Computer Project Final Report

by Indranil Bannerjee

A Report

On

Geolocation App Using HTML 5

By

Indranil Banerjee **2010A7PS096U**

At

BITS, Pilani, Dubai Campus
Dubai International Academic City, Dubai
UAE

Second Semester 2012-13

A Report

On

Geolocation App Using HTML 5

By

Indranil Banerjee 2010A7PS096U Computer Science

Prepared in Partial Fulfillment of the
Project Course: BITS C331 Computer Projects

At

BITS, Pilani, Dubai Campus
Dubai International Academic City, Dubai
UAE

Second Semester 2012-13

BITS, Pilani, Dubai Campus

Dubai International Academic City, Dubai

II - Semester 2012-13

Course Name: Computer Projects

Course No: BITS C331

Duration: 10 February, 2013 - 6 June, 2013

Date of Start: 10 February, 2013

Date of Submission: 6 June, 2013

Title of the Report: Geolocation App Using HTML 5

ID No. / Name of the student: 2010A7PS096U / Indranil Banerjee

Discipline of Student: BE (Hons) Computer Science

Name of the Project Supervisor: Dr. S Vadivel

Key Words: Geolocation API, HTML 5, JavaScript, CSS, Modernizr

Project Area: Geolocation App using HTML 5

Abstract: Knowing the location of your users can help boost the quality of your Web site and the speed of the service. In the past, users had to progressively input their location and submit it to a site, either by typing it, using a long drop-down list, or by clicking a map. Currently, with the HTML5 Geolocation API, finding users (with their permission) is easier than ever [7]. The aim of this project is to create a web application using HTML 5 which will determine the location of the user, then the user can find places near him/her, for example hospitals, banks, ATMs, restaurant, police stations, etc and also give textual directions assistance. He/she can also get the distance he/she is away from the place he/she is searching or search place within his/her desired radius.

Signature of Student

Date: 06-06-2013

Signature of Supervisor

Date: 06-06-2013

Acknowledgements

I express my thanks to Dr. R K Mittal, the Director of BITS Pilani, Dubai Campus, for making it possible for me to do this project.

My deepest thanks to my mentor Dr. S Vadivel, for guiding, correcting and supporting me throughout this project, and providing me with necessary resources to start with. Without his help my project would have been a distant reality.

Table of Contents

Chapter 1

Introduction

1.1 HTML 5, JavaScript and CSS

HTML 5, the latest revision of the HTML standard, is a markup language for constructing and presenting content for the World Wide Web a core technology of the Internet [15] and is a W3C Candidate Recommendation. HTML 5 is a collection of individual features, so instead of looking HTML 5 support in older browsers, one should detect support for individual features like canvas, video, or Geolocation. HTML 5 specification also defines how the syntax and semantics interact with JavaScript, through the Document Object Model (DOM).

HTML 5 supports all the form controls from HTML 4, but it also consists of new input controls. HTML 5 is already supported with ease by the browsers Internet Explorer, Mozilla Firefox, Google Chrome, Opera and Apple Safari. Mobile browsers already support canvas, Geolocation, local storage and more [17].

JavaScript is an interpreted programming language used to develop synergistic web applications supported by web browsers. JavaScript is a language which is easy to use, light, and impulsive [13]. Developers can easily embed the code functionality for developing synergistic applications inside a web page [13].

Cascading Style Sheets (CSS) is a simple mechanism for adding style (e.g., fonts, colors, spacing, etc.) to Web documents [16]. Nearly all browsers nowadays support CSS and many other applications do too. To write CSS one doesn't need more than a text editor, but there are many other tools available that make it even easier.

1.2 Geolocation API

The Geolocation API is developed by the World Wide Web Consortium (W3C) to standardize an interface for retrieval of information on geographical location for a client side device [12]. The latitude and longitude is made available to JavaScript on the website, which sends the values back to the distant web server and perform flamboyant and fluid location-aware functions like finding local landmarks or showing your location on a map [10]. The Geolocation API [10] is supported by most browsers on the desktop and mobile device devices [6]. In addition, some older browsers and devices can be supported by wrapper libraries [6]. The most common sources of location information are IP address, Wi-Fi and Bluetooth Media Access Control address, radio frequency identification (RFID) [9], Wi-Fi connection location or device Global Positioning System (GPS) and GSM or CDMA cell IDs [9]. The location is returned with a given accuracy

depending on the best location information source available [9]. The Geolocation support is opt-in, which means the browser will never force the user to reveal his/her current physical location to a remote server [6].

GEOLOCATION API SUPPORT [6]

IE	Firefox	Chrome	Safari	Opera	iPhone	Android
9.0+	3.5+	5.0+	5.0+	10.6+	3.0+	2.0+

Table 1: Geolocation API support in web browsers

1.3 Uses of Geolocation API

Knowing the location of users can help boost the quality of your Web site and the speed of your service [2]. In the past, users had to actively input their location and submit it to a website, either by typing it, or by using a long drop-down list, or clicking a map [2]. Now, with the HTML5 Geolocation API, finding your users (with their permission) is easier than never before [2]. Figure 1 shows a Web site using geolocation to determine the location of a user, corresponding in latitude and longitude [2]. The numbers can easily be converted into something more graspable, for example the street name or city [2].

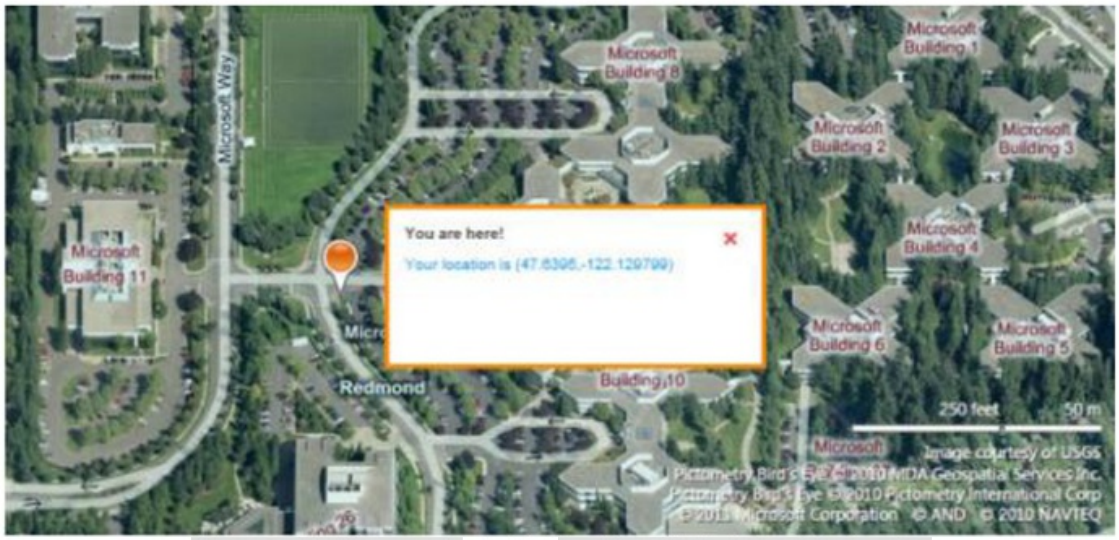

Figure 1: Showing a user's physical location with the help of Geolocation

Using geolocation, the site could recommend optimal travel routes to get people where they're going as fast as possible. Desktop users can get their location of start sorted by proximity to their

computer [2]. Mobile phone users who are trying to get home after a night out could quickly find the closest bus stop within walking distance [2]. The above advantages can be made use of by just another API [2].

Here are simple scenarios that illustrates how a Website can accommodate users and customize their experience by taking their location into consideration [11]. Some of these might look obvious, but the little things often make huge differences [11].

- Provide assistance with directions to users
- Public transport websites are able to list nearby bus stops and metro locations [2].
- Taxi or car service Web sites can find the user's location[2]
- Shopping sites can estimate shipping costs in local currency [2].
- Travel agencies can offer better vacation tips for current location and season [2].
- Content sites can accurately determine the language and dialect of search queries [2].
- Real estate websites can provide average house prices in a particular area, a handy tool when you're driving around to check out a neighborhood or visit open houses [2].
- Movie theater sites can promote regional films based on location [2].
- Online games can blend more reality into the gameplay by giving users missions to accomplish based on real-life locations.
- News websites can include custom made local headlines and local weather on their front page [2].
- Online stores can provide information whether products are in stock at local retailers [2] and discounts based on local festivals.
- Sports and other entertainment ticket sales sites can give promotions of upcoming games and shows nearby [2].
- Job postings can be optimized based on location [2].

1.4 Project Target

The aim of this project named LocationFriend is to deliver location based services using HTML 5 with CSS and JavaScript. The project's main target is to create a web application using HTML 5 which will determine the location of the user, then the user can find places near him/her, for example hospitals, banks, ATMs, restaurant, police stations, etc. He can also get the distance he/she is away from the place he/she is searching or search place within his/her desired radius.

1.5 Summary

The advantages of LocationFriend is to assist the user to know his surroundings better by providing the location of important places with a range the user selects. This will help him/her to find his preferred destination smoothly and effectively without any trouble.

Chapter 2

Coding Steps

Developing the Web app requires the following steps:

2.1 HTML Markup

A new empty HTML file is created. Basic headers are put and next code is put in the header section [8].

```
<script type="text/javascript" src="https://www.google.com/jsapi"></script>
<script type="text/javascript" src="http://maps.googleapis.com/maps/api/js?
libraries=places&sensor=false"></script>
<script src="js/script.js"></script>
```

Figure 2: Linking JavaScript file

Necessary libraries (main Google library with extra library – Places) and the main script.js file (where necessary custom JavaScript code are put) are linked. Now, please put next code into body section [8]:

index.html

```
<div id="gmap_canvas"></div>
<div class="actions">
    <div class="button">
        <label for="gmap_where">Where:</label>
        <input id="gmap_where" type="text" name="gmap_where" /></div>
    <div id="button2" class="button" onclick="findAddress(); return false;">Search for
address</div>
    <div class="button">
        <label for="gmap_keyword">Keyword (optional):</label>
        <input id="gmap_keyword" type="text" name="gmap_keyword" /></div>
    <div class="button">
        <label for="gmap_type">Type:</label>
        <select id="gmap_type">
            <option value="art_gallery">art_gallery</option>
            <option value="atm">atm</option>
            <option value="bank">bank</option>
            <option value="bar">bar</option>
            <option value="cafe">cafe</option>
            <option value="food">food</option>
            <option value="hospital">hospital</option>
            <option value="police">police</option>
            <option value="store">store</option>
        </select>
```

Figure 3: Code snippet for the body section of index.html

2.2 CSS

CSS is required to add nice styles to the webpage. These include

- The resolution and position of the Google map canvas object
- The background color, position and padding of the webpage
- Text alignment, margin and display type of label
- Characteristics of buttons such as background color, cursor, margin, resolution, hovering, etc.

```
#gmap_canvas {
    height: 700px;
    position: relative;
    width: 900px;
}
.actions {
    background-color: #FFFFFF;
    bottom: 30px;
    padding: 10px;
    position: absolute;
    right: 30px;
    z-index: 2;

    border-top: 1px solid #abbbcc;
    border-left: 1px solid #a7b6c7;
    border-bottom: 1px solid #a1afbf;
    border-right: 1px solid #a7b6c7;
    -webkit-border-radius: 12px;
    -moz-border-radius: 12px;
    border-radius: 12px;
}
.actions label {
    display: block;
    margin: 2px 0 5px 10px;
    text-align: left;
}
.actions input, .actions select {
    width: 85%;
}
.button {
    background-color: #d7e5f5;
    background-image: -webkit-gradient(linear, left top, left bottom, color-stop(0%,
color-stop(100%, #cbe0f5));
    background-image: -webkit-linear-gradient(top, #d7e5f5, #cbe0f5);
    background-image: -moz-linear-gradient(top, #d7e5f5, #cbe0f5);
    background-image: -ms-linear-gradient(top, #d7e5f5, #cbe0f5);
    background-image: -o-linear-gradient(top, #d7e5f5, #cbe0f5);
    background-image: linear-gradient(top, #d7e5f5, #cbe0f5);
    border-top: 1px solid #abbbcc;
    border-left: 1px solid #a7b6c7;
    border-bottom: 1px solid #a1afbf;
    border-right: 1px solid #a7b6c7;
    -webkit-border-radius: 12px;
    -moz-border-radius: 12px;
    border-radius: 12px;
    -webkit-box-shadow: inset 0 1px 0 0 white;
    -moz-box-shadow: inset 0 1px 0 0 white;
    box-shadow: inset 0 1px 0 0 white;
```

Figure 4: Adding styles to the webpage

2.3 Modernizr

Modernizr is a JavaScript library that detects HTML5 and CSS3 features in the user's web browser [5]. Taking preference of cool new web technologies is great fun, until one has to support browsers that lag behind [5]. Modernizr makes it easy for the user to write conditional JavaScript and CSS to handle each and every situation, whether a browser supports a feature [5]. It's perfect for doing innovative enhancement easily [5]. Modernizr runs quickly on page load to detect features; it then creates a JavaScript object with the results, and makes addition of classes to the html element for the user to key his/her CSS on [5].

When the developer embeds the Modernizr script in the page, it detects whether the current browser supports CSS3 features like @font-face, border-radius, border-image, box-shadow, rgba() and so on, in addition to HTML5 features like audio, video, localStorage, and the new <input> element types and attributes [4]. With this knowledge, the developer can take advantage of these native implementations in browsers that support these features, and determine whether to create a JavaScript-based fallback or simply gracefully degrade the page in browsers that don't have support for them [4]. Additionally, Modernizr makes the brand-new HTML5 elements available for styling in Internet Explorer, so that developer can start using their improved semantics right away [4].

Modernizr was created based on the principle of progressive enhancement, so it supports, encourages, even builds your website layer by layer, starting with a foundation free from JavaScript and adding layers of enhancement one by one [4]. This is easy with Modernizr, since the developer will know what the browser is capable of [4].

```
#nice {
    background: url(background-one.png) top left repeat-x;
}
.multiplebgs #nice {
    background: url(background-one.png) top left repeat-x,
    url(background-two.png) bottom left repeat-x;
}
```

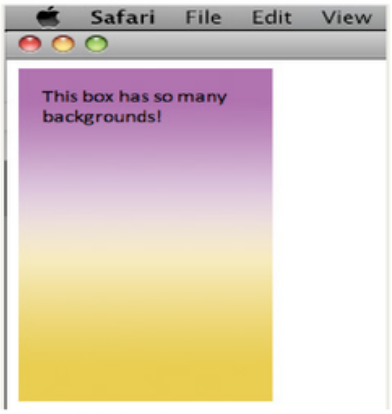

Figure 5: Using Modernizr to create a nice background

2.4 Creating the JavaScript file

An empty file 'js/script.js' is created. Browser support for the Geolocation API is detected with the help of Modernizr object: Modernizr.geolocation. The Geolocation API is used with the help of global navigator object which is navigator.geolocation.

```
function get_location() {
  if (Modernizr.geolocation) {
    navigator.geolocation.getCurrentPosition(show_map);
  } else {
    // no native support; maybe try a fallback?
  }
}
```

Figure 6: Detect browser support for Geolocation API using Modernizr

An error handler is used during locate attempt in case user refuses to reveal location.

```
// handle errors that may occur during locate attempt
function handleLocationErrorCallback(err) {
    if(err.code == 1){
        //user refused to reveal location
    }
}
```

Figure 7: Error Handler

When 'Search for Address' is clicked, script uses Geocoder to search necessary addresses [8].

```
// find address function
function findAddress() {
    var address = document.getElementById("gmap_where").value;

    // script uses our 'geocoder' in order to find location by address name
    geocoder.geocode( { 'address': address}, function(results, status) {
        if (status == google.maps.GeocoderStatus.OK) { // and, if everything is ok

            // we will center map
            var addrLocation = results[0].geometry.location;
            map.setCenter(addrLocation);

            // store current coordinates into hidden variables
            document.getElementById('lat').value = results[0].geometry.location.Xa;
            document.getElementById('lng').value = results[0].geometry.location.Ya;

            // and then - add new custom marker
            var addrMarker = new google.maps.Marker({
                position: addrLocation,
                map: map,
                title: results[0].formatted_address,
                icon: 'marker.png'
            });
        } else {
            alert('Geocode was not successful for the following reason: ' + status);
        }
    });
}
```

Figure 8: Script to get desired address

'findPlaces' function searches local places using API according to parameters, prepares request objects.

```
// find custom places function
function findPlaces() {

    // prepare variables (filter)
    var type = document.getElementById('gmap_type').value;
    var radius = document.getElementById('gmap_radius').value;
    var keyword = document.getElementById('gmap_keyword').value;

    var lat = document.getElementById('lat').value;
    var lng = document.getElementById('lng').value;
    var cur_location = new google.maps.LatLng(lat, lng);

    // prepare request to Places
    var request = {
        location: cur_location,
        radius: radius,
        types: [type]
    };
    if (keyword) {
        request.keyword = [keyword];
    }

    // send request
    service = new google.maps.places.PlacesService(map);
    service.search(request, createMarkers);
}

// create markers (from 'findPlaces' function)
function createMarkers(results, status) {
    if (status == google.maps.places.PlacesServiceStatus.OK) {

        // if we have found something - clear map (overlays)
        clearOverlays();

        // and create new markers by search result
        for (var i = 0; i < results.length; i++) {
            createMarker(results[i]);
        }
```

9

Figure 9: Script to find places within desired radius

'createMarkers' function will draw search result objects as Markers at map [8]. This will help the user to pinpoint the exact location on the map.

```
115 // creare single marker function
116 function createMarker(obj) {
117
118     // prepare new Marker object
119     var mark = new google.maps.Marker({
120         position: obj.geometry.location,
121         map: map,
122         title: obj.name
123     });
124     markers.push(mark);
125
126     // prepare info window
127     var infowindow = new google.maps.InfoWindow({
128         content: '<img src="' + obj.icon + '" /><font style="color:#000;">' + obj.name +
129         '<br />Rating: ' + obj.rating + '<br />Vicinity: ' + obj.vicinity + '</font>'
130     });
131
132     // add event handler to current marker
133     google.maps.event.addListener(mark, 'click', function() {
134         clearInfos();
135         infowindow.open(map,mark);
136     });
137     infos.push(infowindow);
138 }
139
140 // initialization
141 google.maps.event.addDomListener(window, 'load', initialize);
```

Figure 10: Create marker to indicate location

Chapter 3

Running the Web Application

3.1 Requesting User's Location

The index.html file is opened with a browser. Technically speaking, a PC or a mobile device has several ways to find out its own location [2]. When the browser opens, it shows a typical message requesting a user's permission. When it comes to sharing the physical location of users, privacy is a serious concern. According to the Geolocation API, "users must not provide location information to Web sites without the express permission of the user." In other words, a user must always opt in to share location information with a Web site [2].

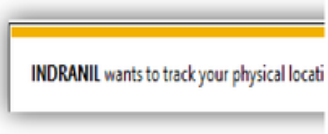

Figure 11: Message from browser requesting user's permission

3.2 The User Interface

Once the user grants permission to the web app to access his physical location, the map loads and shows his location. He will get an option to select an address near which he wants to find places and enter a keyword (optional). Then there will be a dropdown list showing him the type of place he wants to search, like hospitals, café, banks, ATMs, restaurants etc. once he selects his choice he needs to choose from a range of radius within which he wants to find his desired place. Radius ranges are 0.5 km, 1 km, 2 km, 3 km, 4 km and 5 km. after he selects the radius, the user has to click "search for objects" and he finds the locations he want within his desired radius.

3.3 Getting Most Out of It

Supposedly a user wants to find all banks within a radius of 2 km. he gets his address, selects "bank" in the option "Type" and chooses radius as 2000. He gets the names and locations of all the banks located within a radius of 2 km.

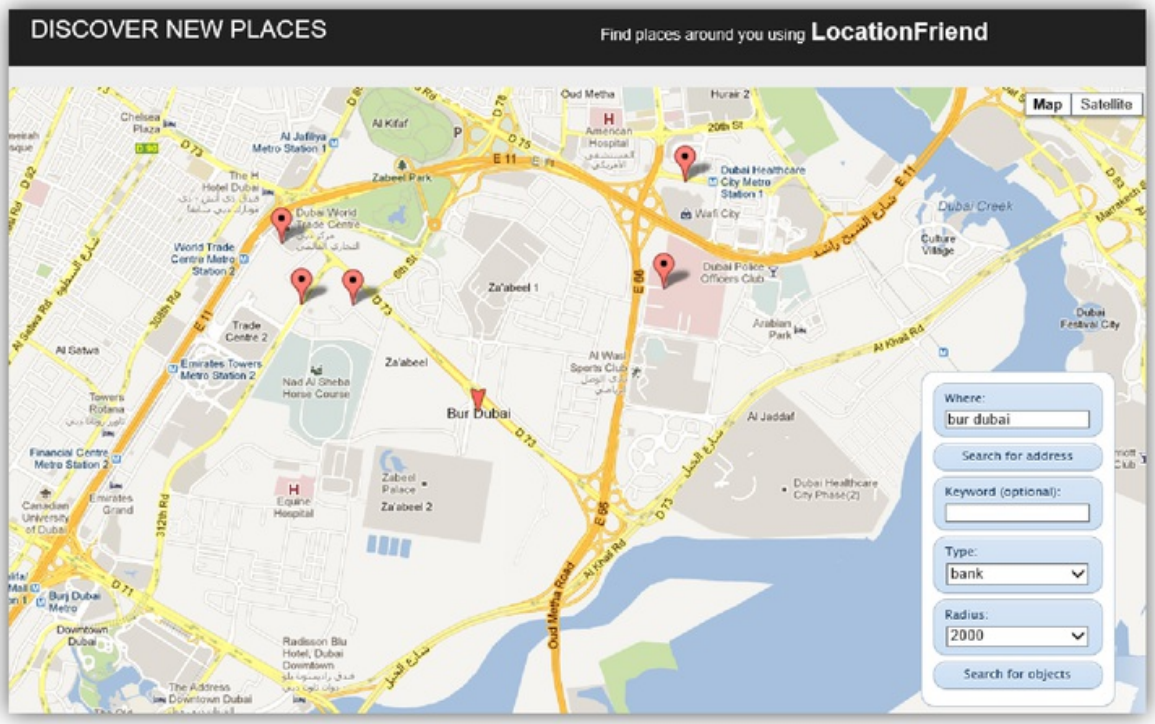

Figure 12: Location of all banks within a radius of 2 km

3.4 Uses of the Website

This website can be used as a travel assistant to help the user find his location, get info about his surroundings and find the location which is his destination. If he wants to look for a restaurant to have lunch in a place he has never been before, he can simply use LocationFriend to find the nearest food. Same thing applies for other import places like banks, hospitals and ATMs. The user can find his destination effectively and smoothly.

Chapter 4
Street View

4.1 Street View Overview

Street view is a feature developed by Google which provides panoramic 360 degree views from desired roads throughout its coverage area. Street view's API covers [14] same as that of Google Maps application [14]. However not all cities have Street View support available.

The JavaScript API for Google Maps provides a Street View service for obtaining and manipulating the imagery utilized in Google Maps [1]. The Street View service used in the new version of the Maps JavaScript API is supported natively within the browser unlike the previous version. [1]

Shown below is a sample image of Street View

Figure 1313: Street View image of Bancroft Way, Berkeley

4.2 Street View Map Usage

The Street View is most useful for indicating a location on a map, although it can be used within a standalone DOM element [1]. By default, Street View gets enabled in the map and a yellow colored Street View Pegman control appears within the navigation (zoom and pan) controls. This control can be hidden within the map's MapOptions by setting streetViewControl to false. [1] The default position of the Street View control may also be changed by setting the Map's streetViewControlOptions.position property to a new property named ControlPosition. [1]

The Street View Pegman allows the user to view the Street View panoramas directly within the map [1]. When the user clicks and drags the Pegman, the map updates automatically to show the street view enabled streets with the help of blue outlines on the map. [1]

Dragging and dropping the yellow Pegman marker on to a Street View enabled street will update the map to display the Street View panorama of location indicated by the user. [1]

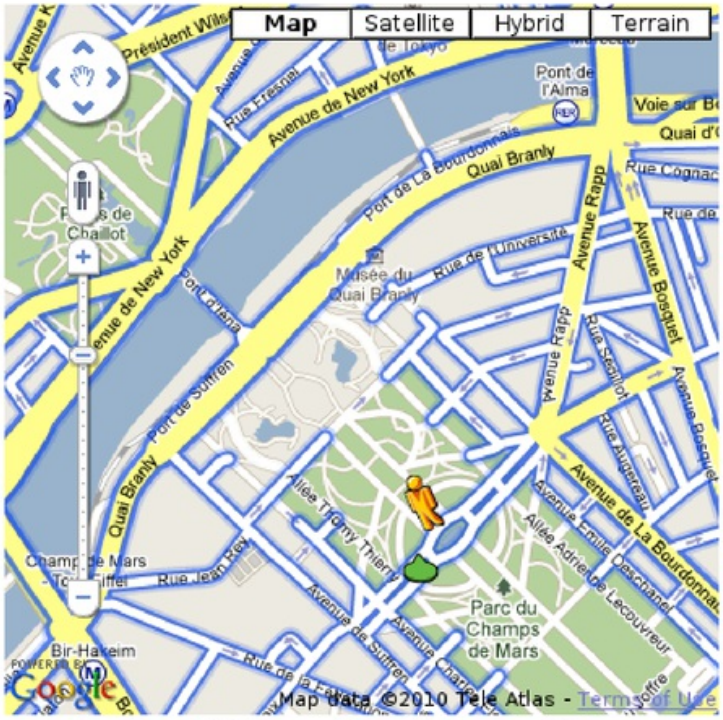

Figure 144: Drag and drop of Pegman on Map

4.3 Street View Panoramas

Street View images are supported by the StreetViewPanorama object, which supports an API interface to a Street View "viewer." Each map contains a default Street View panorama, which the user can retrieve by calling the map's getStreetView() method [1]. By adding a Street View control to the map by setting its streetViewControl option to true, we automatically connect the Pegman control to this default Street View panorama. [1]

A personal StreetViewPanorama object can also be created and the map can be set to use that instead of the default object, by setting the streetView property of the map explicitly to that constructed object. The default panorama can also be overridden if we want to modify default behavior, for example the automatic sharing of overlays between the map and the panorama. [1]

4.4 Street View Containers

A StreetViewPanorama can be displayed within a separate DOM element, often a element. Simply the DOM element needs to be passed within the StreetViewPanorama's constructor. A minimum size of 200 pixels by 200 pixels [1] is recommended for optimum display of images.

4.5 Street View Locations and Point-of-View (POV)

The StreetViewPanorama constructor also allows you to set the Street View location and point of view using the StreetViewOptions parameter [1]. setPosition() and setPov() may be called on the object after construction to change its location[1] and Point of View.

The Street View location defines the placement of the camera focus for an image, but definition the orientation of the camera for that particular image is not obtained. For this purpose, the StreetViewPov object [1] gives definition to two properties:

- heading (default 0) defines the rotation angle around the camera locus in degrees relative from true north. Headings are measured clockwise (90 degrees is true east). [1]

- pitch (default 0) defines the angle variance "up" or "down" from the camera's initial default pitch, which is often (but not always) flat horizontal. (For example, an image which is captured on a hill will likely exhibit a default pitch that is not horizontal). Pitch measurements are with positive values looking up (to +90 degrees straight up and orthogonal to the default pitch) and negative values looking down (to -90 degrees straight down and orthogonal to the default pitch) [1].

The StreetViewPov object is most often used to determine the point of view of the Streetview camera. The point-of-view of the photographer can also be determined — typically the direction the car or trike was facing — with the StreetViewPanorama.getPhotographerPov() method.

Chapter 5
Direction Service

5.1 Overview

Directions can be calculated (using a variety of methods of transportation) by using the DirectionsService object [1]. This object initiates contact request with the Google Maps API Directions Service which receives direction requests and returns computed results [1]. These directions results can be either handled by the user himself/herself or can be rendered using DirectionsRenderer objects.

Directions may specify origins and destinations either as text strings (e.g. "Chicago, IL" or "St Louis, MO") or as LatLng values. The Directions service can return multiple parts of directions using a series of waypoints [1]. Directions are shown as a polyline drawing the route on the map, or additionally marking as a series of textual description within a element (e.g. "Turn right onto the Williamsburg Bridge ramp") [1].

5.2 Direction Requests

Accessing the Directions service is asynchronous, because the Google Maps API requests a call to an external server [1]. As s result, a callback method needs to be passed to execute the request is completed. This callback method processes the result(s). It is to be noted that the Directions service may return more than one possible itinerary as an array of separate routes[][1].

To use directions in the new version (V3), an object of type DirectionsService is created and DirectionsService.route() is called to initiate a request to the Directions service,and forwarding it a DirectionsRequest object containing the input terms and a callback method to execute upon receipt of the response[1] .

The DirectionsRequest object consists of the following fields:

- origin (required) specifies the starting location from which directions are calculated. This value may either be specified as a String (e.g. "Los Angeles, CA") or as a LatLng value [1].

- destination (required) specifies the end location to which directions are calculated. This value may either be specified as a String (e.g. "Kingsman, AZ") or as a LatLng value [1] .

- travelMode (required) specifies what mode of transport to use when directions are[1] calculated.

- transitOptions (optional) specifies values that apply only to requests where travelMode value is google.maps.TravelMode.TRANSIT.

- unitSystem (optional) specifies what unit system to use when displaying results [1].

5.3 Rendering Directions

Starting a directions request to the DirectionsService with the route() method involves passing a callback which executes upon completion of the service request. This callback returns a DirectionsResult and a DirectionsStatus code in the response [1].

Status of Directions Query

The DirectionsStatus will return the following values [1]:

- OK indicates the response contains a valid DirectionsResult[1].
- NOT_FOUND indicates that at least one of the locations specified in the requests' origin, destination, or waypoints could not be geocoded[1] and found.
- ZERO_RESULTS indicates it is unable to find any routhe between origin and destination.
- MAX_WAYPOINTS_EXCEEDED indicates that DirectionsWaypoints[1] exceeded DirectionsRequest. The maximum number of allowed waypoints is 8, in addition to the origin, and destination[1]. For Business customers Maps API allows 23 waypoints, in addition to the origin, and destination. Transit directions don't have waypoint support.
- INVALID_REQUEST indicates that invalid DirectionsRequest is provided. The most common causes of this error code are requests which are missing either an origin or destination, or a transit request which possess waypoints.
- OVER_QUERY_LIMIT indicates the webpage has crossed its request limit within the allowed time period [1].
- REQUEST_DENIED indicates the webpage is denied to use the directions service[1].
- UNKNOWN_ERROR indicates a directions request could not be processed due to a server error. The request [1] should be tried again.

Displaying the DirectionsResult

The DirectionsResult consists of the result of the directions query, which may be either handled by the user, or passed to a DirectionsRenderer object, which can automatically handle displaying the result on a map [1].

To display a DirectionsResult using a DirectionsRenderer, we simply need to do the following [1]:

- Create a DirectionsRenderer object [1].
- Call setMap() on the renderer object to bind it to the passed map [1].
- Call setDirections() on the renderer, and pass it the DirectionsResult as noted above. Since the renderer is an MVCObject, it will detect if any changes are made to its properties and update the map when its associated directions have changed [1].

```
function calcRoute() {
  var start = document.getElementById("start").value;
  var end = document.getElementById("end").value;
  var request = {
    origin:start,
    destination:end,
    travelMode: google.maps.TravelMode.DRIVING
  };
  directionsService.route(request, function(result, status) {
    if (status == google.maps.DirectionsStatus.OK) {
      directionsDisplay.setDirections(result);
    }
  });
```

Figure 155: Code snippet for calculating route

A DirectionsRenderer not only handles display of the polyline and any markers associated with it, but also handles the textual display of directions as a series of steps [1]. To do so, setPanel() needs to be simply called on the DirectionsRenderer, passing it the in which to display this information. Doing so also ensures that the appropriate copyright information is displayed, and any warnings that can come with the result [1].

Textual directions will be provided with the help of the browser's language setting, or the language indicated when loading the API JavaScript using the language parameter [1]. In order to find transit directions, the time will be shown in the time zone at that transit stop.

The DirectionsResult Object

While sending a directions request to the DirectionsService, a response consisting of a status code, including a result, that is a DirectionsResult object is received [1]. The DirectionsResult is an object literal with a single field [1]:

The array variable routes[] contains an array of DirectionsRoute objects. Each route indicates a path to get from the origin to the destination provided in the DirectionsRequest [1]. Usually, only single route is returned for any given request, except the request's provideRouteAlternatives field is initialized to true, in which, multiple routes may be returned [1].

Figure 166: Directions from St Louis, text directions in right

Chapter 6

Conclusions and Future Work

6.1 Conclusions

Privacy policy: When it comes to sharing the physical location of users, privacy is a serious and important concern [2]. According to the Geolocation API, "user agents must not send location information to Web sites without [2] taking permission from the user." In other words, a user must always opt in to share location information with a Web site [2]. LocationFriend ensures that privacy policy is not violated. This website is very useful for finding

6.2 Future Work

- Improving User Interface – At present LocationFriend has a very basic user interface. The future plan is to make the user interface more interactive and fluid so that browsing experience becomes smoother, better and faster.

- LocationFriend now shows the names and location of places within a chosen radius and direction assistance in selected places. In future, there are plans in implement social networking so that users will know places friends are visiting and also find friends nearby.

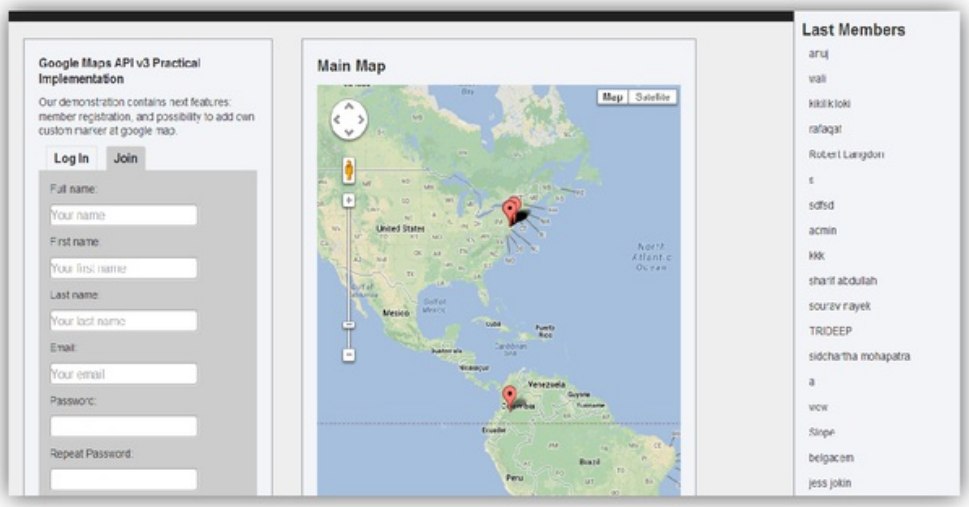

Figure 177: Estimated look on social network integration

- At present there are nine categories of place which the user can search. An increase in number of parameters will allow the user to have more choices.

Chapter 7

References

[1] http://code.google.com

[2] http://sisindotek.com

[3] Eric Freeman, Elisabeth Robson, "Head First HTML5 Programming", October 2011 Edition, PP 51 – 239

[4] http://wwe.alistapart.com

[5] http://alpha.responsivedesign.is

[6] http://diveintohtml5.ep.io

[7] Michael Morrison, "Head First JavaScript", December 2007 Edition, PP 72 – 297

[8] http://script-tutorials.com

[9] http://websitereported.com

[10] http://bogotobogo.com

[11] http://webhelp101.com

[12] http://webauthority.eu

[13] http://a2zinterviews.com

[14] http://diveintohtml5.info/geolocation.html

[15] http://webdesignernotebook.com/css/how-to-use-modernizr/

[16] http://jobskeys.com

[17] http://wickedlysmart.com

[18] http://www.innovexpo.itee.uq.edu.eu

[19] http://images.energieportal24.de

[20] http://www.pedrocorreia.com

[21] http://pc01.lib.ntust.edu.tw

Computer Project Final Report

ORIGINALITY REPORT

50 %	47 %	3 %	12 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	code.google.com *Internet Source*	23%
2	sisindotek.com *Internet Source*	9%
3	Submitted to BITS, Pilani-Dubai *Student Paper*	4%
4	studiosoho.net *Internet Source*	3%
5	alpha.responsivedesign.is *Internet Source*	2%
6	diveintohtml5.ep.io *Internet Source*	2%
7	www.geolocation.com *Internet Source*	2%
8	www.programmieren-in-java.de *Internet Source*	1%
9	www.script-tutorials.com *Internet Source*	1%
10	www.w3.org *Internet Source*	1%
11	webhelp101.com *Internet Source*	1%
12	www.compassdesigns.net *Internet Source*	< 1%
13	Submitted to Royal Melbourne Institute of Technology *Student Paper*	< 1%
14	Submitted to Nanyang Technological University, Singapore *Student Paper*	< 1%
15	Submitted to University of Technology, Sydney	< 1%

	Student Paper	
16	www.innovexpo.itee.uq.edu.au *Internet Source*	< 1%
17	images.energieportal24.de *Internet Source*	< 1%
18	db.s2.chalmers.se *Internet Source*	< 1%
19	Submitted to University of Maryland, University College *Student Paper*	< 1%

EXCLUDE QUOTES OFF
EXCLUDE BIBLIOGRAPHY OFF
EXCLUDE MATCHES OFF