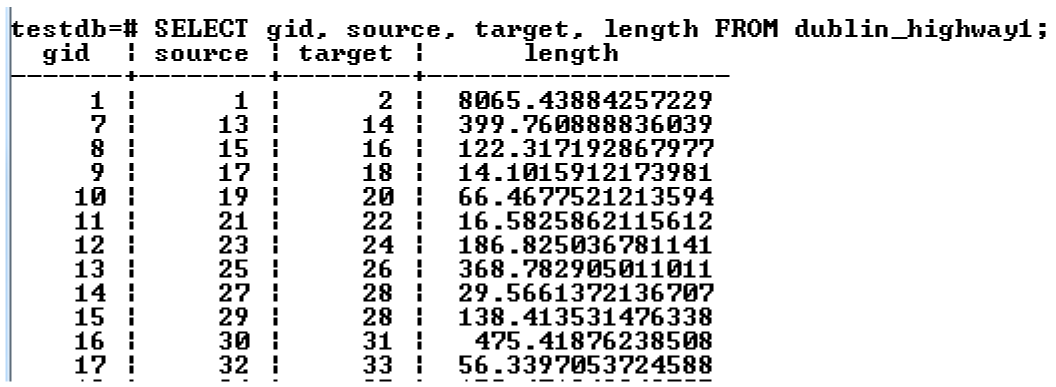
```
CREATE OR REPLACE FUNCTION shortest_path(
                                                sql text,
                                                source_id integer,
                                                target_id integer,
                                                directed boolean,
                                                has_reverse_cost
boolean)
        RETURNS SETOF path_result
```

Listing 4. 1:Shortest path function declaration

4.6 Arguments (Input)

A sql query should return the set of the column with the following columns(pgRouting, 2011).

SQL: `SELECT gid, source, target, length FROM dublin_highway1;`



```
testdb=# SELECT gid, source, target, length FROM dublin_highway1;
  gid  | source | target |       length
-------+--------+--------+--------------------
     1 |      1 |      2 |  8065.43884257229
     7 |     13 |     14 |  399.760888836039
     8 |     15 |     16 |  122.317192867977
     9 |     17 |     18 |  14.1015912173981
    10 |     19 |     20 |  66.4677521213594
    11 |     21 |     22 |  16.5825862115612
    12 |     23 |     24 |  186.825036781141
    13 |     25 |     26 |  368.782905011011
    14 |     27 |     28 |  29.5661372136707
    15 |     29 |     28 |  138.413531476338
    16 |     30 |     31 |   475.41876238508
    17 |     32 |     33 |  56.3397053724588
```

Listing 4. 2: Verfying all of the arguments from shortest_path function are displayed as a result of query from road data dublin_highway1

- id: an int identifier of the edge
- source: an int identifier of the source vertex
- target: an int identifier of the target vertex
- cost: an float8 value, of the edge traversal cost. (a negative cost will prevent the edge from being inserted in the graph).
- reverse_cost (optional): the cost for the reverse traversal of the edge. This is only used when the directed and has_reverse_cost parameters are true (see the above remark about negative costs).

source_id: int id of the start point

directed: true if the graph is directed

has_reverse_cost: if true, the reverse_cost column of the SQL generated set of rows will be used for the cost of the traversal of the edge in the opposite direction.

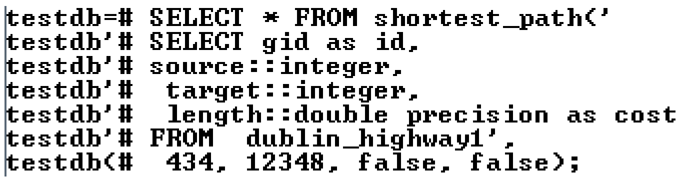
4.6 Output

The function returns a set of rows. There is one row for each crossed edge, and an additional one containing the terminal vertex. The columns of each row are(pgRouting, 2011):

- vertex_id: the identifier of source vertex of each edge. There is one more row after the last edge, which contains the vertex identifier of the target path.
- edge_id: the identifier of the edge crossed
- cost: The cost associated to the current edge. It is 0 for the row after the last edge. Thus, the path total cost can be computated using a sum of all rows in the cost column.

4.7 Query examples

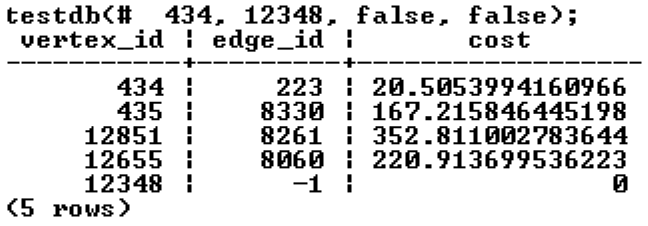
This query examples is based on the open street map road data named dublin_highway1.



```
testdb=# SELECT * FROM shortest_path('
testdb'# SELECT gid as id,
testdb'# source::integer,
testdb'#  target::integer,
testdb'#  length::double precision as cost
testdb'# FROM  dublin_highway1',
testdb(#  434, 12348, false, false);
```

Listing 4. 3: Verifying shortest_path function query successfully runs in psql

Output from the shortest_path function query is shown in listing 4.4



```
testdb(#  434, 12348, false, false);
 vertex_id | edge_id |        cost
-----------+---------+--------------------
       434 |     223 | 20.5053994160966
       435 |    8330 | 167.215846445198
     12851 |    8261 | 352.811002783644
     12655 |    8060 | 220.913699536223
     12348 |      -1 |                  0
(5 rows)
```

Listing 4. 4: Output from the shortest_path function query

4.8 Summary

This project uses Dijkstra shortest path algorithm. Dijkstra shortest path algorithm slove the single source shortest path algorithm for a graph with non negative edge path costs. So this project also only returns the shortest path from the single source route.

Chapter 5 Loading data , setting data for routing and putting postgis data into Geoserver

This chapter explains about how the data was loaded and stored in spatial database which is a part of procedures in GIS and the system integration between PostgreSQL/PostGIS ,openJump, Geoserver and openlayer. The figure 5.1 shows the overview of system integration.

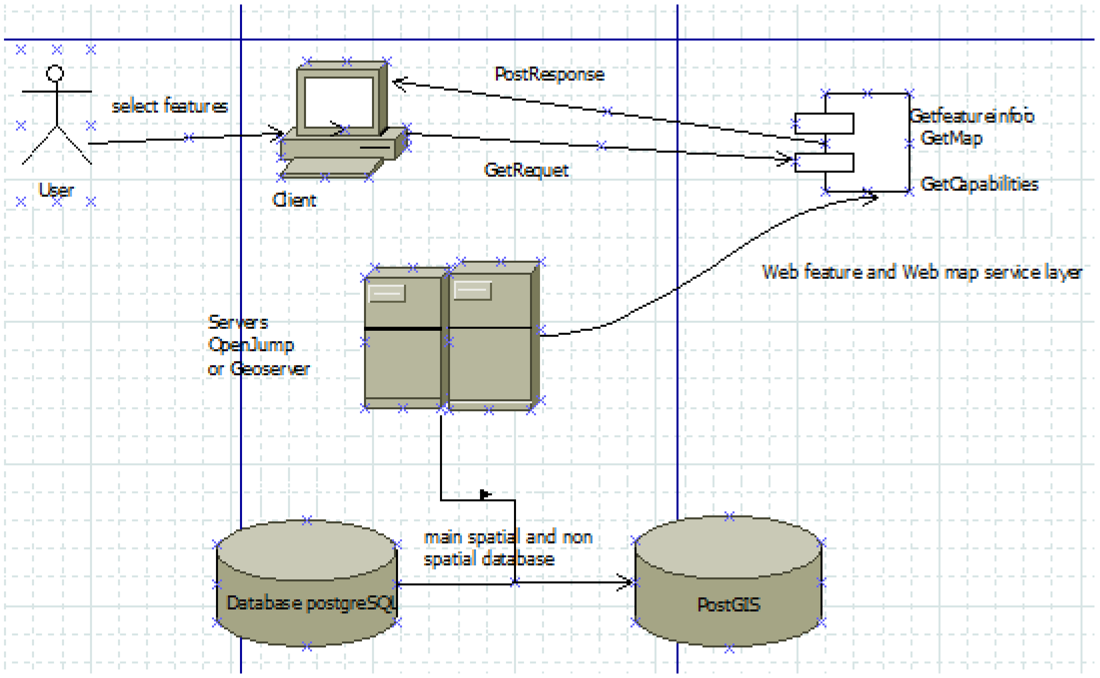


Figure 5. 1:Overview of system integration

5.1 Loading Data

The streets use OSM data from CloudMade.The OSM data was transformed to Irish National Grid (IGN).The buildings are a subset of the Geodirectory from An Post originally in IGN. The road data and buildings data are loaded to postgresql using “psql” SQL terminal monitor(Postgis manual) from the command prompt.

```
psql -d testdb -U postgres -f dublin_highway1.sql
```

```
psql -d testdb -U postgres -f buildings_geodir.sql
```

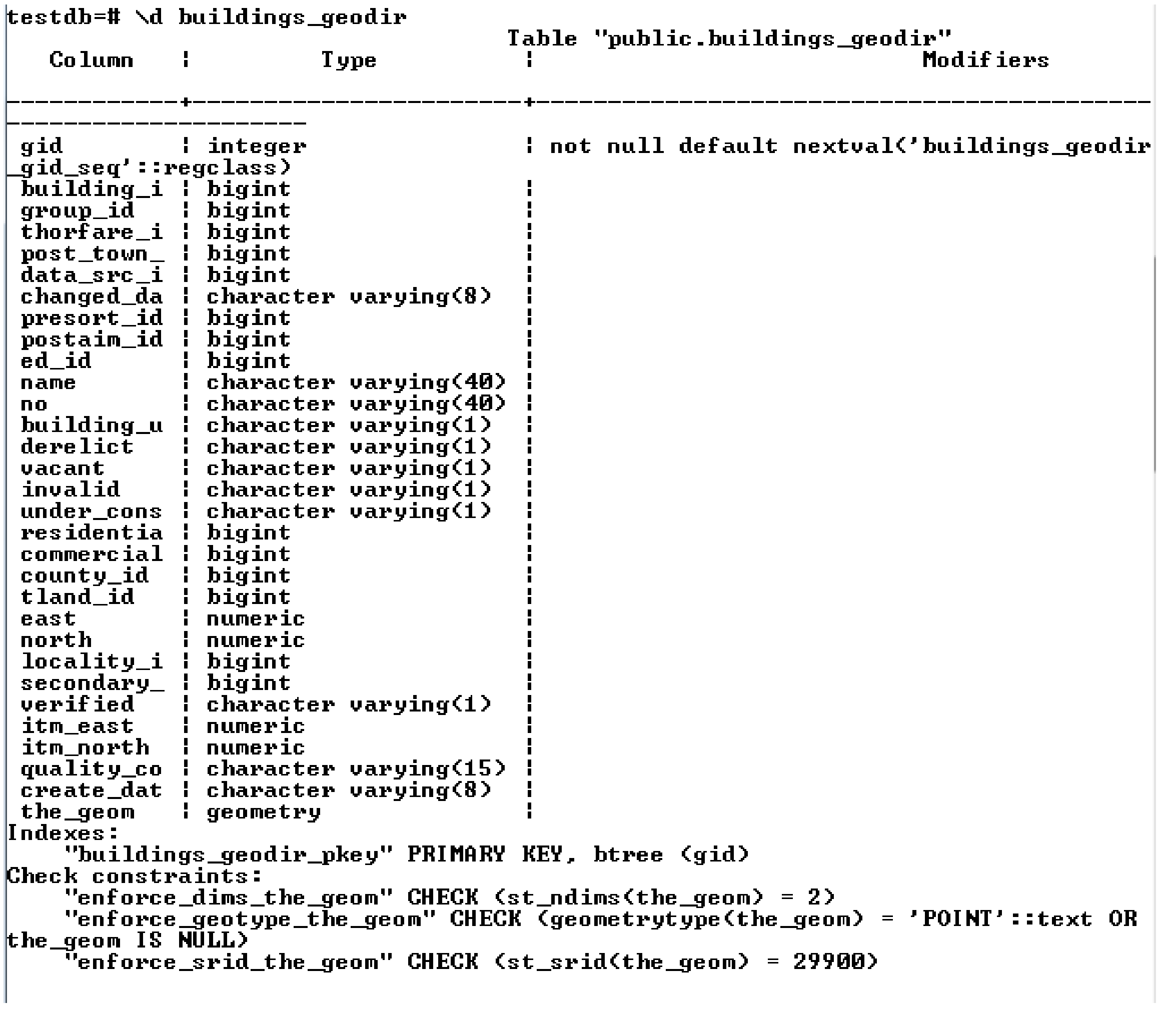
`-d` name of the database

`-U (user)`

The username to use when connecting to database

```
-f (file name)
```

The table named buildings_geodir contains the following attributes



```
testdb=# \d buildings_geodir
                                   Table "public.buildings_geodir"
   Column    |         Type          |                       Modifiers

------------+-----------------------+-----------------------------------------
---------------------
 gid         | integer               | not null default nextval('buildings_geodir
_gid_seq'::regclass)
 building_i | bigint                |
 group_id    | bigint                |
 thorfare_i | bigint                |
 post_town_ | bigint                |
 data_src_i | bigint                |
 changed_da | character varying(8)  |
 presort_id | bigint                |
 postaim_id | bigint                |
 ed_id       | bigint                |
 name        | character varying(40) |
 no          | character varying(40) |
 building_u | character varying(1)  |
 derelict    | character varying(1)  |
 vacant      | character varying(1)  |
 invalid     | character varying(1)  |
 under_cons | character varying(1)  |
 residentia | bigint                |
 commercial | bigint                |
 county_id  | bigint                |
 tland_id    | bigint                |
 east        | numeric               |
 north       | numeric               |
 locality_i | bigint                |
 secondary_ | bigint                |
 verified    | character varying(1)  |
 itm_east    | numeric               |
 itm_north  | numeric               |
 quality_co | character varying(15) |
 create_dat | character varying(8)  |
 the_geom    | geometry              |
Indexes:
    "buildings_geodir_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_the_geom" CHECK (st_ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'POINT'::text OR
the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (st_srid(the_geom) = 29900)
```

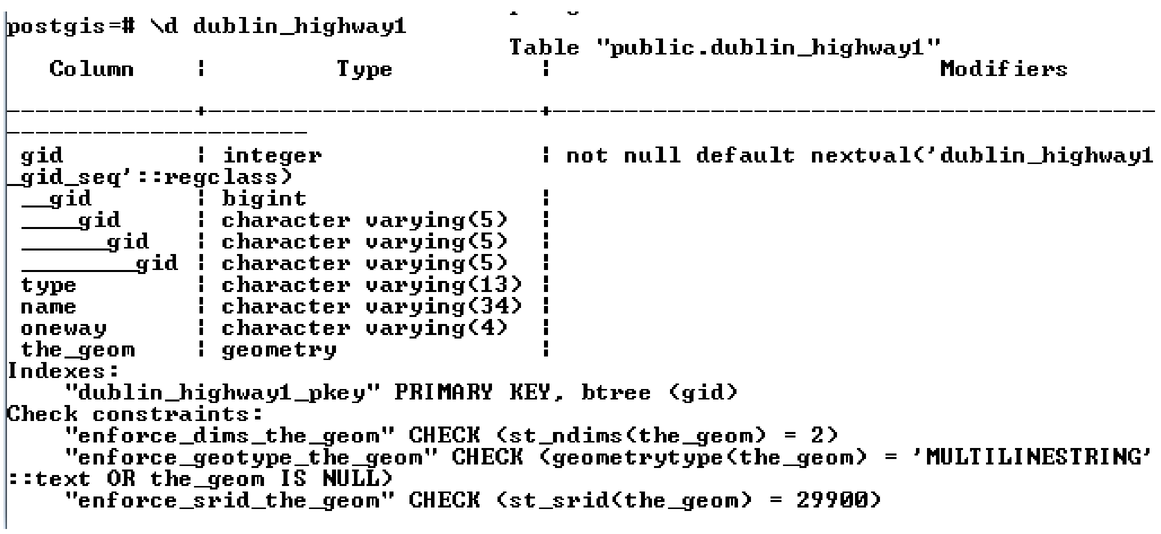
Listing 5. 1: Verifying successful data imports to postgis

Listing 5.1 shows the attributes in the buildings_geodir table. Each column is listed and shows the datatype that each column can hold. PostgreSQL database can handle all these types without PostGIS , except for the geometry data.

Only one column in the table contains geometry data and can be seen in the listing that it is of type point geometry: the the_geom column has geometry listed as its type:

The_geom| geometry

The table named dublin_highway1 containing road network data has the following attributes



```
postgis=# \d dublin_highway1
                                  Table "public.dublin_highway1"
   Column    |         Type          |                       Modifiers

-------------+-----------------------+----------------------------------------
---------------------
 gid         | integer               | not null default nextval('dublin_highway1
_gid_seq'::regclass)
 __gid       | bigint                |
 ____gid     | character varying(5)  |
 ______gid   | character varying(5)  |
 ________gid | character varying(5)  |
 type        | character varying(13) |
 name        | character varying(34) |
 oneway      | character varying(4)  |
 the_geom    | geometry              |
Indexes:
    "dublin_highway1_pkey" PRIMARY KEY, btree (gid)
Check constraints:
    "enforce_dims_the_geom" CHECK (st_ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'MULTILINESTRING'
::text OR the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (st_srid(the_geom) = 29900)
```

Listing 5. 2: Verifying successful data imports to postgis

Listing 5.2 shows the attributes in the dublin_highway1 table. Each column is listed and shows the datatype that each column can hold.PostgreSQL database can handle all these types without PostGIS , except for the geometry data.

Only one column in the table contains geometry data and can be seen in the listing that it is of type MULTLINESTRING geometry: the the_geom column has geometry listed as its type:

The_geom| geometry

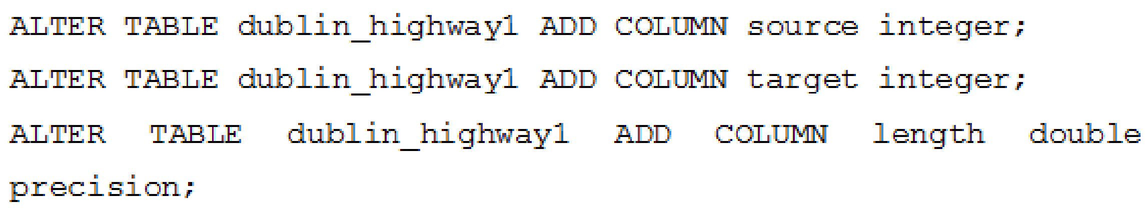
The network data dublin_highway1 provides the following information

- Road link id(gid)
- Road type(for ex primary, secondary,tertiary)
- Road name(name)
- Road geometry(the_geom)

However this data allows to display the road data as a PostGIS layer for example in openJump which I am using for this project. But for the routing function to work the network data should contain a network topology information. The steps for creating network topology is explained in section 5.2

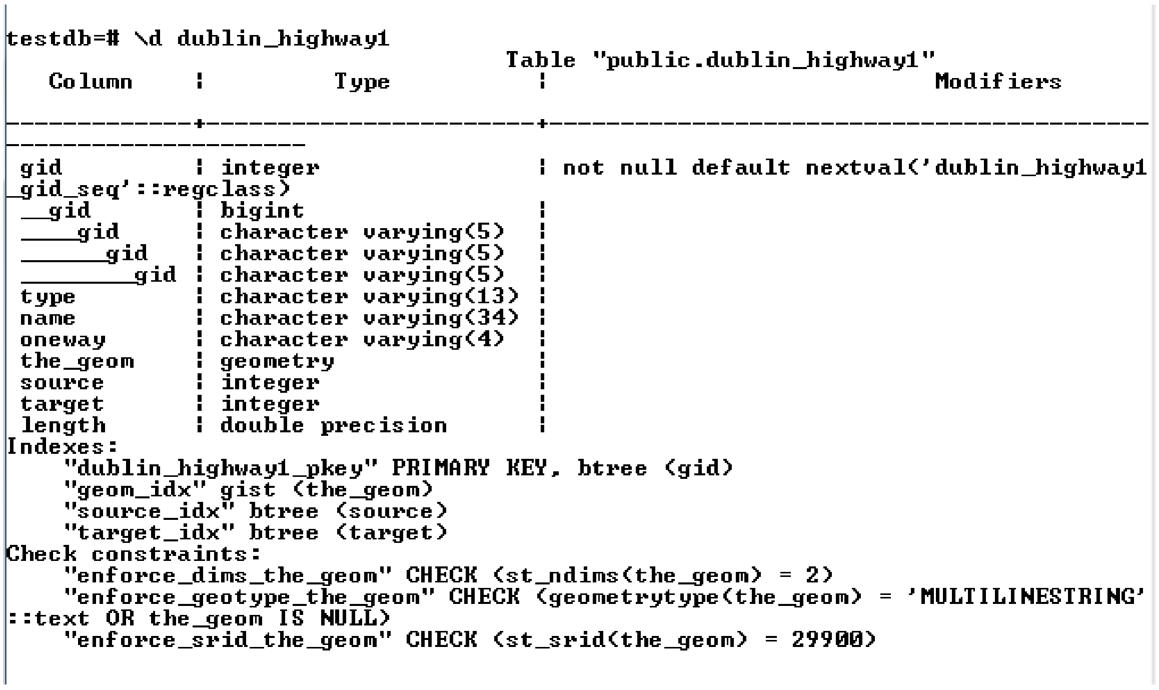
5.2 Setting data for pgrouting

After the data is imported in spatial database usually requires one more step for pgRouting to work. For pgRouting to work edges table should contain the network topology or connectivity information data should provide the network topology which consists of links to source and target id each. So first source, target and length column is added as shown in listing 5.3 and then the assign_vertex_id function is ran. This function assigns a source and a target ID to each link and it can “snap” nearby vertices within a certain tolerance.



```
ALTER TABLE dublin_highway1 ADD COLUMN source integer;
ALTER TABLE dublin_highway1 ADD COLUMN target integer;
ALTER TABLE dublin_highway1 ADD COLUMN length double
precision;
```

Listing 5. 3: shows that the table was successfully altered. Now road data contains source , target and length column



```
testdb=# \d dublin_highway1
                                   Table "public.dublin_highway1"
   Column     |          Type          |                            Modifiers

-------------+------------------------+------------------------------------------
----------------------
 gid          | integer                | not null default nextval('dublin_highway1
_gid_seq'::regclass)
 __gid        | bigint                 |
 ____gid      | character varying(5)   |
 ______gid    | character varying(5)   |
 ________gid  | character varying(5)   |
 type         | character varying(13)  |
 name         | character varying(34)  |
 oneway       | character varying(4)   |
 the_geom     | geometry               |
 source       | integer                |
 target       | integer                |
 length       | double precision       |
Indexes:
    "dublin_highway1_pkey" PRIMARY KEY, btree (gid)
    "geom_idx" gist (the_geom)
    "source_idx" btree (source)
    "target_idx" btree (target)
Check constraints:
    "enforce_dims_the_geom" CHECK (st_ndims(the_geom) = 2)
    "enforce_geotype_the_geom" CHECK (geometrytype(the_geom) = 'MULTILINESTRING'
::text OR the_geom IS NULL)
    "enforce_srid_the_geom" CHECK (st_srid(the_geom) = 29900)
```

Listing 5. 4: Verfying table dublin_highway1 was successfully altered adding source, target and length column

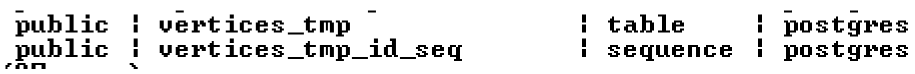
After this step `assign_vertex_id` function was ran to create network topology in dublin_highway1 table containing road data. Listing 5.5 shows that the `assign_vertex_id` was successfully run to create network topology(connectivity information).

```
testdb=#   select   assign_vertex_id('dublin_highway1',   cast   (1.5   as
float), 'the_ge

om', 'gid');

NOTICE:         CREATE    TABLE    will    create    implicit    sequence
"vertices_tmp_id_seq" for se

rial column "vertices_tmp.id"

CONTEXT:  SQL statement "CREATE TABLE vertices_tmp (id serial)"

PL/pgSQL function "assign_vertex_id" line 14 at EXECUTE statement

 assign_vertex_id

------------------

 OK

(1 row)
```

Listing 5. 5: Verfying the network topology was successfully created in psql

For dublin_highway1 table

After these steps testdb database looks like this:



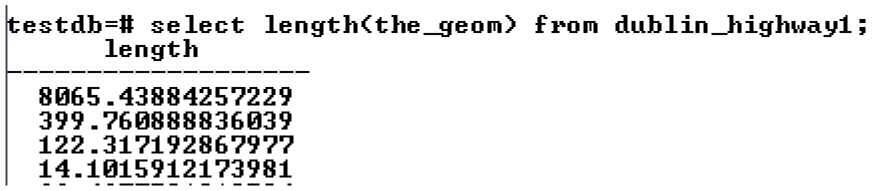
```
 public | vertices_tmp                 | table    | postgres
 public | vertices_tmp_id_seq          | sequence | postgres
```

Listing 5. 6: Verifying the network topology was successfully created

The following SQL command populates the “length” field which will be used as the edge cost in the network topology.

```
      UPDATE dublin_highway1 SET length = length(the_geom);
```

```
Test the lengths have been calculated
```

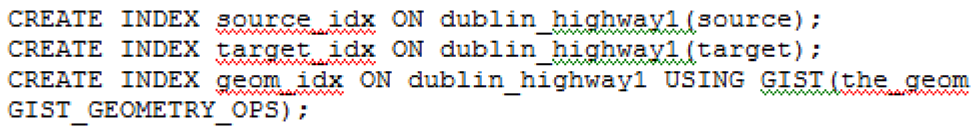


```
testdb=# select length(the_geom) from dublin_highway1;
      length
--------------------
  8065.43884257229
  399.760888836039
  122.317192867977
  14.1015912173981
```

Listing 5. 7: Verifying the lengths column has updated successfully

5.2.1 Add indices

Adding indices speeds up the data access so it improves the query performance. This is done using the CREATE INDEX command . The command to create an index for the geometry in the dublin_highway1 table looks like this.

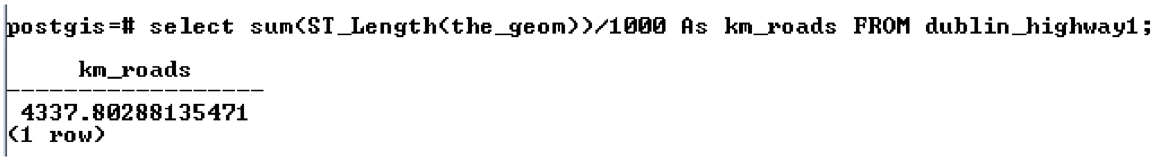


```
CREATE INDEX source_idx ON dublin_highway1(source);
CREATE INDEX target_idx ON dublin_highway1(target);
CREATE INDEX geom_idx ON dublin_highway1 USING GIST(the_geom
GIST_GEOMETRY_OPS);
```

Listing 5. 8 Command to create index for the geometry

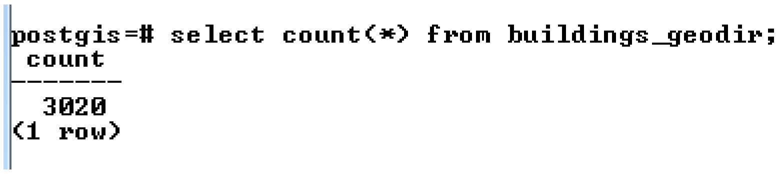
The source_idx, target_idx and geom._idx is used for the name of the index. The second parameter dublin_highway1 is the name of the table for which indexes are being created. The field that hold geometry data is identified as the_geom. The rest of the command consists of keywords needed for indexing functions to run and must be used as shown (Mitchell, 2005).

5.3 Example of Spatial SQL queries



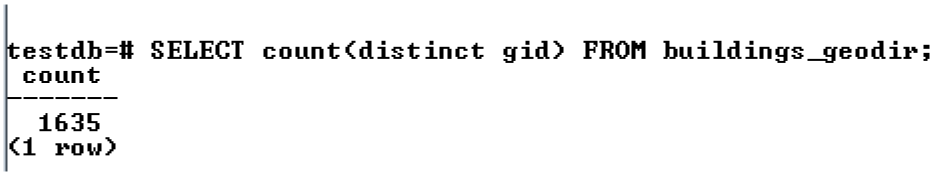
```
postgis=# select sum(ST_Length(the_geom))/1000 As km_roads FROM dublin_highway1;
     km_roads
------------------
 4337.80288135471
(1 row)
```

This spatial query is selecting the total length of the roads in dublin_highway1 table, expressed in kilometres.



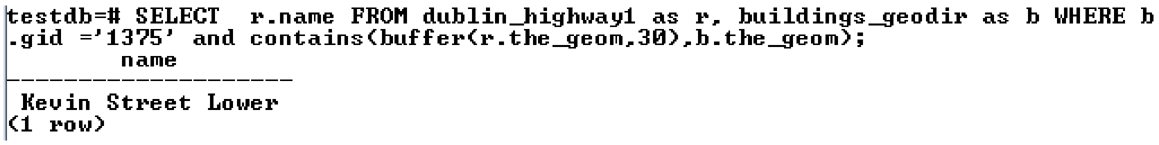
```
postgis=# select count(*) from buildings_geodir;
 count
-------
  3020
(1 row)
```

This query reports the number of features in the buildings_geodir table



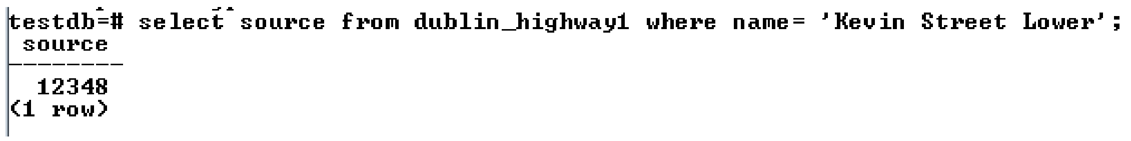
```
testdb=# SELECT count(distinct gid) FROM buildings_geodir;
 count
-------
  1635
(1 row)
```

This query counts the total number of distinct buildings in buildings_geodir table.



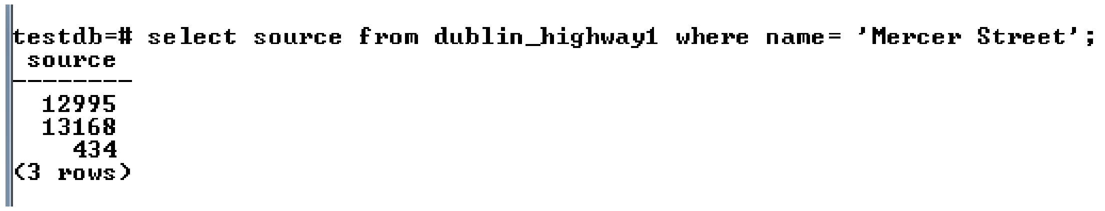
```
testdb=# SELECT  r.name FROM dublin_highway1 as r, buildings_geodir as b WHERE b
.gid ='1375' and contains(buffer(r.the_geom,30),b.the_geom);
        name
--------------------
 Kevin Street Lower
(1 row)
```

This query is selecting the road name in dublin_highway1 table which is within 30 m distance from the buildings with gid ‘1375’ in buildings_geodir table. So two table are joined together pair at time using topology and set comparison operator ‘`contains`’ and spatial analysis operator ‘`buffer`’.



```
testdb=# select source from dublin_highway1 where name= 'Kevin Street Lower';
 source
--------
  12348
(1 row)
```

This query is selecting the source of the road named ‘Kevin Street Lower’ in dublin_highway1 table



```
testdb=# select source from dublin_highway1 where name= 'Mercer Street';
 source
--------
  12995
  13168
    434
(3 rows)
```

This query is selecting the source of the road name ‘Mercer Street’ in dublin_highway1 table which returns three source as an output

5.4 Putting PostGIS data into Geoserver

There is a certain order to which data must be loaded to GeoServer. First a workspace must be created, then the data store and finally the individual layer must be loaded.

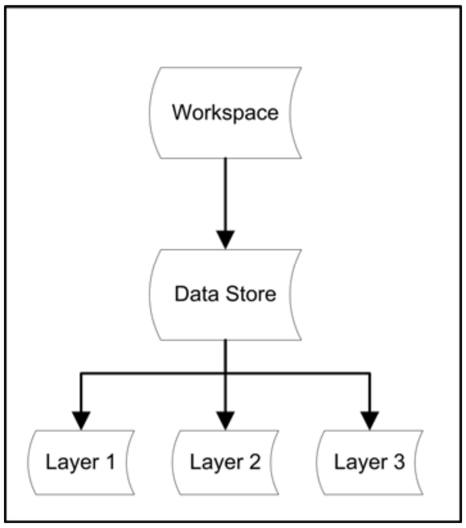


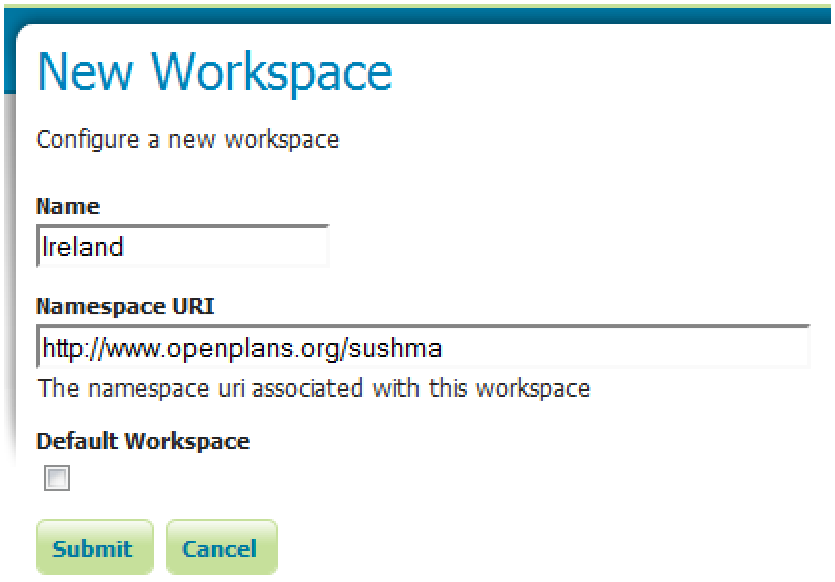
Figure 5. 2: Geoserver datastorage

Step1: Login to GeoServer

The defaults are username=admin, password=geoserver.

Step2: Create Workspace

Workspaces is a container which is used to organise layer. A Workspaces consists of a name and a Namespace URI In GeoServer, a workspace is often used to group similar layers together. Individual layers are often referred to by their workspace name, colon, then store. (Ex: topp:states) Two different layers having the same name can exist as long as they exist in different workspaces. (Ex: sf:states, topp:states) (Geoserver,2009).



New Workspace

Configure a new workspace

Name

Ireland

Namespace URI

http://www.openplans.org/sushma

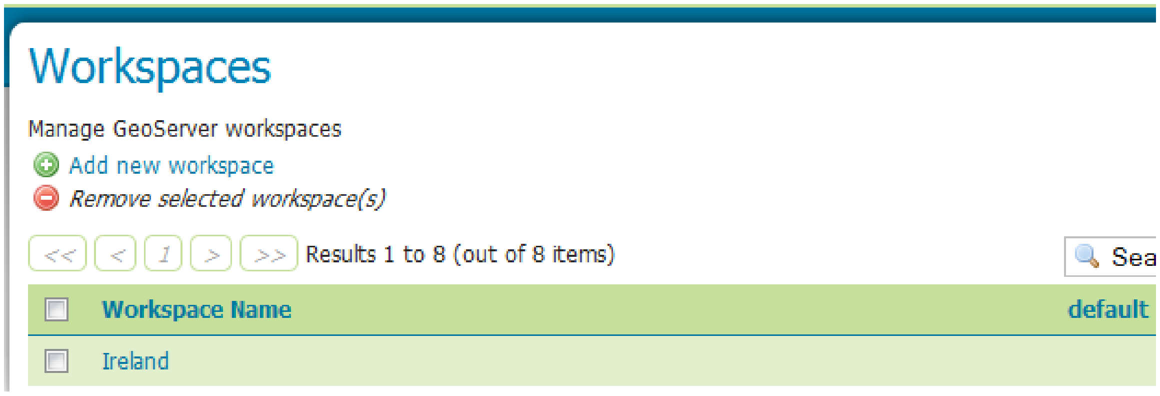
The namespace uri associated with this workspace

Default Workspace

☐

Figure 5. 3Adding new workspace Ireland

After clicking submit the newly created workspace appears as follows:



Workspaces

Manage GeoServer workspaces

Results 1 to 8 (out of 8 items)

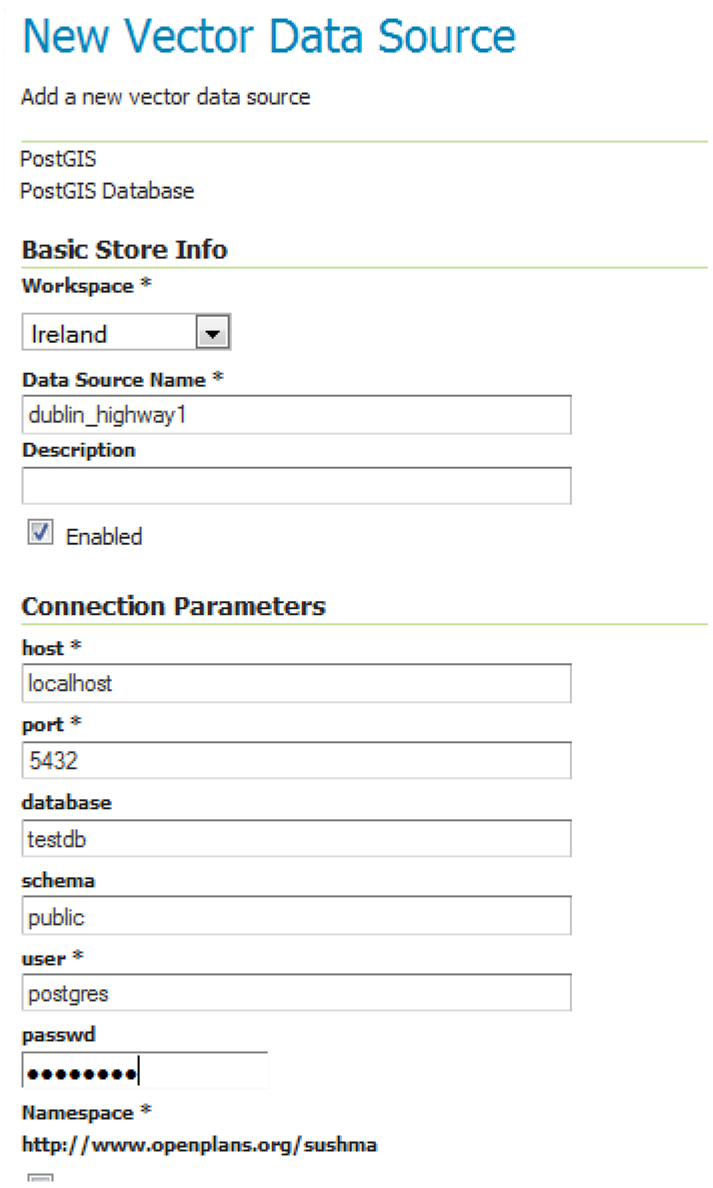
	Workspace Name	default
☐	Ireland	

Figure 5. 4: Workspace view

Step 3: Add new store

A store connects to a data source that contains raster or vector data. A data source can be a file or group of files such as a table in a database, a single file (such as a shapefile), or a directory (such as Vector Product Format library). The store construct is used so that connection parameters are defined once, rather than for each piece of data in a source. As such, it is necessary to register a store before loading any data.

Select Stores | New data source | PostGIS then the form was filled out as follows;



New Vector Data Source

Add a new vector data source

PostGIS
PostGIS Database

Basic Store Info

Workspace *
Ireland

Data Source Name *
dublin_highway1

Description

☑ Enabled

Connection Parameters

host *
localhost

port *
5432

database
testdb

schema
public

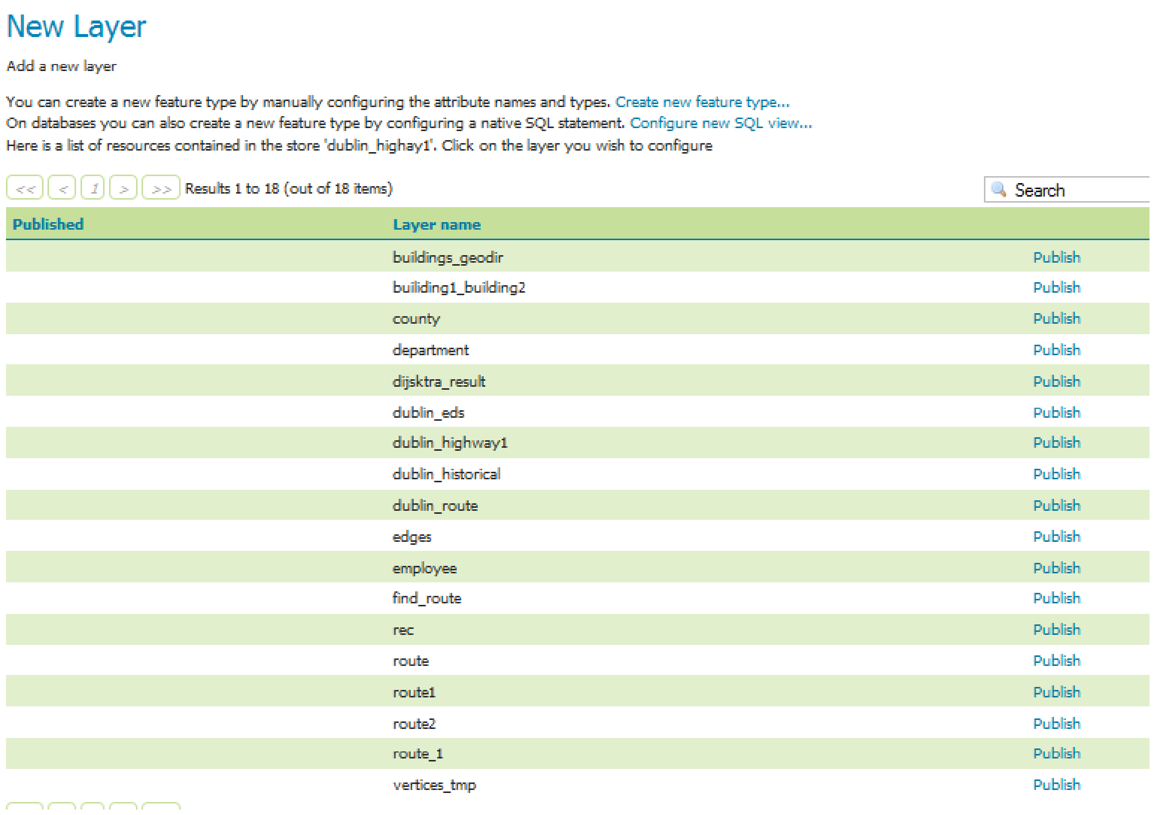
user *
postgres

passwd
●●●●●●●●●

Namespace *
http://www.openplans.org/sushma

Listing 5. 9 Filling out the form for the new data source

On Saving was presented with a list of all the tables in the `testdb` database



New Layer

Add a new layer

You can create a new feature type by manually configuring the attribute names and types. Create new feature type...
On databases you can also create a new feature type by configuring a native SQL statement. Configure new SQL view...
Here is a list of resources contained in the store 'dublin_highay1'. Click on the layer you wish to configure

<< < 1 > >> Results 1 to 18 (out of 18 items)

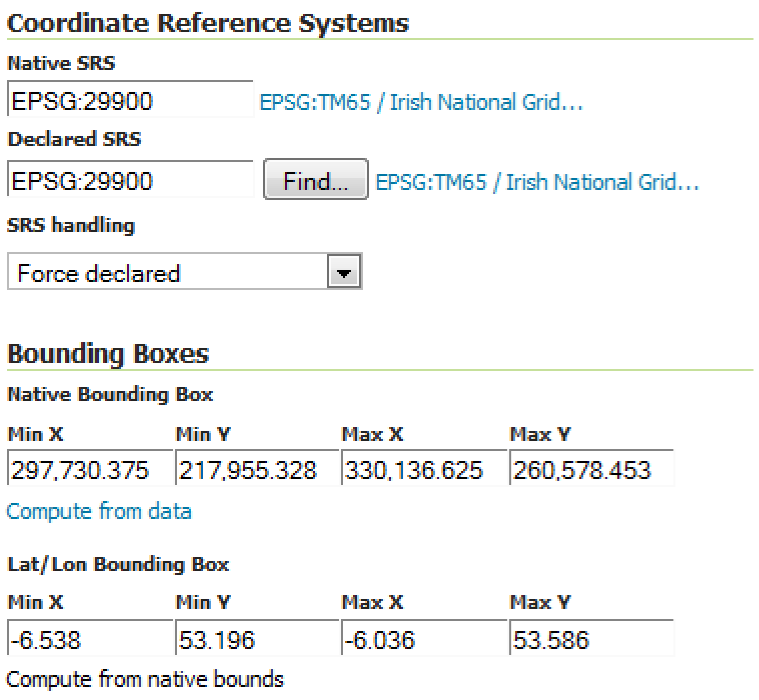
Published	Layer name	
	buildings_geodir	Publish
	builiding1_building2	Publish
	county	Publish
	department	Publish
	dijsktra_result	Publish
	dublin_eds	Publish
	dublin_highway1	Publish
	dublin_historical	Publish
	dublin_route	Publish
	edges	Publish
	employee	Publish
	find_route	Publish
	rec	Publish
	route	Publish
	route1	Publish
	route2	Publish
	route_1	Publish
	vertices_tmp	Publish

Figure 5. 5: Layer view

Step 4: Publish

Clicked on publish for dublin_highway1

Then it returned detailed layer information for the `dublin_highway1` table/layer.


Coordinate Reference Systems

Native SRS: EPSG:29900 EPSG:TM65 / Irish National Grid...

Declared SRS: EPSG:29900 EPSG:TM65 / Irish National Grid...

SRS handling: Force declared

Bounding Boxes

Native Bounding Box

Min X	Min Y	Max X	Max Y
297,730.375	217,955.328	330,136.625	260,578.453

Compute from data

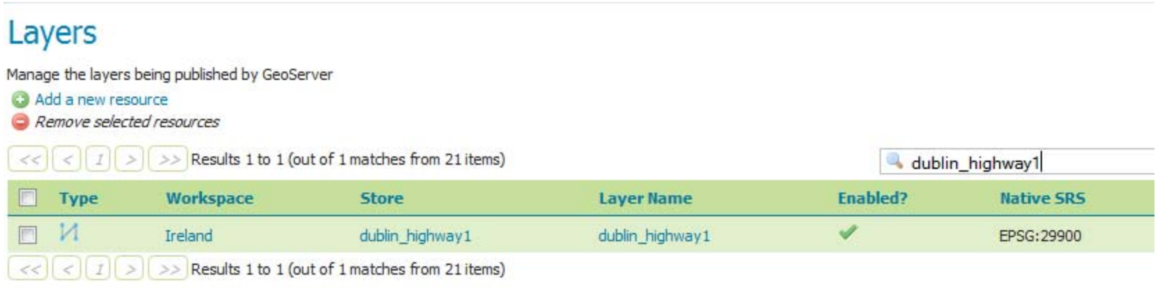
Lat/Lon Bounding Box

Min X	Min Y	Max X	Max Y
-6.538	53.196	-6.036	53.586

Compute from native bounds

Figure 5. 6 Computing bounding boxes in Irish National Grid

Figure 5.6 shows system computing the bounding boxes by clicking on "Computer from data" and "Compute from native bounds".
When clicked Save, got:


Layers

Manage the layers being published by GeoServer

Add a new resource

Remove selected resources

Results 1 to 1 (out of 1 matches from 21 items)

dublin_highway1

Type	Workspace	Store	Layer Name	Enabled?	Native SRS
	Ireland	dublin_highway1	dublin_highway1	✓	EPSG:29900

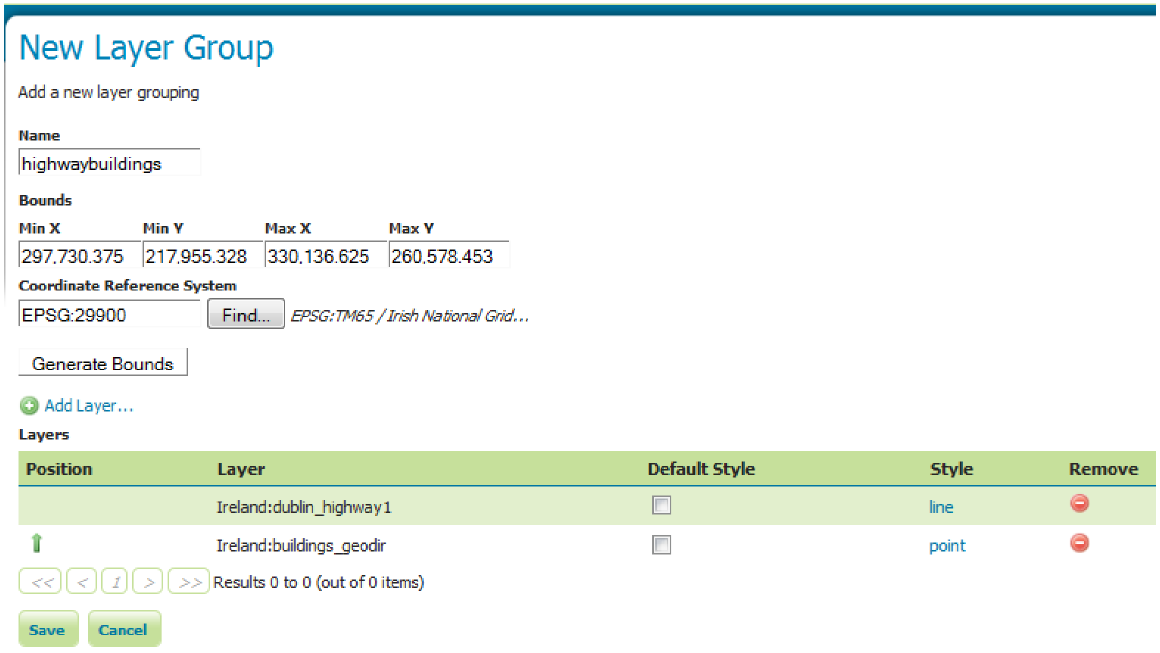
Results 1 to 1 (out of 1 matches from 21 items)

Figure 5. 7 Result after the coordinate system was set to Irish National Grid(29900).

Again the step3 and step4 was repeated to add the buildings data to GeoServer but this time Data Source name is buildings_geodir.

Step5: Add a layer group

This step was done to display the road table and buildings table together



New Layer Group

Add a new layer grouping

Name	highwaybuildings

Bounds

Min X	Min Y	Max X	Max Y
297,730.375	217,955.328	330,136.625	260,578.453

Coordinate Reference System: EPSG:29900 Find... *EPSG:TM65 / Irish National Grid...*

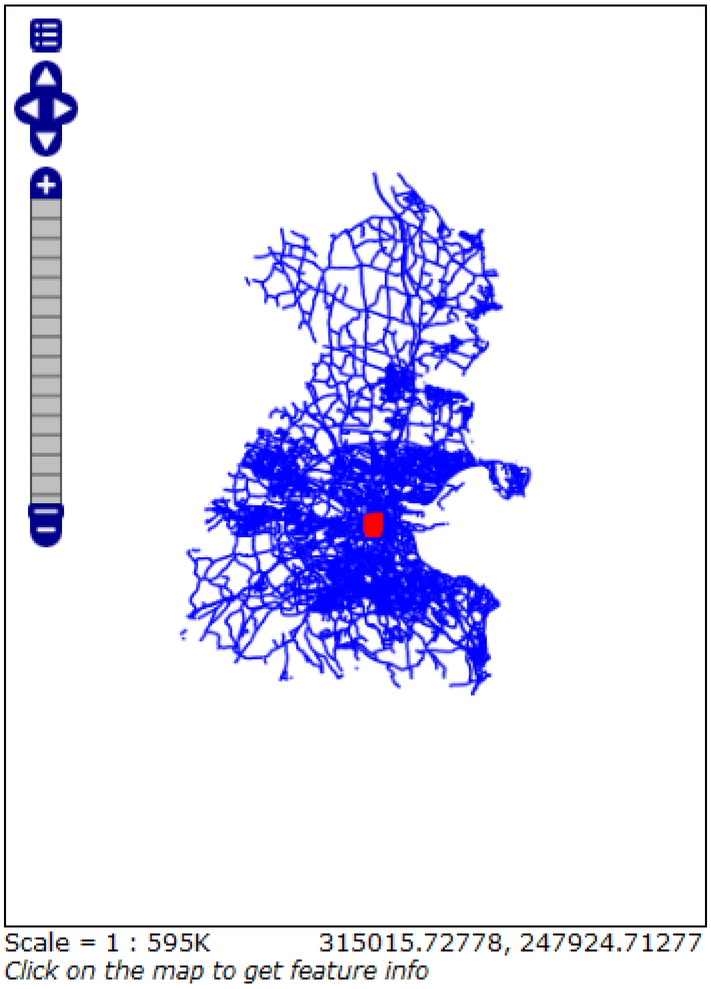
Layers

Position	Layer	Default Style	Style	Remove
	Ireland:dublin_highway1	☐	line	⊖
↑	Ireland:buildings_geodir	☐	point	⊖

Results 0 to 0 (out of 0 items)

Figure 5. 8 Adding two different geometry table in one group name highwaybuildings

By viewing the highwaybuildings group map in Layer Preview selecting OpenLayer format got both data were displayed as shown in figure



.

Figure 5. 9 Displaying group map in openlayer format

Figure 5.9 shows the both data dublin_highway1 and buildings_geodir from postgis spatial database were successfully integrated with GeoServer.

5.5 Summary

Creating a GIS involves integrating multiple technologies. Each of these has to be configured to communicate with each other to allow automated flow of data throughout the system.

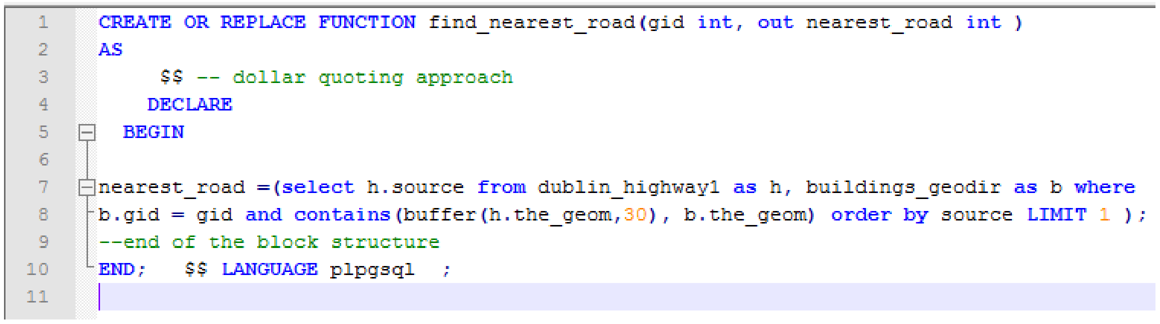
Chapter 6 Code

The main aim of this project is to find route from one building to another building using PostgreSQL/PostGIS ,initially display the routes in OpenJump and then display the routes using openlayers. I have already explained how the data were loaded in database. Now this chapter explains the languages that were used in this project to meet the main objectives of this project.

6.1 PL/pgSQL code

PL/pgSQL programming language was used to put the queries together. By using this language first the small function named as find_nearest_road was created and this function was passed as a variable in another function named find_route which is described in section 6.1.2.

6.1.1 find_nearest_road function



```
1   CREATE OR REPLACE FUNCTION find_nearest_road(gid int, out nearest_road int )
2   AS
3        $$ -- dollar quoting approach
4       DECLARE
5     BEGIN
6
7   nearest_road =(select h.source from dublin_highway1 as h, buildings_geodir as b where
8   b.gid = gid and contains(buffer(h.the_geom,30), b.the_geom) order by source LIMIT 1 );
9   --end of the block structure
10  END;   $$ LANGUAGE plpgsql  ;
11
```

Listing 6. 1:find_nearest_road function

Line 1 Creates the function named find_nearest_road and states its argument gid as integer and states the OUT parameter named nearest_road of type integer. This OUT parameter allows to return outputs from a function without having to declare a PostgreSQL type as output of the function.

In Line 3 the dollar sign represents the dollar quoting approach. Dollar quoting approach is used beacause ‘`CREATE FUNCTION`’ call that creates the PL/pgSQL function is specified as a string literal. If the string literal is written in ordinary way with single quotes, then any single quotes inside the function body must be doubled; likewise any backslashes must be doubled. In dollar quoting approach there is no need to double any quote marks.

In Line 7 and Line 8 “SELECT” clause selects desired column source from table dublin_highway1

--“WHERE” clause specifies the required condition for rows here the condition is buildings gid from the table buildings_geodir which should match with the argument gid at the time of function call

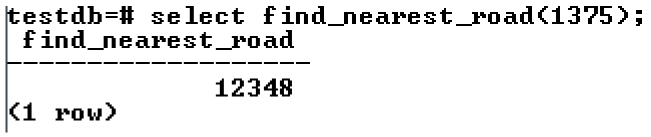
--This query involves spatial join between two tables

--spatial join is done between two spatial table dublin_highway1 and buildings_geodir

--spatial predicate “Contains” is used for spatial join to test wheather the dublin_highway1 geometry contains another geometry buildings_geodir

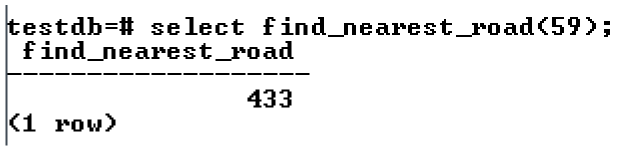
--spatial operator “Buffer” is used to buffer within the 30m of the building with gid and then finally through this query find_nearest_road function returns the source of the road order by source and limiting the result to return only one row

After calling this function passing building gid as an argument find_nearest_road in psql shell returns the source of the road from the dublin_ighway1 street table. Listings below shows the function find_nearest_road returning the source 12348 and 433 of the buildings with gid 1375 and 59 respectively



```
testdb=# select find_nearest_road(1375);
 find_nearest_road
-------------------
             12348
(1 row)
```

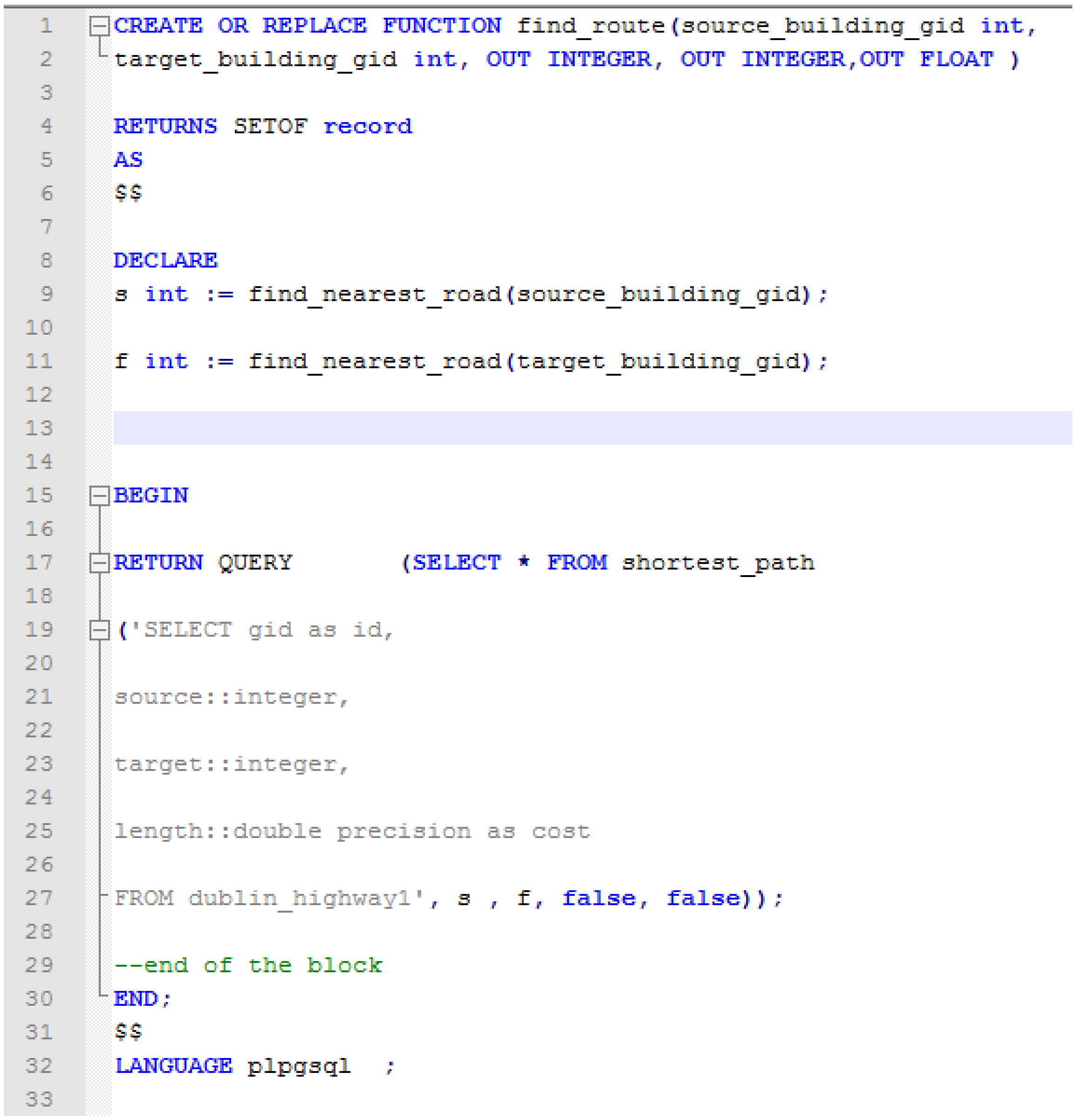
Listing 6. 2 calling find_nearest_road function with gid as argument



```
testdb=# select find_nearest_road(59);
 find_nearest_road
-------------------
               433
(1 row)
```

Listing 6. 3 calling find_nearest_road function again with another gid as argument

6.1.2 find_route function



```
1  CREATE OR REPLACE FUNCTION find_route(source_building_gid int,
2  target_building_gid int, OUT INTEGER, OUT INTEGER,OUT FLOAT )
3
4  RETURNS SETOF record
5  AS
6  $$
7
8  DECLARE
9  s int := find_nearest_road(source_building_gid);
10
11 f int := find_nearest_road(target_building_gid);
12
13
14
15 BEGIN
16
17 RETURN QUERY        (SELECT * FROM shortest_path
18
19 ('SELECT gid as id,
20
21 source::integer,
22
23 target::integer,
24
25 length::double precision as cost
26
27 FROM dublin_highway1', s , f, false, false));
28
29 --end of the block
30 END;
31 $$
32 LANGUAGE plpgsql   ;
33
```

Listing 6. 4 find_route function

In line 1 and 2 `CREATE FUNCTION` call creates the PL/pgSQL function in the PostgreSQL database. --This `CREATE FUNCTION` command names the new function find_route, states its argument source_building_gid and target_building_gid as integers types, OUT parameter allows to return outputs from a function without having to declare a PostgreSQL type as output of the function.The function's main code block then starts with a declaration section.

Line 4 defines return type which is set of record, “RETURNS SETOF record” returns the multiple rows

Line 9 and 10 declares the variable as int and another PL/pgSQL function find_nearest _road function is assigned as a variables within this function

Line 17 to 27 -- return query returns out value of the function through Dijkstra shortest path core function

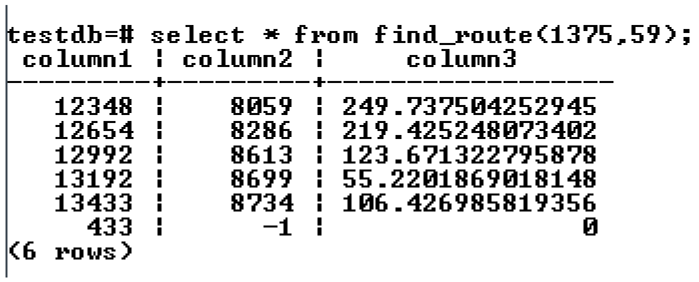
--source and target ids are vertex ids

--source and target is cast into integer and length to double precision

--s and f is passed as variable which came from declaration section

--false,false means has_reverse_cost is set to false

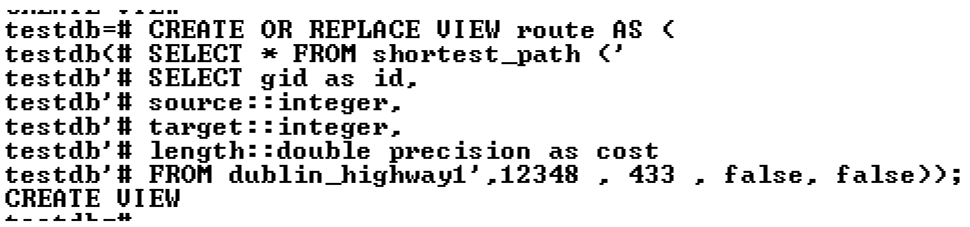
After calling this function find_nearest_road in psql shell got the shortest path between the two buildings with gid 1375 and 59 .



```
testdb=# select * from find_route(1375,59);
 column1 | column2 |      column3
---------+---------+-------------------
   12348 |    8059 | 249.737504252945
   12654 |    8286 | 219.425248073402
   12992 |    8613 | 123.671322795878
   13192 |    8699 | 55.2201869018148
   13433 |    8734 | 106.426985819356
     433 |      -1 |                0
(6 rows)
```

Listing 6. 5 calling the find_route function with two different gid as arguments

- Then the view was created to see the shortest path between buildings with gid 1375 and 59. Listing below shows how the view was created in psql shell.



```
testdb=# CREATE OR REPLACE VIEW route AS (
testdb(# SELECT * FROM shortest_path ('
testdb'# SELECT gid as id,
testdb'# source::integer,
testdb'# target::integer,
testdb'# length::double precision as cost
testdb'# FROM dublin_highway1',12348 , 433 , false, false));
CREATE VIEW
```

- The view was then can be viewed from the OpenJump and for that the following query was passed from the OpenJump query tool

select asbinary(h.the_geom) from route r, dublin_highway1 h where h.gid = r.edge_id;

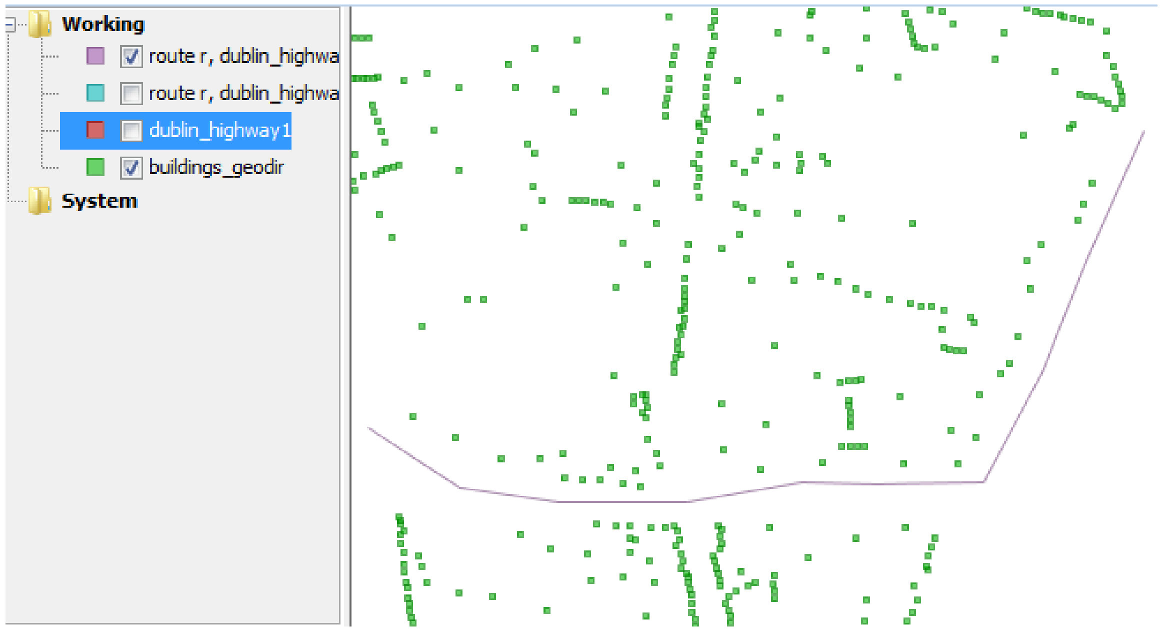


Figure 6. 1:showing the the shortest path between two buildings with gids 1375 and 59 in OpenJump.

6.2 OpenLayer Code

This section explains interface design code. It includes HTML, CSS and javascript code. JavaScript was selected as it's provides functionality need to keep track of mouse position or capture when someone clicks the mouse, or execute something when the page is fully loaded.

Therefore JavaScript was used in conjunction with OpenLayer API to load the map and enable user interaction.

<div id="map" ></div>

This `<div>` element will serve as the container for map viewport..

```
<style type="text/css" >
#map {
width: 500px;
height: 500px;
border: 1px solid black;
}
</style>
```

In this case, we're using the map containers value as a selector, and map is specified with the width (`512px`) and the height (`256px`) for the map container and 1 px for the border.

```
<script src="openlayers/OpenLayers.js" ></script>
```

This includes the OpenLayers library. The location of the file is specified by the src='openlayers/OpenLayers.js' attribute. Here, I am using a relative path. As the index.html page is in the same folder as the OpenLayers.js file
An absolute path can also be used that is it can be pass in a URL that the script is located at. OpenLayers.org hosts the script file as well; the following line of code can be used to link to the library file directly:

```
<script type='text/javascirpt' src='http://openlayers.org/api/
OpenLayers.js'></script>
```

The advantage of using absolute path is that you do not need to maintain a local copy of the JavaScript library on your own web server ensuring that you always have the most up-to-date product on offer. The disadvantage of this is that you must always have internet connectivity to access their functionality. So I have chosen to use the localcopy for this project.

<script> starts a block where all our code is set up inside it to create map.

```
var map;
```

global variable called map is created. In JavaScript, anytime a variable is created it is needed to place var in front of it to ensure that we don't run into scope issues (what functions can access which variables). When accessing a variable, no need to put var in front of it.

```
<!----init() used to initialize the map using the onload=”init”  in the body of the HTML page, therefore telling the web browser to run this function at the same time as the webpage body is loaded---!>
function init(){



// creates a new openlayers.Bounds object, Bound sets the geographic limit of the display layer
            var bounds = new OpenLayers.Bounds(
                297730.375, 217955.328,
                330136.625, 260578.453
             );

//In this example, we set some custom map options. First, we set    controls: [new OpenLayers.Control.PanZoom() ,],
Next  the map units to "m" for meters was set,  and projection  was set  in 29900 Irish national grid .

             var options = {
                controls: [new OpenLayers.Control.PanZoom() ,],

                maxExtent: bounds,
                maxResolution: 166.49658203125,
                projection: "EPSG:29900",
                units: 'm'
             };


// The OpenLayers.Layer.WMS constructor requires 3 arguments and an optional fourth.


dublin = new OpenLayers.Layer.WMS( "dublin historical" ,
"http://localhost:8080/geoserver/wms?" ,
{layers: 'Ireland:buildings_geodir', transparent:true} );
```

The first argument, `"dublin"`, is a string name for the layer. This is only used by components that display layer names (like a layer switcher) and can be anything of choosing.

The second argument, "http://localhost:8080/geoserver/wms?" is the string URL for a Web Map Service.

The third argument, `{layers: 'Ireland:buildings_geodir'}` is an object literal with properties that become parameters in our WMS request. In this case, we're requesting images rendered from a single layer identified by the name `'Ireland:buildings_geodir'`.

```
//add layers to the map can be done together but here I have done it separetly

map.addLayer(counties);
map.addLayer(dublin);
map.addLayer(start);
map.addLayer(stop);

// support GetFeatureInfo
            <! Capture mouse events and pass request to function to be execute, so that an
user can click on the map to get map information of the area of interest !>
  map.events.register('click', map, function (e) {


//define the result of getfeatureInfo, the parameters are listed below
var params = {
                  REQUEST: "GetFeatureInfo",
                  EXCEPTIONS: "application/vnd.ogc.se_xml",
                  BBOX: map.getExtent().toBBOX(),
                  X: e.xy.x,
                  Y: e.xy.y,
                  INFO_FORMAT: 'text/html',
//get map query retrieve information from the database
                  QUERY_LAYERS=Ireland:dublin_highway1,buildings_geodir,
//set maximum number of return feature
                  FEATURE_COUNT: 5,
          LAYERS=Ireland:dublin_highway1,buildings_geodir,
Styles: '',
  //set Irish coordinate
                  Srs: 'EPSG:29900',
                  WIDTH: map.size.w,
                  HEIGHT: map.size.h,
                  format: format};


// If you activate a radio-buttion, eg. for the startpoint
// the function toggleControl is activated.

<input type="radio" name="control" value="start" id="startToggle"
                onclick="toggleControl(this);" />
        <label for="startToggle">set start point</label>



// draw feature control is created
```

this creates an `OpenLayers.Control.DrawFeature` control . We construct this layer with an `OpenLayers.Handler.Point` to allow drawing of point.

```
controls = {
   start: new OpenLayers.Control.DrawFeature(start, SinglePoint),
      stop: new OpenLayers.Control.DrawFeature(stop, SinglePoint)
    }
    for (var key in controls) {
        map.addControl(controls[key]);
    }

// This section deals with vector layers - where the data is rendered for viewing in your
browser. The two vector layer is given a title Start point and End point respectively.

start = new OpenLayers.Layer.Vector("Start point", {style: start_style});
stop = new OpenLayers.Layer.Vector("End point", {style: stop_style});



// these code provides style to the vector layer Start point and End point
var start_style = OpenLayers.Util.applyDefaults({
                externalGraphic: "marker-green.png",
                graphicWidth: 18,
                graphicHeight: 26,
                graphicYOffset: -26,
                graphicOpacity: 1
            }, OpenLayers.Feature.Vector.style['default']);

                            var stop_style = OpenLayers.Util.applyDefaults({
                externalGraphic: "marker.png",
                graphicWidth: 18,
                graphicHeight: 26,
                graphicYOffset: -26,
                graphicOpacity: 1
            }, OpenLayers.Feature.Vector.style['default']);
```

OpenLayers provides controls for drawing and modifying vector features. The `OpenLayers.Control.DrawFeature` control can be used in conjunction with an `OpenLayers.Handler.Point`, an `OpenLayers.Handler.Path`, or an `OpenLayers.Handler.Polygon` instance to draw points, lines, polygons, and their multi-part counterparts. The `OpenLayers.Control.ModifyFeature` control can be used to allow modification of geometries for existing features.
openlayer uses event listener to handle the mouse events.

```
var SinglePoint = OpenLayers.Class.create();
 SinglePoint.prototype = OpenLayers.Class.inherit(OpenLayers.Handler.Point, {
    createFeature: function(evt) {
        this.control.layer.removeFeatures(this.control.layer.features);
        OpenLayers.Handler.Point.prototype.createFeature.apply(this, arguments);
    }
 });
```

Finally, behavior was added to the `<input>` element in order to activate and deactivate the draw control when the user clicks the checkbox. We'll also call the `toggle` function when the page loads to synchronize the checkbox and control states. Add the following to your map initialization code:

```
function toggleControl(element) {

   for (key in controls) {
      if (element.value == key && element.checked) {


         controls[key].activate();
      } else {
         controls[key].deactivate();
      }
   }
}
```

In order that users can also navigate with the mouse, we don't want this control to be active all the time. We need to add some elements to the page that will allow for control activation and deactivation. In the `<body>` of your document, add the following markup.

```
<input type="radio" name="control" id="noneToggle"
            onclick="toggleControl(this);" checked="checked" />
        <label for="noneToggle">navigate</label>
```

Finally figure below shows the two tables dublin_highway1 and buildings_geodir displayed as a layer through openlayers.

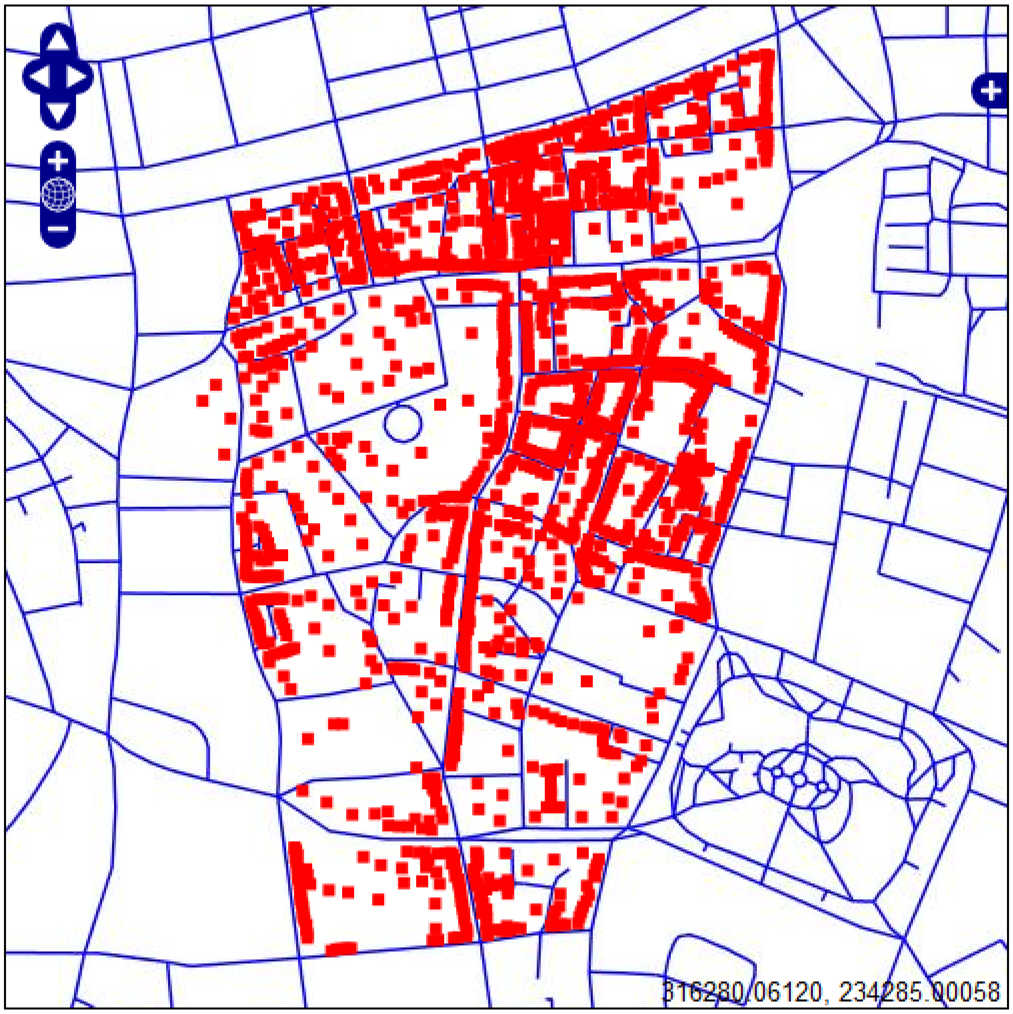
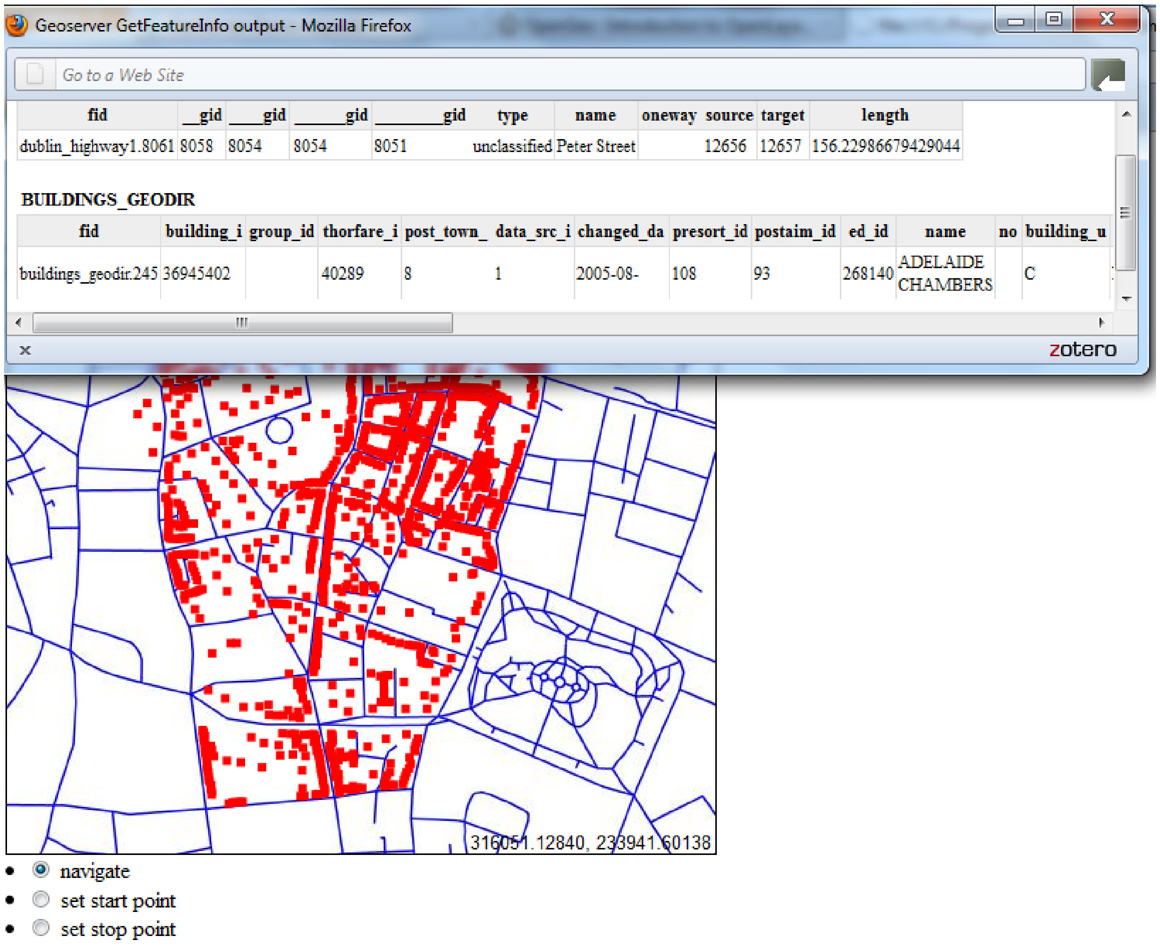


Figure 6. 2 Showing the road and buildings table as a layer



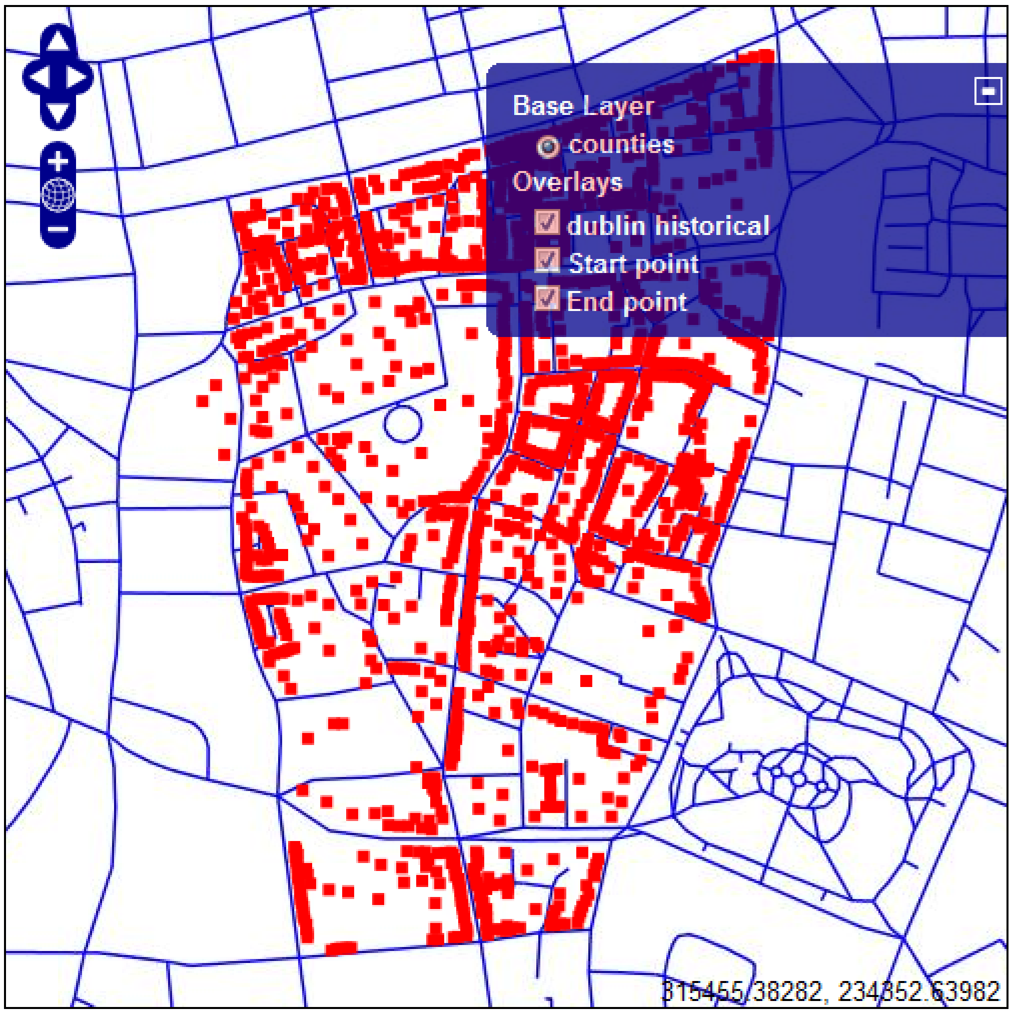
fid	_gid	__gid	____gid	_______gid	type	name	oneway	source	target	length
dublin_highway1.8061	8058	8054	8054	8051	unclassified	Peter Street		12656	12657	156.22986679429044

BUILDINGS_GEODIR

fid	building_i	group_id	thorfare_i	post_town_	data_src_i	changed_da	presort_id	postaim_id	ed_id	name	no	building_u
buildings_geodir.245	36945402		40289	8	1	2005-08-	108	93	268140	ADELAIDE CHAMBERS		C

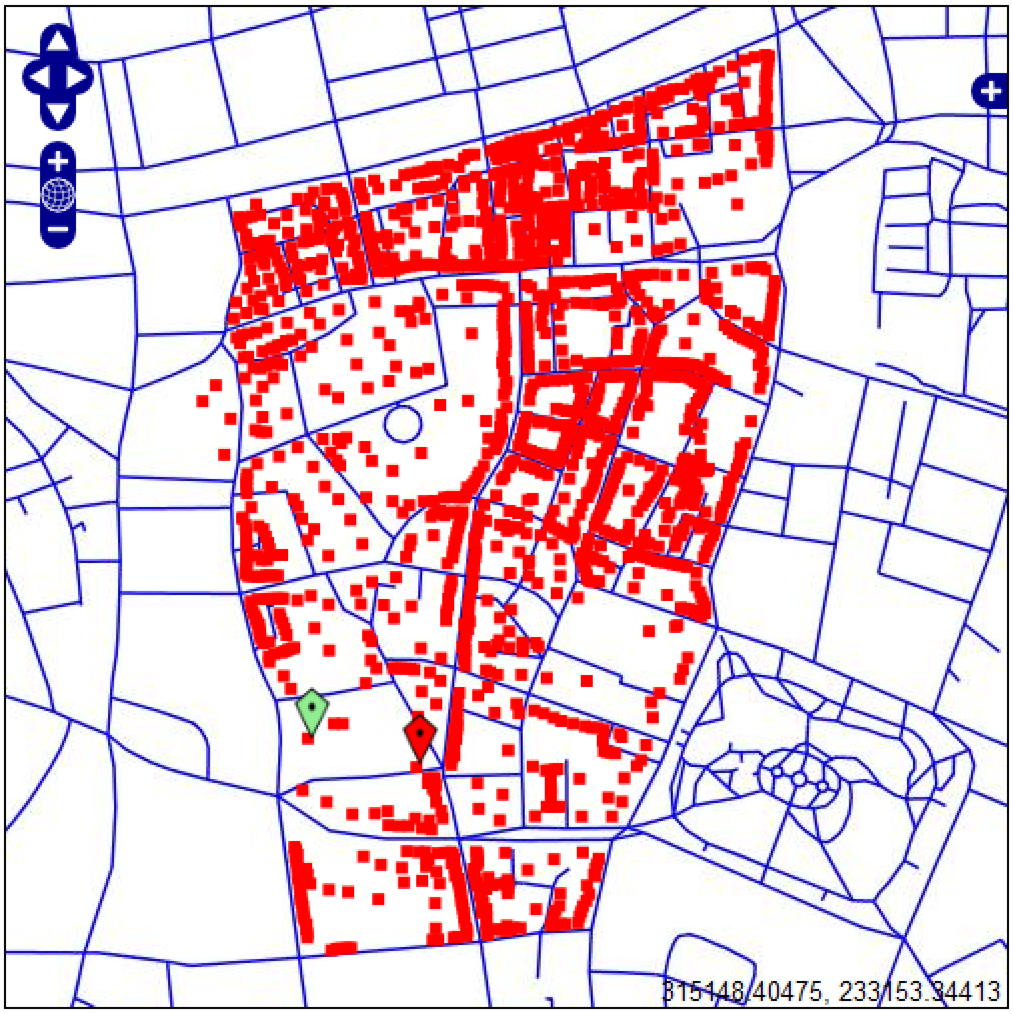
- navigate
- set start point
- set stop point

Figure 6. 3 Showing the features of the road and building data.



- [✓ navigate]
- set start point
- set stop point

Figure 6. 4 showing the Layer switcher containing layers name



- ◉ navigate
- ○ set start point
- ○ set stop point

Figure 6. 5 showing the start and end point as a layer

Chapter 7 Conclusions

However achieving the aims of this project was very challenging since technical and non technical Challenges needed to be addressed. Personal Research and reading where essential in order to succeed. Most Importantly understand the concept of spatial database and Web mapping.

A lot of time and resources were used to investigate, testing and integrating the different tools needed to support the implementation process of this project, because it was difficult to implement extra features on standalone earth client without previous knowledge of PL/pgSQL, OpenLayers, GeoServer and javascript language.

7.1 Overview of objectives

Objective	Deliverables
Research, install and configure a spatial database.	PostGIS
Research, install and configure coding language that interact with spatial database.	PL/pgSQL
Acquire competency using the tool OpenJump, PostgreSql and GeoServer.	These tools were successfully downloaded and integrated
Research, install and configure software to act as the client interface in conjuction with the user's browser	openLayers
Research, install and configure client that act as gis viewer to perform analysis of datasets, query the database.	OpenJump

To acquire the data which will be used for the route finding	The streets use OSM data from CloudMade.The OSM data was transformed to Irish National Grid (IGN).The buildings are a subset of the Geodirectory from An Post originally in IGN.
To load the database with valid data	Database was loaded with both data i.e street data and buildings data
To design the systems that integrates all the components	Integrated system is designed in chapter 5
To create a queries that will allow the user to initially interact with the data in GIS client	OpenJump
To create an interface that allows the user to interact with the system	Only the part of the interface was designed Using HTML, CSS, OpenLayers and Geoserver.

Table 7. 1: Objectives and their corresponding Deliverables of the project

There were other emergent benefits that were a consequence of the planning and preparation that went into preparing this work:

- project management
- work planning
- project report making / writing

The following points illustrate the main learning outcomes.

- I’ve also improved my research skills. The assignments from my previous studies had given me an introduction into having to perform research to find answers. For this project it was on a much grander scale. By the end of the project I found that I had learned how to evaluate sources better and how to better pick the information that was relevant to my aims.

- Building the client side component gave the author a working understanding of the OGC’s standards and specifications in general. Detailed knowledge of the OGC Web

Map and the Simple Features was also gained through their practical application in the prototype.

- Data is freely available from a variety of sources. The main source of data for this project was open street map data from cloudmade and the buildings are a subset of the Geodirectory from An Post originally in IGN.
- Becoming familiar with so many different applications and technologies has an initial steep learning curve but the knowledge attained provides a strong foundation for building similar systems in the future
- Lastly, I also learned a lot about the Open Source software world. Previous to this project, my knowledge of the open source world was almost zero. By working withopen source software on this project, I learned how the open source projects are typically run and how one can make a contribution.

7.4 Benefits of undertaking this project

This project has been personally very beneficial from the point of view of acquiring in-depth knowledge into integrating a system that includes an Postgis database, a GIS server, and front-end technology to display geographic data. Each of these had to be researched so in addition of gaining hands-on experience with the chosen software, information was also learned about other similar applications and the functionality they offer.

The underlying services of PostGis combined with GeoServer can be used for deploying any geographic data and allow the use of this information in many ways i.e. using OpenJump to access either the database directly or through GeoServer to process the data. It makes viewing this geographic data very easy over the web using a product such as OpenLayers.

7.5 Future research could be done

The software developed in this work performs to show routes between one building to another building in OpenJump using dijkstra shortest path algorithm and can display the layers in openlayer format and javascript was written two pick up the two points as a layer.

The following features could be added:

- Create table dynamically after picking the two points and display features as a layer
- Create the routes from one name place to another
- Create a web service where the routing algorithm runs on the web server and accepts queries from web clients.The server performs all required computations and generates images with recommended routes that are sent back to the client.
- Create route based on more than one routing algorithms.

8 References

GeoServer,(2009) Web Administration Interface — GeoServer 2.1.0 User Manual. Available at: http://docs.geoserver.org/2.1.0/user/webadmin/index.html [Accessed August 11, 2011].

Chen ,C.J , "Dijkstra's Shortest Path Algorithm", *JOURNAL OF FORMALIZED MATHEMATICS* Volume 15, Released 2003, Published 2003
Inst. of Computer Science, Univ. of Bialystok

pgRouting, 2011, Shortest Path Dijkstra — Open Source Routing Library. Available at: http://www.pgrouting.org/docs/1.x/dijkstra.html [Accessed July 18, 2011].

Sutskever.V,2008, Dijkstra Algorithm implementation in Java. Available at: http://www.cs.nyu.edu/~vs667/development/~DijkstraAlgorithm/ [Accessed July 18, 2011].

Waldura.R, 2007 ,Dijkstra's Shortest Path Algorithm in Java. Available at: http://renaud.waldura.com/doc/java/dijkstra/ [Accessed July 18, 2011].

Lecture 6,2011, spatial databases lectures. Available at: http://www.comp.dit.ie/pbrowne/Spatial%20Databases%20SDEV4005/Spatial%20Databases%20SDEV4005.htm [Accessed July 18, 2011].

RUMBAUGH, J. JACOBSON, I. and BOOCH, G.(1999): *The Unified Modeling Language Reference Manual.* Reading, Mass, Addison Wesley Longman Inc.

Purchase, H.C., Colpoys, L., McGill, M., Carrington, D., and Britton, C. (2001):*UML class diagram syntax: an empirical study of comprehension*
School of Information Technology and Electrical Engineering
University of Queensland

Obe ,O.R and Hsu, S.Leo (2010)/ *PostGIS in Action,* Manning publications

FOSS4G ,(2011). Shortest Path search for real road networks with pgRouting | Free and Open Source Software for Geospatial. Available at: http://2011.foss4g.org/sessions/shortest-path-search-real-road-networks-pgrouting [Accessed July 17, 2011].

Geoserver 2.1.0, Overview — GeoServer 2.1.0 User Manual. Available at: http://docs.geoserver.org/2.1.0/user/introduction/overview.html [Accessed August 13, 2011].

OpenJUMP, OpenJUMP GIS. Available at:
http://www.openjump.org/index.html [Accessed July 21, 2011].

Informer, OpenJUMP Software Informer: version 1.2 information. Available at: http://openjump.software.informer.com/1.2/ [Accessed July 21, 2011].

PostGIS, PostGIS: Home. Available at:
http://postgis.refractions.net/ [Accessed May 14, 2011].

PostgreSQL, (1996-2010). PostgreSQL: About. Available at:
http://www.postgresql.org/about/ [Accessed July 21, 2011].

SourceForge, SourceForge.net: What is OpenJUMP - jump-pilot. Available at:
http://sourceforge.net/apps/mediawiki/jump-pilot/index.php?title=What_is_OpenJUMP [Accessed July 21, 2011].

Windows 7, Windows 7 - Microsoft Windows. Available at:
http://windows.microsoft.com/en-US/windows7/products/home [Accessed July 21, 2011].

Mitchell,T(2005) *Web mapping illustrated* - Google Books. Available at: http://books.google.com/books?hl=en&lr=&id=IdGoy2rZSyIC&oi=fnd&pg=PR3&dq=web+mapping&ots=toVZFMkbMN&sig=kuqnTOEeCPBpSSXKnjNSDVX64RQ#v=onepage&q&f=false [Accessed August 1, 2011].

Hazzard,E (2011) *Openlayers 2.10 Beginners Guide*, Packt publishing

Mozilla, Mozilla.com | Mobile. Available at:
http://www.mozilla.com/en-US/m/faq [Accessed August 13, 2011].

OGCWMS 1.3. Web Map Service | OGC(R). Available at:
http://www.opengeospatial.org/standards/wms [Accessed August 3, 2011].

GeoServer, (2011). Welcome - GeoServer. Available at:
http://geoserver.org/display/GEOS/Welcome [Accessed August 2, 2011].

WSA, (2004). Web Services Architecture. Available at:
http://www.w3.org/TR/ws-arch/ [Accessed August 2, 2011].

OGC ,(2002) Web Map Service Implementation Specification. Available at: http://cite.opengeospatial.org/OGCTestData/wms/1.1.1/spec/wms1.1.1.html [Accessed August 2, 2011].

Deoliveira. J, GeoServer: Uniting the “GeoWeb” and Spatial Data Infrastructures http://www.gsdi.org/gsdiconf/gsdi10/papers/TS26.4paper.pdf

OGCWFS 1.1.0, (2005) Web Feature Service | OGC(R). Available at: http://www.opengeospatial.org/standards/wfs [Accessed August 7, 2011].

OSM, (2011a):OpenStreetMap. Available at:

http://www.openstreetmap.org/ [Accessed July 28, 2011].

Haklay,M. & Weber,P , OpenStreetMap: User-Generated Street Maps. *Pervasive computing,IEEE*,vol.7,no.4, 2008 , pp.12-18

Bennett, J. (2010) *OpenStreetMap.* Birmingham :Packt publishing

ESRI, (2011a).ESRI Virtual Campus. Available at:

http://training.esri.com/Courses/LearnArcGIS/index.cfm?c=188 [Accessed May 21, 2011a].

GIS, (2008). Essays on Geography and GIS. Available at:http://www.esri.com/library/bestpractices/essays-on-geography-gis.pdf

Longley, P.A. et al., 2010. *Geographic Information Systems and Science*, John Wiley & Sons. Available at: http://books.google.com/books?id=wUkZQAAACAAJ.

Ian N. Gregory and Richard G. Healey Historical GIS: structuring, mapping and analysing geographies of the past

DOI: 10.1177/0309132507081495
Prog Hum Geogr 2007 31: 638
http://0-phg.sagepub.com.ditlib.dit.ie/content/31/5/638.full.pdf+html

ISS (2006). Information Software Systems. Available at:

http://www.informationsoftwaresystems.com/ [Accessed June 4, 2011].

ESRI (2011b): vector. Available at:

http://training.esri.com/Courses/_Shared2003/vc/_deftemplate.cfm?Term=vector &CourseKbaseID=-1 [Accessed June 5, 2011b].

Guting.R.H. 1994. An introduction to spatial database systems. *The VLDB Journal* 3, 4 (October 1994), 357-399.

FOSS4G, (2011). Shortest Path search for real road networks with pgRouting | Free and Open Source Software for Geospatial. Available at: http://2011.foss4g.org/sessions/shortest-path-search-real-road-networks-pgrouting [Accessed July 17, 2011].

pgRouting, pgRouting Project — Open Source Routing Library. Available at: http://www.pgrouting.org/ [Accessed July 17, 2011].

Lecture 2. (2011) spatial databases lectures. Available at: http://www.comp.dit.ie/pbrowne/Spatial%20Databases%20SDEV4005/Spatial%20Databases%20SDEV4005.htm [Accessed July 18, 2011].

GIS.com |. Available at: http://gis.com/ [Accessed May 14, 2011].

OGCSFS 1.1, OpenGIS Simple Features Specification For SQL ref: 99-049 http://portal.opengeospatial.org/files/?artifact_id=829[Accessed 15th August 2011]

OGC ,(2002) Web Map Service Implementation Specification. Available at: http://cite.opengeospatial.org/OGCTestData/wms/1.1.1/spec/wms1.1.1.html [Accessed August 2, 2011].

OWS (2003). Open GIS Consortium Inc. portal.opengeospatial.org/files/?artifact_id=1320

OGC History, OGC History (abbreviated) | OGC(R). Available at: http://www.opengeospatial.org/ogc/history [Accessed August 14, 2011].

Carl N. Reed (2011). *Geospatial Web Services: Advances in Information Interoperability* http://www.igi-global.com/viewtitlesample.aspx?id=51480 [Accessed August 14, 2011].

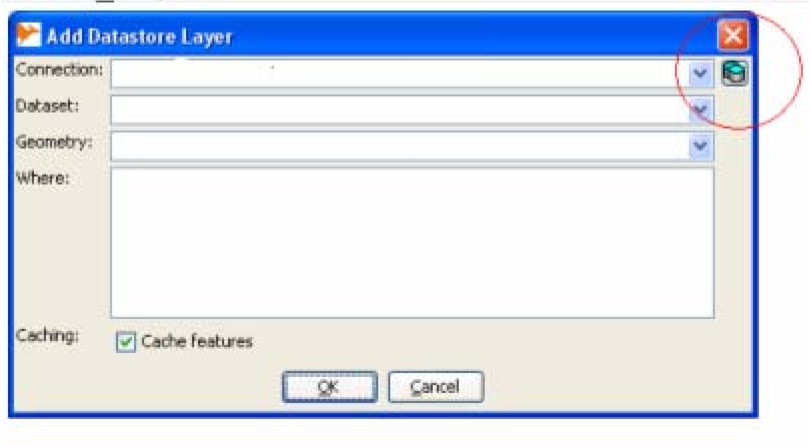
Whitten J. L., Bentley L. D., Dittman K. C. (2001) Systems Analysis and DesignMethods, 5th Ed., McGraw Hill

OpenLayers, OpenLayers: Home. Available at: http://openlayers.org/ [Accessed August 29, 2011].

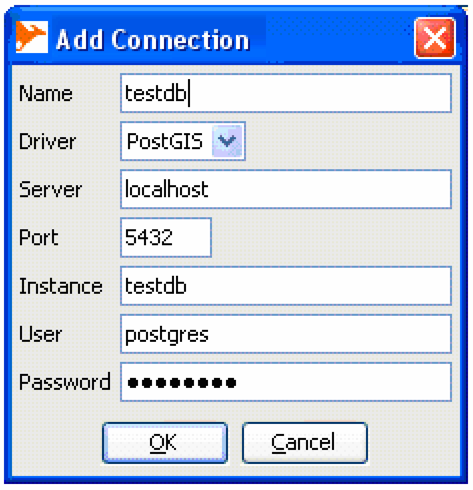
9. Appendix

9.1 Appendix A Using OpenJump with PostgreSQL/PostGIS

- To add a PostGIS table to OpenJump select
- Layer | Run Datastore Query
- The first time you do this you will have to establish a connection with PostgreSQL/PostGIS.



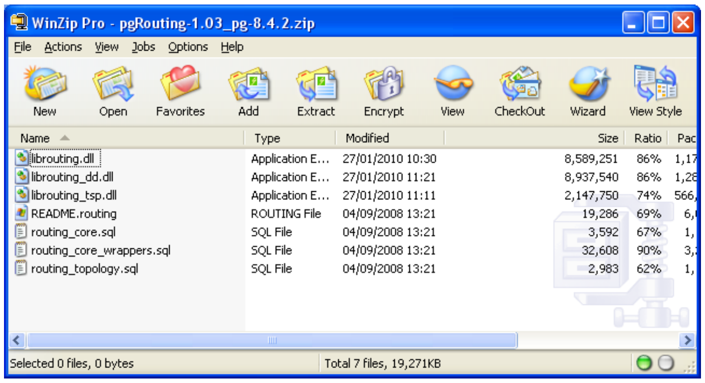
- In the Run Datastore Query Window click on the icon on the top right (2 small disks)
- This will open the Connection Manager
- Click Add



Name testdb
Driver PostGIS
Server localhost
Port 5432
Instance testdb
User postgres
Password ●●●●●●●●●

9.2 Appendix B- pgRouting Installation

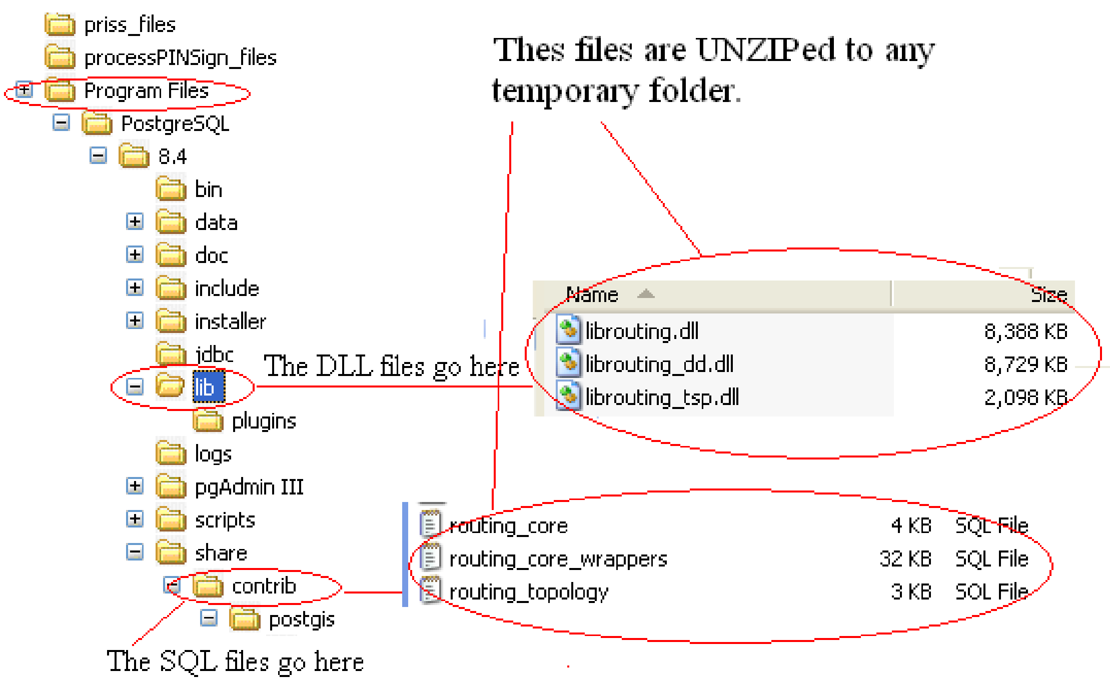
The Zip contains source SQL files and Windows DLL files



Name	Type	Modified	Size	Ratio	Pac
librouting.dll	Application E...	27/01/2010 10:30	8,589,251	86%	1,17
librouting_dd.dll	Application E...	27/01/2010 11:21	8,937,540	86%	1,28
librouting_tsp.dll	Application E...	27/01/2010 11:11	2,147,750	74%	566,
README.routing	ROUTING File	04/09/2008 13:21	19,286	69%	6,
routing_core.sql	SQL File	04/09/2008 13:21	3,592	67%	1,
routing_core_wrappers.sql	SQL File	04/09/2008 13:21	32,608	90%	3,
routing_topology.sql	SQL File	04/09/2008 13:21	2,983	62%	1,

Selected 0 files, 0 bytes   Total 7 files, 19,271KB

- Unzip (extract) to a folder (say pgTemp).
- The extraction process will make two sub-folders called “lib” and “share”
- You should copy the DLLs and SQL files provided in these folders the correct location under your PostgreSQL installation, which should be:
- Copy the DLL files to C:\Program Files\PostgreSQL\8.4\lib
- Copy the SQL files to C:\Program Files\PostgreSQL\8.4\share\contrib
- These folders are shown below:



Name	Size
librouting.dll	8,388 KB
librouting_dd.dll	8,729 KB
librouting_tsp.dll	2,098 KB

routing_core	4 KB	SQL File
routing_core_wrappers	32 KB	SQL File
routing_topology	3 KB	SQL File

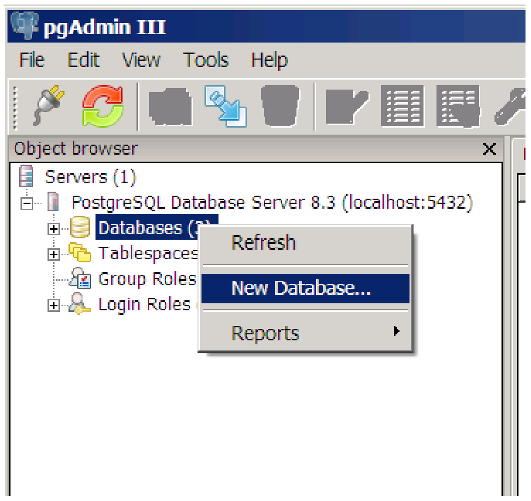
9.3 Appendix C

I had created separate database named testdb and postGIS functionality was added to it. It is described below how it was done.

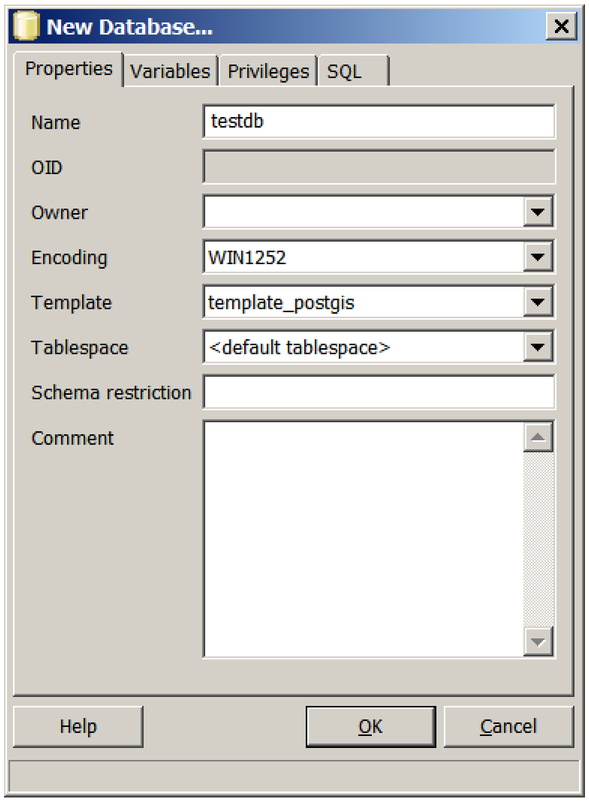
Start pgAdmin III, a graphical tool for administering PostgreSQL databases .

Step 2: Create a database called "testdb" using pgAdmin III and add PostGIS functionality via the default template. A template can be used to construct many database with the same properties.

- Open pgAdmin III from the Windows Start Menu ("Start->Programs->PostgreSQL 8.4->pgAdmin III").
- Connect to your database by double clicking it in the object browser. You may need to enter password information.
- In pgAdmin III, right click on "Databases" in the table and click "New Database…".

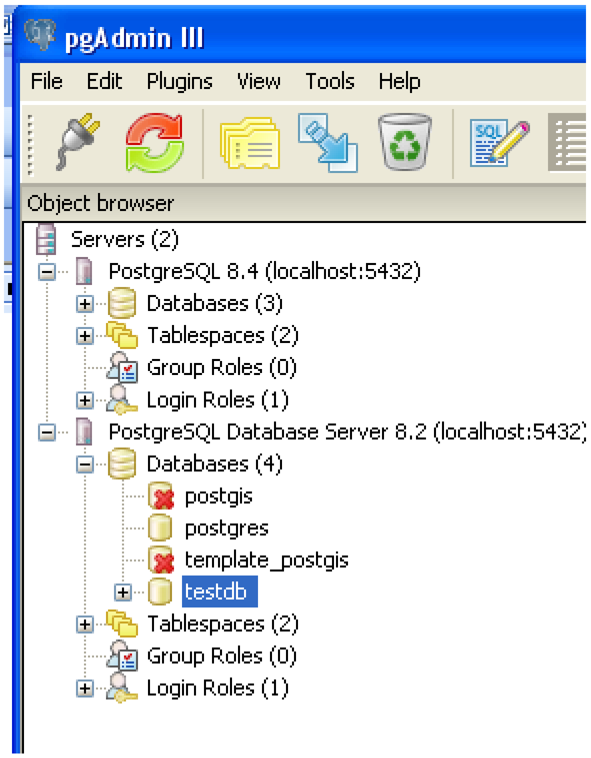


- Name the database “testdb” and for the template, select “template_postgis”.
- Click “OK”.



Name	testdb
OID	
Owner	
Encoding	WIN1252
Template	template_postgis
Tablespace	<default tablespace>
Schema restriction	
Comment	

Your pgAdmin should look something like this:



Servers (2)
PostgreSQL 8.4 (localhost:5432)
Databases (3)
Tablespaces (2)
Group Roles (0)
Login Roles (1)
PostgreSQL Database Server 8.2 (localhost:5432)
Databases (4)
postgis
postgres
template_postgis
testdb
Tablespaces (2)
Group Roles (0)
Login Roles (1)

<u>Step 3</u>: Add the core pgRouting functionality to the newly created database.

- In pgAdmin III, <u>select</u> the newly created "testdb" database in the object browser.

- Look at the top toolbar in pgAdmin III. There is a SQL query tool. Click on this tool to open it, or click "Tools->Query Tool" from the application menu.
- In the SQL query tool window, click "File->Open" and select

"C:\Program Files\PostgreSQL\8.4\share\contrib\routing_core.sql"

- To execute this query, click the green "play" button or navigate the application menu by clicking "Query->Execute".
- Repeat the same process for

"C:\Program Files\PostgreSQL\8.4\share\contrib\routing_core_wrappers.sql".

"C:\Program Files\PostgreSQL\8.4\share\contrib\routing_topology.sql".