ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

A WEB MAPPING INFRASTRUCTURE DESIGN AND IMPLEMENTATION WITH OPEN SOURCE GEO INFORMATION TECHNOLOGY.A CASE STUDY OF ITU SMART CAMPUS

M.Sc. THESIS

Rouhollah NASIRZADEH DIZAJI
(706131019)

Informatics Institute

Geographical Information Technologies Programme

Thesis Advisor: Prof. Dr. Rahmi Nurhan ÇELİK

May 2015

Rouhollah Nasirzadeh Dizaji, a **M.Sc.** student of ITU Informatics Institute student ID **706131019**, successfully defended the **thesis** entitled "**A WEB MAPPING INFRASTRUCTURE DESIGN AND IMPLEMENTATION WITH OPEN SOURCE GEO INFORMATION TECHNOLOGY. A CASE STUDY OF ITU SMART CAMPUS**", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Rahmi Nurhan ÇELİK** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Melih BAŞARANER** Yıldız Technical University	.....
	Assist. Prof. Dr. Ahmet Özgür DOĞRU Istanbul Technical University	.....

Date of Submission : 04 May 2015
Date of Defense : 25 May 2015

To my dear family,

FOREWORD

The Conceptual Web Map Design and Implementation a Sample Application, ITU Ayazağa Smart Campus is a thesis project that aims to transform the Istanbul Technical University's (ITU) into a "Smart Campus". Several applications are part of the Smart Campus such as ITU Place Finder, Energy Consumption, Routes, Resources Management, and Indoor Mapping. Part of this thesis project is the creation of the web GIS map of the ITU Ayazağa campus using open source software such as QGIS, PostgreSQL/PostGIS and GeoServer.

ACKNOWLEDGMENTS

I wish to thank, first and foremost, my supervisor, ***Prof. Dr. Rahmi Nurhan ÇELİK*** for his advice, guidance and most of all, for introducing me this wonderful and exciting topic for my thesis.

I would like to thank, ***Assoc. Prof. Dr. Ziyadin Çakır*** within the National Innovation and Research Center for Geographical Information Technologies of the Istanbul Technical University (ITU), for sharing their data and most of all, for his support and advice in the development of my thesis.

To my friend, I would especially like to thank ***Mohammed Zia***, who kindly helped and motivated me throughout this Master thesis that I will never forget.

Finally thank you to ***my parents*** who have been an authentic proof of eternal love and support throughout my life.

May 2015

Rouhollah NASIRZADEH DIZAJI
(Civil Engineer)

TABLE OF CONTENTS

Page

ABBREVIATIONS

ITU : Istanbul Technical University
GIS : Geographic Information Systems
API : Application Programmer Interface
WMS : Web Map Service
WFS : Web Feature Service
OGC : Open Geospatial Consortium
DBMS : Database Management System
KML : Keyhole Markup Language
DOM : Document Object Model
EPSG : European Petroleum Survey Group
OSRM : Open Source Route Machine
QGIS : Quantum GIS
XML : Extensible Markup Language

LIST OF TABLES

Page

LIST OF FIGURES

Page

A WEB MAPPING INFRASTRUCTURE DESIGN AND IMPLEMENTATION WITH OPEN SOURCE GEO INFORMATION TECHNOLOGY.A CASE STUDY OF ITU SMART CAMPUS

SUMMARY

Large, dense places can be highly demand for immediate services. However the in Large, dense places rapid influx of people, customers, and users presents overwhelming challenges to their managements.

In a professional community with highly productive, innovative members of the architecture, planning, engineering, transportation, utilities, information technology, operations research, social sciences, geography and environmental science, public finance and policy, and communications profession, systems collaborative can be formed to foster mutual learning among them. Using of new and rich sources of information about what is going on in the city and understanding of the impact that information technology can have on the urban fabric and norms of behavior is requested.

Systems Science, in particular work on systems of systems and scaling laws, has provided new observations of urban systems at a macro-level. The Smart City provides new instrumentation that enables observation of urban systems at a micro-level. Smart Cities provide a new form of instrumentation for observing in fine detail the way that people use the city and so may enable new approaches to theories of cities.

To some extent, university's campus can be seen as "small cities," raising similar concerns for their unique populations. They contain athletic facilities, concert halls, housing, hospitals, libraries, museums, offices, parking, restaurants, stores, theaters, and, of course, classrooms. Smart campus projects are improving a lot of aspects of universities management, using the newest information and communication technologies for the benefit of the students and citizens.In this thesis, creating web Gis application of ITU Ayazağa campus is a part of a project that aims to transform the Istanbul Technical University into a "Smart Campus".

Application of geographic information today is mainly performed through Geographic Information Systems. GIS developed through the 1990s, can be described partly as a digital map producer, partly as a complex tool for problem solving, registration, unit of storage and partly as a piece of software. From a superior point of view, a combination of general information (maps) and specific information (attributes) is GIS. Many other definitions of GIS have been suggested depending on shape, function, dynamic processes and depending upon how it is used. Geographic systems are built upon a data model. A data model is the mechanism used to represent real-world objects and processes digitally in the computer system. In this way data models are closely related to the topic of data representation but usually data models are used as a formalization of representations. In a general GIS system, real-world objects are described and represented digitally, allowing for the user to see and analyze the results.

Web mapping is the process of designing, implementing, generating, and delivering maps on the World Wide Web and its products, which is the process of using maps delivered by GIS. Web GIS emphasizes geodata processing aspects more involved with design aspects such as data acquisition and server software architecture such as data storage and algorithms, than it does the end-user reports themselves.The terms web GIS and web mapping remain somewhat synonymous. Web GIS uses web maps, and end users who are web mapping are gaining analytical capabilities. The term location-based services refers to web mapping consumer goods and services. Web mapping usually involves a web browser or other user agent capable of client-server interactions. Figure1. Shows what are the main components of the web gis application system and how they are interacting with each other.

This thesis explains conceptually design of web GIS by implementing a sample application for ITU Ayazağa campus. It is demonstrating structure of web GIS application and interoperability amoung the compunends by using open source and free softwares.

ITU Ayazağa campus web GIS application is a dynamic map with a web based capability. The map with some different base layers and overlays which each layer has a feature to come up on the map. Layers can be the campus buildings such as faculties, library, banks, markets, trees or locations of car parking and so on.

There is other possibility to switch the map to the navigation map. By selecting the start and end points on menu or by droping the markers to one as the start point and

the other as end point, it will show the shortest main route on the map. It also gives more information such as the distance and duration both by car and walking between start and end points and some other tools with a differernt features.

WEB HARİTALAMA ALTYAPI TASARIMI VE AÇIK KAYNAK KODLU COĞRAFİ BİLGİ TEKNOLOJİLERİ İLE İTÜ AKILLI KAMPÜS UYGULAMASI ÇALIŞMASI

ÖZET

Geniş ve kalabalık bölgelerde anlık servislerin kullanımına duyulan ihtiyaç artmaktadır. Ancak geniş ve kalabalık bölgelerde kullanıcıların yoğun talebinin karşılanması ve yönetilmesi zorlayıcı olabilmektedir.

Alanlarında yetkin, üretken ve yenilikçi üyelerden oluşan profesyonel bir komunite ile mimarlık, şehir ve bölge planlama, mühendislik, ulaşım, kamu hizmetleri, bilgi teknolojileri, yöneylem araştırması, sosyal bilimler, coğrafya ve çevre bilimleri, iletişim, kamu maliyesi ve politika gibi alanların birlikte çalışabilirliği ve ortak öğrenim süreçleri sağlanabilir. Oluşan yeni ve zengin bilgi kaynakları kullanılarak kentin güncel durumu takip edilebilir ve bilgi teknolojilerinden yararlanarak kent dokusu ve davranış normları anlaşılabilir.

Sistem Bilimi, sistemlerin sistemi ve ölçeklendirme yasalarına göre yapılan çalışmalar kent sistemlerinde makro düzeyde yeni gözlemler sağlarken, Akıllı Kent Sistemleri ise mikro düzeyde gözlemleri olanaklı kılmıştır. Akıllı kentler, gözleme yöntemlerine getirdikleri yeni gözlem araçları sayesinde insanların kentleri kullanım şekillerini daha detaylı gözlemlemeye olanak sağlamış ve kent teorileri için yeni yaklaşımları olanaklı kılmıştır.

“Küçük kentler” olarak kabul edilebilen üniversite kampüsleri; spor alanları, tiyatro ve konser salonları, yurt ve lojmanlar, sağlık merkezleri, kütüphaneler, müzeler, ofisler, park ve dinlenme alanları, kafe ve restoranlar ile sınıf ve laboratuvarlardan oluşan kendilerine özgü yapıları için benzer gereksinimlere sahiptir. Akıllı kampüs projeleri üniversite yönetimini öğrenciler ve vatandaşlar için en yeni bilgi ve iletişim teknolojilerinden yararlanarak pek çok açıdan geliştirmektedir. Bu tezde, İstanbul Teknik Üniversitesini Akıllı Kampüse dönüştürmeyi amaçlayan projenin bir parçası olarak Ayazağa Yerleşkesi için bir Web CBS uygulaması gerçekleştirilmiştir.
Günümüzde coğrafi bilgilerle ilgili uygulamalar genel olarak Coğrafi Bilgi Sistemleri ile gerçekleştirilmektedir. 90’lı yıllarda geliştirilmeye başlayan Coğrafi Bilgi

Sistemleri, sayısal harita üretimi, karmaşık problemlerin çözülmesi, kayıt ve depolama birimlerinden oluşan bir yazılım olarak tanımlanabilmektedir. Bir başka deyişle, Coğrafi Bilgi Sistemleri genel bilgiler (harita) ve ilgili özel bilgilerin (öznitelik) birleşimidir. Coğrafi Bilgi Sistemlerinin şekline, fonksiyonlarına, dinamik süreçlere ve sistemlerin nasıl kullanıldığına göre önerilen pek çok tanım da mevcuttur. Coğrafi sistemler veri modellerine göre inşa edilmektedir. Veri modelleri ile gerçek dünya objeleri, bilgisayar ortamında sayısal olarak temsil edilir. Bu açıdan veri modelleri veri sunumu/görselleştirmesi ile doğrudan ilişkilidir, ancak veri modelleri genellikle veri sunumunun formalizasyonu için kullanılmaktadır. Coğrafi Bilgi Sistemleri ile gerçek dünya objelerinin sayısal olarak tanımlanması, kullanıcıların sonuçları görmesini ve analiz etmesini olanaklı kılmıştır.

Web haritalama; haritaların World Wild Web üzerinde tasarlanması, uygulanması, oluşturulması, sunulması ve oluşan ürünlerin Coğrafi Bilgi Sistemleri'nde kullanılması süreçlerini kapsar. Web CBS, son kullanıcıların kendileri için oluşturdukları raporlardan daha çok veri elde etme, depolama algoritmaları gibi sunucu yazılım mimarisi işlemleri ile ilgilidir. Web CBS ve web haritalama terimleri eşanlamlı olarak kullanılabilmektedir. Web CBS, web haritalarını kullanır ve web haritalama yapan son kullanıcılara analitik yetenekler kazandırır. Konum tabanlı servisler terimi web haritalama tüketim malları ve hizmetlerini tanımlamaktadır. Web haritalama, genellikle internet tarayıcısına ya da diğer istemci-sunucu etkileşimli kullanıcı ara yüzlerine gereksinim duyar. Web CBS uygulama sisteminin ana bileşenleri ve birbirleriyle etkileşimleri Şekil 1' de gösterilmektedir.

Bu tezde, İstanbul Teknik Üniversitesi Ayazağa Yerleşkesi için örnek bir web CBS uygulaması kavramsal olarak açıklanmıştır. Web CBS uygulamasının yapısı ve bileşenlerin birlikte çalışabilirliği açık kaynaklar ve ücretsiz yazılımlar kullanılarak gösterilmiştir.

İstanbul Teknik Üniversitesi Ayazağa Yerleşkesi Web CBS uygulaması; web tabanlı dinamik bir haritadır. Harita, farklı özelliklerdeki temel katmanların üst üste bindirilmesi ile oluşturulmuştur. Katmanlar, yerleşkedeki fakülte, kütüphane, banka, market gibi binaları, ağaçları, otopark alanları ve diğer yapı ve objeleri temsil etmektedir.

Harita, navigasyon haritası olarak da kullanılabilmektedir. Başlangıç ve bitiş noktalarının menüden seçilmesi ya da işaretleyicinin başlangıç ve bitiş noktaları

üzerine bırakılması ile iki nokta arasındaki en kısa ana yol hesaplanarak harita üzerinde gösterilmektedir. Ayrıca, seçilen başlangıç ve bitiş noktaları arasındaki mesafe ile araçla ve yaya olarak geçecek olan ulaşım süresi bilgisi ve daha farklı özellikler de uygulama ile kullanıcılara sunulmaktadır.

1. INTRODUCTION

1.1 Motivation

According to the fast growth and development of the urban environment and its subsidiaries, including universities, research centers, health, sports and recreation, shopping malls and stores ... the growing needs to access and rapid web-based responses, thereby need for an intelligent or smart system of visual and web-based are inevitable.

Large, dense places can be highly demand for immediate services. However, the in Large, dense places rapid influx of people, customers, and users presents overwhelming challenges to their managements.

At the same time, globalization has connected cities on opposite sides of the planet in forms of competition previously unknown for resources, and for the Creative Class. These challenges lead to approves with new methods to the planning, design, finance, construction, management, and operation of civic infrastructure and services that are extensively known as Smart Cities. Some of these methods related to appearing impress of information technology.

In a professional community with highly productive, innovative members of the architecture, planning, engineering, transportation, utilities, information technology, operations research, social sciences, geography and environmental science, public finance and policy, and communications profession, systems collaborative can formed to foster mutual learning among them. Using of new and rich sources of information about what is going on in the city and understanding of the impact, that information technology can have on the urban fabric and norms of behavior requested.

Smart Cities provide a new form of instrumentation for observing in fine detail the way that people use the city and so may enable new approaches to theories of cities.

Through new sources of information, cities hope to create insight, innovation, opportunity and real jobs that will increase prosperity and quality of life. Smart Cities is a field in want of a good theoretical base. [1]

“This is notable that revolutions in theory are often preceded by revolutions in instrumentation “(Albert Einstein). Smart Cities provide a new form of instrumentation for observing in fine detail the way that people use the city and so may enable new approaches to theories of cities. Through new sources of information, cities hope to create insight, innovation, opportunity and real jobs that will increase prosperity and quality of life. Therefore, a good theoretical base are require in a Smart Cities field.

Big universities are like small cities and the same principles of the Smart cities can applied to university campus, producing similar results for the university community. They contain athletic facilities, concert halls, housing, hospitals, libraries, museums, offices, parking, restaurants, stores, theaters, and, of course, classrooms.
Smart campus projects are improving many aspects of universities management, using the newest information and communication technologies for the benefit of the students and citizens.

The Virtual Smart Campus for the Istanbul Technical University is a project that aims to transform the Istanbul Technical University (ITU) into a “Smart Campus”. Several applications are part of the Smart Campus such as ITU Place Finder, Energy Consumption, Routes, Resources Management, and Indoor Mapping.

Istanbul Technical University (Tr. İstanbul Teknik Üniversitesi commonly referred to as ITU or Technical University) is an international technical university located in Istanbul, Turkey. It is the world's third oldest technical university dedicated to engineering sciences as well as social sciences recently, and is one of the most prominent educational institutions in Turkey. The university has 39 undergraduate, 144 graduate programs, 13 colleges, 346 labs and 12 research centers. Its student-to-faculty ratio is 12:1. [2]

The structure of each "Faculty" at ITU is comparable to those of "colleges" in the U.S. institutions, where each faculty is composed of two or more departments. For example, the Faculty of Electrical and Electronics Engineering consists of the departments of electrical engineering, control engineering, electronics engineering and telecommunications engineering.

ITU is a public (state) university. It has five campuses, 12 faculties and 5 institutes, which are located in the most important areas of Istanbul. Among ITU's five campuses, the main campus of Maslak is a suburban campus, covering a total area of 2.64 km². The University Rectorate, swimming pool, stadium, along with most of the faculties, student residence halls and the central library of ITU are located there. Another suburban campus of ITU is the Tuzla Campus. It serves the Maritime Faculty students and faculty members. It is located in the Tuzla district of Istanbul, which is a dockyard area. One of the three urban campuses of ITU is the Taşkışla (TR) Campus. It was a military barracks in the Ottoman era. Taşkışla is one of the most renowned historical buildings in Istanbul. Gümuşsuyu (Mechanical Engineering Faculty) and Maçka (Management Faculty) campuses are close to Taksim Square, and they are among the important historical buildings of Istanbul.

ITU offers many options to students who like doing extra-curricular work during their studying years. The most popular ones are Rock Club, Cinema Club, Model United Nations, EPGIK, and International Engineering Club. In addition, ITU has an option for those who like to organize events and socialize with people from various European countries in the Local Board of European Students of Technology Group, which had 40 members in 2007. Despite all these, it can still be a little quiet in the campus from time to time because students can choose the city of Istanbul over the campus life.

ITU dormitories have a capacity of 3000 students. They include Lakeside Housings, IMKB Dormitory, Verda Urundul, Ayazaga Dormitory and Gumussuyu dormitory. Additional dormitories are planned.

Ayazaga Campus, being the main campus, is located in the Maslak region, which is now the new business and trade center. Stretching over a 247-hectare area, Ayazaga campus hosts Rectorate and Administrative units along with eight of the 13 faculties and 5 of the 6 institutes. There are 576 student-capacity “Gölet Dorms” composing of 12 blocks, 1424 student-capacity “Vadi Dorms” composing of 4 blocks, 384 student-capacity “Ayazağa Dorms for Girls” composing of 3 blocks, and 220 student-capacity “Arı, Gök and Verda Üründül Dorms” offer our students a quality of life beyond the standards.

1.2 Objective

A campus is a platform where there are people spending same hours at a day many days a week. These people have a different role in the campus (students, professors, campus staff and campus managers/administrators). This platform has different expenses (energy, maintenance, information, etc. expenses) and offers same services. For these reasons, the campus platform and the city platform have several similarities.

Optimize the resources and improve the services is a key factor for the campus managers. Regarding the improvement of the services in a campus, about 80% of the data that the campus administration works with daily are georeferenced. In the same way, most of the campus information that the students use are georeferenced too.

The concept of Smart Campus appears in order to use the new technologies and apply it to improve the services and reduce the expenses. Smart Campus can improve all services and it is a powerful tool to manage all information in a campus: wastes, transports, access, classrooms and labs, energy expenditure, maintenance incidents ….

Besides this it possible to use the smart campus as attesting bench and then apply these investigations to implement smart cities with tested applications and with a background. An important part of any Smart Campus or any Smart City is the realizations of tools or application that use the new technologies.

This thesis shows the basis for creating a Web-GIS application in order to leading ITU Ayazaga campus as a smart campus. For creating a Web-GIS application first step is to create GIS data. There are many GIS software application to generate, modify and manipulate GIS data. Storing the created GIS data is the second step. In order to storing GIS data for the web there are many DBMS that support spatial data. After that GIS data stored on the database, publishing map and the data from database is the third step. To do so, there are applications which allow to share, process and edit geospatial data. Therefore, by interaction and interoperability of user interface applications, server applications and database applications enable to create ITU Ayazaga campus web GIS application.

2. Web-GIS

2.1 Anatomy of a web-mapping application

Web mapping is the process of designing, implementing, generating, and delivering maps on the World Wide Web and its products, which is the process of using maps delivered by geographical information systems (GIS). [3] Since a web map on the World Wide Web is both served and consumed, web mapping is more than just web cartography, it is both a service activity and consumer activity. To put it bluntly, it's some type of Internet application that makes use of a map. This could be a site that displays the latest geo-tagged images from Flickr, a map that shows markers of locations we've traveled to, or an application that tracks invasive plant species and displays them. If it contains a map and it does something, we could argue that it is a web map application. The term can be used in a pretty broad sense. Web-GIS emphasizes geodata processing aspects more involved with design aspects such as data acquisition and server software architecture such as data storage and algorithms, than it does the end-user reports themselves.[4] The terms web-GIS and web mapping remain somewhat synonymous. Web-GIS uses web maps, and end users who are web mapping are gaining analytical capabilities. The term location-based services refers to web mapping consumer goods and services. Web mapping usually involves a web browser or other user agent capable of client-server interactions. Figure 2.1 shows what are the main components of the web-GIS application system and how they are interacting with each other.

So how we can make our first web map easily? Thesis tried to demonstrate the web GIS structure separately and their functionality and then represented how we can put these parts in the same cycle, which mentioned in the Figure 2.1, in order to make our first web GIS application.

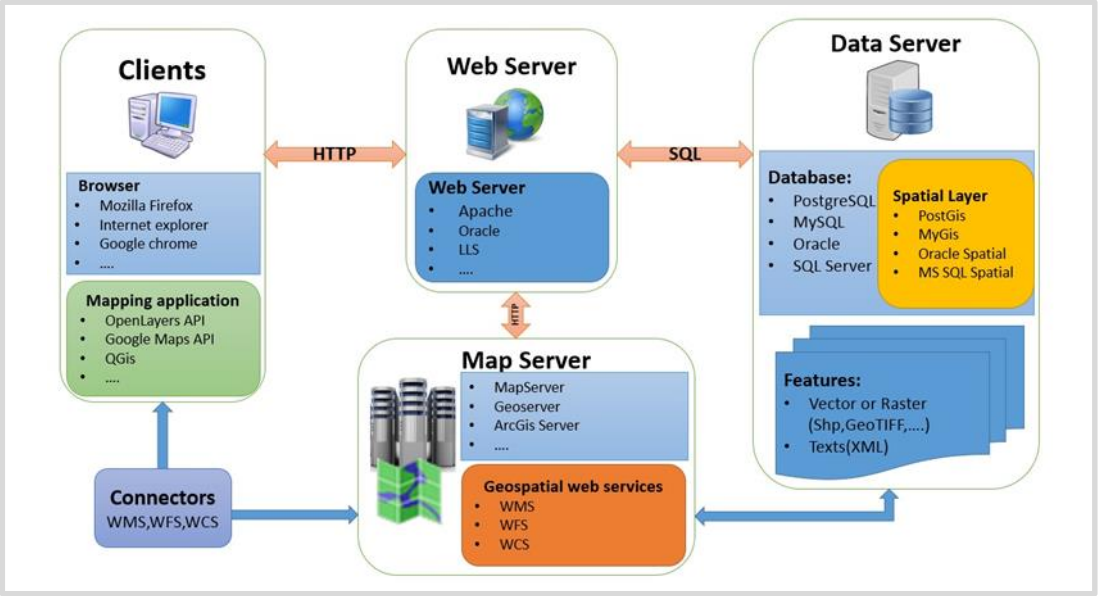

Figure 2.1: Structure of Web Mapping (Web-GIS).

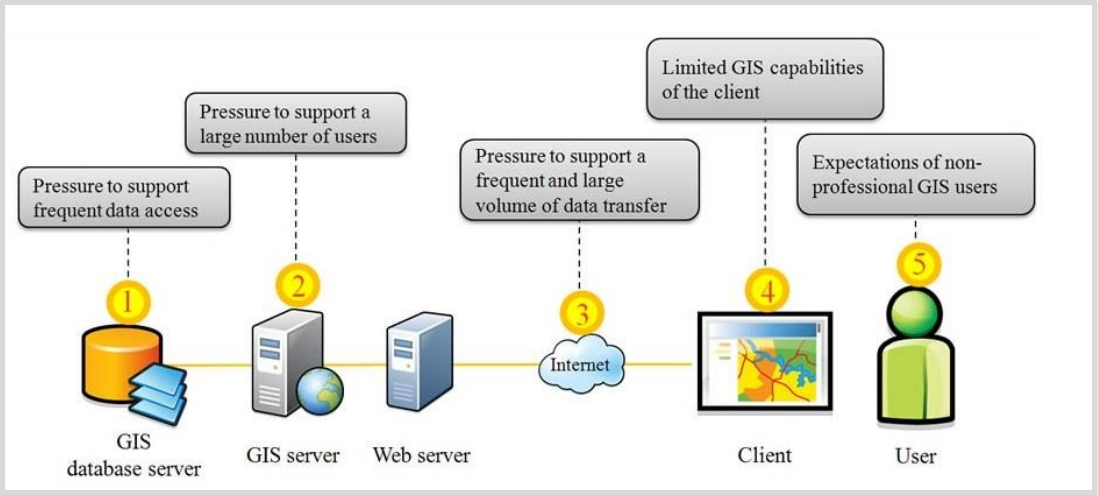

Figure 2.2: Basic System Architecture. [5]

Within the past few years, the popularity of interactive web maps has exploded. In the past, creating interactive maps was reserved for large companies or experts with lots of money. [6] But now, with the advent of free services like Google and Yahoo! Maps, online mapping is easily accessible to everyone. Today, with the right tools, anyone can easily create a web map with little or even no knowledge of geography, cartography, or programming. Web maps are expected to be fast, accurate, and easy to use. Since they are online, they are expected to be accessible from anywhere on nearly any platform. There are only a few tools that fulfill all these expectations.

OpenLayers is one such tool. It's free, open source, and very powerful. It's providing both novice developers and seasoned GIS professionals with a robust library, OpenLayers makes it easy to create modern, fast, and interactive web-mapping

applications. OpenLayers is a powerful, community driven, open source, pure JavaScript web-mapping library. With it, we can easily create our own web map mashup using WMS, Google Maps, and a myriad of other map backends.

2.2 Realization of the System

2.2.1 Software and data

What we need to know before going through? The only thing we will need for this is a computer and text editor. The operating system will come with a text editor, and any will do, but if we are using Windows, it recommended using Notepad++, VI if we are using Linux, and Textmate if on OSX. We are not required more knowledge of Geographic Information Systems (GIS), or too much extensive JavaScript experience. A basic understanding of JavaScript syntax and HTML / CSS will help to understanding the material, but having good knowledge about them is not required. An Internet connection will be required to view the maps, and we will need a modern web browser such as Firefox, Google Chrome, Safari, or Opera. While a modern browser is required to get the most of the library, OpenLayers even provides support for non-standards based browsers such as Internet Explorer (even IE6, to some extent).

What is OpenLayers? OpenLayers is an open source, client side JavaScript library for making 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; we can use it without even downloading anything! Originally developed by Metacarta, as a response, in part, to Google Maps, it has grown into a mature, popular framework with many passionate developers and a very helpful community.

OpenLayers makes creating powerful web-mapping applications easy and fun. It is very powerful but also easy to use we do not even need to be a programmer to make a great map with it. It is open source and free.
OpenLayers allows us to build entire mapping applications from the ground up, with the ability to customize every aspect of our map layers, controls, events, etc. we can use a multitude of different map server backends together, including a powerful vector layer. It makes creating map 'mashups' extremely easy.

OpenLayers provides a powerful, but easy-to-use, pure JavaScript and HTML toolkit to quickly make crossbrowser web maps. A basic understanding of JavaScript will be helpful, but there is no prior knowledge required to use this book. If we have never

worked with maps before, this thesis will introduce us to some common mapping topics and gently guide us through the OpenLayers library.

However, what is the technical meaning of OpenLayers? Previously it mentioned that OpenLayers is a client side JavaScript library.

2.2.2 Client side

Once we are saying client side we are referring to the user's computer, specifically their web browser. The only thing we need to have to make OpenLayers work is the OpenLayers code itself and a web browser. we can either download it and use it on our computer locally, or download nothing and simply link to the JavaScript file served on the site that hosts the OpenLayers project .[6] OpenLayers works on nearly all browsers and can served by any web server or our own computer. It is usable in a modern, standard-based browser such as Firefox, Google Chrome, Safari, or Opera also can be recommend.

Nevertheless, whenever we say library we mean that OpenLayers is an **API** that provides us with tools to develop our own web maps. Instead of building a mapping application from scratch, we can use OpenLayers for the mapping part, which maintained and developed by a bunch of brilliant people. For instance, if we wanted to write a blog we could either write our own blog engine, or use an existing one such as WordPress or Blogger and build on top of it. Similarly, if we wanted to create a web map, we could write our own from scratch, or use software that has been developed and tested by a group of developers with a strong community behind it. By choosing to use OpenLayers, we have to learn how to use the library.

OpenLayers is written in **JavaScript**, but do not worry if we do not know it very well. All things that we really need is some knowledge of the basic method.

2.2.3 Web map client

So now, we know OpenLayers is a client side mapping library.

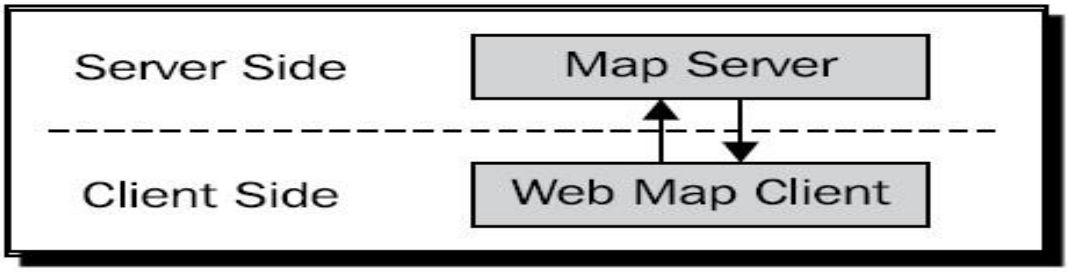

Figure 2.3: Core of web map application. [7]

Figure 2.3, is called the **Client / Server Model** and it is, essentially, 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., a WMS server or the Google Maps backend).

OpenLayers lives on the client side. One of the primary tasks the client performs is to get map images from a map server. Essentially, the client has to ask a map server for what we want to look at. Every time we navigate or zoom around on the map, the client has to make new requests to the server because we are asking to look at something different. OpenLayers handles this all for us, and it is happening via asynchronous JavaScript (AJAX) calls to a map server. To reiterate the basic concept is that OpenLayers sends requests to a map server for map images every time we interact with the map, then OpenLayers pieces together all the returned map images so it looks like one big, seamless map.

2.2.4 Server side

A **web server** is a computer system that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. The term can refer either to the entire system, or specifically to the software that accepts and supervises the HTTP requests. [8]

The most common use of web servers is to host websites, but there are other uses such as gaming, data storage, running enterprise applications, handling email, FTP, or other web uses. The primary function of a web server is to store, process and deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML

documents, which may include images, style sheets and scripts in addition to text content.

A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that the behavior of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

Web servers are not always used for serving the World Wide Web. They can also be found embedded in devices such as printers, routers, webcams and serving only a local network. The web server may then be used as a part of a system for monitoring and/or administering the device in question. This usually means that no additional software has to be installed on the client computer, since only a web browser is required (which now is included with most operating systems). [9]

2.2.5 Web map server

A **map server** (or map service) provides the map itself. There are a myriad of different map server backends. A small sample includes WMS, Google Maps, Yahoo! Maps, ESRI ArcGIS, WFS, and OpenStreetMaps. [10]The basic principle behind all those services is that they allow us to specify the area of the map we want to look at (by sending a request), and then the map servers send back a response containing the map image. With OpenLayers, we can choose to use as many different backends in any sort of combination as we would like. OpenLayers is not a web map server; it only consumes data from them. Therefore, we will need to be able to access some type of

web map service. Fortunately, there are numerous of free and/or open source web map servers available that are remotely hosted or easy to set up our self, such as GeoServer.

With many web map servers we do not have to do anything to use those, just supplying a URL to them in OpenLayers is enough. OSGeo, OpenStreetMaps, Google, Yahoo!, and Bing Maps, for instance, provide access to their map servers (although, some commercial restrictions may apply with various services in some situations).

GeoServer (one of the most popular and open source map servers) which it is applied in this thesis for web mapping application, explained in detailed in sections 6.3.

2.2.6 DBMS

Briefly, a DBMS is a database program. Technically speaking, it is a software system that uses a standard method of cataloging, retrieving, and running queries on data. The DBMS manages incoming data, organizes it, and provides ways for the data to be modified or extracted by users or other programs.

There many DBMS's but some more known examples include MySQL, PostgreSQL, Microsoft Access, Microsoft SQL Server, SQL Server, FileMaker, Oracle, RDBMS, dBASE, Clipper, FoxPro, SAP and IBM DB2.[10] Since there are so many database management systems available, it is important for there to be a way for them to communicate with each other. For this reason, most database software comes with an Open Database Connectivity (ODBC) driver that allows the database to integrate with other databases. For example, common SQL statements such as SELECT and INSERT are translated from a program's proprietary syntax into a syntax other databases can understand. Database management systems are often classified according to the database model that they support; the most popular database systems since the 1980s have all supported the relational model as represented by the SQL language. Sometimes a DBMS is loosely referred to as a 'database'.

PostgreSQL (one of the most popular and open source DBMS) which it is applied in this thesis for web mapping application, explained in detailed in sections 6.2.1.

2.2.7 Web Map services

The Open Geospatial Consortium (OGC) has defined Web Mapping Service (WMS) specification protocols for the transfer of geospatial data from servers to client applications. A Web Map Service (WMS) produces an image (e.g. GIF, JPG) of

geospatial data. In addition, vector graphics can be included: such as points, lines, curves and text, expressed in SVG or WebCGM format. [11]

WMS gives clients access to an image of data without storing a local copy of the dataset. Using a specially structured HTTP request, we can call a dataset (e.g. orthoimagery, highways) into our GIS software application. Receiving an image of the data works well for imagery and other raster data, but means that standard GIS analysis is not feasible using WMS. Web Feature Services (WFS) protocols, on the other hand, do support actual data feature transfer, but very few services currently exist.

2.2.7.1 Map service capabilities

When we publish a map service, we can enable capabilities such as WMS, KML and others. Capabilities create additional services that work from or with the map service. They allow users to access our map in an expanded variety of applications and devices. Capabilities also allow users to do a greater variety of things with our map service, such as network analysis and geoprocessing.

Some capabilities require that our map document contain specific types of layers. The available capabilities also depend on which edition of servers we have and whether our map service is based on an .msd or a .mxd. Table 2.1 lists the capabilities that are available with map services and any special requirements for enabling the capability. This help system contains topics on each of the service types created by these capabilities.

Table 2.1: Lists of the capabilities that are available with map services.

Capability	What it does	Special requirements
Mapping	Provides access to the contents of a map document.	This capability is always enabled for any map document.
WCS	Uses the raster layers in the map document to create a service compliant with the Open Geospatial Consortium's (OGC) Web Coverage Service (WCS) specification.	Requires raster layers.
WFS	Uses the layers in a map document to create a service compliant with the Open Geospatial Consortium's (OGC) Web Feature Service (WFS) specification.	Requires vector layers. Raster layers are not included in the service, since the purpose of WFS is to serve vector feature geometry.
WMS	Uses a map document to create a service compliant with the Open Geospatial Consortium's (OGC) Web Map Service (WMS) specification.	None
KML	Uses a map document to create Keyhole Markup Language (KML) features.	None
Geodata Access	Allows an end user to perform replication and data extraction in ArcMap.	Requires a layer from a geodatabase. Creates a geodata service that works with the map service.
Geoprocessing	Provides access to geoprocessing models from a tool layer. A tool layer represents a model that has been dragged from ArcToolbox into a map document's table of contents.	Requires a tool layer. Creates a geoprocessing service that works with the map service.

3. Procedures for creating web GIS application

In this part, procedures for creating web app is presented step by step, which is slightly in the simple form and we have tried to make sufficient description (e.g., OpenLayers, method, parameters,..) in order to achieve to the goal which is to create GIS web-application for Istanbul Technical University (ITU) Ayazaga (Maslak)campus. It was proposed to use ITU GIS web-application as a part of GIS ITU smart campus project in Turkey.

In the following we are going through the OpenLayers and make a map:

First of all as it mentioned little about APIs on the last part, there are some more information about APIs. Relation to Google / Yahoo! / and other mapping APIs which:

The Google, Yahoo!, Bing, and ESRI Mappings API allow us to connect with their map server backend. Their APIs also usually provide a client side interface (at least in the case of Google Maps). The Google Maps API, for example, is fairly powerful, so that we have the ability to add markers, plot routes, and use KML data (things that it is possible to do also in OpenLayers) but the main difficulty is that our mapping application relies totally on Google. The map client and map server are provided by a third party. This is not inherently a bad thing, and for many projects, Google Maps and the like are a good fit.

However, there are quite a few problems.

- We are not in control of the backend
- we can't really customize the map server backend, and it can change at any time
- There may be some commercial restrictions, or some costs involved
- These other APIs also cannot provide us with anything near the amount of flexibility and customization that an open source mapping application framework (i.e., OpenLayers) offers

OpenLayers allows us to have multiple different 'backend' servers that our map can use. To access a web map server, we create a layer object and add it to our map with OpenLayers. For example, if we wanted to have a Google Maps and a WMS service

displayed on our map, we would use OpenLayers to create a GoogleMaps layer object and a WMS layer object, and then add them to our OpenLayers map.

Like layers of an onion, each layer is above and will cover up the previous one; the order that we add in the layers is important. With OpenLayers, we can arbitrarily set the overall transparency of any layer, so we are easily able to control how much layers cover each other up, and dynamically change the layer order at any time. For instance, we could have a Google map as our base layer, a layer with satellite imagery that is semi-transparent, and a vector layer all active on our map at once. A vector layer is a powerful layer that lets us add markers and various geometric objects to our maps.

For starting to create our first map we need to download the OpenLayers library. For downloading the OpenLayers library we should Go to the OpenLayers website (*http://openlayers.org*) and download the.*zip* version (or if we prefer the *.tar.gz* version). After we're done, we should have the OpenLayers library files set up on our computer. For this we extract the file just downloaded. Once we extract it, we will end up with a folder called OpenLayers-2.10 (or whatever our version is).

Then open up the OpenLayers folder. Once inside, we'll see a lot of folders and files, but the ones we are concerned with right now is a file called *OpenLayers.js* and two folders, */img* and */theme*. We'll be copying these to a new folder. Figure 3.1

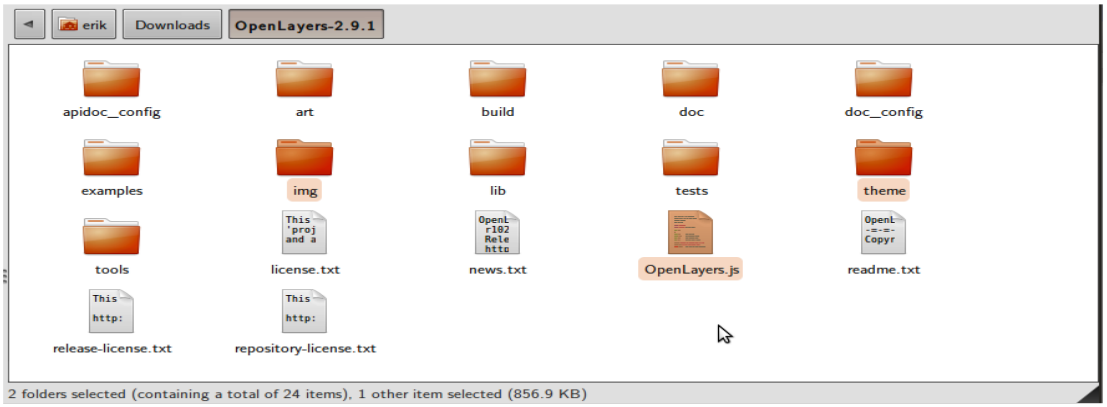

Figure 3.1: Inside the OpenLayers folder that we downloaded.

At this time we need to create a new folder outside the OpenLayers directory; we'll use *~/code/* (if we are on Windows, then *c:/code*). We can name the folder whatever we like, but we'll refer to it as the code folder. Inside the code folder, copy over the *OpenLayers. js* and two folders *(/img and /theme*) from the previous step. Our new folder structure should look similar to Figure 3.2.

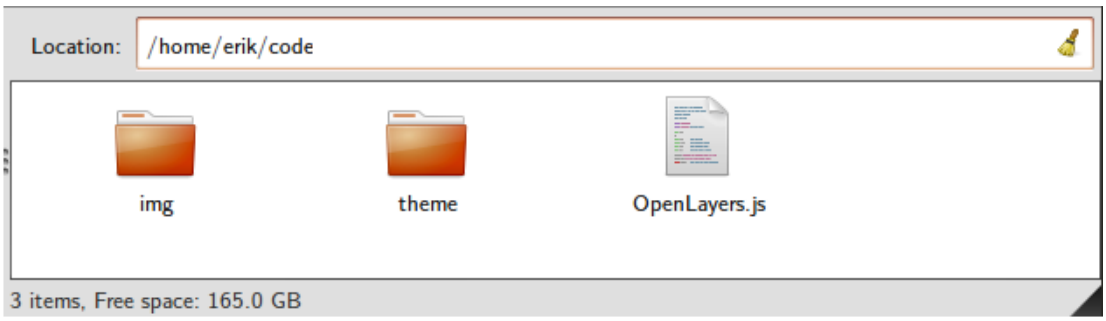

Figure 3.2: Creating new folder and selecting the folders just we need to do.

If we open the *OpenLayers.js* file, we will notice it is nearly unreadable. This is because this is a minified version, which basically means extra white space and unnecessary characters have been stripped out to cut down on the file size. While it is no longer readable, it is a bit smaller and thus requires less time to download. If we want to look at the uncompressed source code, we can view it by looking in the OpenLayers source code folder we extracted.

Now we are ready to making our map, but for creating a map with OpenLayers, at a minimum, we require to do following things:

- Including the OpenLayers library files
- Creating an HTML element that the map will appear in
- Creating a map object from the Map class
- Creating a layer object from a Layer class
- Adding the layer to the map
- Defining the map's extent (setting the area the map will initially be displaying)

After we did above mentioned steps, we are ready to crate our map.

It will help us to do (create a map) without errors by implementing the following steps one by one so for this;

1. Navigate to the code directory that contains the *OpenLayers.js* file, */img* and */ theme* directories. Create a file here called *index.html*. This directory (*/code*) will be referred to as our root directory, because it is the base (root) folder where all our files reside.

2. Add in the following code to *index.html* and save the file as an *.html* file, if we are using Windows, it will be good to using Notepad++. We will use the following code also as the base template code for other future examples and so we'll be coming back to it many times.

3. We will write the code on our Notepad++ editor as following;

```
<!DOCTYPE html>
<html lang='en'>
<head>
    <meta charset='utf-8' />
    <title>My OpenLayers Map</title>
    <script type='text/javascript' src='OpenLayers.js'></script>
     <script type='text/javascript'>

    var map;

function init () {
    map = new OpenLayers.Map('map_element', {});
    var wms = new OpenLayers.Layer.WMS(
    'OpenLayers WMS',
    'http://vmap0.tiles.osgeo.org/wms/vmap0',
    {layers: 'basic'},
    {}
);

    map.addLayer(wms);
    if(!map.getCenter()){
     map.zoomToMaxExtent();
 }
}

</script>
</head>

<body onload='init();'>
    <div id='map_element' style='width: 500px; height: 500px;'>
    </div>
</body>
</html>
```

4. We almost done and now we should go to open up *index.html* in our web browser. We should see something similar to Figure 3.3.

Figure 3.3: Creating map by using OpenLayers.

By this way we could just create our first map using OpenLayers, If it did not work for some reason, we need to try double checking the code and making sure all the commas and parentheses are in place.

In our map we put the control on the left side (the navigation buttons) which is called the PanZoom control. We can click the buttons to navigate around the map, drag the map with our mouse/use the scroll wheel to zoom in, or use our keyboard's arrow keys.

As this is the first map that we created so it’s worth to look at the code and explain line by line to better understanding the code. Before we do that, we’ll include a quick reference to the line numbers at which the requirements from the previous section occur at. These are the core things that we need to do to have a functioning map. It’ll denote line numbers with brackets[x], where x is the line number.

1. Including the OpenLayers library files:

Line [6]

```
<script type=’ text/javascript’  src=’ OpenLayers.js’ ></script>
```

2. Creating an HTML element for our map:

Lines [30] and [31]

```
        <div id=' map_element'  style=' width: 500px; height: 500px' >
</div>
```

1. Creating a map object from the Map class:

   Line [12]

```
        map = new OpenLayers.Map( 'map_element' , { });
```

2. Creating a layer object from a Layer class:

   Lines [13] to [18]

```
        var wms_layer = new OpenLayers.Layer.WMS(
                'WMS Layer Title' ,
                'http://vmap0.tiles.osgeo.org/wms/vmap0' ,
               {layers:  'basic' },
        {}
);
```

3. Adding the layer to the map:

   Line [20]

```
        map.addLayer(wms_layer);
```

4. Defining the map's extent:

   Lines [21] to [23]

```
        if(!map.getCenter()){
          map.zoomToMaxExtent();
}
```

In lines [1] to [5] they are using to set up the HTML page. Every HTML page needs an <html> and <head> tag, and the extraneous code we see specifies various settings that inform our browser that this is an HTML5 compliant page. For example, we include the *DOCTYPE* declaration in line [1] to specify that the page conforms to standards set by the WC3. We also specify a <title> tag, which contains the title that will be displayed on the page.

Line [6] **:** `<script type='text/javascript' src='OpenLayers.js'></script>`

This line includes the OpenLayers library. The location of the file is specified by the src='OpenLayers.js' attribute. We're using a relative path as the index. Html page is in the same folder as the OpenLayers.js file, we don't have to worry about specifying the path to it. The file could be either on our computer or another computer, it doesn't matter much, as long as the browser can load it. We can also use an absolute path, which means we pass in a URL that the script is located at. OpenLayers.org hosts the script file as well; we could use the following line of code to link to the library file directly:

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

Notice that how the *src* specifies an actual URL, this is how we use absolute paths. Either way works, however, throughout this thesis but not in all of it, we'll use relative path and we'll have the OpenLayers library on own computer/server. But while we are using the hosted OpenLayers library, we cannot be sure that it will always be available, and it may change overnight (and changes when the library is updated), so using a local copy is good idea. After some progresses we'll try to use an absolute path which is related to MapServer.

Line [7]: Starts a <script> block. We will set up all our code inside it to create our map. Since the OpenLayers library has been included in line [5], we are able to use all the classes and functions the library contains.

Line [8]: `var map;`

Here we create a global variable called map. In JavaScript, anytime we create a variable we need to place `var` in front of it to ensure that we do not run into scope issues (what functions can access which variables). When accessing a variable, we do not need to put var in front of it. Since defining map as a variable at the global level (outside of any functions), we can access it anywhere in the code. Later we will make this map variable on map object, but right now it is just an empty global variable.

Line [11]: by this line we are create a function called *init*. When the page loads (via body `onload='init();'` on line [29]), this function will get called. This function contains all of our code to set up our OpenLayers map. If we are familiar with JavaScript, we do not have to put all the code in a function call we could, for instance, just put the code at the bottom of the page and avoid a function call all together.

Creating a function that gets called when the page loads is a common practice and so we will be doing it throughout in this thesis.

Line [12]: `map = new OpenLayers.Map('map_element', { });`

If you remember that global map variable, now we're making it a map object, created from the `OpenLayers.Map` class. It is also referred to as an instance of the Map class. The map object is the crux of our OpenLayers application, we call its functions to tell the map to zoom to areas, fire off events, keep track of layers, etc. in right hand side of the equal sign (=): new means that we are creating a new object from the class that follows it. `OpenLayers.Map` is the class name which we are creating an object from. Notice that something is inside the parenthesis: `('map_element', {})`. This means we are passing two things into the class (called arguments, and we pass them in separated by a comma). Every class in OpenLayers expects different arguments to be passed into it, and some classes don't expect anything.

The Map class expects two parameters. The first argument, map_element, is the ID of the HTML element that the map will appear in. The second argument, `{ }`, are the map options, consisting of key: value pairs (`e.g., {key: value}`). Because we passed in map_element as the first parameter, we will have an HTML element (almost always a) with the ID of map_element. The HTML element ID can be anything, but for the sake of clarity and to avoid confusion, we call it `map_element`.

Line [13]: `var wms = new OpenLayers.Layer.WMS(`

Here, we create a layer object for the map to use from the WMS subclass of the Layer class. In OpenLayers, every map needs to have at least one layer. The layer points to the 'back end', or the server side map server, as we discussed earlier. The layer can be any of a multitude of different services, but we are using WMS here. WMS, which stands for Web Map Service, is an international standard defined by the Open Geospatial Consortium (OGC). The arguments we can pass in for layers are dependent on the layer class.

Notice that we don't include everything on one line when creating our layer object, this improves readability, making it easier to see what we pass in. The only difference is that we are also adding a new line after the commas which separate arguments, which doesn't affect the code (but does make it easier to read).

Line [14]: 'WMS Layer Title',

This is the first parameter passed in; the layer's title. Most layer classes expect the first parameter passed in to be the title of the layer. This title can be anything we would like, the main purpose of it is for human readability.it is displayed in controls such as the layer list.

Line [15]: 'http://vmap0.tiles.osgeo.org/wms/vmap0',

The URL is the second parameter that the WMS layer class we need to put in our code. For now, we're using a publicly available WMS service from OSGeo. I will cover in depth the WMS later. For now, all we need to know is that this is the base URL, which the layer will be using.

Line [16]: {layers: 'basic'},

The third parameter is an anonymous object containing the layer properties (similar in format to the previous options object on line [12]), and is specific to the WMS layer class. These are the things that are actually added (more or less) straight into the GET call to the map server backend when OpenLayers makes requests for the map images.

Line [17]: { }

The fourth parameter is an optional options object, an anonymous object in the format we just discussed. These properties are generally shared by every OpenLayers Layer class. For instance, regardless of the Layer type (e.g., WMS or Google Layer), we can pass in an opacity setting (e.g., {opacity: .8} for 80 percent opacity). So, regardless of whether we are working with a WMS or a Vector layer, this opacity property can apply to either layer.

Since this is the last thing passed into the Layer object creation call, we must be sure there is not a leading trailing comma. Trailing commas are a common error and are often tedious to debug. This options object is optional, but we will often use it, so it's a good habit to keep our code consistent and provide an empty object (by {}), even if we aren't passing anything into it yet.

Line [18]:);

This simply finalizes the object creation call.

Line [20]: map.addLayer(wms);

Now that we have a wms_layer object created, we need to add it to the map object. Notice we are calling a function of the map object. There are actually a few ways to go about adding a layer to a map object. We can use the above code (by calling map.addLayer), where we pass in an individual layer, or we could use map.addLayers:

```
map.addLayers( [layer1, layer2, ...] );
```

Here, we pass an array of layers. Both methods are equally valid, but it may be easier to pass in an array when we have multiple layers. We can also create the layer objects before we create the map object and pass the layer objects into the map when we create it, for instance:

```
map = new OpenLayers.Map('map_element', {layers: [layer1,
layer2,
…]});
```

All ways are valid, but we will usually use addLayer or addLayers throughout thesis. Line [21] - [23]:

```
if(!map.getCenter()){
map.zoomToMaxExtent();
}
```

Finally, we must specify the map's viewable area. Here, the actual code that moves the map is map.zoomToMaxExtent(), which zooms the map to the map's maximum extent. It is inside an if statement. if statement checks to see whether the map already has a center point. The reason why we add in this check is because, by default, our map can accept a specially formatted URL that can contain an extent and layers to turn on/off. This is, in more common terms, referred to as a permalink. If we did not check to see if a center has already been set, permalinks would not work.

There are a few ways to set the map's extent. If we know that we want to show everything, the map.zoomToMaxExtent() function is a quick and good way to do it. There are other ways as well, such as:

```
map.zoomToExtent(new OpenLayers.Bounds([minx,miny,maxx,maxy]);
```

There are even more ways though. If we know a specific location we want the map to start at, this is another way to do it:

```
    map.setCenter(new OpenLayers.LonLat(x,y));
    map.zoomTo(5);
```

Where x, y are the Lon/Lat values, and 5 is the zoom level we wish to zoom to. By default, our map will have 16 zoom levels, which can be configured by setting the numZoomLevels property when creating our map object.

There are many ways, but these are the most common strategies. The basic idea is that we need to specify a center location and zoom level, setting the extent accomplishes this, as does explicitly setting the center and zoom level.

Line [24]: }

This simply finishes the init() function.

Lines [26], [27]:

These lines close the script tag and head tag.

Line [29] : <body onload='init();'>

This starts the body tag. When the page is finished loading, via the onload='init();' attribute in the body tag, it will call the JavaScript init() function. We have to wait until the page loads to do this because we cannot use the map div (or any HTML element) until the page has been loaded. Another way to do this would be to put the init() call in a JavaScript tag at the bottom of the page (which would not be called until the page loads), but both methods accomplish the same thing.

When browsers load a page, they load it from top to bottom. To use any DOM elements (any HTML element on our page) in JavaScript, they first have to be loaded by the browser. So, we cannot reference HTML with JavaScript before the browser sees the element. It'd be similar to trying to access a variable that hasn't yet been created. Even though we have JavaScript code that references the map_element div at the top of the page, it is not actually executed until the page is loaded (hence the need for the onload and init() function call).

Line [30] and [31]:

```
<div id='map_element' style='width: 500px; height:500px'></div>
```

To make an OpenLayers map, we need an HTML element where the map will be displayed in. Almost always this element will be a div. we can give it whatever ID we

would like, and the ID of this HTML element is passed into the call to create the map object. We can style the div however we would like setting the width and height to be 100 percent, for instance, if we wanted a full page map. It would be best to style the elements using CSS, but styling the div in line like this works as well.

Lines [32] and [33]: These lines finalize the page by closing the remaining tag.

4. The 'Layers' in OpenLayers

Maps can contain a very great amount of information, but some maps don't show enough, like the first map that we created previously. Detecting what information to display on a map is certainly an art form, and creating printed maps with just the right balance of information is quite difficult.

Fortunately, creating maps for the web is slightly easier in this respect, because we can let the user determine what information they want to see. Imagine two people looking at a city map, one person just cares about the bus routes, while the other wants to only know about bicycle routes. Instead of creating two maps, we could create a single map with two different layers, one for each route. Then the user can decide if they want to see the bus routes, bicycle routes, both, or none at all. OpenLayers provides us with a variety of layer types to choose from and use. We can do all sorts of things, such as changing layer opacity, turning the layers on or off, changing the layer order, and much more.

A layer is basically a way to show multiple levels of information independent of each other. Layers are not just a mapping or cartography concept; graphic designers and digital artists make heavy use of layers.

Imagine a printed-out map of a city. We also have two sheets of transparent paper. One sheet has blue lines that indicate bus routes, and the other sheet contains green lines that indicate bicycle routes. Now, if we placed the transparent sheet of paper with bicycle routes on top of the map, we would see a map of the city with the bicycle routes outlined. Putting on or taking off these transparent pieces of paper would be equivalent to turning a layer on or off. The order we place the sheets on top of each other also affects what the map will look like, if two lines intersect, we would either see the green line on top or the blue line on top. That's the basic concept of a layer.

OpenLayers is a JavaScript framework, and when we want to actually create a layer, we create (or instantiate) an object from an OpenLayers Layer class. OpenLayers has many different Layer classes, each allowing us to connect to a different type of map server 'back end.' For example, if we wanted to connect to a WMS map server, we would use the `Layer.WMS` class, and if we wanted to use Google Maps we'll be use

the `Layer.Google` class. Each layer object is independent of other layer objects, so doing things to one layer won't necessarily affect the other.

The safest maximum amount of layers we can have on a map at one time depends largely on the user's machine (i.e., their processing power and memory). Too many layers can also overwhelm users; many popular web maps (e.g., Google and Yahoo!) contain just a few layers.

As it is clear from last paragraph creating how many layers it depends of our web map application, but we will need at least one layer to have a usable map. An OpenLayers map without any layers would be sort of like an atlas without any maps. Therefor we need at least one layer, at least one Base layer. All other layers that 'sit above' the base layer are called Overlay layers. So these are the two 'types' of layers in OpenLayers.

4.1 Base layer

A base layer is at the very bottom of the layer list, and all other layers are on top of it. This would be our printed out map as we did in the earlier example. The order of the other layers can change, but the base layer is always below the overlay layers. By default, the first layer that we add to our map acts as the base layer. We can, however, change the property of any layer on our map to act as the base layer (by setting the `isBaseLayer` property to `True`). We may also have multiple base layers. However, only one base layer can be active at a time.

When one base layer is turned on, all the other base layers are turned off. Base layers are similar to radio buttons, only one can be active at a time.

4.2 Overlay layers

Any layer that is not a base layer is called an overlay layer. Overlay layers (non base layers), however, do not behave like Base layer, turning on or off overlay layers will not affect other overlay layers. Overlay layers are similar to check boxes, we can have as many on or off as we would like. As we talked about, the order that we add layers to our map is important. Every time we add a layer to the map, it is placed above the previous one.

Now let us try to creating a map with multiple layers; at this time, we want to create a map with to WMS layers which one layer will act as our base layer, and the other will be an overlay layer containing labels for country, state, and city names.

We will act as following:

1. We‘ll create a copy of the file that we made for the last example and remove the existing WMS layer code. We can name it whatever we would like. We should be sure that, it is in the same directory as my OpenLayers. js file.
2. First we are going to remove everything that was in the init() function. The function should now look like this:

```
function init() {
}
```

3. Then , inside the init() function, we are going to setup the map object like before:

```
map = new OpenLayers.Map('map_element', {});
```

4. Latter we'll create the first layer and we'll use a WMS layer and ask the WMS server for the layer 'basic' (a layer on the WMS service). We'll also explicitly set it to be a base layer.

```
var wms_layer_map = new OpenLayers.Layer.WMS(
      'Base layer',
      'http://vmap0.tiles.osgeo.org/wms/vmap0',
      {layers: 'basic'},
      {isBaseLayer: true}
);
```

5. Now creating a second layer object. It will also be a WMS layer. This time, we are going to do few different layers from the WMS service a bunch of labels. We are also going to set the transparent property to true, so the map images which the server sends back will be transparent. We'll also set the opacity to be 50 percent (by setting the opacity to .5). This layer will be an overlay layer.

```
var wms_layer_labels = new OpenLayers.Layer.WMS(
      'Location Labels',
      'http://vmap0.tiles.osgeo.org/wms/vmap0',
      {layers: 'clabel,ctylabel,statelabel',
      transparent: true},
      {opacity: .5}
);
```

6. At this time we need to add the layers to the map. We'll use the addLayers function and pass in an array of layer objects.

```
map.addLayers([wms_layer_map, wms_layer_labels]);
```

7. Now, we'll add a *Layer Switcher* control that will show us the layers on the map.

```
        map.addControl(new
OpenLayers.Control.LayerSwitcher({}));
```

8. Finally, we need to set the map center information. All our maps will need to have their extent set somehow, and this is one standard way to do so.

```
        if(!map.getCenter()){
                map.zoomToMaxExtent();
        }
```

9. After that saving the file, then we'll open it up in web browser (preferably Firefox) because we are just working with HTML and JavaScript, we don't need to place this on a server or anything. We can simply open the file with our web browser directly from the folder. Therefore, we should see something like Figure 4.1.

Figure 4.1: Map with two layers (Base layer and Overlay).

We just created a map with two WMS layers and a **Layer Switcher Control** that allows us to turn on and off layers. **Controls** are what OpenLayers provides that allows us to actually interact with the map and layers. To use controls, we create objects from different Control classes. By default, all maps get a Navigation control object which allows us to pan and zoom the map.

4.2.1 Creating layer objects

There are two steps to work with layers which the first one is create the layer object and second is adding the layer object to the map. We can use either `map.addLayer`(layer) to add an individual layer, or `map.addLayers([layer1, layer2, ...])` to add an array of layers, like in the previous example.

These two steps can actually be combined into one step (by instantiating the layer object when calling the addLayer function. By now, we have a bit of experience instantiating objects from the WMS Layer class. By looking at the code that creates `wms_base layer` object.

```
var wms_layer_map = new OpenLayers.Layer.WMS(
            'Base layer',
            'http://vmap0.tiles.osgeo.org/wms/vmap0',
            {layers: 'basic'},
            {isBaseLayer: true}
     );
```

Each item inside the parentheses, after `OpenLayers.Layer.WMS` (, are called arguments which we pass in while creating the object. But how can we find what arguments to pass in?

We did not write the class, so we do not know what it expects to take in. Therefore, as with nearly any third party library, we have to refer to the documentation to see what arguments the class expects. OpenLayers has great documentation.

Layer.WMS class:

There are a lot of information in the API docs, but let us look specifically at the section titled Constructor. This will tell us how to create a WMS layer object. The process varies for different types of layer but focus on the WMS class for now. In Table 4.1 in the Parameters sub section specifies what arguments this specific class expects to take in.

Table 4.1: WMS layer parameters sub section.

Parameters	Description
name	{String} A name for the layer.
url	{String} Base url for the WMS (e.g. http://vmap0.tiles.osgeo.org/wms/vmap0).
params	{Object} An object with key/value pairs representing the GetMap query string parameters and parameter values.
options	{Object} Hashtable of extra options to tag onto the layer.

The parameters tell us about the order of arguments to pass in, and what each argument means. The word in between the curly brackets ({ }) refers to the data type of parameter. Therefore, {string} means the parameter should be a string.

Now we'll get a bit more familiar with it by configuring the options parameter which is something that is present all layer classes. For this we need to do as following;

1. Again we need to use the template that we were created in the first example with the WMS layer code removed. We'll be adding some WMS layers, and we'll refer to this file as previous example.
2. First we'll add in a layer that contains the 'basic' layer from the WMS server, like in the previous example.

   // Setup our two layer objects

```
        var wms_layer_map = new OpenLayers.Layer.WMS(
          'Base layer',
          'http://vmap0.tiles.osgeo.org/wms/vmap0',
          {layers: 'basic'},
    {isBaseLayer: true}
    );
```

3. After that we are going to create another layer object. We'll create a layer using labels, like in the previous example. This time though, we set the layer's options to include `visibility: false`. This will cause the layer to be hidden by default. The layer definition should now look like:

```
        var wms_layer_labels = new OpenLayers.Layer.WMS(
           'Location Labels',
           'http://vmap0.tiles.osgeo.org/wms/vmap0',
           {layers: 'clabel,ctylabel,statelabel',
           transparent: true},
           {visibility: false, opacity:0.5}    );
```

4. Now this is the time to create another layer. We will set the layer params to ask the WMS service for the stateboundary layer. Then, we will specify the options so that it will not display in the layer switcher control (via `displayInLayerSwitcher: false`) and set a scale at which it will be visible (via `minScale`). That means this layer will only show up once we have reached a certain scale. We'll add the following to `init()` function.

```
        var wms_state_lines = new OpenLayers.Layer.WMS(
          'State Line Layer',
          'http://labs.metacarta.com/wms/vmap0',
          {layers: 'stateboundary',
           transparent: true},
          {displayInLayerSwitcher: false,
           minScale: 13841995.078125}
      );
```

5. Later, we will create a layer object that shows some road layers from the WMS service and has an options object containing the `transitionEffect: resize` property. This causes the layer to have a 'resize' animation when zooming in or out.

```
        var wms_roads = new OpenLayers.Layer.WMS(
          'Roads',
          'http://labs.metacarta.com/wms/vmap0',
          {layers: 'priroad,secroad,rail',
           transparent: true},
          {transistionEffect:'resize'}
      );
```

6. finally we just need to add the layers to the map (by replacing the previous `addLayers` function call with following):

```
        map.addLayers([
              wms_layer_map,
              wms_layer_labels,
              wms_state_lines,
              wms_water_depth,
              wms_roads]);
```

7. After saving the file, we will open it in web browser.

8. While we zoom in a few times around, for example, the Gulf of Mexico. We'll start to notice black contour lines in the ocean when we zoom in, and we should see a resize effect from the base ground layer. Depending on where we zoom, should see something like Figure 4.2.

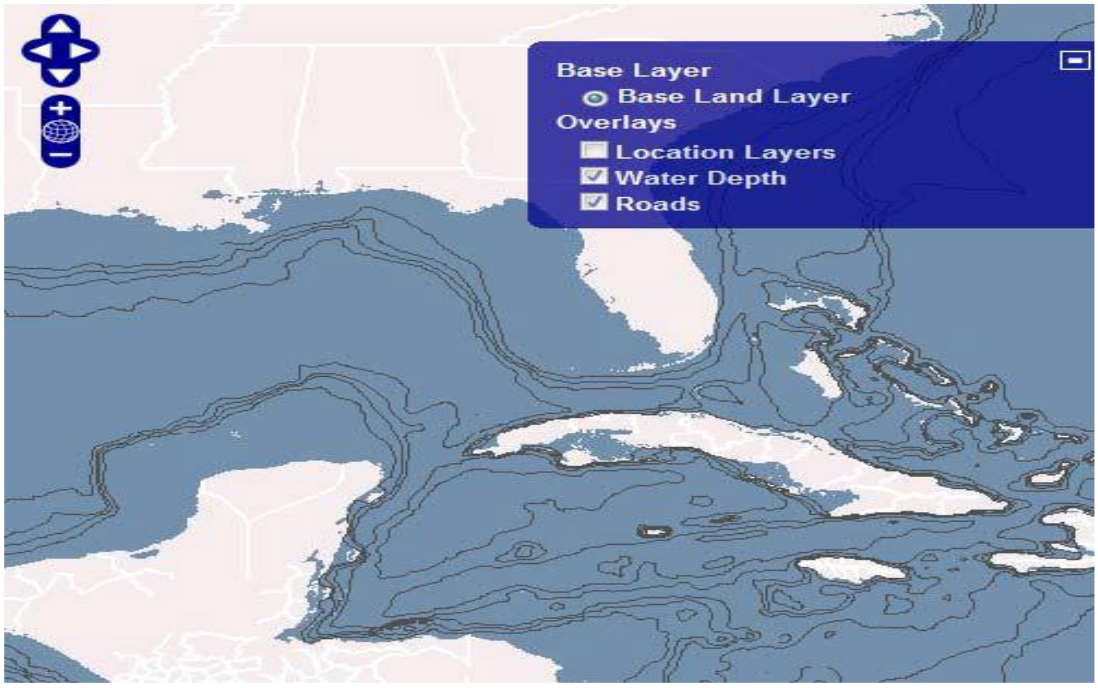

Figure 4.2: Map with more different layers and some more layer options.

Now we are going to look at a couple of the layer's options argument we passed in during the previous example. This layer's options are: *wms_state_lines layer options*

```
{ displayInLayerSwitcher: false, minScale: 13841995.078125}
```

The first property, displayInLayerSwitcher, can be either true or false, and determines if the layer will appear in the layer switcher control (if there is a layer switcher control). In our example, we do have a layer switcher control and we'll notice that we don't see it in the list of layers. The layer is still there, and we can programmatically turn it on or off through the `setVisibility()` function, e.g. `wms_state_lines.setVisibility(false);`. In fact, when we turn on or off a layer through the layer switcher control, this setVisibility function is what is being called. This property is useful when we want to include layers in our map that we don't want to let the user control. The second property is `minScale: 13841995.078125`. The value for this property is float, a number that can contain a decimal point. This minScale property determines the minimum scale the map must be at before the layer is displayed, which basically means how far we must be zoomed in before the layer will be turned on.

There is also a maxScale property which determines how far we can zoom in before the layer is turned off. The term for this behavior is referred to as scale dependency.

Scale dependency can be controlled either from the client or server side (or both). We'll notice that the wms_water_depth layer (with black contour lines in the water to specify depth) does not turn on until we start to zoom in. We haven't set the minScale property for the wms_water_depth layer, so why don't we see it at all zoom levels? The reason is WMS server has its own scale dependencies on the server side, so even if we wanted to see this layer when the map is zoomed out we can't because the server does not allow it.

When we set the minScale or maxScale properties on a layer, we are specifying client side scale dependency, so even if the server allows it, we're telling OpenLayers not to show it. To determine the current scale of the map, we can call the map.getScale(); function which will show us the current scale value.

wms_layer_labels layer options

This layer's options property consists of:

```
{visibility: false, opacity:0.5}
```

The visibility property

We have already covered the first property, so we'll talk about the visibility property. Its value can be either true (by default) or false. This visibility property controls if a layer is visible or not. Setting it to false will make it hidden, but the user can turn it on by enabling it in the layer switcher control. The map.getVisibility(); and map.setVisibility({{Boolean}}); functions refer to this property. {{Boolean}} means we can pass in a Boolean, in other words, we can pass in either true or false.

The opacity property

The next property is opacity. It accepts a float with values between 0 and 1. A value of 0 means the layer will be completely transparent, and a value of 1 means the layer will be completely opaque. We set it to 0.5 here, so it will be 50 percent opaque. If we turn on the layer in the map (we can click on the layer in the layer switcher to enable it), we'll notice the labels are sort of see-through. This opacity setting helps us to create more visually pleasing maps, as by enabling multiple layers and changing their opacities we can produce some nifty effects.

There are many other layer types such as Layer.Grid, Layer.Image, Layer.Vector, Layer.WFS, Layer.WMS, Layer.VirtualEarth, Layer.Yahoo ... OpenLayers supports a multitude of different Layer classes. Each Layer class is associated with a different map server back end. The Layer.WMS class is used to connect to a WMS map server, and the Layer.Google class is used to connect to the Google Maps service.

Some layer classes are:

Layer.Vector

The vector layer is one of the more powerful features of OpenLayers. It allows us to draw points, line and polygons on the map, style them however we want, retrieve and use KML and other geo data formats, etc. The vector layer makes uses of Protocols (such as HTTP), Formats (such as KML), and Strategies (such as Clustering) to display and provide an interactive layer. The vector layer also allows us to create and edit vector features.

Layer.WFS

WFS allow us to interact with data that is stored on the server. If we would like to use WFS, the general procedure is to use the Layer.Vector class and use WFS as the format.

Layer.WMS

As it was explained at previous sections, WMS is a popular standard and the way we interact with it is similar to how we interact with other Layer classes.

Layer.VirtualEarth

This layer allows us to interact with Microsoft's VirtualEarth API. It works in a similar way to the Google Layer, so we will not spend more time discussing it here.

Layer.Yahoo

This Layer class allows us to interact with the Yahoo! API. It works in a similar way to Layer.Google and Layer.VirtualEarth.

Things we have done until now are the base for rest of work. As it mentioned at the previous part, in this thesis we intended to make web GIS application, in order to

achieve our goal, which is to create GIS-web application for Istanbul Technical University (ITU) Ayazaga (Maslak) campus.

While its pointed in the introduction part, ITU Ayazaga campus with a total area of 2.64 km²and with population of more than 20,000 student. [2] Is one of the largest and populous universities campus in Turkey or may in the world. So Ayazaga campus can be considering as a small city. It contains athletic facilities, concert halls, housing, hospitals, libraries, offices, parking, restaurants, stores, theaters, and, of course, classrooms. For the newcomers (specially foreigners) who coming for the first time to ITU that will be difficult to find place for example, faculties, student affairs,…and or how they can go to desired place easily without waiting to someone to show the route.

Therefor this is the best reason to help that people by creating ITU Ayazaga campus web GIS application. To do so, we will use the previously mentioned procedures and by implementing other approach, we'll continue to create the web GIS application.

During of creation web GIS application it will explain the structure of the web mapping on Figure 2.1.

We are going to create a web map with base layer and some overlays. Base layers that we want to put are OpenStreetMap (OSM), Google Map Hybrid, Google Map Satellite and Google Map Terrain.

In overlays we'll put all the layers that we want to show on the map such as, faculties, social buildings, sport buildings, offices, banks, dormitories, markets, hospital, roads and some others layers.

The outline is that we want to make a map and provide it on the web, which it has, those pointed layers with some tools in it. These tools include, zoom in or out, pan, scalebar, some editing tools for drawing vector layers like point, line, and polygon. In addition, there is a marker layer on the overlays. Markers are located on faculties on the map and each markers has a pop up window include of picture and web site of faculty.

After that, we could find desired location on the map or any other things related to the ITU campus, there is a navigation button, which will switch us from current map to navigation map. In the navigation page there are options that we can chose, from where to go where by selecting starting and ending point on the map. Then is will shows us the shortest route and specifies destination along with duration ether by using car and

by walking. In addition, it will gives some extra information and signs to navigate us well.

There is a possibility that if the place that we want to go there is not exist on start and end options, it is possible to click and drop the marker where that is our start point and the same work for other marker as a destination. Therefore it will draw, the shortest route and other information as it done for selecting exist options. There are few other tools like base layers, zoom in or out, editor and some other as well.

We are going to start to create web GIS map. As well as it stated at the previous example that how to create map with Base Layer and Overlays, so here we will carry out similar task. There is a little difference that in this map, we will create more than one base layer and we have so many overlays, which we need to embed on the map.

Let us create the map using OpenLayers. As we already done after opening Notepad++ editor start to write the code. We need to set up the HTML page. HTML used, as we have seen, to create the structure and content of a webpage. We also specify a <title> tag, which contains the title that will displayed on the page.

OpenLayers library, which it explained before, will be locate on the next line of the code. The location of the file specified by the `src='OpenLayers.js'` attribute. We will be asking the map's API and specify that we want v3.0 of the Google's API. We will add this before our OpenLayers inclusion script. Above this line, we have <link> line. By this line, we will connect to CSS page. CSS is an acronym for **Cascading Style Sheets**. It is a type of markup language used to specify the appearance of HTML elements. CSS, in the other hand, it using to control the site's presentation, or how the page should look.

Therefore, when we make a site we will have at least three discrete things to consider the HTML behind it, the CSS that styles the HTML, and the JavaScript that handles any logic or user interaction. Ideally, HTML, CSS, and JavaScript should not mix (i.e., we should not use style tags, or tags like , in our HTML). Our HTML pages should link to external JavaScript and CSS files, especially in a production environment. Therefore, the first lines we be like following:

```
<!DOCTYPE>

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<link rel="stylesheet" href="theme/default/style1.css" type="text/css"/>

<script src="https://maps.googleapis.com/maps/api/js?v=3.exp"></scrip>

<meta name="viewport" content="initial-scale=1.0, user-scalable=no"; charset="UTF-8">

<title>Compus Mapper</title>

<script type='text/javascript' src='OpenLayers.js'></script>

<script type='text/javascript'>
```

Here we create a global variable called map. In JavaScript, anytime we create a variable we need to place var in front of it to ensure that we do not run into scope issues (what functions can access which variables). Since defining map as a variable at the global level (outside of any functions), we can access it anywhere in our code. We will add like this:

```
function init() {

    document.getElementById("navigation-map").style.display = "none";

    map = new OpenLayers.Map("map_element", {

    projection : new OpenLayers.Projection("EPSG:900913"),

    displayProjection : new OpenLayers.Projection("EPSG:4326"),

    maxExtent : new OpenLayers.Bounds(-20037508, -20037508, 20037508,20037508.34),

        maxResolution : 156543.0339,

        numZoomLevels : 16,

        units : "m",

            controls : controlArray,

        panMethod : OpenLayers.Easing.Expo.easeOut,

    panDuration : 0,

    });
```

The map class expects two parameters. The first argument, map_element, is the ID of the HTML element that the map will appear in. The second argument, { }, are the map options, consisting of key:value pairs (e.g., {key:value}). This is also called

JavaScript Object Notation, a way to create objects on the fly. In code above there are projection and displayProjection, `key:value`. Now a little explanation about projection.

No maps of the earth are truly perfect representations; all maps have some distortion. The reason for this is because they are attempting to represent a three dimensional object (an ellipsoid: the earth) in two dimensions (a plane: the map itself). A projection is a representation of the entire, or parts of a, surface of a three dimensional sphere on a two dimensional plane (or other type of geometry).

Every map has some sort of projection, it is an inherent attribute of maps. Imagine unpeeling an orange and then flattening the peel out. Some kind of distortion will occur, and if we try to fully fit the peel into a square or rectangle (like a flat, two dimensional map), we'd have a very hard time.

To get the peel to fit perfectly onto a flat square or rectangle, we could try to stretch out certain parts of the peel or cut some pieces of the peel off and rearrange them. The same sort of idea applies while trying to create a map.

There are literately an infinite amount of possible map projections; an unlimited number of ways to represent a three dimensional surface in two dimensions, but none of them are totally distortion free.

So, if there are so many different map projections, how do we decide on what one to use? Is there a best one? The answer is no. The 'best' projection to use depends on the context in which we use our map, what we are looking at, and what characteristics we wish to preserve.
For the purpose of OpenLayers, EPSG codes referred to as EPSG: 4326. The numbers (4326 in this case) after EPSG: refer to the projection identification number. This projection, EPSG: 4326, is the default projection which OpenLayers uses.

We set the maxExtent to that coordinates. The reason we must specify these properties is that different projections use different coordinates. These values are much different from the default values; they are not longitude and latitude values. Instead, they use an **x/y** coordinate system, and to OpenLayers the longitude is the **x** value and latitude is the **y** value.

`maxResolution` this property sets the maxResolution of the map, which is based on fitting the map's extent into 256 pixels. This is a `{Float}` data type. The

maxResolution property specifies what the maximum resolution of the map can be. In other words, how far 'zoomed out' the map can be. Setting this property will affect what the base zoom level is and how far we can zoom out.

numZoomLevels specifies the number of zoom levels a layer has. The layer will not be displayed if the zoom level of the map is greater than this value. For many applications, giving users 16 zoom levels is not always desirable. In most cases, just specifying the numZoomLevels (along with maxExtent, if using a different projection) is all we need to do to set the number of zoom levels.

Another key: value is controls. Controls allow us to interact with our map. They also allow us to display extra information, such as displaying a scale bar with the *ScaleLine* control. Some controls do not have a visual appearance, such as the *ArgParser* control, but others do, such as the *OverviewMap* control. We can have as many controls on our map as we would like. There are even some cases where we may not want any controls, such as embedding an unmovable map in a page, or showing a static map for printing. Most controls are added directly to the map, such as the Navigation control. Others are added to the map, but can also be placed in a tag outside the map, like the overview map control.

Other controls act similar to buttons which can be clicked or toggled on and off. These types of controls can be placed inside of a *panel*, which can also be placed outside the map. Furthermore, we can even create our own controls and place them directly in the map or inside of a panel.

panMethod ：this property specifies what type of tween animation will be used when panning the map. The default value is a function called OpenLayers. Easing. Expo. easeOut, which causes the animation to 'ease out' (or slow down) when the panning is finished.

panDuration ：specifying this will control how long panning takes to complete. Like the panMethod property, this property only effects the panning animation of the map.

Units ：This specifies that the map is in meters. It is a {String} data type. By default, the units property is set to degrees. Since Spherical Mercator is a projection that uses meters, we need to specify it here.

this time we'll put the layer to the map. This step needs to add Base Layers. To do so, start by adding a hybrid layer to the map. The type property is in the form of google.maps.MapTypeId.TYPE, where TYPE in this case is HYBRID:

```
var google_map_hybrid = new OpenLayers.Layer.Google(
"Google Map Hybrid",
{type: google.maps.MapTypeId.HYBRID}
);
```

Now we'll add a physical (topographic type) layer:

```
var google_map_TERRAIN = new OpenLayers.Layer.Google(
"Google Map Terrain",
{type: google.maps.MapTypeId.TERRAIN}
);
```

To add a satellite layer type:

```
        var google_map_SATELLITE = new OpenLayers.Layer.Google(
        "Google Map Satellite",
        {type: google.maps.MapTypeId.SATELLITE}
});
```

Now, we will create a streets layer. If we do not pass in a layer type, the streets layer is used by default. If we wish to manually specify the type, the streets map type is google.maps.MapTypeId.ROADMAP.

```
var osm_layer = new OpenLayers.Layer.OSM("OpenStreetMap
Layer");
```

We'll add the layers to the map:

```
map.addLayers([var google_map_TERRAIN, osm_layer,
google_map_SATELLITE,
google_map_hybrid]);
```

Finally, add a layer switcher control. We will also notice that all the layers on the map that we have added are all base layers. By default, third party map API layers act as base layers.

```
var controls_array = [
            navigation_control,
            new OpenLayers.Control.PanZoomBar ({}),
            new OpenLayers.Control.LayerSwitcher({}),
            new OpenLayers.Control.Permalink(),
            new OpenLayers.Control.MousePosition({})
      ];
      var argparser_control = new OpenLayers.Control.ArgParser();
      map = new OpenLayers.Map('map_element', {controls:
      controls_array
      });
            map.addControl(new OpenLayers.Control.OverviewMap());
            map.addControl(new
            OpenLayers.Control.KeyboardDefaults());
            map.addControl(new OpenLayers.Control.ScaleLine());
```

Latter, we need to set the map center information. This last step needs to add to code and we need to be sure to include it even if it is not explicitly asked for. All our maps will need to have their extent set somehow, and this is one standard way to do so.

```
      if(!map.getCenter()){
      map.zoomToMaxExtent();
      }
```

Finally, we will close the script by writing </script>, and <style>, </style> which we put style information (map width and height, specifying position of map on screen…) between two style tags. Then we will close </head>, , </body> and the last job is to close all the code by </html>tag. We have <body> tag, When the page loads (via body onload='init();' this function will get called. This function contains all of our code to set up our OpenLayers map. Moreover, we are going to demonstrate how to create and use the Panel class. We will create a panel control object that will be placed in a div outside the map, and then add control objects to the panel. We are going to place the panel in a div element outside of the map. We will first create an HTML element to contain our panel. We should add last tags as I mentioned above as following:

```
</script>
<style>
      #map_element {
      width: 1000px;
      height: 975px;
      float: left;
}
</style>
</head>
<body onload='init();'>
<div id="map_element">
</div>
</body>
</html>
```

For checking the result after saving, open up the page. We should see something similar to Figure 4.3.

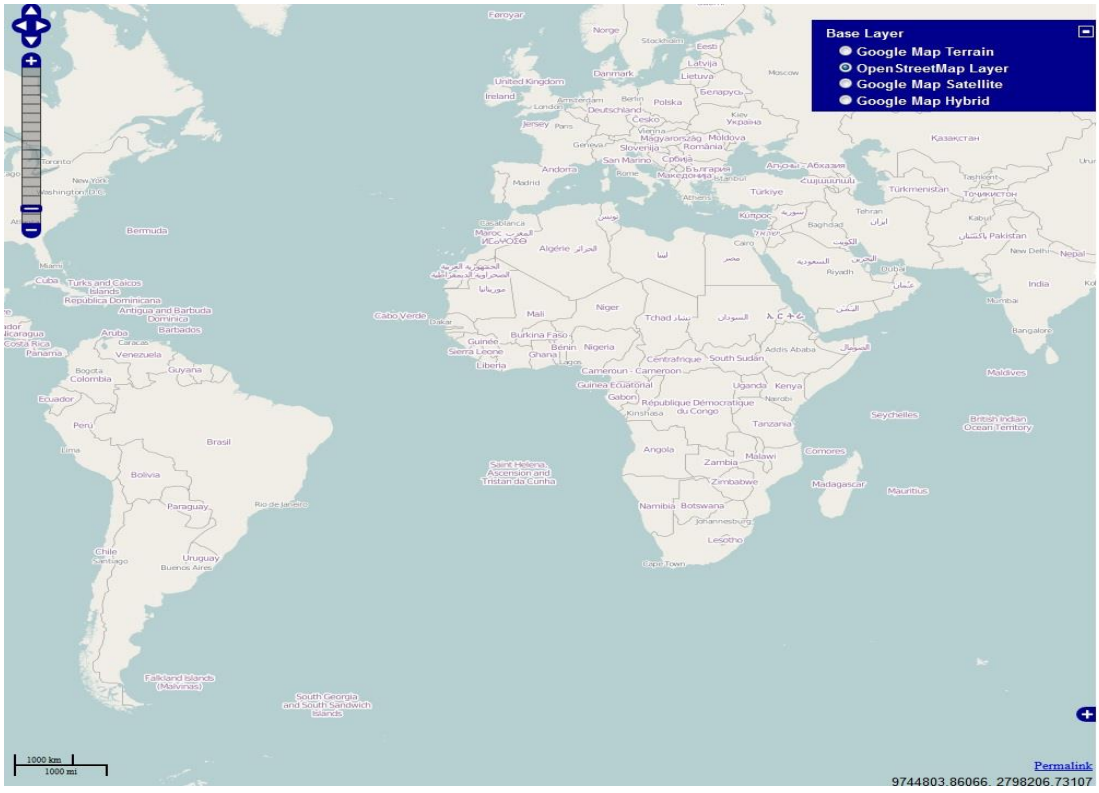

Figure 4.3: Map with different Base Layers.

By now we have done some important parts of our web map but we have slightly more things to do here in after. By adding overlays we'll make the map more interesting. As

it pointed in overlays we‘ll put all the layers that we want to show on our map such as, faculties, social buildings, sport buildings, offices, banks, dormitories, markets, hospital, roads and some others layers. Now is the time to do.

We are going add pointed overlays as a WMS layers. We had some explanations that what does the meaning of WMS briefly but there is question here that how WMS layers work and how can call them to our maps. On the other hand, essentially how we created these layers to apply on the map via WMS.

It will explain about these questions on coming section in deep. In order to add WMS layers (Overlays) to our code, for each layer we will act like following order and for each layer we should repeat same order but the only thing that we need to change is the name of the WMS layer. Here the important point, which we required to do, related to database and map server side of our job, which we will discuss later.

For now we would like add first WMS layers with a name of “Idari Birimler" (student’s affair) as an example:

```
var wms_layer_Idari_Birimler = new OpenLayers.Layer.WMS("Idari
Birimler",
      "http://localhost:8080/geoserver/itu/wms", {
            layers : [
            "idaribirimler",
            "rektorluk",
            "otomasyon_ogrenciisleri",
            "teknoloji_bligi_merkezi",
      ],
            sphericalMercator : true,
            transparent : true
   }, {
            visibility : false,
            isBaseLayer : false,
            opacity : .5,
            attribution : "ITU Shapefile from IITR"
});
```

Here is another WMS layer with a name of "Fakulteler (Group Layer)", (faculties). This layer include of many sub layers (faculties) so that each layer should be separate from other by putting comma (,) between the layers.

```
var wms_layer_fakulteler = new OpenLayers.Layer.WMS("Fakulteler
(Group Layer)",
"http://localhost:8080/geoserver/itu/wms", {
      layers : [
            "insaat_block_a", "insaat_block_b", "insaat_block_c",
      "insaat_block_d",
            "insaat_block_e", "insaat_block_f", "insaat_block_g",
            "elektrik_fakultesi_block_a",
            "elektrik_fakultesi_block_b",
            "elektrik_fakultesi_block_c",
            "elektrik_fakultesi_block_d",
            "fen_edebiyat_fakultesi",
            "gemi_deniz_blim_block_a", "gemi_deniz_blim_block_b",
            "kimya_metaloji_fakultesi_block_a",
      "kimya_metaloji_fakultesi_block_b",
            "kimya_metaloji_fakultesi_block_c",
      "madan_fakultesi_block_a",
            "madan_fakultesi_block_b", "madan_fakultesi_block_c",
            "merkezi_derslik_binasi",
            "molekuler_bioloji_genetik_bolumu",
            "ucak_uzay_bilim_fakultesi",
      ],
      sphericalMercator : true,
      transparent : true
      }, {
      visibility : false,
      transitionEffect : "resize",
      isBaseLayer : false,
      opacity : .5
      });
```

After adding this layer to the map, when turn on the layer with "Fakulteler (Group Layer)" name, all the faculties will be appear on the web. Because some faculties had built on different blocks, so in this layer for example for civil engineering (Insaat) faculties we make each block as separate layer to access easily.

We have to pay attention that code is specific for the WMS Layer class. `var wms = new OpenLayers.Layer.WMS(`

`var wms` , will be an `OpenLayer.Layer.WMS` object. i.e., wms is an instantiated object based on the `Layer.WMS` class. (If we are unfamiliar with object oriented programming, the basic idea is that a class is similar to a blueprint of a house and an object is the actual house created based on the blueprint. We interact with objects that are generated from classes.) new `OpenLayers.Layer.WMS` ,this creates a new object using the `OpenLayers.Layer.WMS` class. Each item inside the parentheses, after `OpenLayers.Layer.WMS(`, are called arguments which we pass in while creating the object. Here we begin passing in parameters "OpenLayers WMS", this is the title of the layer. The title what is displayed on the layer switcher control and is completely arbitrary, we can call it whatever we would like.

```
"http://localhost:8080/geoserver/itu/wms",
```

The URL is the second parameter that the WMS layer class expects to receive. This is the URL we'll be using for the layer and we'll ask the WMS server for the layers (a layers on the WMS service). This URL is dependent on our installation of Map Server. In thesis GeoServer used as server which it will give more information about GeoServer at the section 6.3.

```
{layers: ''}
```

Now we start passing in params (parameters). We pass in *key: value* pairs separated by commas. e.g., `{ layers: '', transparent: 'true' }`. The params represent the GetMap URL string discussed above. The GetMap call will provide us with parameters we can define.

```
);
```

And after we passed in parameters we should finish the call.

Some explanation about parameters:

sphericalMercator , this property determines if the map should behave as a Mercator-projected map. Setting to true will allow us to use other layers, such as the Vector layer, with the actual map projection. We'll specify sphericalMercator: true so the layer is properly projected to the projection.

transparent : true , We're also going to set the transparent property to true, so the map images which the server sends back will be transparent.

visibility : false, This visibility property controls if a layer is visible or not. Setting it to false will make it hidden, but the user can turn it on by enabling it in the layer switcher control. The map. getVisibility(); and map. setVisibility({{Boolean}}); functions refer to this property. {{Boolean}} means we can pass in a Boolean, in other words, we can pass in either true or false. setVisibility(false); property is useful when we want to include layers in our map that we don't want to let the user control.

transitionEffect: resize , we'll create a layer object that shows some road layers from the WMS service and has an options object containing the transitionEffect: resize property. This causes the layer to have a 'resize' animation when zooming in or out. So this property specifies the transition effect to use when the map is panned or zoomed. Possible values at the time of writing are null (no effect) and 'resize.'

isBaseLayer : false, Determines if a layer is to act as a base layer. We can change the property of any layer on our map to act as the base layer (by setting the isBaseLayer property to True). We set isBaseLayer: false so we're sure it will be an overlay layer.

opacity : 0.5, As it explained before it accepts a float with values between 0 and 1. A value of 0 means the layer will be completely transparent, and a value of 1 means the layer will be completely opaque. We set it to 0.5 here, so it will be 50 percent opaque. If we turn on the layer in the map (we can click on the layer in the layer switcher to enable it), we'll notice the labels are sort of see-through. This opacity setting helps us to create more visually pleasing maps, as by enabling multiple layers and changing their opacities we can produce some nifty effects.

By this way we will add other WMS layers to the code. Since we add all the layers to the code we need to add layers to the map object. Notice we are calling a function of

the map object. There are actually a few ways to go about adding a layer to a map object. We can use as we write at the first example code (by calling `map.addLayer`), where we pass in an individual layer, or we could use `map.addLayers`:

```
map.addLayers( [layer1, layer2, ...] );
```

Here, we pass an array of layers. Both methods are equally valid, but it may be easier to pass in an array when we have multiple layers.

We can also create the layer objects before we create the map object and pass the layer objects into the map when we create it, for instance:

```
map = new OpenLayers.Map('map_element', {layers: [layer1,
layer2,
…]});
```

All ways are valid. Here again each layer should be separate from other by putting comma (,) between the layers. There is just one difference that the layer will appear on

overlays as sequence of layers which we put in the map object. So we will add our layers as following:

```
map.addLayers([osmMap, google_map_hybrid, google_map_SATELLITE,
google_map_TERRAIN,vector_layer,wms_layer_Idari_Birimler,wms_l
ayer_fakulteler,wmsSosyal_tesisle,wmsSpor_tesisleri,wmsLojmanl
ar, wmsYurtlar, wmsOtopark, wmsOkullar, wmsMarket, wmsHavuzgol,
wmsKampusbaglantiyollari,wmsKapaliyayayol,wmsBankalar,
]);
```

After saving the code and opening up the page, we will see something like Figure 4.4.

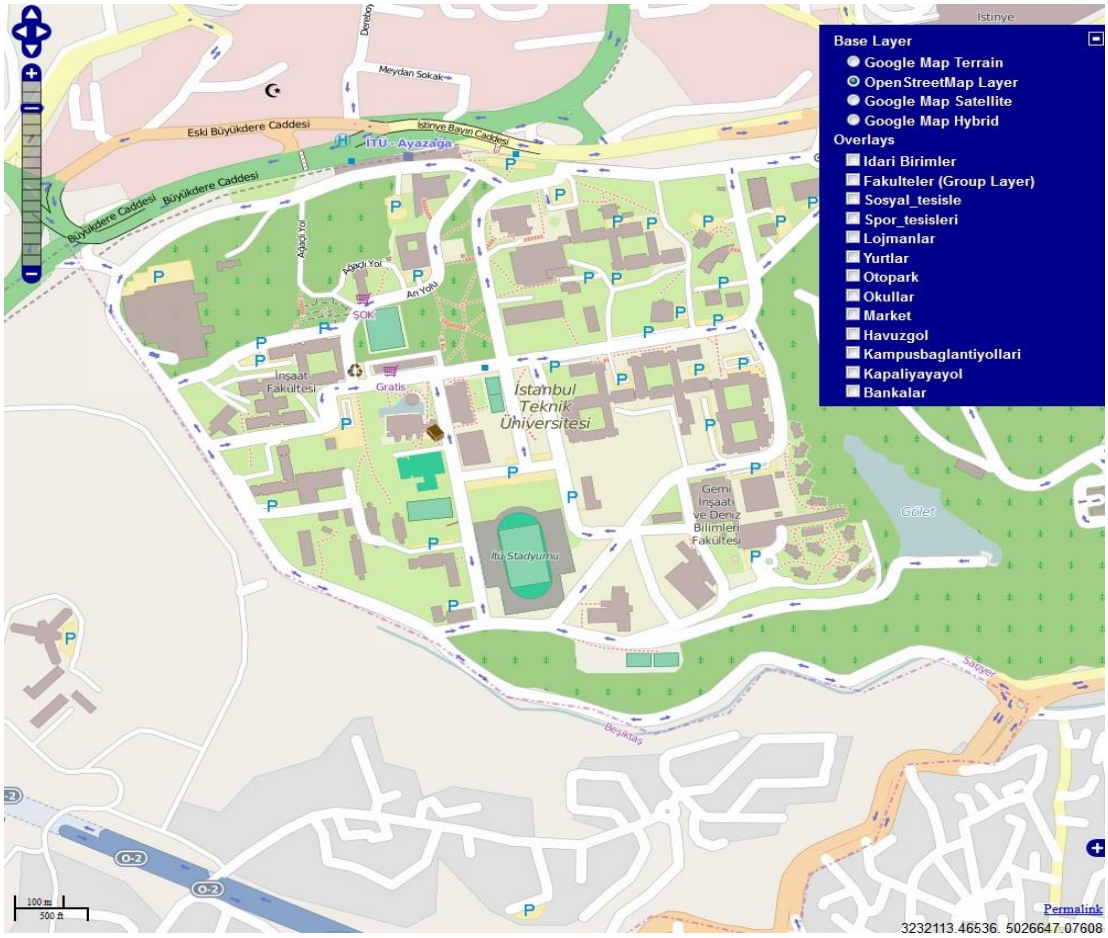

Figure 4.4: Map with Base Layers and Overlays.

Now we want to add vector layer to the map. In terms of graphics, there are essentially two types of images: raster and vector. Most images we see are raster images, meaning, basically, they are comprised of a grid of pixels and their quality degrades as we zoom in on them. A photograph, for example, would be a raster image. If we enlarge it, it tends to get blurry or stretched out. The majority of image files .jpegs, .png, .gifs, any bitmap image are raster images.

A vector, on the other hand, uses geometrical shapes based on math equations to form an image. Meaning that when we zoom in, the quality is preserved. If we were to zoom in on a vector image of a circle, the lines would always appear curved with raster image, the lines would appear straight, as raster images are made up of a grid of colors. Vector graphics are not constrained to a grid, so they preserve shape at all scales. Now begin to create a basic vector layer. As we discussed before it allows us to draw point line and polygon on the map.

Therefore, we would like add one point, polygon and other feature types to the vector layer. We are going to add point and polygon which will show us ITU campus on the

map. We start to add point and polygon separately to the code out of init() function as following:

This time we'll create the vector layer itself. We'll use the default projection and default values for the vector layer, so to create the layer all we need to do is create it:

```
var vector_layer = new OpenLayers.Layer.Vector('Basic Vector Layer')
var argparser_control = new OpenLayers.Control.ArgParser();
var nav_history_control=new OpenLayers.Control.NavigationHistory();
```

We are going to create some features first. To do so, we'll need to make use of the OpenLayers.Geometry classes. We'll start off by creating an OpenLayers.Geometry.Point object, passing in a longitude and latitude:

```
      new OpenLayers.Geometry.Point(4552568, 418396),
      {
            'location': 'Istanbul Technical University',
            'description': "Turkey"
      }
);
```

Now that we have a geometry object, we can create a feature object from it. We'll use the OpenLayers.Feature.Vector class to create a feature object using the point object we just created. When instantiating, the constructor can take in three arguments ,geometry, attributes (optional), and style (optional).

```
      var feature_point = new OpenLayers.Feature.Vector(
```

We'll make a polygon from a linear ring object, which consists of points.

```
      var feature_polygon = new OpenLayers.Feature.Vector(
          new      OpenLayers.Geometry.Polygon(new
OpenLayers.Geometry.LinearRing(
            [
                  New OpenLayers.Geometry.Point(3230221.64891, 5028157.90171),
                  new OpenLayers.Geometry.Point(3230087.88411, 5028105.35126),
                  new OpenLayers.Geometry.Point(3230006.66977, 5028014.28228),
                  new OpenLayers.Geometry.Point(3230001.89245, 5027885.59480),
```

```
                new OpenLayers.Geometry.Point(3230336.30445, 5027331.42634),
                new OpenLayers.Geometry.Point(3230708.93497, 5027149.88840),
                new OpenLayers.Geometry.Point(3231100.67474, 5027116.44720),
                new OpenLayers.Geometry.Point(3231640.51125, 5027259.76663),
                new OpenLayers.Geometry.Point(3231358.64971, 5028014.58228),
                new OpenLayers.Geometry.Point(3231516.30108, 5028181.78828),
                new OpenLayers.Geometry.Point(3231482.87988, 5028358.54891),
                new OpenLayers.Geometry.Point(3230699.38034, 5028301.22114),
```

We will not pass in the first point, the polygon will close automatically.

```
            ]
            )),
            {
                'location': 'ITU Ayazaga Campus',
                'description': 'Istanbul_Turkey'
            }
    );
```

We add the layers to the map now:

```
    map.addLayers([osmMap, google_map_hybrid, google_map_SATELLITE,
    google_map_TERRAIN, vector_layer, wms_layer_Idari_Birimler, wms_l
    ayer_fakulteler, wmsSosyal_tesisle, wmsSpor_tesisleri, wmsLojmanl
    ar, wmsYurtlar, wmsOtopark, wmsOkullar, wmsMarket, wmsHavuzgol, wmsK
    ampusbaglantiyollari, wmsKapaliyayayol, wmsBankalar,
    ]);
```

Now we are ready to add the feature to the map with `addFeatures`:

```
    vector_layer.addFeatures([feature_point, feature_polygon]);
```

We added ITU campus as a point and polygon to the map. If we looked at the map now, we would just see a simple map, our vector layer does not have any data loaded into it, nor do we have any controls to let us add vector data.

We are adding the *EditingToolbar* control to the map, which allows us to add points and draw polygons on a vector layer. To do so, we just need to instantiate an object from `OpenLayers.Control.EditingToolbar` and pass in a vector layer. We'll pass in the vector_layer object we previously created:

```
map.addControl(new OpenLayers.Control.EditingToolbar(vector_layer));
            map.addControl(customControlPanel);
            customControlPanel.moveTo(new OpenLayers.Pixel(70, 0));
```

Take a look at the map now. We should see the *EditingToolbar* control (which is basically a panel control with control buttons). Selecting different controls will allow us to place vector objects (called features) on the vector layer.

We've placed some features (points, polygons, etc.) on the map, but if we were to refresh the page they would disappear. We can, however, get the information about those features and then export it to a geospatial file. Now we'll grab the information about the features we've created. To access the information about the vector layer's features, all we need to do is access its features array.

To make the features interactive, we'll first need a *SelectFeature* control. When we instantiate it, we pass in the vector layer that we want the control to use (alternatively, we could pass in an {Array} of vector layers if we wanted the control to use multiple vector layers).

The multiple property will allow multiple features to be selected at once, the toggle property will cause features to be unselected when selecting a different feature, and the *multipleKey* specifies which key to press to allow multiple features to be selected:

```
var select_feature_control = new OpenLayers.Control.
    SelectFeature(
    vector_layer,
    {
    multiple: false,
    toggle: true,
    multipleKey: 'shiftKey'
    }
    );
    map.addControl(select_feature_control);
```

At this point, the control is created and has been added to the map. However, before we can use it we must activate it by calling it's activate method.

```
    select_feature_control.activate();
```

Now we can select our features. Because we passed in the *multipleKey* property, we can select multiple controls by holding Shift and clicking on them. Holding control will toggle a feature selection.

This time we'll do something when the user clicks on a feature. To do this, we'll create two functions that will be called when the *featureselected* and *featureunselected* events are fired (they get fired from the *SelectFeature* control). First, create the function to call when a feature is selected. It will clear the *map_feature_log* div and then look at the attributes object of the passed in feature. Finally, it will loop through all selected features and display the location property of all selected features. The sentences between the lines define functionality of each line.

```
function selected_feature(event){
```

//clear out the log's contents

```
        document.getElementById('map_feature_log').innerHTML = '';
```

//Show the current selected feature (passed in from the event object)

```
        var display_text = 'Clicked on: '
    + '<strong>' + event.feature.attributes.location + '</strong>'
        + ': ' + event.feature.attributes.description + '<hr/>';
        document.getElementById('map_feature_log').innerHTML
    =display_text;
```

//Show all the selected features

```
    document.getElementById('map_feature_log').innerHTML += 'All
        selected features: ';
```

//Now, loop through the selected feature array

```
    for(var i=0; i<vector_layer.selectedFeatures.length; i++){
        document.getElementById('map_feature_log').innerHTML+=
        vector_layer.selectedFeatures[i].attributes.
        location + ' | ';
      }
    }
```

We need a function to call when a feature is unselected now. It will do a similar thing to the previous function, display the feature the user clicked on, and then show all the selected features.

```
function unselected_feature(event){
   var display_text=event.feature.attributes.location+'
   unselected!'+    '<hr />';
document.getElementById('map_feature_log').innerHTML= display_text;
```

//Show all the selected features

```
 document.getElementById('map_feature_log').innerHTML+=
'Allselected features: ';
```

//Now, loop through the selected feature array

```
      for(var i=0; i<vector_layer.selectedFeatures.length; i++){
      document.getElementById('map_feature_log').innerHTML+=
      vector_layer.selectedFeatures[i].attributes.
      location + ' | ';
   }
}
```

Just one more thing now left to do. We have to register the events to call those functions when a feature is selected or unselected:

```
vector_layer.events.register('featureselected',this,
selected_feature);
      vector_layer.events.register('featureunselected',
      this,unselected_feature);
```

Well, we finished, we will save the code and then will open up our map and select some features (we will use *Shift* to select multiple features). We should see something as Figure 4.5.

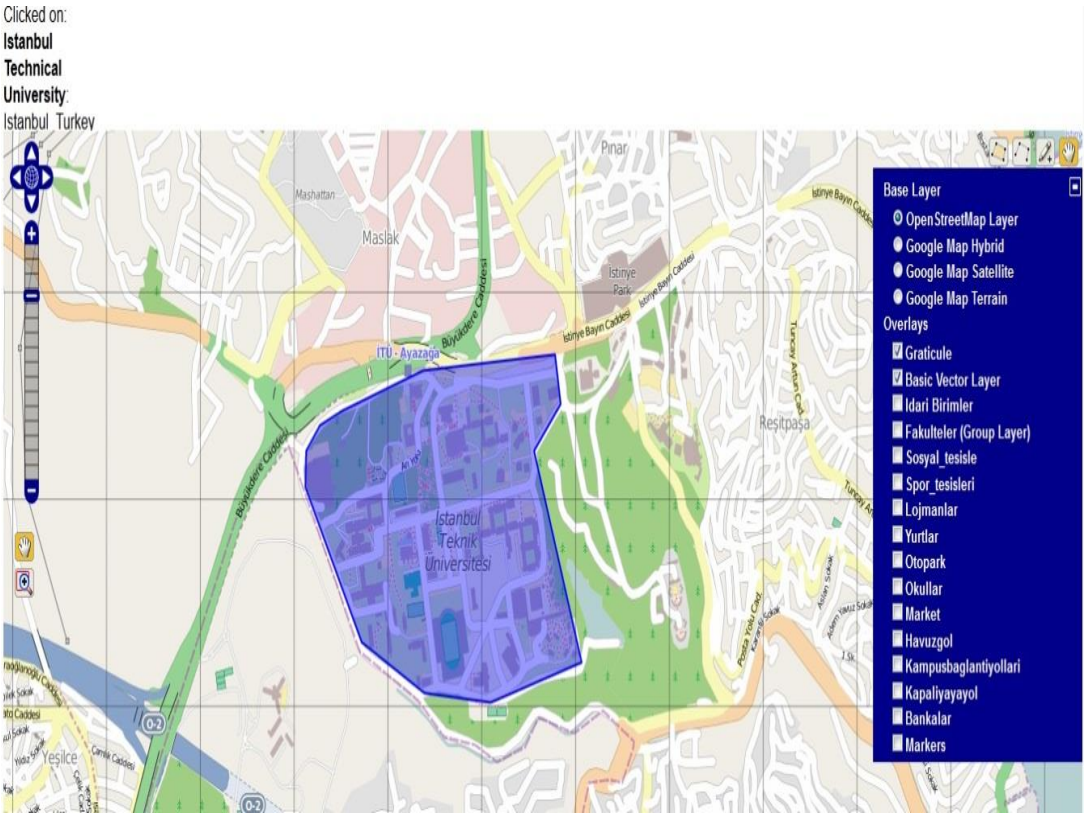

Figure 4.5: Basic vector layer by suing the *SelectFeature* control.

In this part, we would like add some markers to the map. These markers will be locate where faculties are placed in ITU Ayazaga campus on the map. Each marker has a pop up window so that once we click on the marker on the map pop up window will be open. This window include of each faculty's photos and by clicking on the photo in will link us to the considered faculty.

For creating the Markers layer itself we are doing as following. We will add this order inside of `init()` function. So to create the Markers layer all we need to do is create it:

```
var markers = new OpenLayers.Layer.Markers( "Markers" );
```

After we create Marker layer we should add Markers to the map, like:

```
map.addLayer(markers);
```

Then we will add Markers layer properties as:

```
var size = new OpenLayers.Size(21,25);
var offset = new OpenLayers.Pixel(-(size.w/2), -size.h);
var icon = new OpenLayers.Icon('img/pins9.png', size, offset);
```

As it is clear about these Markers properties, by adding them we can specify things like size, offset and icon. We can set any form of icons in order to appear on the map.

Only thing that we need to notice is, we call this Markers icon from OpenLayers.js which we downloaded before start to create our map. For this, after opening the .img folder among the folders which we downloaded and then we will add any icons that we want to display on the map inside this folder in format of .PNG , .JPG or any other image format. Then we will call it, in a same way that it is called above ('img/pins9.png').

Now, we'll create some features first. To do so, we'll need to make use of the OpenLayers.Marker classes. We will start off by creating an OpenLayers.Marker1 object, passing in a longitude and latitude:

```
marker1 = new OpenLayers.Marker(new OpenLayers.LonLat(3230415.8, 5027803.9));
```

We would like to add a marker which appears on the map where ITU civil engineering faculty (Insaat facultesi) is located, so we need to put the ITU civil engineering coordinates on the code as we done above. Then we are going to add marker1 to the map like:

```
markers.addMarker(new OpenLayers.Marker(new OpenLayers.LonLat(3230415.8, 5027803.9),icon));

markers.addMarker(new OpenLayers.Marker(new OpenLayers.LonLat(3230415.8, 5027803.9),icon.clone()));

markers.addMarker(marker1);
```

As it mentioned before we want to add pop up window to the marker. To do so, I just need to register the events as following:

```
marker1.events.register("click", marker1, function(e){

var popup = new OpenLayers.Popup.FramedCloud("chicken", new OpenLayers.LonLat(3230415.8, 5027803.9),

new OpenLayers.Size(200, 200),

"Insaat Fakultesi <br> <a href='http://www.ins.itu.edu.tr' target='_blank'> <img src='img/Insaat.jpg' href='www.google.com' width='100' height='100'></a>",

null, true);
```

In the above code we created pop up window by suing OpenLayers.Popup.FramedCloud class.it will be appear in the same coordinate (actually inside) which Marker1 appeared. On the next line of the code we can set the size of the pop up window by changing the numbers between parentheses. Then we will be add civil engineering (Insaat facultesi) web site into the pop up window or

actually into the image of civil engineering photo in the opening window. At this time by click on the image it will open blank page for civil engineering web page. At end of this line we'll put null, true in which case we can simply include CSS file link tags in our page (or put style information in <style> tags).

Finally, we should add pop up window to the map as following:

```
        map. addPopup(popup);

        });
```

We almost done and we can add other Markers and also popup windows after previous code.

When we finished adding all of them again we'll save the code and will open up on the browser. The result should be like as Figure 4.6.

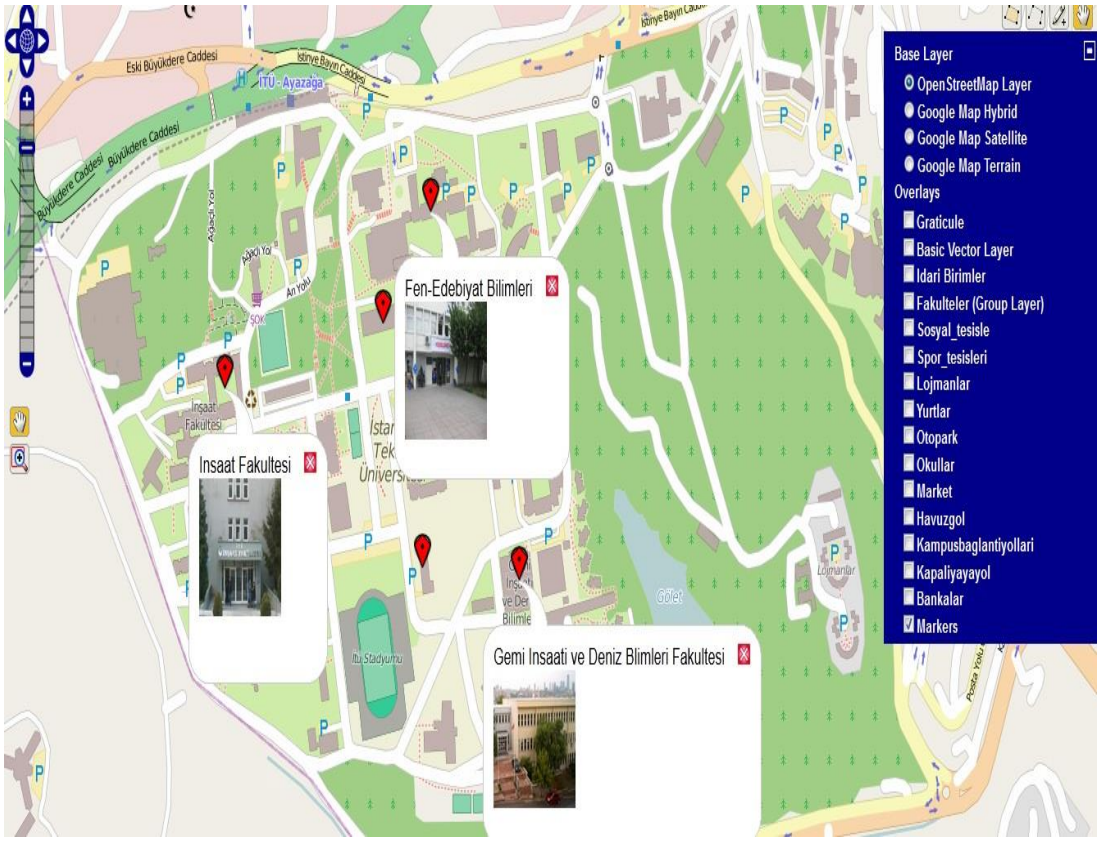

Figure 4.6: Map with Markers and popup windows.

Until now we are created a map with some Base Layers, Overlays (Basic vector layer, faculties, library, market, auto parks, etc.,) and Markers with popup windows. So now we can find the place or any buildings by suing this map but there is another question that how can I access or go to the considered location and what is the shortest route and how many it will takes me to go there?

At the rest of this dissertation, we want to add a possibility to the map to answer all these questions. We are going to add a button on the map that will switch us to navigation page. So that once we could find desired location on the map or any other place related to the ITU Ayazaga campus and In the navigation page there are options that we can chose, from where to go where by selecting starting and ending point on the map. Then is will shows us the shortest route and specifies destination along with duration ether by using car and by walking. In addition, it will gives some extra information and signs to navigate us well.

5. Navigation map

Base code for the navigation page is related to OSRM but we need to add this to the code and also make some changes in ORSM code to collaborate and work properly with together.

Here it is just explained some part of code that we need to work about it.

First of all, in order to join navigation to the map, we need to add following orders to the main code. We can add it exactly after we closed the </script>, and it can be like:

```
<script type='text/javascript'>switchNav = function () {
      if (document.getElementById("navigation-map").style.display == "none") {
            document.getElementById("navigation-map").style.display = "block";
            //                  document.getElementById("navigation-map").style.display = "block";
            document.getElementById("map_element").style.display    = "none";
      } else {
            document.getElementById("navigation-map").style.display = "none";
            //                  document.getElementById("navigation-map").style.display = "block";
            document.getElementById("map_element").style.display    = "block";
      }
};
</script>
```

At this time we'll work with OSRM code. In this part we would like to specify the starting and ending points in ITU Ayazaga campus. For instance, we want to select Electricity and electronic faculty (Elektrik-Elektronik) as a starting point and Education Center Classrooms (Merkezi Eğitim Dersliği) as an end point. The thing we should pay attention is that these two location can be replace on the starting and ending points. It means that Education Center Classrooms (Merkezi Eğitim Dersliği) can be starting point and Electricity and electronic faculty (Elektrik-Elektronik) can be our end point.

The second step that we need to do is adding the coordinates of starting and ending point to the code. Here again we will consider that each point can be replace and asking as start or end point. Therefore we add both coordinates in starting and ending point in the code. It should be as following:

```
      <div class="full">

      <div id="input-source" class="input-marker">

      <div  class="left"><span      id="gui-search-source-label"
class="input-label nowrap">Start:</span></div>

      <div class="center stretch">

      <select id="gui-input-source" class="input-box"
autocomplete="off">

      <option       value="İstanbul       Teknik       Üniversitesi"
selected="selected">

      Eğitim Binası Seçin</option>

            <option    value="41.104928,    29.024399">Elektrik-Elektronik
(EEB)</option>

            <option    value="41.105506,    29.022949">Merkezi    Eğitim
Dersliği(MED)</option>

      </select>

      </div>

      <div  class="left"><div  id="gui-delete-source"  class="iconic-button
cancel-marker input-delete"></div></div>

      <div      class="right"><a      class="button"      id="gui-search-
source">Show</a></div>

      </div>
```

```
<div id="input-target" class="input-marker">
<div class="left"><span id="gui-search-target-label" class="input-label nowrap">End:</span></div>
<div class="center stretch">
<select id="gui-input-target" class="input-box" autocomplete="off">
<option value="İstanbul Teknik Üniversitesi" selected="selected">Eğitim Binası Seçin</option>
<option value="41.104928, 29.024399">Elektrik-Elektronik (EEB)</option>
<option value="41.105506, 29.022949">Merkezi Eğitim Dersliği(MED)</option>
</select>
</div>
<div class="left"><div id="gui-delete-target" class="iconic-button cancel-marker input-delete"></div></div>
<div class="right"><a class="button" id="gui-search-target">Show</a></div>
</div>
</div>
```

We just put two points as the starting and ending point to the map in upper code, and we will place rest of the points as mentioned above.

After setting all points and saving, the code will open the map on the browser and we will switch to the navigation map. We want to examine so for instance, we will select Civil engineering block B (Insaat Block B) as the start point and Mines faculty (Maden facultesi) as the end, then we will see the result. As we can see on the Figure 5.1, it will show the shortest route between that two point and on the left hand side of the map there is a window, which tells us distance between this two points and the duration both by car and by walking and route description.

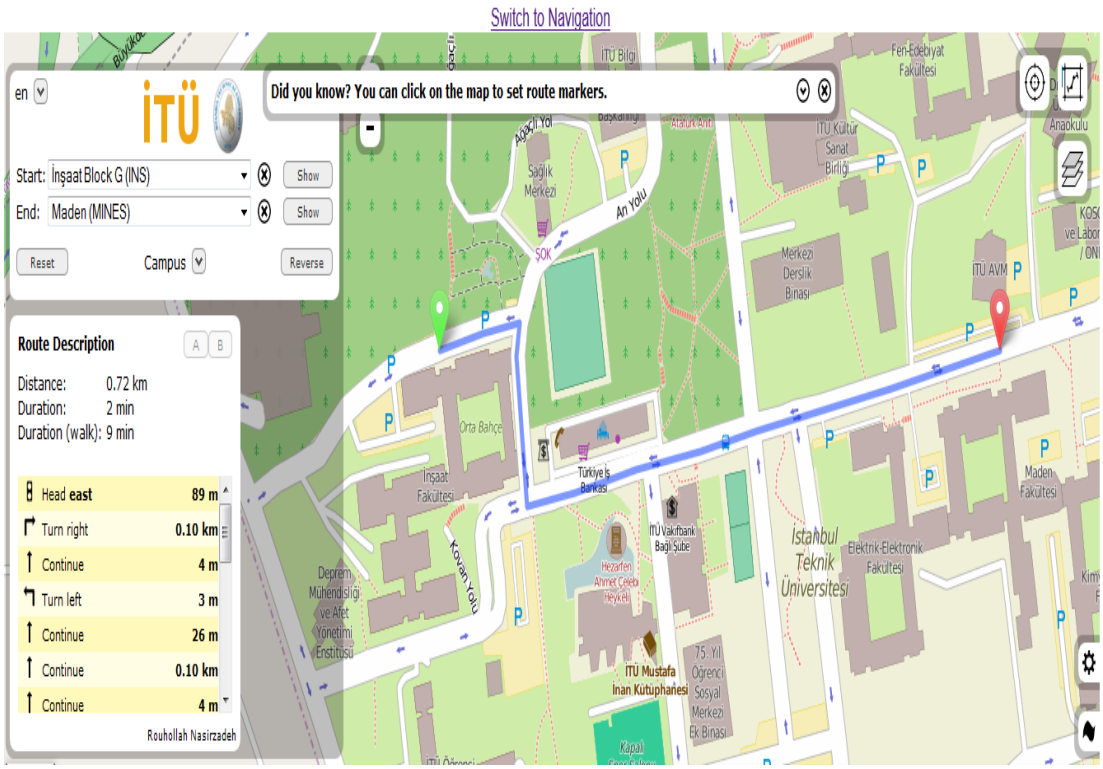

Figure 5.1: Navigation map**.**

As it mentioned before in this map there is another possibility. So that if the place that we want to go is not exist on start and end option we can click on marker and put it wherever as our start point and the same work for other marker as a destination so it will draw the shortest route and other information as it is done for selecting exist options. There are few other tool like base layers, zoom in or out, editor and some other as well. Editor tool is on the right hand bottom side of map. It is possible to edit the map, just before starting to edit anything we need to log in. after getting log in we will connect to the OpeenStreetMap editing page and then we can do any edition.

The only thing to mention about navigation map is that, it shows the shorten route by considering and selecting main routes and sidewalks are not included.

Well, until now we tried to explain the web-GIS application in terms of ITU Ayazaga campus project. At the next part of this thesis, we want to present the structure of our web-GIS application. As we remember from Figure 2.1, which it showed the main components of a web GIS application, here we are going to demonstrate the procedures of the web GIS application that applied.

6. Web-GIS Architectural Model used in this thesis

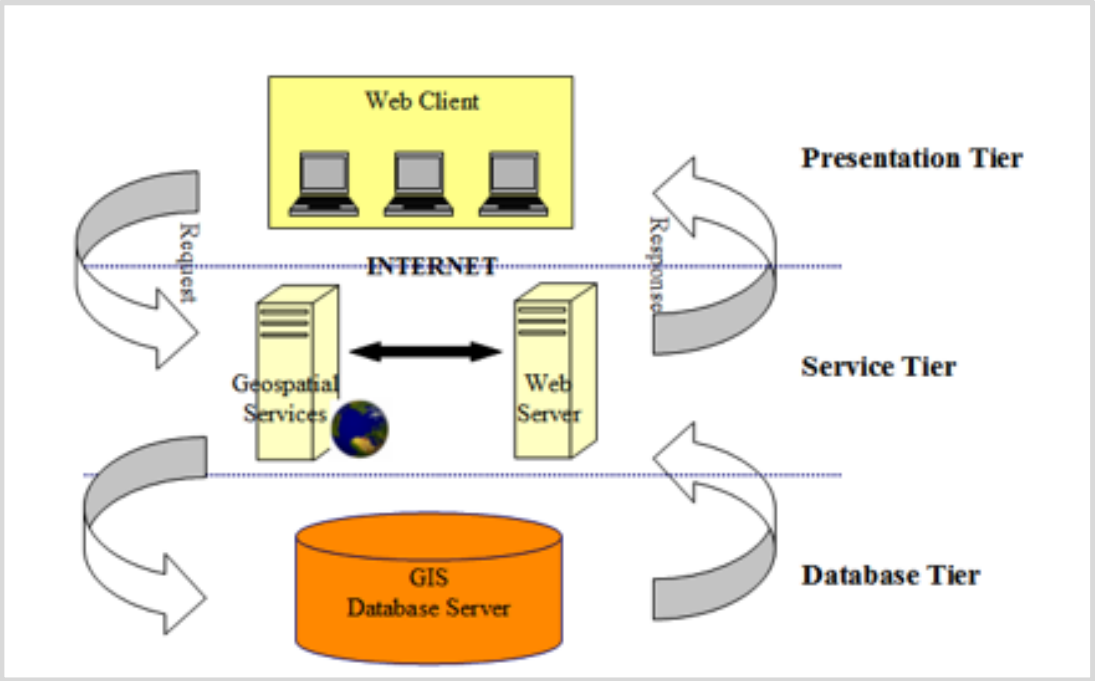

Figure 6.1: System Architecture of web GIS. [12]

In Figure 6.1, system structure has divided in three parts and each section has a separate role to play, but at the same time they can all work together. On the other word, they have interoperability.

As we see on the Figure 6.2, each sections can be produced by different products and any component of the structure can be replaced with other products and vice versa.

On the Figure 6.2 , the products that we applied to creating our web-GIS application are bold and colored on the each sections.

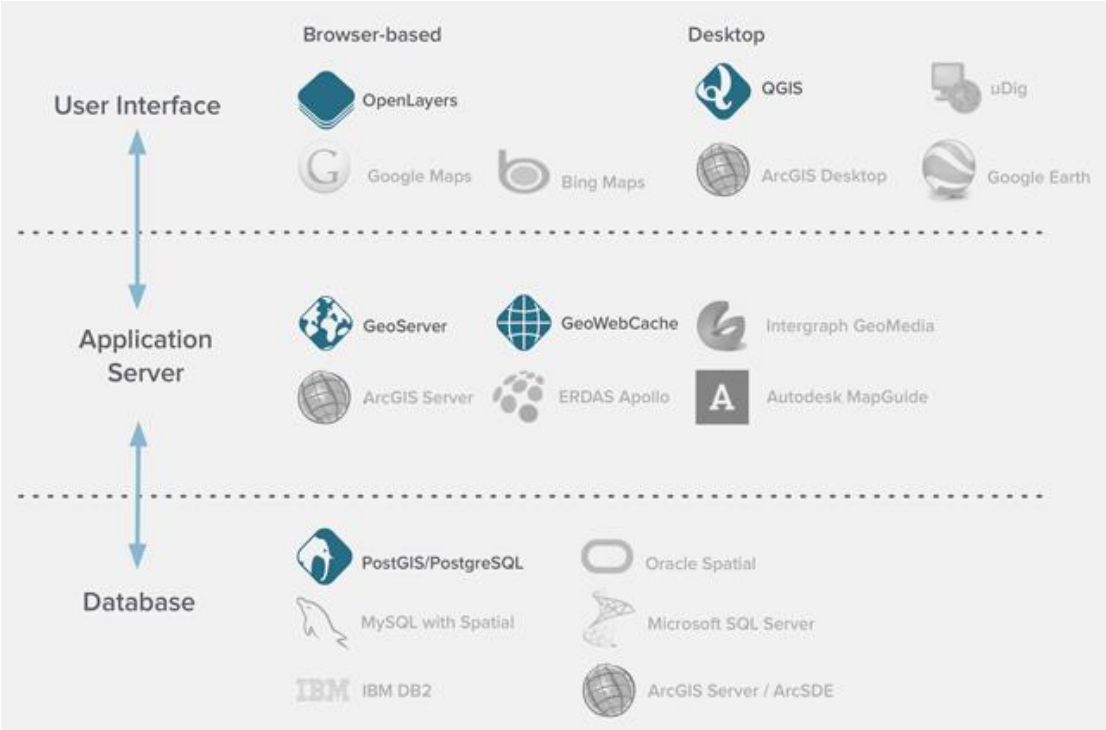

Figure 6.2: Interoperability of components of the web GIS structure. [13]

As it mentioned above, any component can be replaced with other products so that:

- For the database, **PostGIS/PostgreSQL** can replace Oracle Spatial, Microsoft SQL Server, IBM DB2, and more.
- For web map access, **GeoServer** can replace ArcGIS for Server or any other OGC Web Map Service (WMS) server.
- For web feature access, **GeoServer** can replace ArcGIS for Server, Ionic Red Spider, CubeWerx or any other Web Feature Service (WFS) server.
- For tile deliver, **GeoWebCache** can replace or optimize ArcGIS for Server, Google Maps, Bing Maps, or other tile servers.
- For browser map components, **OpenLayers** can replace or leverage Google Maps, Bing Maps, and other map APIs.
- For desktop analysis and editing, **QGIS** can replace ArcGIS for Desktop.

In the following, components that is used in this thesis for creating Web-GIS application will be investigated.

We applied **QGIS** for desktop analysis and editing, **OpenLayers** for the browser map, **GeoServer** for accessing the web map, **PostGIS/PostgreSQL** For the database and **GeoWebCache** for tile delivering.

Figure 6.3, illustrates how these components are working together to create a web map GIS application.

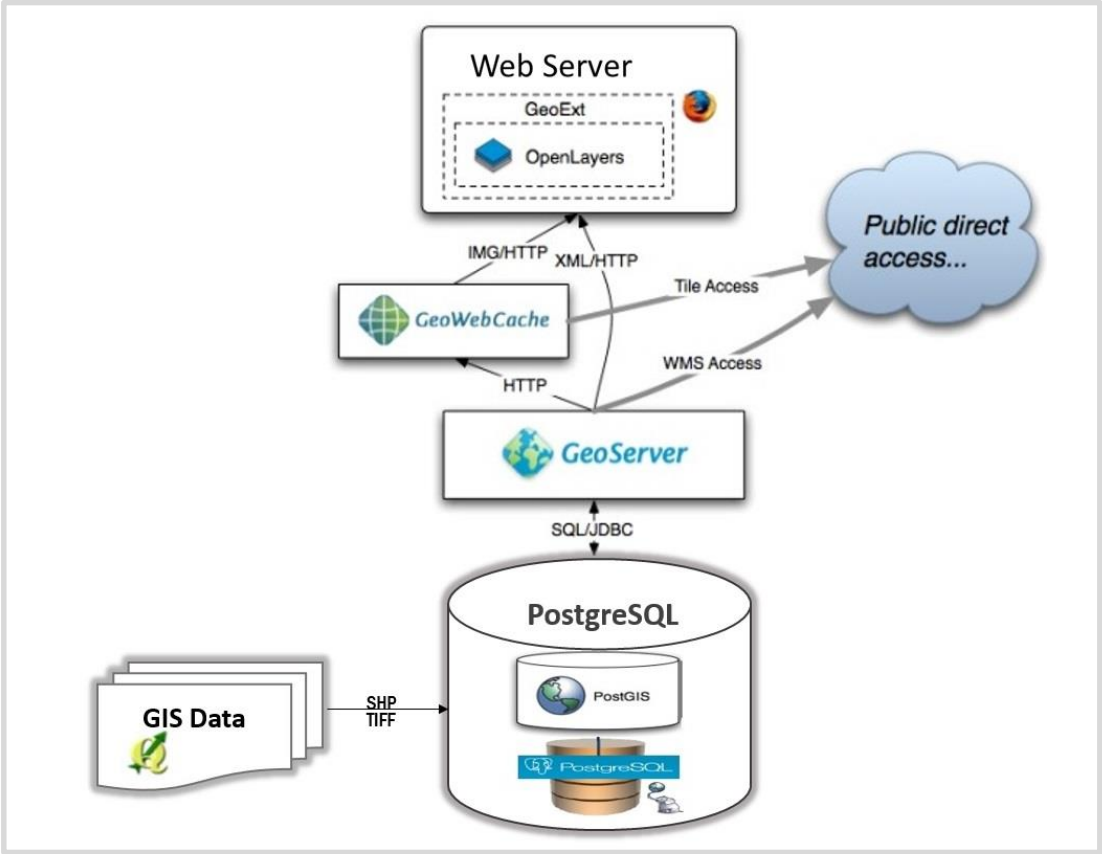

Figure 6.3: Arrangement of products to create web GIS map application.

As it shown on the Figure 6.3, in order to display any feature once user ask to appear on the map so we need to import GIS dada. Features, which we are using as a GIS data, can be any vector or raster layers (Shp, GeoTIFF…) and can be Text (XML). Therefore, the first step is to create GIS data, but how?

Just as we use a word processor to write documents and deal with words on a computer, we can use a GIS application to deal with spatial information on a computer. A GIS consists of:

• **Digital Data** - the geographical information that we are view and analyze using computer hardware and software.

• **Computer Hardware** - computers used for storing data, displaying graphics and processing data.

• **Computer Software** - computer programs that run on the computer hardware and allow us to work with digital data. A software program that forms part of the GIS is called a GIS Application.

With a GIS application, we can open digital maps on our computer, create new spatial information to add to a map, create printed maps customized to our needs and perform spatial analysis.

6.1 GIS Software and GIS Application

We can see an example of what a GIS Application looks like in Figure 6.4 below. GIS Applications are normally programs with a graphical user interface that can be manipulated using the mouse and keyboard. The application provides menus near to the top of the window (File, Edit etc.) which, when clicked using the mouse, show a panel of actions. These actions provide a way for us to tell the GIS Application what we want to do. For example we may use the menus to tell the GIS Application to add a new layer to the display output.

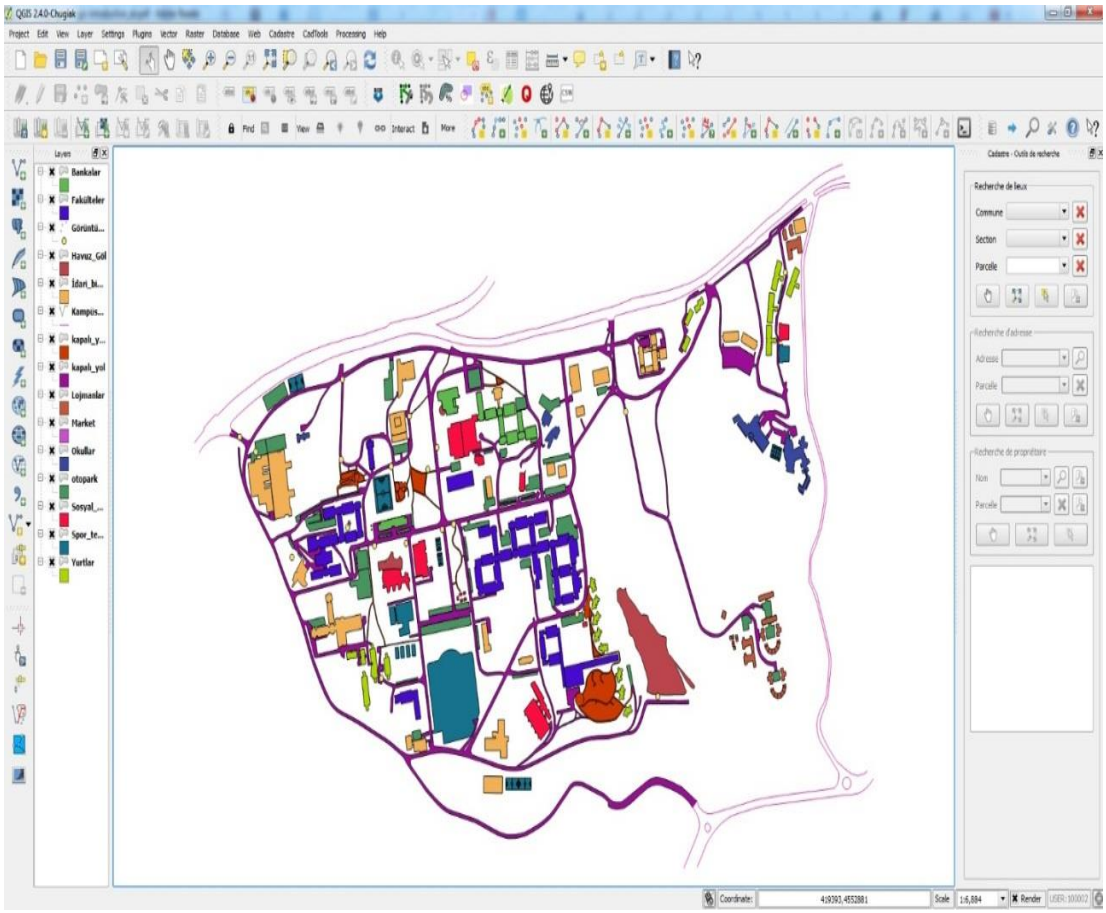

Figure 6.4: Screenshot showing ITU campus data in a GIS application.

6.1.1 Quantum GIS

There are many different GIS Applications available. Some have many sophisticated features and it costs too expensive. In other cases, we can obtain a GIS Application for

free. Deciding which GIS Application to use is a question of how much money we can afford and personal preference. For these work, we will be using the **Quantum GIS** Application, also known as **QGIS**. Quantum GIS is completely free and we can copy it and share it with our friends as much as we like. We can always download free copy **Quantum GIS** Application. [14]

Therefore, QGIS is a cross-platform free and open-source desktop geographic information systems (GIS) application for viewing, editing, and analyzing geospatial data from a variety of vector, raster, and database formats. [15]

QGIS provides integration with other open source GIS packages, including PostGIS, GRASS, and MapServer (e.g. GeoServer) to give users extensive functionality. Plugins, written in Python or C++, extend the capabilities of QGIS. There are plugins to geocode using the Google Geocoding API, perform geoprocessing (fTools) similar to the standard tools found in ArcGIS, interface with PostgreSQL/PostGIS, SpatiaLite and MySQL databases. Then:

- Viewing of vector and raster data from different formats, including PostGIS, Esri Shapefiles, OGC services, and more
- Interactive exploration of spatial data, including on-the-fly reprojection
- Create, edit, and export spatial data
- Perform spatial analysis, including map algebra, terrain analysis, network analysis, and more
- Publish maps to the internet
- Adapt QGIS to our needs with the extensive plugin library

After a brief introduction to QGIS, we start to create or modify our ITU Ayazaga campus spatial data (GIS Data which data is another word for information). To do so, we'll add layers (vector or raster) to the QGIS by using add vector or add raster layer tools.

We wanted to create a map with a different layers such as faculties, dormitories, markets, library and many other layers, so we need to create separate layers for each faculty (like: Insaat, Maden, Fen Bilimleri….) in order to call as a discrete layer on the map. We can do this or any edition in QGIS. For example if the layers that we are using have different projection then our MapServer (GeoServer) is used, it well not appear on the map. Therefore, we need to change and adjust by using QGIS.

As in this dissertation, the main purpose is to create a web GIS map and explicating procedures, so we are not go in more detail about QGIS. Then imagine that we have done all that we need to do in QGIS.

Now we have spatial data and ready to use on the map. As we can find from Figure 6.3 and also Figure 6.5 we need to import, store and keep these data in a database in order to use for the web.

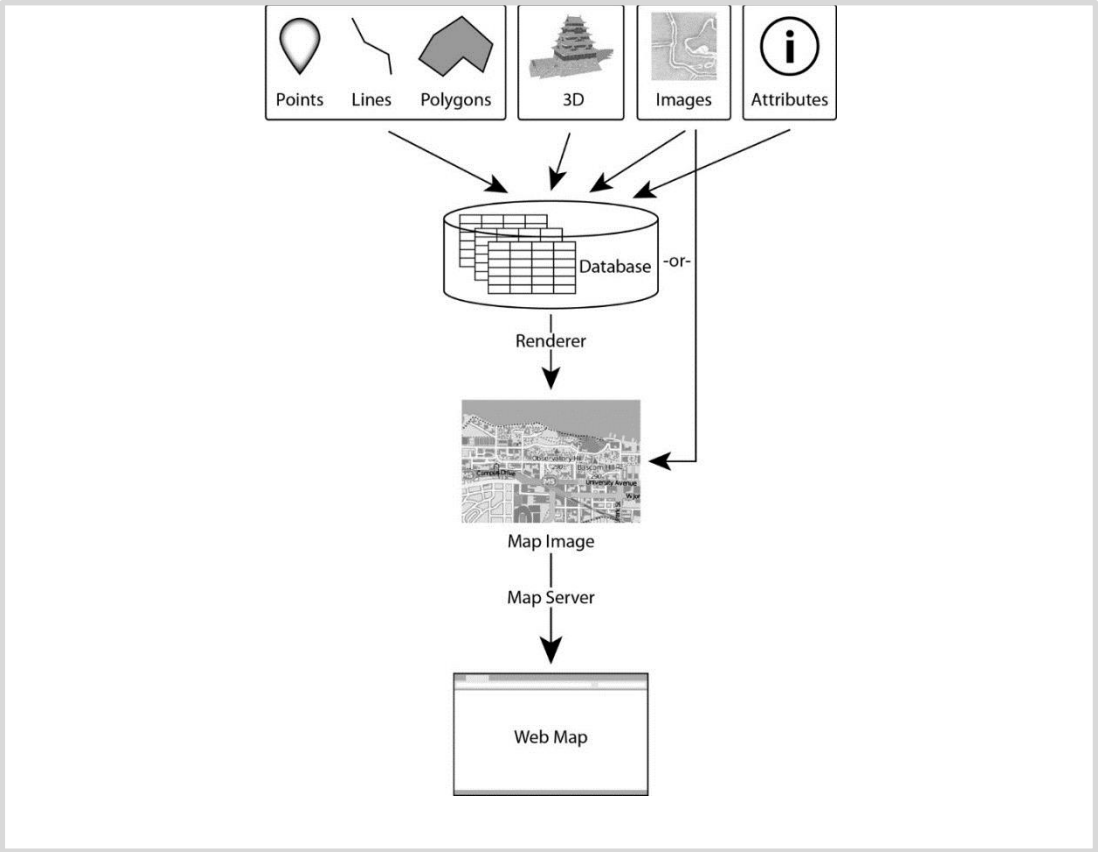

Figure 6.5: Storing GIS Data for the Web. [16]

6.1.2 Storing GIS Data for the Web

There are lots of ways to turn an existing tile service into something new and interesting, which usually don't require setting up our own server. As previously we discussed about using a library or API to add layers to the map, which can be drawn dynamically from the script, can come from a WMS, or can be hosted privately in a separate GIS file. But if we have been surfing around the net for map services and we want to try to making our own, there are basically three steps:

1. Store the data
2. Make the images (tile or otherwise)
3. Serve the map

The steps of serving a map are include of: 1) Data is aggregated, typically in a database; 2) a rendering engine is used to transform the data into an image, which can be done either on-the-fly or in advance with the image cached; 3) the image is served through a map server, which may also provide a rendering engine depending on the software used.

The first order of business is to know how the data—the points, lines, polygons, and/or images we want to use on our map—are stored and how to access them.

Without delving too deeply into the fundamentals of GIS, we'll review the two main types of GIS data. Vector data (points, lines, and polygons) are stored in tables as sets of geographic coordinates and attributes. They can be in CSV, ESRI Shapefile (SHP), XML/KML, GeoJSON, SVG, or a database format. Raster data, or collections of pixels that make up images, can be stored in a number of formats, including database formats, some of which have geographic data embedded. If the data includes coordinates that enable it to be projected accurately onto a map, it is said to be georeferenced.

An important web standard for data transfer is XML (Extensible Markup Language), a form of script that can be used to carry geographic and attribute data and metadata in a way that's readable by humans and JavaScript/HTML. It has become an important language for transferring geographic data over the internet, so much so that early on the Open Geospatial Consortium (OGC) defined a Geography Markup Language (GML) standard, which continues to be updated and refined. It includes specific definitions for spatial data in vector, imagery, and real-time sensor formats. It doesn't care about platforms or file structures, making it a universal tool. To repeat: GML is a language standard, not a file type. The transfer of GML-encoded data over the internet is accomplished with WFS, or Web Feature Service, another OGC specification like Web Map Service (WMS), only for GML data instead of map images. Data transferred through WFS can be displayed as a map in a client browser if it first gets rendered by a map server using a Feature Portrayal Service, which adds style rules to it to turn it into a map.

Of course, a lot of geographic data exists in a variety of other formats that can be used in desktop mapping applications like ArcGIS and Google Earth or parsed by script. In many cases, the files need to be interpreted or transformed for web use. A pair of open-source libraries provided by the Open Source Geospatial Foundation (OSGeo), called

GDAL and OGR, are key for geospatial file processing. These may be incorporated into other programs to retrieve file information or convert between file types. GDAL deals entirely with raster files, while OGR is for vector files. As of this writing, OGR is able to read, create, and modify 25 different georeferenced file types, and GDAL can do the same with 45 raster types, plus many more they can either read or write but not both. These include a number of database formats. Obviously this is too much to cover in a short paper, but a few of those file types are the most commonly used on the web and worth discussing.

6.1.2.1 Vector Formats

1. CSV: Comma-Separated Values (.csv), the simplest possible data table format, consists of a continuous string of values separated by commas and line breaks, and is usually read/modified using Microsoft Excel or similar table software. CSV is typically only used as a format for holding attribute data. In order for the data to be georeferenced, there must be columns for coordinates, almost always stored as latitude and longitude values.

2. ESRI Shapefile: Actually a collection of four or more files (.shp, .dbx, .shx, and .prj, optionally .sbn, .sbx, .xml, or .obr), each representing a different type of table or data manipulation, for years this has been the most widely-used format for transporting GIS data. It's owned by ESRI, and lately being challenged for supremacy by KML and GeoJSON formats. Uploading shapefiles to web applications is typically done by first archiving them together in a zip file.

3. KML: Keyhole Markup Language (.kml, .kmz compressed; so-called because it was invented by Keyhole, Inc. before they were bought by Google) is an XML-based format for use in Google Earth. It's readable/changeable in any text editor, and able to be projected by most mapping software that's not made by ESRI. Now its own OGC standard, it mostly complies with the GML standard, but is much narrower in scope.16 For instance, KML doesn't include datum or coordinate reference system information or metadata.

4. GeoJSON: JavaScript Object Notation (.json) is similar in concept to KML, but writes the data in JavaScript instead of XML. Theoretically this makes it easier for scripts written in various languages to access.

5. SVG: Scalable Vector Graphics (.svg) are XML-based graphics files, like KML but without the georeferencing. Unlike KML files, they can be rendered by any modern

browser. They are typically created using graphics software such as the free program Inkscape or Adobe Illustrator, and can be layered directly onto a web map using an API or JavaScript library. [17]

6.1.2.2 Raster Formats

6. GeoTIFF: Despite the large profusion of raster formats used in GIS and supported by GDAL, many have very specific uses or are proprietary to particular software packages. For specialized purposes, it's worth consulting the documentation to find the best format. GeoTIFF is the most widely-used image format with embedded georeference data. It is organized just like a regular TIFF (Tagged Image File Format; .tif) file, but tagged with a header that includes geographic coordinate and projection information.

7. PNG: The Portable Network Graphic (.png) was invented as a successor to GIF and has become a standard format for raster web graphics. It was designed for the web, not for print, so it only uses RGB (light-based) colors. An open format, it is commonly used for the 256×256 pixel tiles in tile maps. While it doesn't natively support georeferencing, tiles are assigned addresses when rendered that correspond to their geographic positions, and methods can be used in tile services that translate pixel addresses to lat/long coordinates.

8. JPEG2000: The folks who brought us the wildly popular JPEG (.jpg) compressed image format designed JPEG2000 (.jp2) as its successor. But the format was essentially stillborn, failing to be adopted by most of its intended users due to a closed architecture and lack of support for specific applications. It has superior compression and can store GML data to make it georeferenced and even add vector layers. Despite these selling points, .jp2 is rarely seen in web maps. Its intended purpose was to speed the transfer of large images, but tile services using PNG files have bypassed that need.

9. DEM: Digital Elevation Model (.dem) raster are used strictly to show elevation data. They are in black and white, and each pixel is assigned a value from 0-255 according to the elevation of that location. Freely available from USGS and NASA, DEMs can be used by GIS software to create a variety of useful layers, from hillshading to contours to 3D models. [15]

6.1.2.3 Database Formats

Say we have a layer of features on our web map that we want to add new features to or delete features from, or add new information about the features. Where can we store these features to make them easy to change? Databases give the advantage of neat organization and quick access to the data. A relational database is a collection of tables that can be cross-referenced using a common identifying attribute (column). An index that just stores the feature identifiers and the tables they appear in is used to speed up operations with the database. A database management system (DBMS) is typically used to access and write to the database. This is a program designed to interface with the database, usually using a version of Structured Query Language (SQL). A server needs a DBMS to serve geographic data stored in a database to clients on demand (but not to serve map images that are not stored in a database). Several DBMS systems can interpret geographic data. The three most popular are listed below.

MySQL: Oracle's open-source DBMS is massively popular for all kinds of uses, including several of the web's biggest sites. There is a free "Community" version and paid editions with more functionality. It comes with spatial extensions to manipulate geographic data. One caution is that it uses some syntax that departs from the SQL standard. [15]

PostgreSQL/PostGIS: A complete, functional open source DBMS, Postgres (as it's commonly referred to) seems to be the most popular for open-source web mapping. It requires OSGeo's PostGIS extension, which comes with the standard Postgres installation, to handle georeferenced data. It is used by OpenStreetMap, GeoServer, MapServer, and CartoDB. [15]

SQLite/Spatialite: A "light" open-source DBMS, unlike MySQL or Postgres, SQLite doesn't need a server to run. It also doesn't require any set-up or configuration to run, but just uses a command-line interface to create and manipulate databases and tables, and writes the databases as .db files. Spatialite, an extension that gives SQLite geospatial support, must be downloaded separately. It is modeled on PostGIS and comes with its own graphics user interface.

6.2 Database

There are bunches of other DBMS out there that support spatial data. Big-name products include IBM DB2, Oracle Database, Microsoft SQL server, Sybase/Boeing

Spatial Query Server, and GE's Version Managed Data Store. The growing number of "NoSQL" databases which, as we might expect, use a different language than SQL might prove worth experimenting with for advanced web programmers. Graph databases such as AllegroGraph and Neo4j store data as linked nodes. CouchDB and MongoDB store data as JSON documents. SpaceBase is built to store complex spatial objects and uses a Java or C++ interface. Many database systems can handle raster, though storing them in a database may not give any performance gain. Which is a better storage strategy apparently depends on who we ask.

6.2.1 PostgreSQL/PostGIS

For mange and store the GIS data will be use **PostgreSQL/PostGIS.** Because as it mentioned above **PostGIS** is an open source software program that adds support for geographic objects to the **PostgreSQL** object-relational database. **PostGIS** follows the Simple Features for SQL specification from the Open Geospatial Consortium (OGC). [18]

Now we are going to start to import our GIS data to postgreSQL database by using postGIS. We will download it from the internet and then will install it our PC system. During installation the program will ask us to put user and password for our database and also for PostGIS, and we should to keep them for future uses.

Now we are ready to add our data (layers) to database but before adding data will start postgreSQL. There is a ready-made database, which can put data inside it otherwise, we can create a new database by expanding the postgreSQL and right click on the Database and clicking on New Database will give a name for our database, will click on Ok button Figure 6.6. Our database will be appear in the main Database trees. We need to expand our database so that the red X sign on the database icon should be removed, so now we are ready to import our data to the new database.

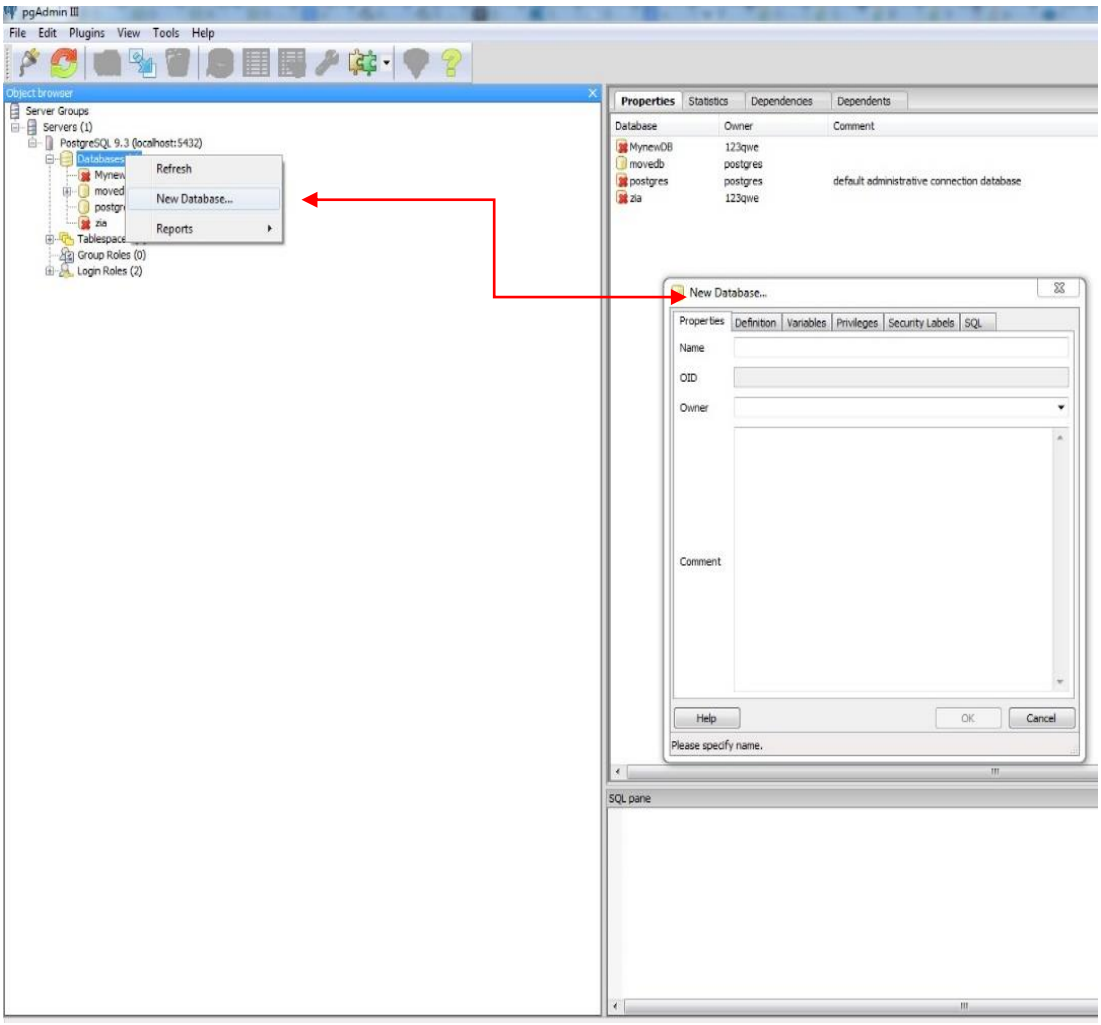

Figure 6.6: Creating own new database in postgreSQL.

Once postGIS is installed, it needs to be enabled in each individual database we want to use it in. for this as Figure 6.7, we need to run a following command to enable the database with PostGIS extensions. Click on the SQL button at the top of PgAdmin III, then click the “Execute query” button.

Create extension postgis;

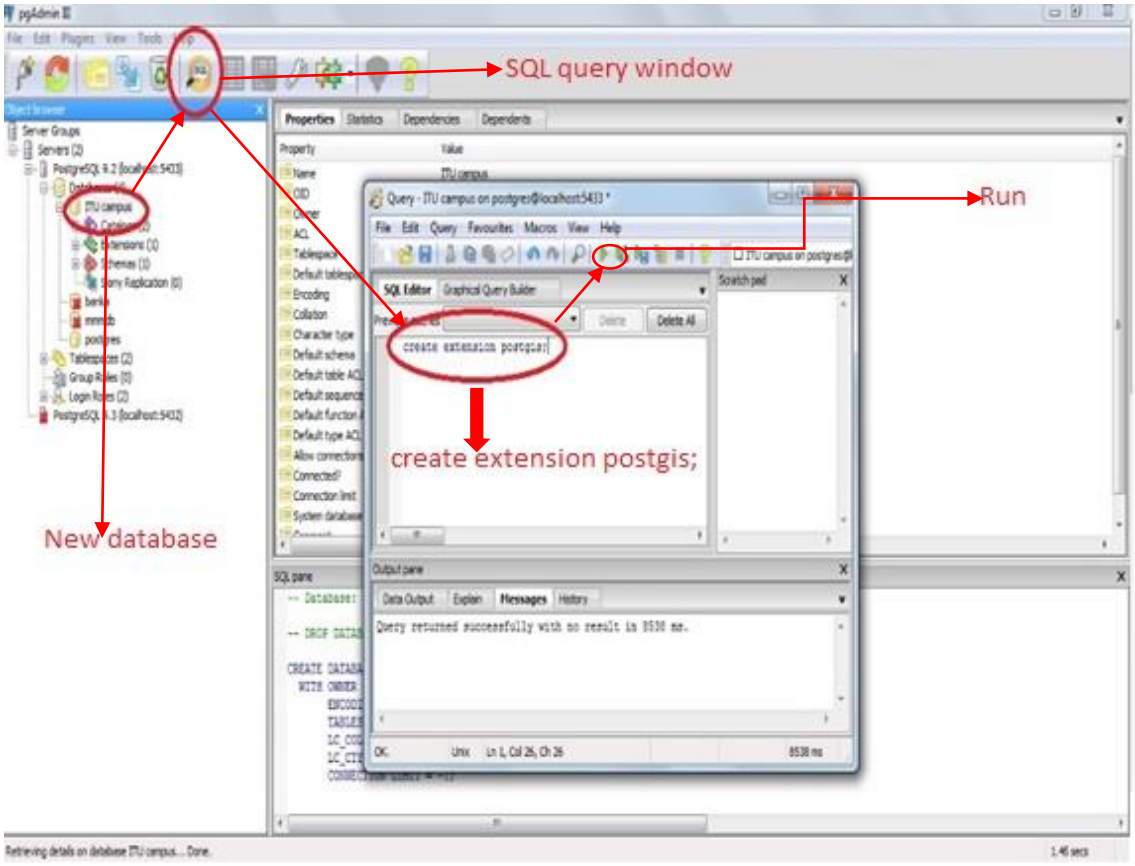

Figure 6.7: Enabling new database with PostGIS extensions**.**

Now we'll try to import data. At this time, we need to open PostGIS program Figure 6.8.

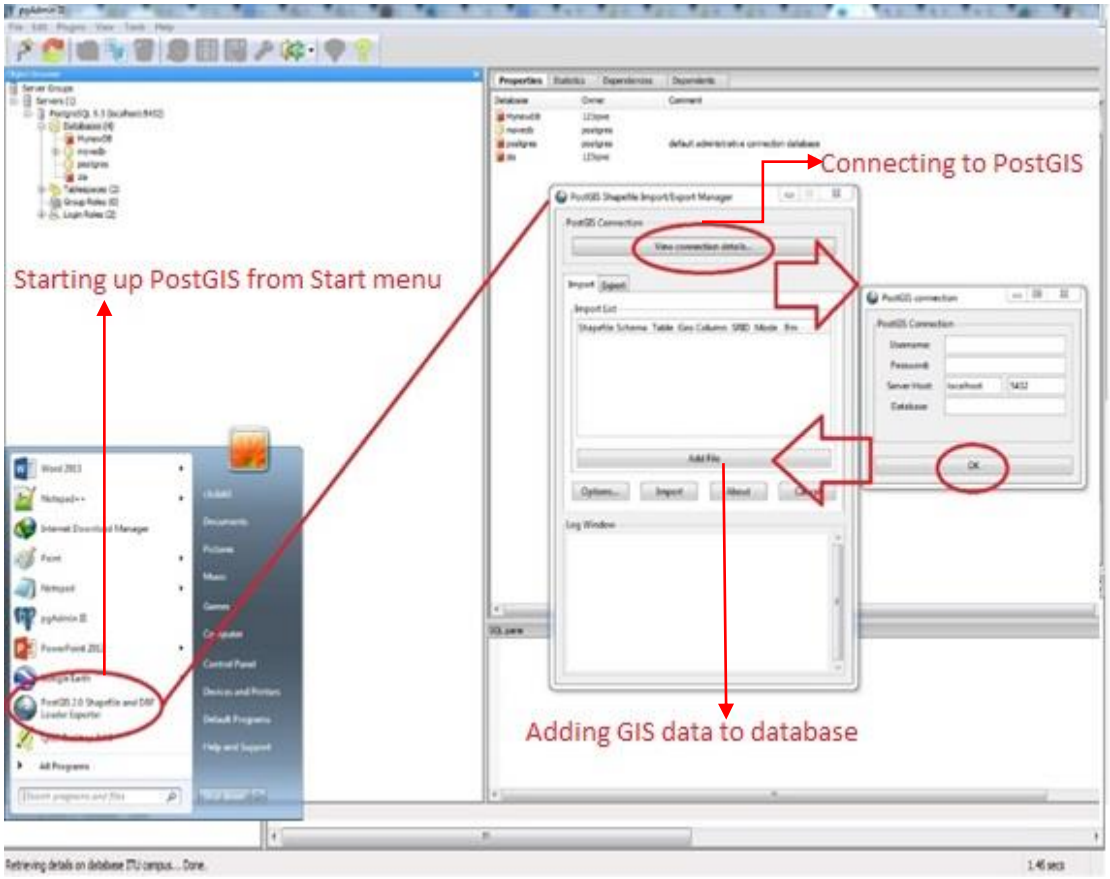

Figure 6.8: Connecting to PostGIS.

As we can see from Figure 6.8, as it is mentioned before, in order to connect PostGIS to postgreSQL and let us to import data, we need to put user and password. There are two other options that we need to fill them. One is Sever Host, which we should put localhost and the port number that this part are related to GeoServer section and we will discuss on part 6.3. Second and last option is Database which here we must put our new database name or any other database that we want to add date into it. After connecting will click on Add file button and continue to importing data Figure 6.9.

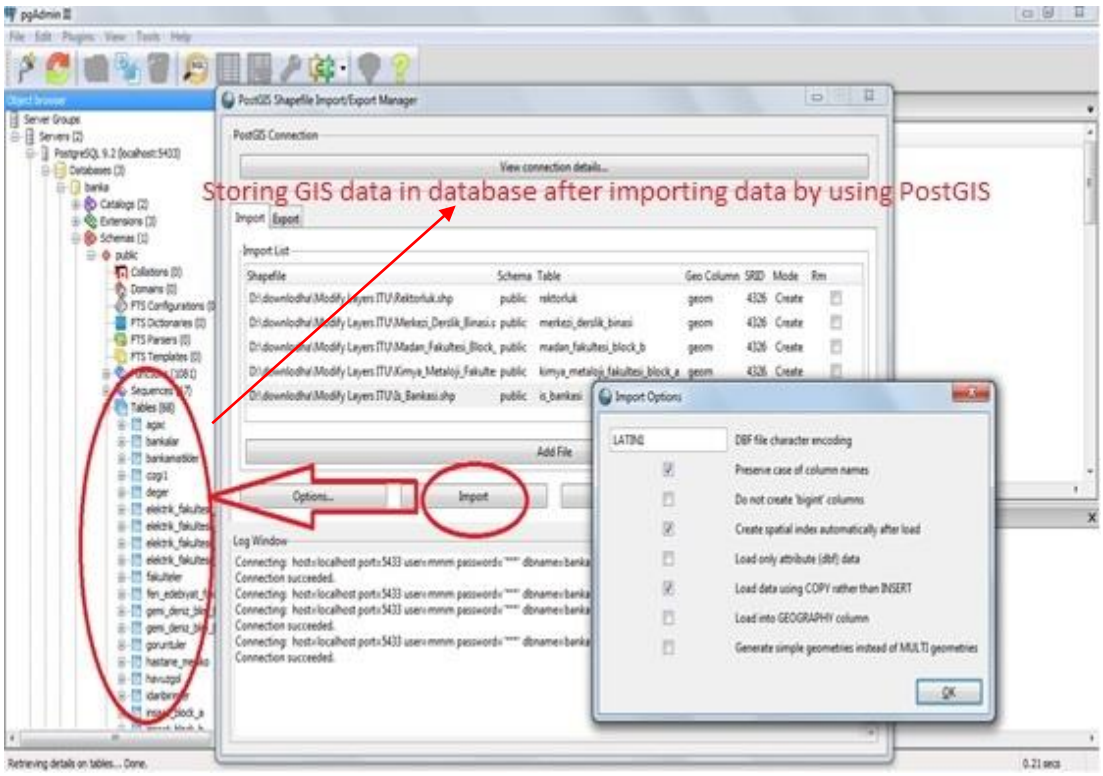

Figure 6.9: Importing data to PostgreSQL by using PostGIS.

If it connects correctly, we will see Connection Succeeded message on log window. During importing data, in the opened window there is Option button which we should set as it shown on Figure 6.9. Things that we should notice during importing data is setting projection correctly, otherwise we cannot access to the layers because of incorrect map projection. As it pointed before, we are using EPSG 4326 map projection. Therefore, we should set SRID (Spatial Reference System Identifier) on the Import options as 4326. Then we'll import and after data added to the database as it shown on Figure 6.9, we need make a refresh the database by right clicking on the database, on Refresh option, we can see all the data in the database on PostgreSQL.

6.3 GeoServer

Well, we stored our GIS data on the database. Now we want to publish map and the data from database. To do so, we are going to use **GeoServer**, which is an open-source server written in Java allows us to share, process and edit geospatial data. Designed for interoperability, it publishes data from any major spatial data source using open standards, as shown in Figure 6.10. GeoServer has evolved to become an easy method of connecting existing information to Virtual Globes such as Google Earth and NASA World Wind as well as to web-based maps such as OpenLayers, Google Maps and

Bing Maps. GeoServer functions as the reference implementation of the Open Geospatial Consortium Web Feature Service standard, and also implements the Web Map Service, Web Coverage Service and Web Processing Service specifications.

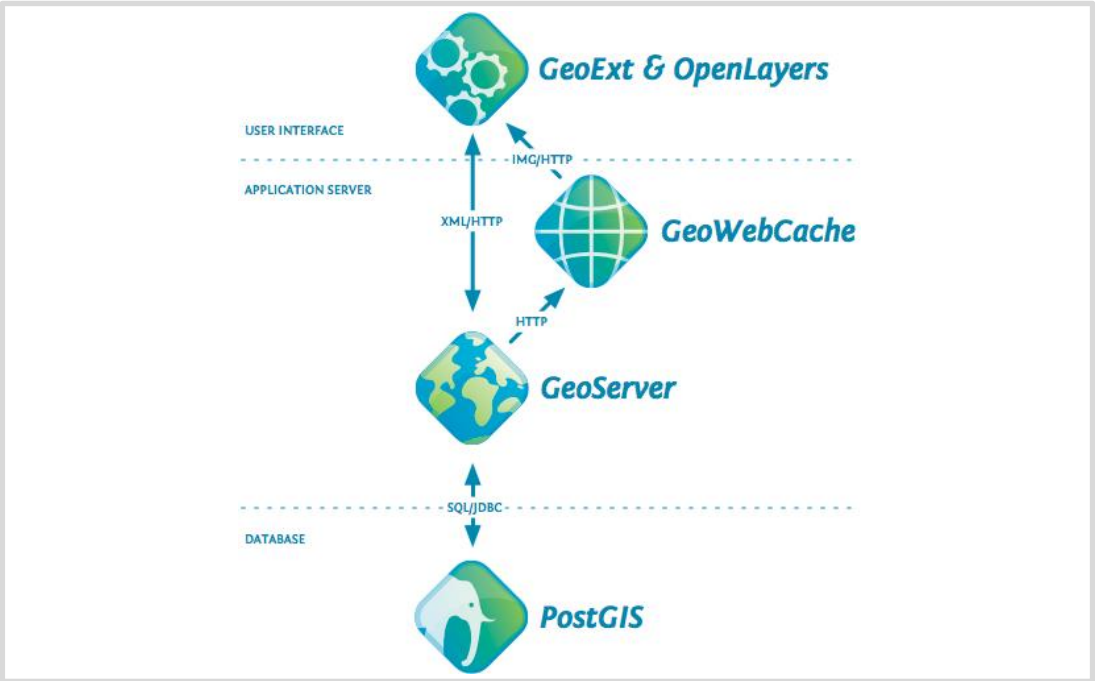

Figure 6.10: Architecture of interaction between GeoServer and other parts. [19]

As it mentioned, GeoServer is a java application for serving maps (and data) for other clients to draw. So by this way we can:

- add a vector and raster data source to GeoServer
- apply color to map features using styling
- test the layers in a simple web map
- learn about clients that can display our maps

For starting to work with GeoServer, as it is an open-source server so we can download it from internet. While we are installing GeoServer same as postgreSQL it will ask us to specify a user name and password. We'll keep them for future uses.

For starting GeoServer, we click on our computer's Start menu and write GeoServer on search program or file place. We'll see some applications go up. There is start GeoServer application which we'll use it to start GeoServer as Figure 6.11, and initializing the program.

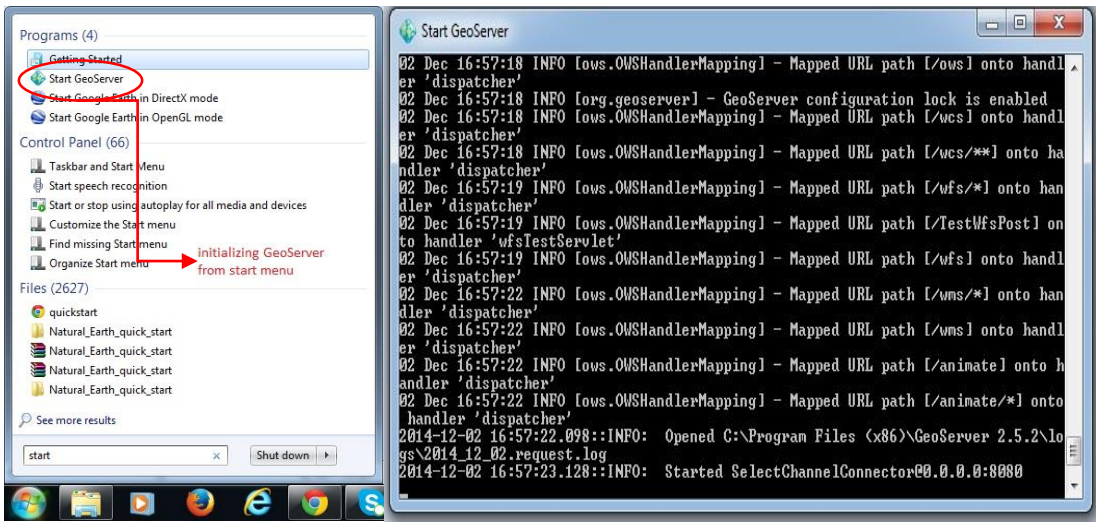

Figure 6.11: starting and initializing GeoServer.

The application will take a few moments to start up and will open a web page at *http://localhost:8080/geoserver/web*. When we first open the GeoServer page, we will see the screen below, first we need to log in using the username admin and password GeoServer.

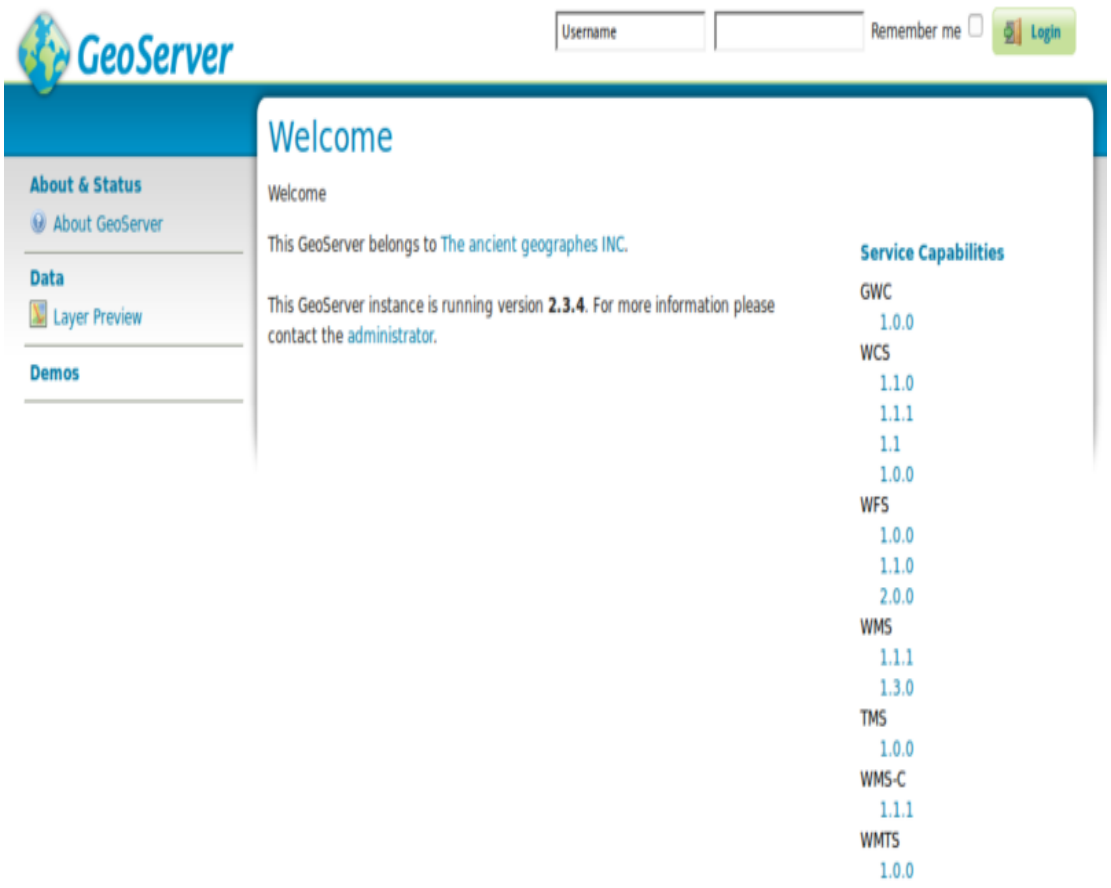

Figure 6.12: Logging page of GeoServer.

We will now see the admin page. Figure 6.13

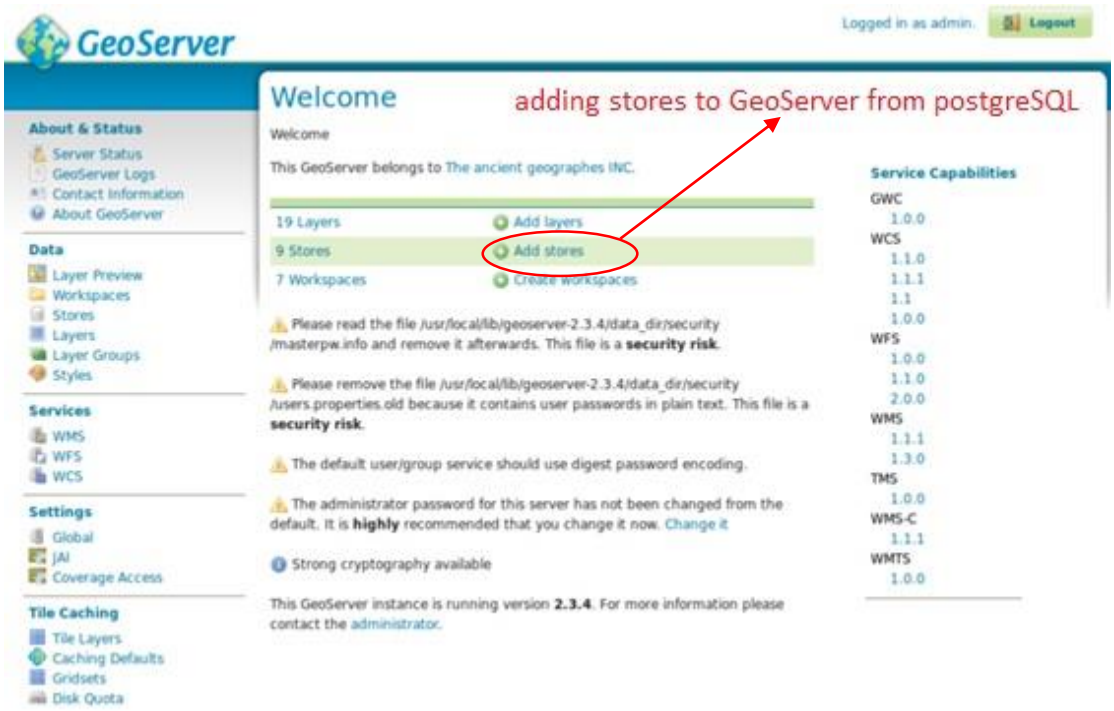

Figure 6.13: GeoServer admin page.

We need to create a Store for our data. From the GeoServer admin page go to Stores and then click on Add new Store. We will see this page:

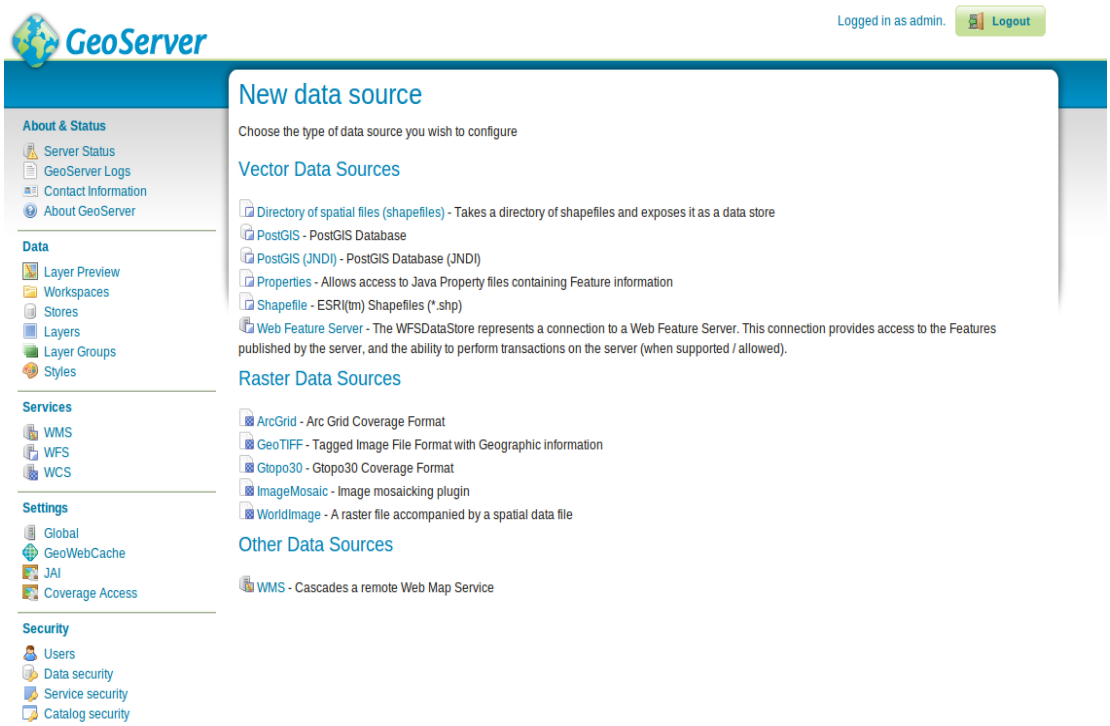

Figure 6.14: Choosing the type of data source to configure.

We will select the PostGIS. So we will see following page.

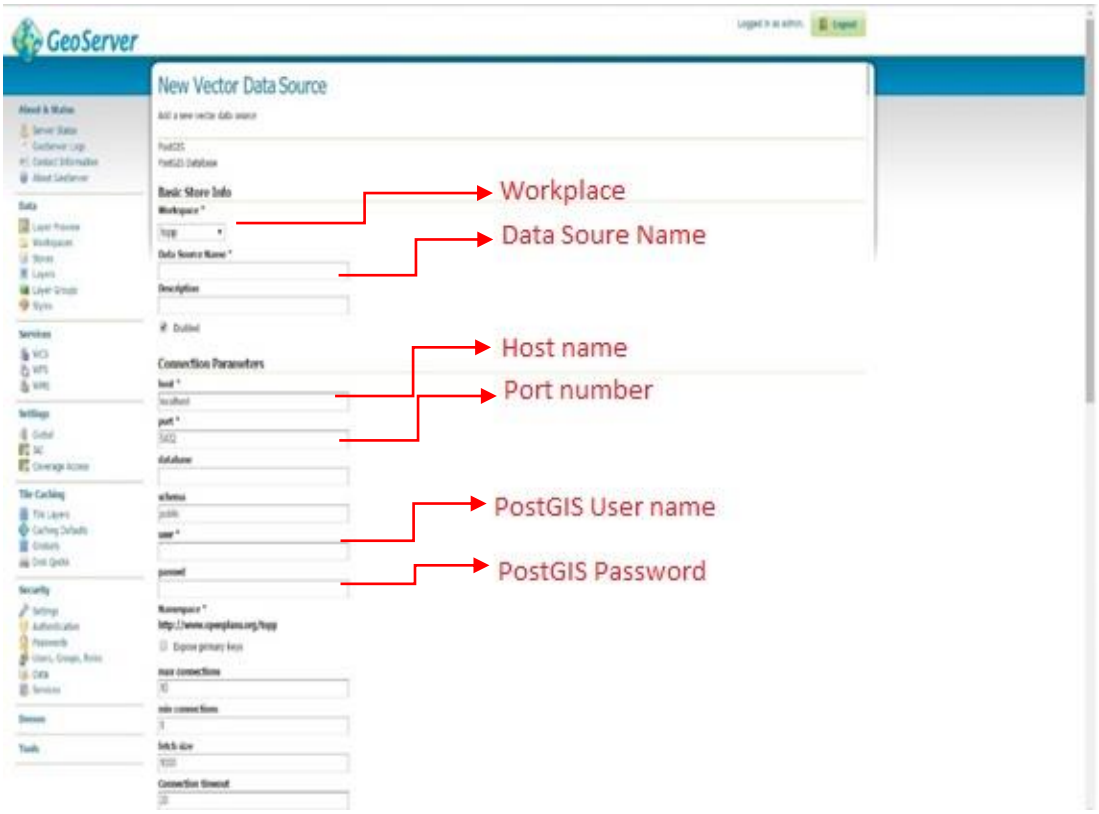

Figure 6.15: Adding vector data source.

We need to fill all options which specified by star (*).on the Basic Store Info section, we'll choose one of exist workspace otherwise we can create the new workspace with a desired name. To do this, as we can see on the left hand side of this page there is some heading with some applications. On the Data menu, select the workspace and add new workspace. We should select a Data source name. It can be anything we would like. In *Connection Parameters*, in host option we'll write localhost. In port option we should put the port number which we are selected during installing the GeoServer. By default it is 5432. We'll put our database name on the next option. We need to put PostGIS user and password at the next steps. After we filled required options we'll save the steps.

Now we have new layers and in next page, we need to publish them as Figure 6.16.

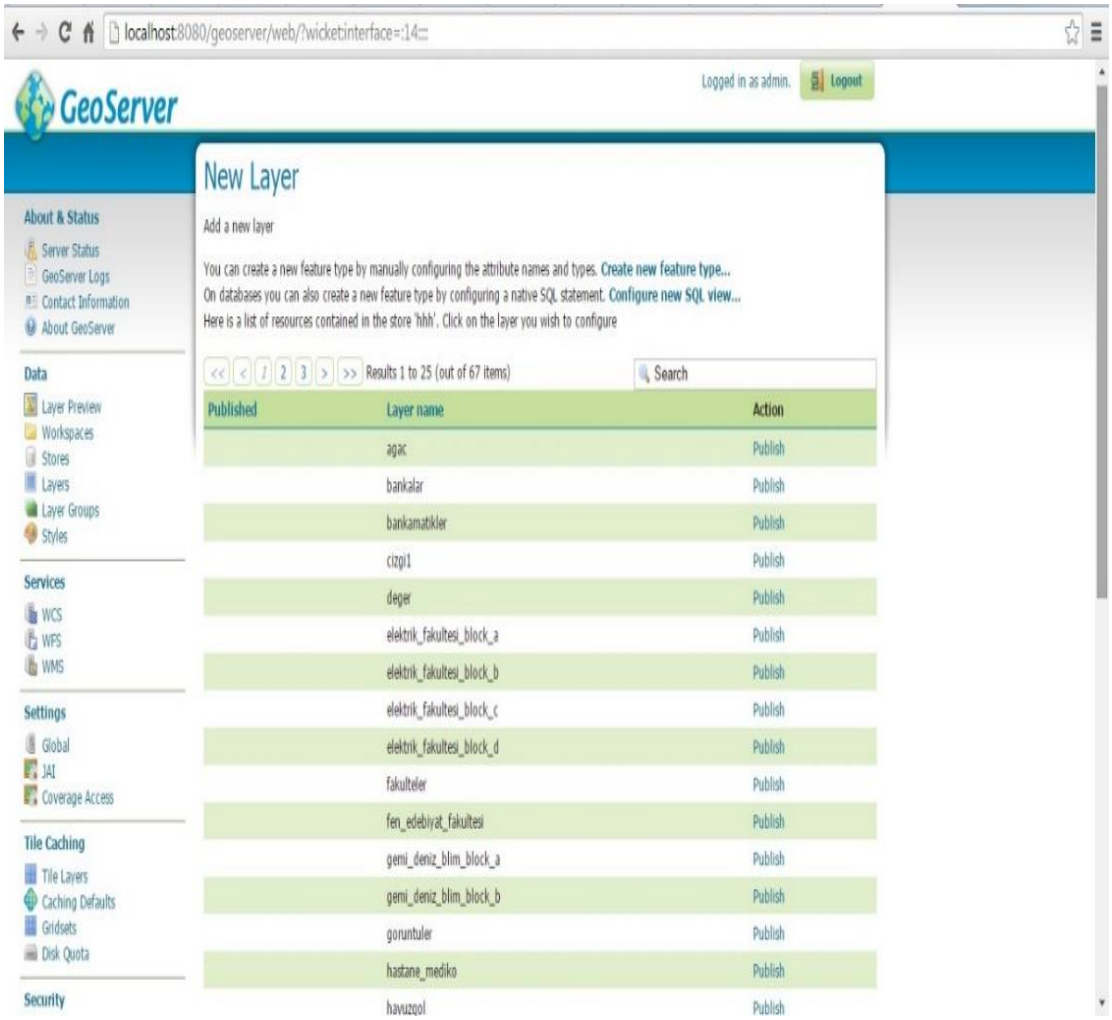

Figure 6.16: Publishing page to finish adding the data.

Under Data menu there is Layers, which we can manage the layers being published by GeoServer (Add a new resource or remove selected resources) as Figure 6.17.

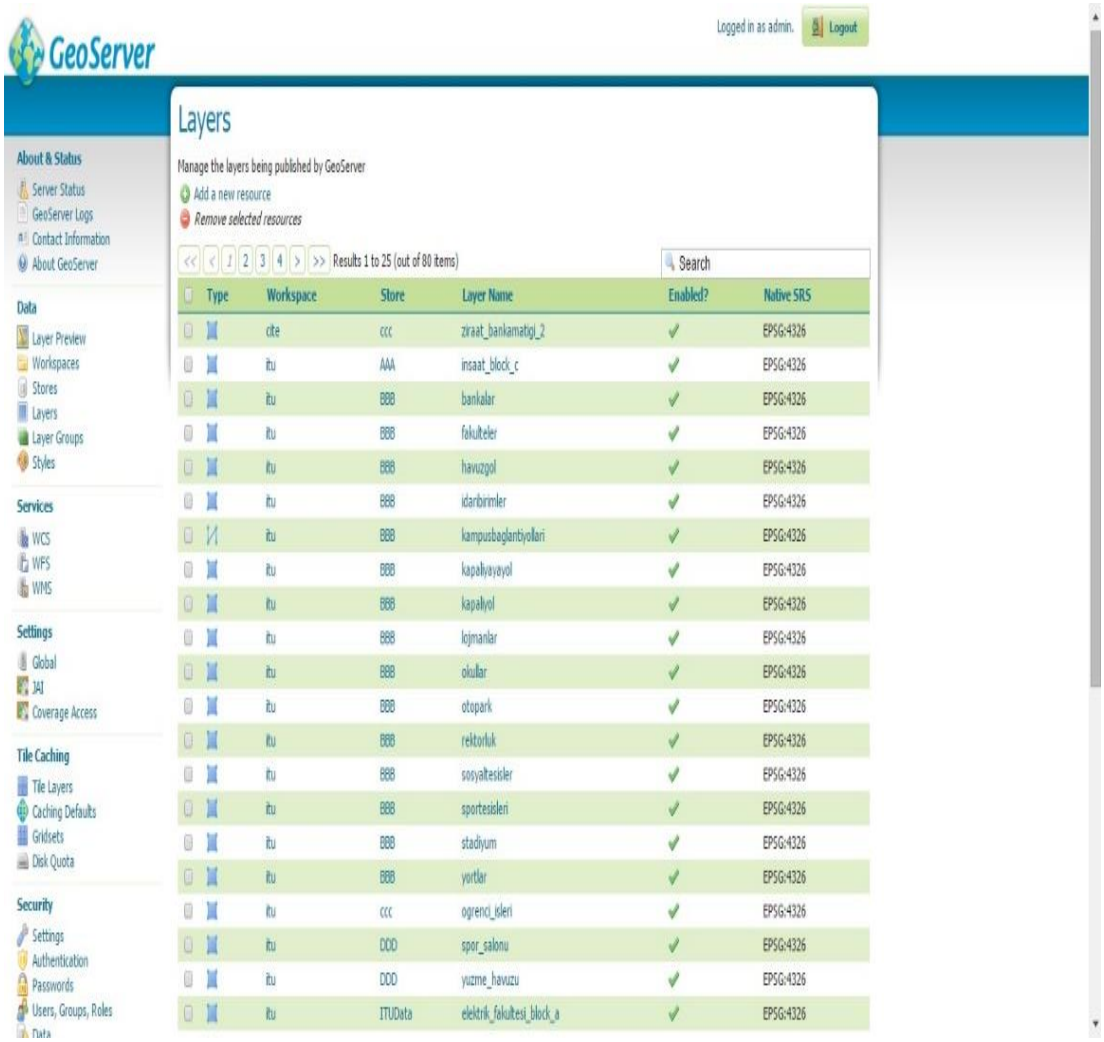

Figure 6.17: Managing the published layers.

We should publish each layer separately. Once we click on each layer to publish, this will take us to the Layers page (Edit Layer) which by editing layer we configure the resource and publishing information for the current layer, Figure 6.18.

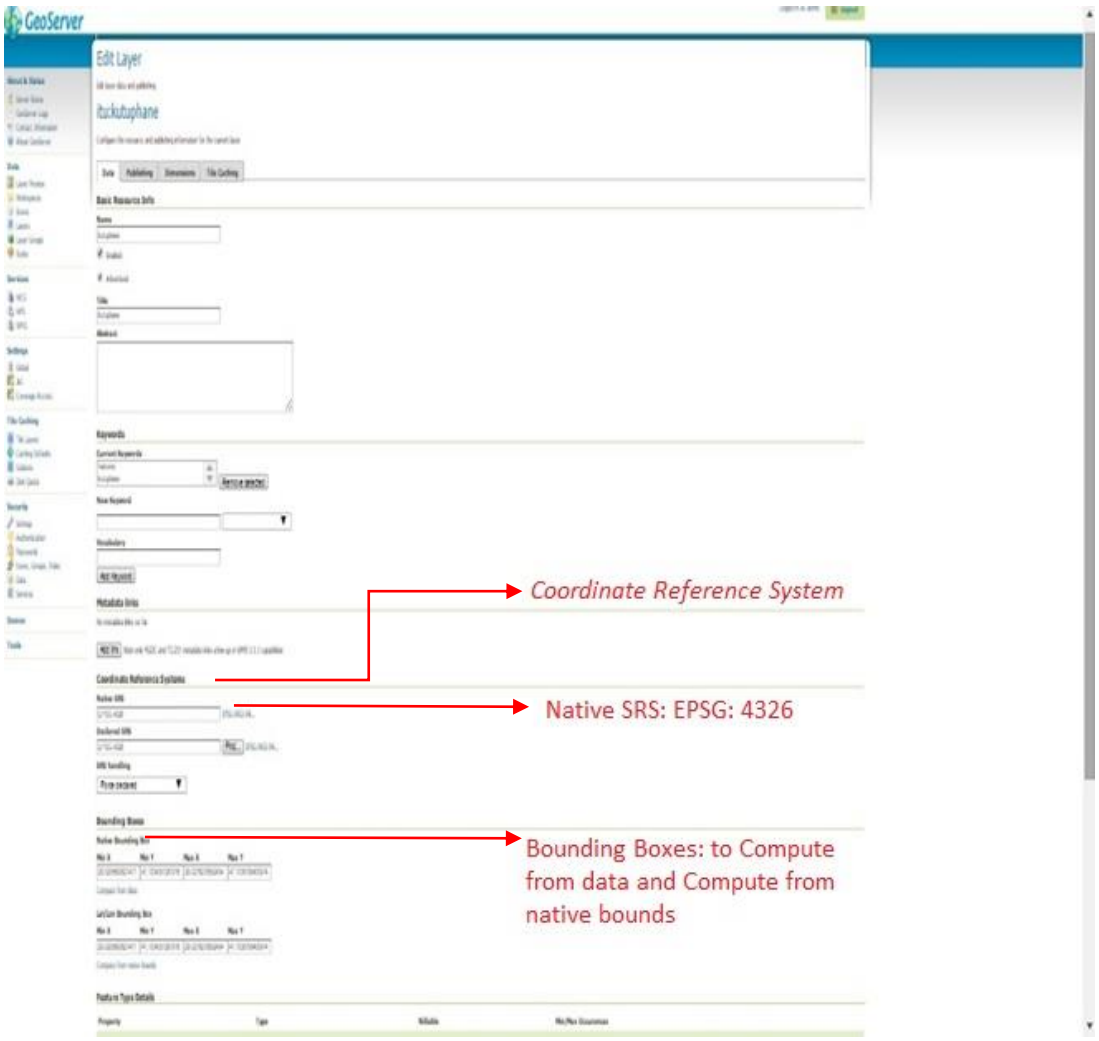

Figure 6.18: Editing layers.

As we scroll down the page we will see that GeoServer has filled in many of the fields for us. When we reach *Coordinate Reference System* we will notice that under *Native SRS* that it says UNKNOWN we will need to fill in the next box (*declared SRS*) to make sure GeoServer knows where the data is. We'll type *EPSG: 4326* in the box. Then click on *Compute from data* and *Compute from native bounds* to fill in the Bounding Boxes. Finally hit save and we have published our first layer. We'll do this for each layer separately. List of all layers configured in GeoServer will being on Layer Preview on the left hand side in Data application section and by pressing Layer Preview it provides previews in various formats for each layer, Figure 6.19.

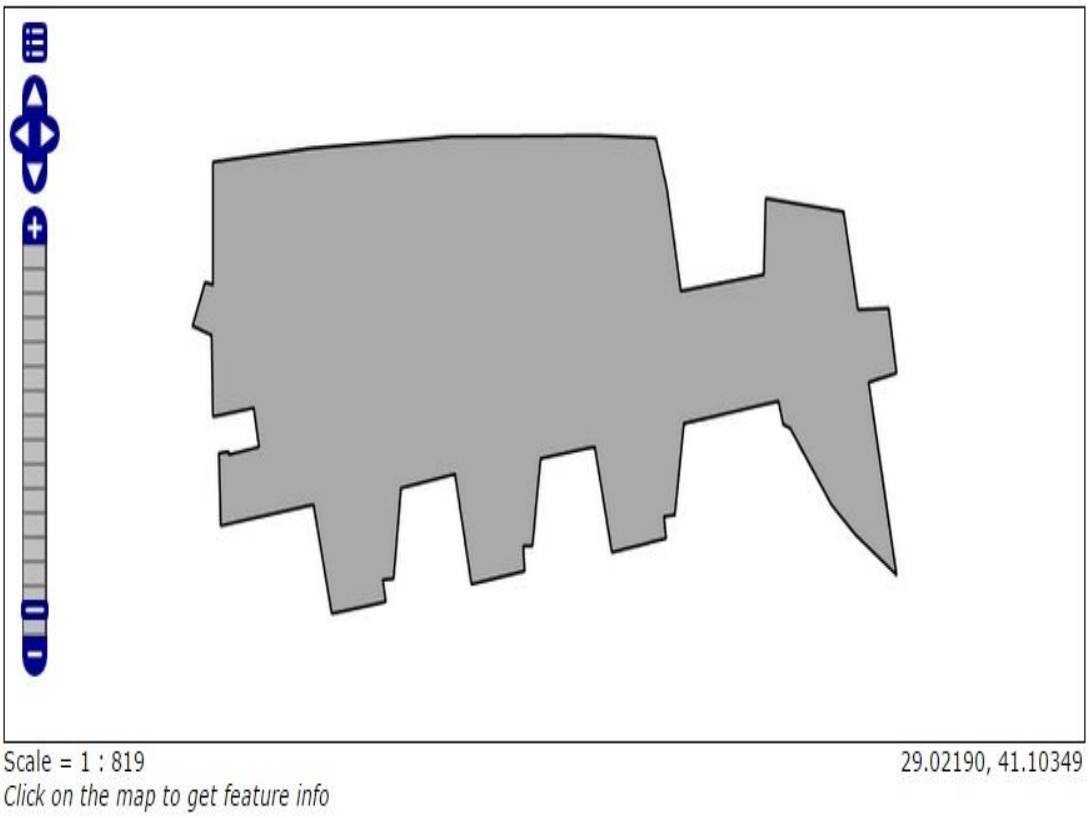

Figure 6.19: Layer preview of the layer that are loaded on the server.

6.3.1 Styling data in GeoServer

To style a data set into a map layer GeoServer uses an OGC standard called *Styled Layer Descriptors* (SLD). These are represented as XML files which describe the rules that are used to apply various symbolizers to the data.

On the style window there is an export button which allows to save the SLD file that defines the style. Once we saved the two styles we can go to the GeoServer admin page again and select *Styles* (at the bottom of the *Data* section). Then I select the *Add New Style* link, at the bottom of that page is a file upload box and a browse button. Clicking this allows to hunt around on the hard drive to find the files we just saved. Once we've found one we want, we click the upload link (next to the browse button) and a copy the file appears in the editor as Figure 6.20.

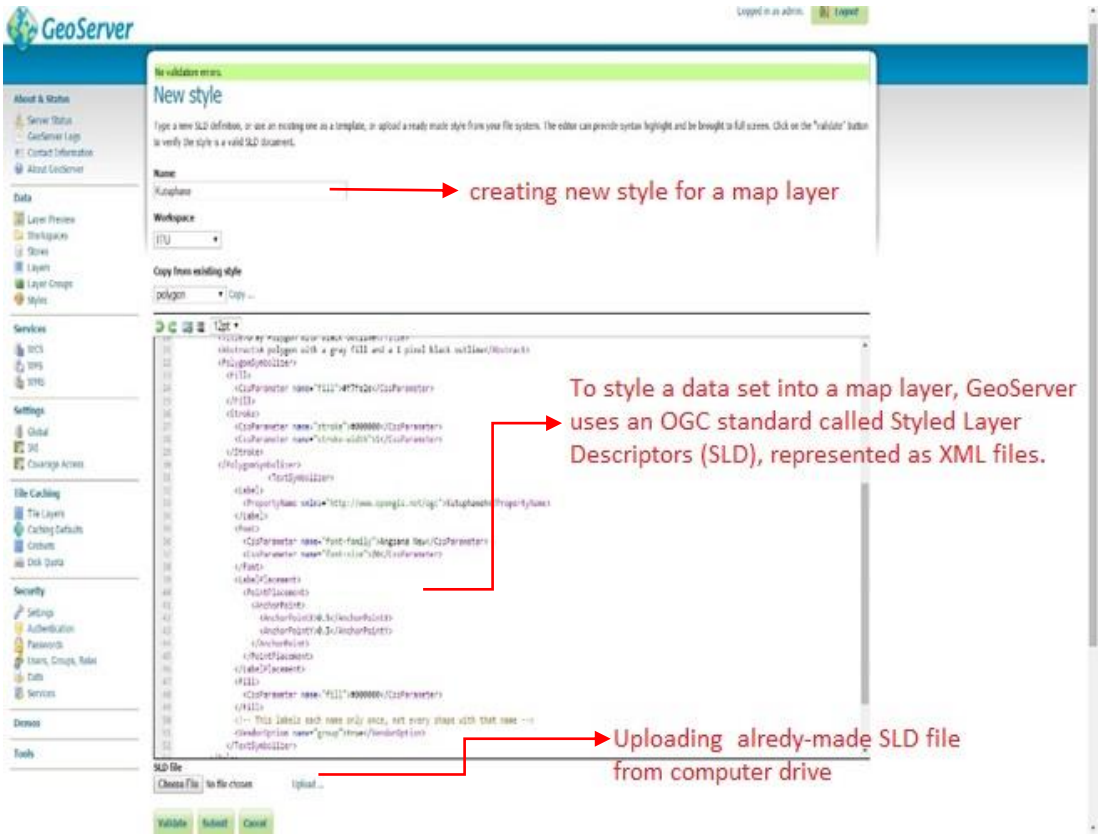

Figure 6.20: Creating new style.

Now we would like to create new style for the data. For instance, we'll create style for the Kutuphane layer which it has no style as it shown on Figure 6.19. We want to change background color of layer to yellow and border color of layer to black. Therefore, we will use HTML color codes as #F7FE2E for yellow and #000000 for black. I also want to give name for this layer so that as it appears on the map, name of layer (Kutuphane) will be seen on the map too with font name of *Angsasa New* and font size of *20*, black color code for the text (#000000), and also we can place the word (Kutuphane) in any position. To do this, on the style window, in the XML code we'll do as following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<StyledLayerDescriptor version="1.0.0"
 xsi:schemaLocation="http://www.opengis.net/sld
StyledLayerDescriptor.xsd"
 xmlns="http://www.opengis.net/sld"
 xmlns:ogc="http://www.opengis.net/ogc"
 xmlns:xlink="http://www.w3.org/1999/xlink"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <!-- a Named Layer is the basic building block of an SLD document
-->
```

```
  <NamedLayer>
    <Name>default_polygon</Name>
    <UserStyle>
    <!-- Styles can have names, titles and abstracts -->
      <Title>Default Polygon</Title>
      <Abstract>A sample style that draws a polygon</Abstract>
      <!-- FeatureTypeStyles describe how to render different
features -->
      <!-- A FeatureTypeStyle for rendering polygons -->
      <FeatureTypeStyle>
        <Rule>
          <Name>rule1</Name>
          <Title>Gray Polygon with Black Outline</Title>
          <Abstract>A polygon with a gray fill and a 1 pixel black
outline</Abstract>
          <PolygonSymbolizer>
            <Fill>
              <CssParameter name="fill">#f7fe2e</CssParameter>
            </Fill>
            <Stroke>
              <CssParameter name="stroke">#000000</CssParameter>
              <CssParameter name="stroke-width">1</CssParameter>
            </Stroke>
          </PolygonSymbolizer>
                      <TextSymbolizer>
            <Label>
            <PropertyName xmlns="http://www.opengis.net/ogc">Kutupha
neh</PropertyName>
              </Label>
              <Font>
                <CssParameter name="font-family">Angsana
New</CssParameter>
                <CssParameter name="font-size">20</CssParameter>
              </Font>
              <LabelPlacement>
                <PointPlacement>
                  <AnchorPoint>
                    <AnchorPointX>0.5</AnchorPointX>
                    <AnchorPointY>0.5</AnchorPointY>
                  </AnchorPoint>
                </PointPlacement>
              </LabelPlacement>
```

```
            <Fill>
              <CssParameter name="fill">#000000</CssParameter>
            </Fill>
            <!-- This labels each name only once, not every shape
with that name -->
            <VendorOption name="group">true</VendorOption>
          </TextSymbolizer>
        </Rule>
      </FeatureTypeStyle>
    </UserStyle>
  </NamedLayer>
</StyledLayerDescriptor>
```

All right now we click on the validate button. If our style was not validated the highlighted lines will give us an error but we can safely ignore the error (or delete those lines as they don't do anything) and we are ready press the Submit at the bottom of the page.

6.3.2 Adding the Style to the Layer

Now we are going to add the style that we created on the previous part to the layer. So we click on the Layers link in the Menu on the left of the GeoServer window. Then we click on the layer (Kutuphane), and select the Publishing tab (Figure 6.21) and change the Default Style box to the name of the style we uploaded in the previous section. Now we click Save and go to the Layer Preview page to check that it looks good.

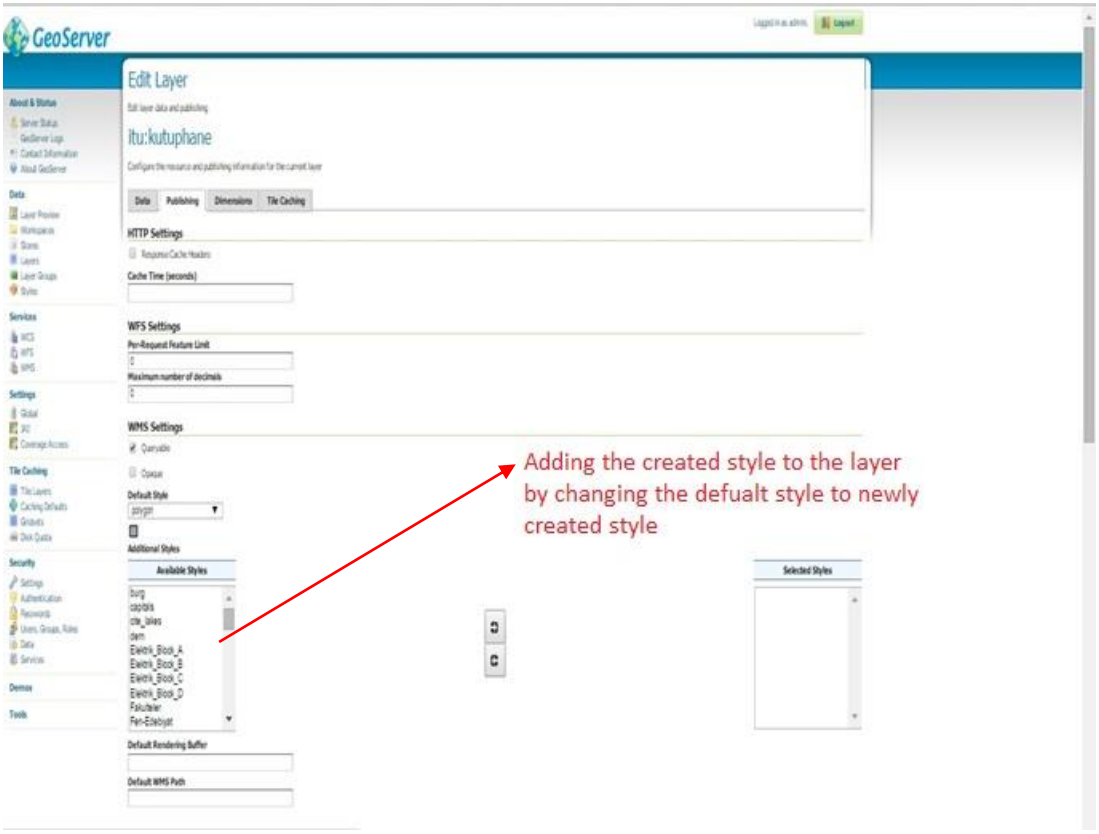

Figure 6.21: Adding the Style to the Layer.

As we can see from the Figure 6.22, we added the style to our Kutuphane layer.

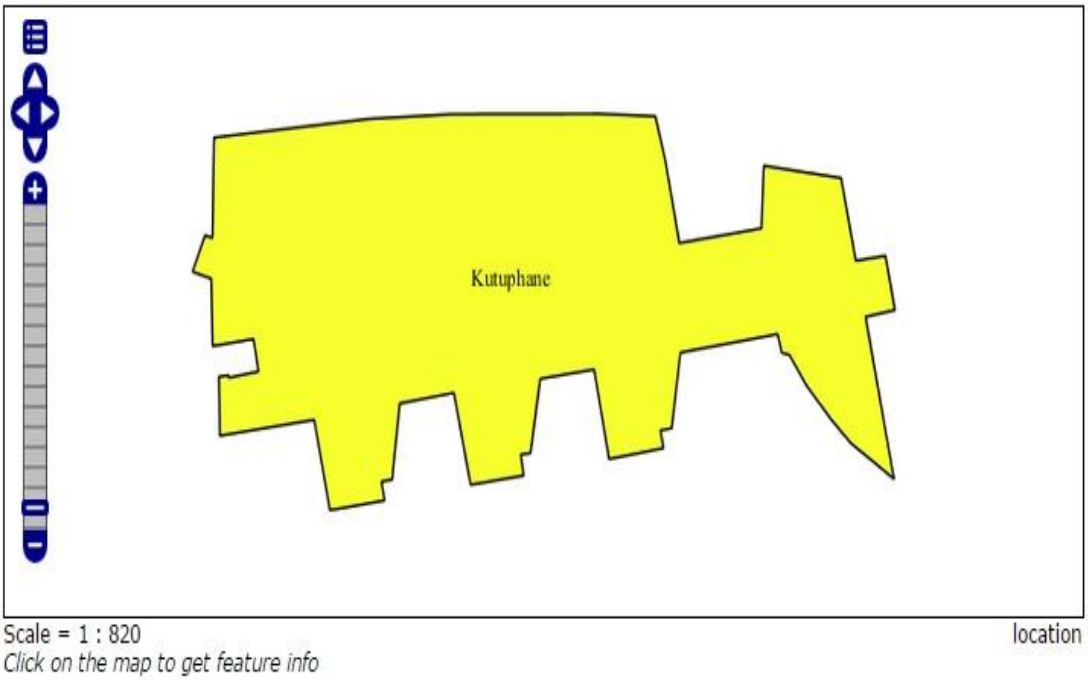

Figure 6.22: Layer with style.

We will create styles and add them to other layers as we have done above.

Now that we finished to styling all layers, which want to use in our ITU web GIS application, they can applied to the map. As we remember from the code that contains the OpenLayers.js file from previous sections where we were added layers to the JavaScript code, we used following link:

```
    "http://localhost:8080/geoserver/itu/wms", {
```

In computer networking, localhost means our computer. It is a hostname that the computer's software and users may employ to access the computer's own network services via its loopback network interface. Using the loopback interface bypasses local network interface hardware. The local loopback mechanism is useful for testing software during development independent of any networking configurations. If a computer has been configured to provide a website, directing its web browser to *http://localhost* may display its home page. [8] On most computer systems, localhost resolves to the address *127.0.0.1*, which is the most-commonly used IPv4 loopback address. 8080 is the port it uses to connect on and is mostly a debugging/testing convention. Usually http servers have the ports of 80 or 8080. Then all that is doing tell what computer to connect to and what port to connect on and *'localhost: 8080'* is showing us the default web page from our PC, not the Internet.

The next part (geoserver/itu/wms) related to our map server name which we are using GeoServer as our map server. *Itu* is our workplace that we selected on GeoServer and Web Map Service (WMS) is a standard protocol for serving georeferenced map images over the Internet that are generated by a map server using data from a GIS database.

Therefore, by interaction and interoperability of user interface (OpenLayers, QGIS), server application (GeoServer) and database (postGIS/postgreSQL) we are enable to create ITU Ayazaga campus web GIS application, Figure 6.23.

Figure 6.23: ITU Ayazaga smart campus map on the web.

7. Result & Discussions

A Smart Campus has the goal to provide an interactive map and services that allows people to visualize, locate and access information about the campus street network, buildings, classrooms, departments, programs, offices, cafeterias, residence, green spaces, and so forth. The concept of Smart Campus follows the same principles than the known Smart Cities concept/policy: find “smart” solutions to provide quantitative and qualitative improvement of life of the populations, whether to an urban population or students in a campus.

Smart ITU intends to improve the resources and behavior management in the campus and, most of all, aims to produce spatial data and manage information to optimize the use of all resources related to the campus. Being a university campus a smaller enclosure than a city, this concept serves real testing ground for the implementation of real Smart City.

Optimize the resources and improve the services is a key factor for the university campus managers. Regarding the improvement of the services in a campus, about 80% of the data that the campus administration works with daily are georeferenced.

Map is the most efficient and effective means is to inform about spatial information. It locates geographic objects, while the shape and color of signs and symbols representing the objects inform about their characteristics. It reveals spatial relations and patterns, and offer the user insight in and overview of the distribution of particular phenomena. An additional characteristic of particular on-screen map is that it is often interactive and has a link to a database, and as such allow for more complicated queries. So map is the result of the visualization process. The purpose of the map could be to present the data to wide audience or to explore the data to obtain better understanding.

This thesis showed the basis for creating a Web-GIS application in order to leading ITU Ayazaga campus as a smart campus. For creating a Web-GIS application first step is to create GIS data. In order to display any feature once user ask to appear on the map so we need to import GIS dada. Features, which we are using as a GIS data, can be any vector or raster layers (Shp, GeoTIFF…) and can be Text (XML). Therefore, the first

step is to create GIS data. There are many different GIS Applications available to generate, modify and manipulate GIS data such as ArcGIS and Quantum GIS Applications. Some have many sophisticated features and it costs too expensive. In other cases, it is possible to obtain a GIS Application for free. Deciding which GIS Application to use is a question of how much money we can afford and personal preference. For these work, Quantum GIS Application, also known as QGIS is used. Storing the created GIS data is the second step. In order to Storing GIS Data for the Web there are many DBMS that support spatial data. Well-known products include IBM DB2, Oracle Database, Microsoft SQL server, Sybase Spatial Query Server, and some others.

In order to manage and store GIS data PostgreSQL/PostGIS is used in this thesis. PostGIS is an open source software program that adds support for geographic objects to the PostgreSQL object-relational database. There is a ready-made database in postgreSQL otherwise we can Create own new database in postgreSQL. In this step after creating new database, enabling it with PostGIS extension is important.

After that GIS data stored on the database, publishing map and the data from database is the third step. To do so, there are applications like as MapServer, GeoServer and ArcGIS Server which allow to share, process and edit geospatial data. GeoServer, which is an open-source server written in Java is used as a map server.

Therefore, by interaction and interoperability of user interface applications, server applications and database applications enable to create ITU Ayazaga campus web GIS application. The advantage of this system is that each sections can be produced by different products and any component of the structure can be replaced with other products and vice versa.

Web-GIS is a fast-changing field indeed, the technology has gone through many iterations. The Web is the ideal platform for GIS. Because they complement each other so well, the Web and GIS are a natural fit. Web-GIS allows professionals, organizations, and the public share and collaborate, making GIS eminently more accessible to a wide range of users and at a much lower cost.

Open-Source Web-GIS software systems have reached a stage of maturity, sophistication, robustness and stability, and usability and user friendliness rivalling that of commercial, proprietary GIS and Web-GIS server products. The Open-Source Web-GIS community is also actively embracing OGC standards, including WMS.

WMS enables the creation of Web maps that have layers coming from multiple different remote servers/sources.

Users can switch to browse the e-map or the satellite image and manipulate the map functions, like pan, zoom in/out, zoom to full extent, etc. Users can also utilize query tool to query specific information of locations. Therefore, the map display content can be greatly enriched and users can understand the process of GIS data collecting, managing and displaying by manipulation.

In conclusion, The Virtual Smart Campus for the Istanbul Technical University is a project that aims to transform the Istanbul Technical University (ITU) into a “Smart Campus” and ITU Ayazaga campus web GIS application is a part of this project.

8. Future work Suggestions

Show more info:

The app can include more feature layers to show other kind of places: toilets, canteens, stationery, etc. Moreover, functionalities to search those places can implemented.

Indoor positioning:

The app can get the position of the device that is running, but only uses Global Positioning System (GPS) and does not work well indoors. A good improvement would be to include the indoor positioning; there are many technological options for doing so.

3D Campus:

The app can has more intuitive perspective by using 3D building models and 3D visualization, so that all the campus buildings, street network and all the outdoor spaces as well as, indoor spaces will be displayed future in 3D.

Other improvements:

With the progress of time, adding applications that increase efficiency and getting more user-friendly. For instance, adding other languages can be included, and even new screen sizes if necessary.

REFERENCES

[1] Colin Harrison & Ian Abbott Donnelly (2011), A THEORY OF SMART CITIES**,** harrisco@us.ibm.com, abbottia@uk.ibm.com 1IBM Corporation, New Meadow Road, Armonk, NY 10504, USA 2IBM UK Limite Northminster House, Natural England, Peterborough PE1 1 UA, UK, Received date: 22.01.2015

[2] Url-1 History of Istanbul Technical University, < http://en.wikipedia.org/ Wiki/Istanbul_Technical_University>, received date: 12.02.2015

[3] Pinde Fu, Ph.D.(ESRI,2012), pfu@esri.com Project Lead / Senior Developer Professional Services Division ,Web GIS: Principles and Applications, received date: 17.02.2015

[4] Jan-Menno Kraak & Allan Brow (2001), Web Cartography – Developments and prospects , received date: 17.02.2015

[5] Url-2 Basic System Architecture, <http://www.esri.com/news/ arcwatch/1110/ Web- gis.html>, received date: 20.02.2015

[6] Url-3 OpenLayers Official page, <http://www.openlayers.org>, Received date: 25.02.2015

[7] Zhejiang University of Finance & Economics Hangzhou, China (2006), Develop Web GIS Based Intelligent Transportation Application Systems with Web Service Technology, received date: 10.03.2015

[8] Mikael Andersson (2005), Department of Communication Systems Lund Institute of Technology Email: mike@telecom.lth.se, Introduction to Web Server Modeling and Control Research, Received date: 18.03.2015

[9] Url-4 ArcGIS Server 9.3.1 Help,<http://webhelp.esri.com/arcgisserver/9.3.1/java/ index.htm#what_is_a_map_service.htm>, received date: 20.03.2015

[10] Url-5 Database Management System, < http://www.techterms.com /definition/dbms>, received date: 22.03.2015

[11] Url-6 Open Geospatial Consortium (OGC) Web Mapping Services (WMS) <http://www.lib.ncsu.edu/ gis/ogcwms.html>, received date: 27.03.2015

[12] Url-7 Application architecture, < http://www.geotec.uji.es/improving-environmental-decision-making-utilizing-a-web-gis-to-monitor-hazardous industrial-emissions-in-the-valencian-community-of-spain/>, received date: 11.04.2015

[13] Url-8 Interoperability of standards-based platform, <http://boundlessgeo.com /solutions /opengeo-suite/>, received date: 19.04.2015

[14] Url-10 QGIS Official page, <http://qgis.org>, received date: 23.04.2015

[15] Url-9 Databases and data access APIs, <http://wiki.openstreetmap.org/wiki/ Databases_and_data_access_APIs#Choice_of_DBMS>, received date: 27.04.2015

[16] Url-11 Web Mapping Services Overview, http://wicoastalatlas.wordpress.com/ Received date: 01.05.2015

[17] Wei Chen, (2009), The Design and Implementation of a Web-based GIS for Political Redistricting, received date: 08.05.2015

[18] Url-12 PostgreSQL/PostGIS Official page, < http://www.postgresql.org/>, Received date: 09.05.2015

[19] Url-13 OpenGeo Architecture, < http://www.prodevelop.es/en/opengeo-enterprise-suit-0>, received date: 10.05.2015

CURRICULUM VITAE

Name Surname: Rouhollah NASIRZADEH DIZAJI

Place and Date of Birth: Tabriz, Iran, 23.04.1982

Address: ITU Informatics Institute Maslak / Istanbul

E-Mail: nasirzadeh.rouhollah@gmail.com

B.Sc.: Civil Engineering

Professional Experience and Rewards:

İmensaze-e Shabestar Co. 2005-2006, (Site Supervisor)

Tehran Doman Construction Co. 2006–2007, (Supervisor of project)

Tosae -e-Maskan-e Azarbayjan Construction Co. 2008–2012 , (Supervisor of project)

Professional Certificate of Buildings Supervisor Grade 3 from Iran Road & Urbanization Ministry. 2010