Integration of Data from Electric Vehicle Charging Station networks

Team 10

Nathaniel Lum

Nitesh Sood

&

Team

Jeyenth Veerarahavan

Allen Webb

Overview

In this project, data for the locations of EV (electric vehicle) charging stations was collected from the websites of the different providers. This data was then imported into a MySQL database and two different methods were developed for viewing the information, a website and an Android application.

Team Responsibilities:

Jeyenth Veerarahavan and Allen Webb: Researching networks(Charge Point, GE, eVgo, Shorepower); Initial database design, Android application design and development.

Nathaniel Lum and Nitesh Sood: Researching networks(US Department of Energy, Blink, SemaCharge); Extending and enhancing the database design; Website design and development.

Network Research

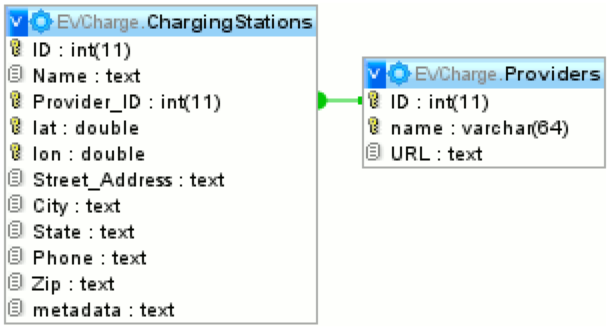
The first step of the project was to find various networks from which to take our data. The networks we found were Charge Point, GE, eVgo, Shorepower, Blink, SemaConnect, and the network data provided by the US Department of Energy website. In the end, we did not use the Blink or SemaConnect maps. The Blink network was not used because its data was only available through a limited access API. The data from SemaConnect was not used because it overlapped with data from other networks. The Department of Energy data was by far the largest, with over 6000 different stations belonging to different networks spread across the United States.

Data Collection, Import, and Storage

The format of the location data gathered from each provider fell into one of two categories. The easiest of the two categories to work with was latitude and longitude coordinates. Both for the Google Maps API and for Android there are already existing tools for working with coordinates. The harder category to deal with were lists of addresses. These needed to be converted into coordinates to be used for our purpose. Google has a Geocoder API that can convert location strings such as addresses into latitude and longitude coordinates or the reverse. It is free to use in conjunction with their Maps API or for app development.

PhpMyAdmin is a browser-based MySQL client which provided us a simple interface to use to create a user, database and setup the privileges. It also was useful for troubleshooting SQL queries and viewing the data in the database.

Here is a diagram of the database model used taken from PhpMyAdmin:



One table, *Providers*, contains a list of the names of the EV charging station networks. The other table, *ChargingStations*, has a foreign key linking to the primary key of the Providers table. An index was added to speed up searches based on latitude and longitude pairs. The metadata field ended up not being used but was intended to contain a JSON serialized string of other information given by the provider would could be displayed. This would allow for great flexibility with collecting the data without having to force it all to conform to the same structure. An SQL dump containing the structure of the database is included in the appendix.

Some of the providers had direct links to access their full list of charging stations. This was not the case for all the providers. One method used to get a full list from a provider which didn't provide a directly link but had a map marking all the points was to look for the javascript function which retrieved the list from their server. By adding a "console.log()" call to the client-side javascript using the developer tools of a web browser (Chrome) the JSON objects received from their server were recovered.

Nodejs, an event driven server-side javascript environment, was used to parse the websites which had a list of addresses. Nodejs has a package manager which allows for easy installation of third party libraries. After installing Nodejs run the following command to download the additional libraries needed:

```
npm install geocoder jsdom jquery
```

Nodejs was used because it can pull the HTML of a page given the URL string and then setup an environment where jQuery, a javascript library, can be used to reduce the development time to find and retrieve data embedded on the page. Once the data was isolated, the geocoder library which simplifies the process of interfacing with Google's Geocoder API was used to convert the addresses into latitude and longitude coordinates. This information was then saved to a file in a JSON format easily read by the PHP code written to read JSON objects and import them into the database. One file used to convert the list for GE is included in the appendix "import_ge_data.js". The main PHP import file is "importer.php" (not included because of length) which depends on some variables being predefined. An example of the definitions for these variables and the usage of importer.php is "import_charge_point.php" (not included because of length. Probably the biggest disadvantage of Nodejs is that event driven code can be very difficult to read and troubleshoot because the execution order isn't always clear at first glance.

One obstacle of using the geocoder library is that there is a 2500 request limit per day per IP address and it is necessary to spread the requests out over time so that the requests are not denied. A timer was used to limit the rate requests were made to 2 per second.

Website Development

A locator website, **locator.html,** has been developed using which users can find charging stations near any given location, within a specified radius. The page is powered by the JavaScript Google Maps API v3 and styled using HTML/CSS. The backend used is a MySQL database (with tables ChargingStations and Providers) as described in the preceding sections. Client requests are served using the AJAX framework, handled by PHP scripts on the server-side.

Setting up the site infrastructure:

The site is designed to make two kinds of requests to the server, viz., a request to load all stations and a request to load stations within a specified radius of a given location in the United States.

We had to first setup the database and push in all the relevant data collected from various sources. We had the data dump from the Department of Energy website as an XML file. This had information like latitude, longitude, name, provider, street address, city, phone for more than 6000 stations. We wrote Python scripts to parse the xml and generate SQL insert statements to push in all the data. Initially, we only used the latitudes and longitudes from the xml. After extending the database schema, we also extracted meta-information like name, provider, street address, city, phone etc. for each station and generated SQL update statements to be used during database installation. At submission time, we provided an exported database dump with the structure and data generated in PHPMyAdmin. We only used the Python script-generated sql statements to arrive at the final database state that our site needs.

PHP scripts have been written to handle client requests on the server side. There are two scripts handling both the major functions of the site. ajax_load_all.php to load all stations, and ajax_search_near.php to find nearby locations. These scripts use phpsqlsearch_dbinfo.php for the database information. Both the above scripts follow the AJAX framework, i.e. accept requests over http, query the database, and send back results in xml form. It is up to the client to parse and render the xml output appropriately. The SQL used to find nearby stations w.r.t a given location is based on **The Haversine formula.**

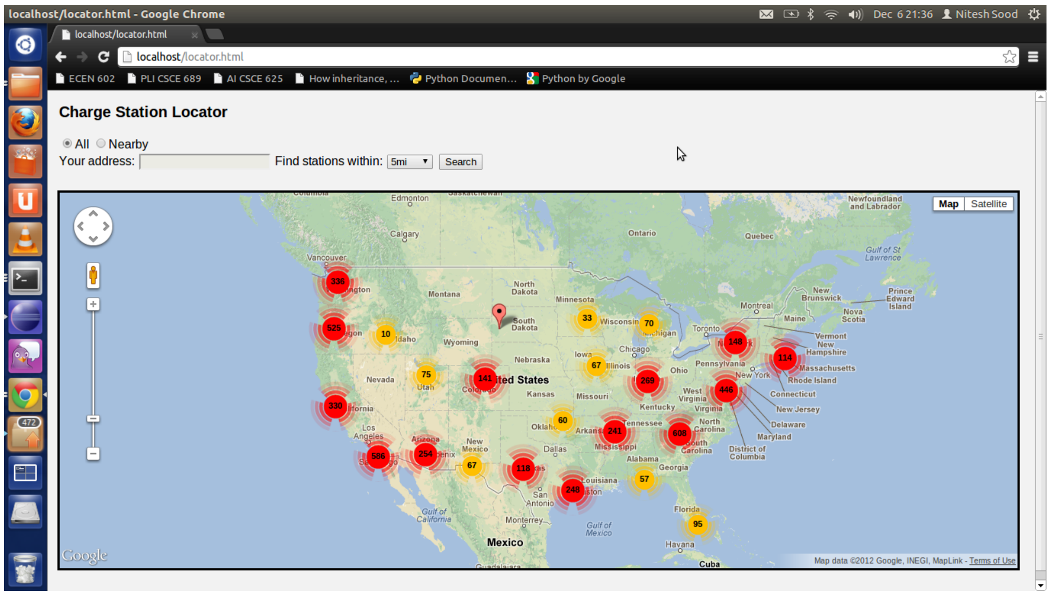
```
SELECT ChargingStations.ID as id,
    ChargingStations.lat as lat,
    ChargingStations.lon as lon,
    ChargingStations.Name as name,
    Providers.name as provider,
    ChargingStations.Street_Address as street,
    ChargingStations.City as city,
    ChargingStations.State as state,
    ChargingStations.Zip as zipcode,
    ChargingStations.Phone as phone,
    (3959 * acos(
                    cos(radians(xxx)) -- user's latitude
                   * cos(radians(ChargingStations.lat))
                   * cos(radians(ChargingStations.lon)-radians(yyy)) -- user's longitude
                   + sin(radians(xxx)) -- user's latitude
                   * sin(radians(ChargingStations.lat))
                   )
          )
          as distance
    FROM ChargingStations, Providers
   WHERE ChargingStations.Provider_ID = Providers.ID
  HAVING distance < zzz -- radius within which to find locations
ORDER BY distance;
```

<u>Site features:</u>

- Displaying all stations within the US, using MarkerClusterer API for Google Maps (http://google-maps-utility-library-v3.googlecode.com/svn/tags/markerclusterer/1.0/docs/reference.html)
- Displaying stations near a given user location, within a specified mile radius
- Displaying the path from the user's location to the station selected
- Dynamic Zoom-in/Zoom-out
- Address translation using Google's Geocoder API

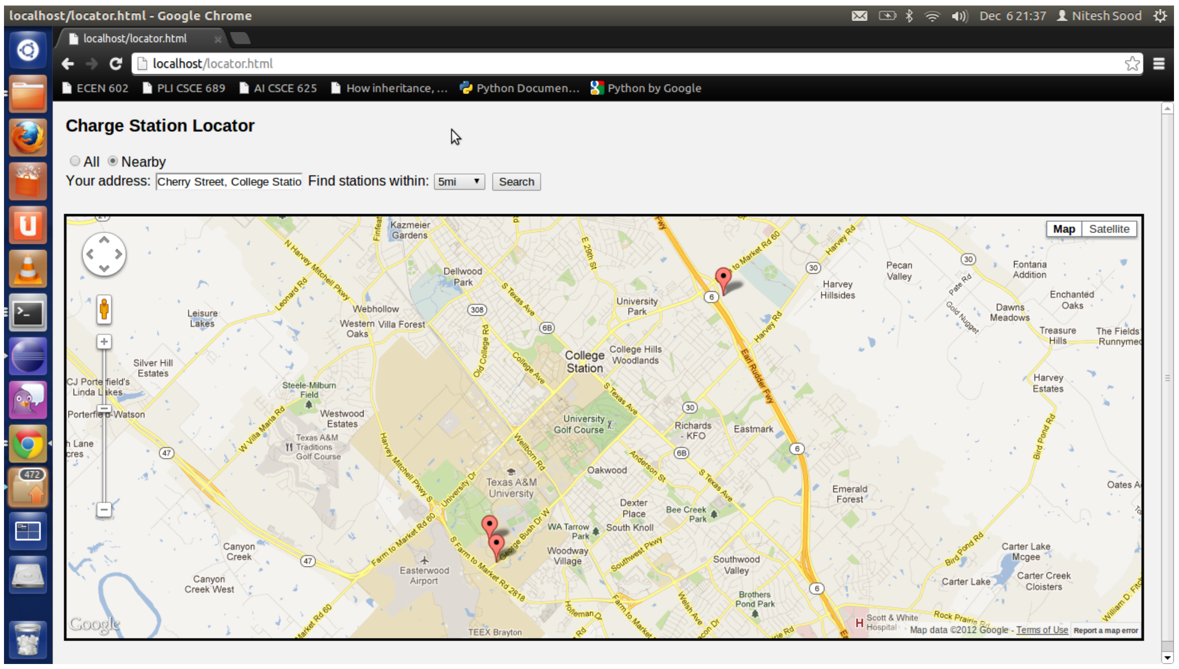
Upon site load, all stations within the US that we have data for are displayed.

Clusters are created using the MarkerClusterer library.



The “All” button at the top of page can be selected by the user at any time to zoom out and display the entire U.S. map with all points plotted. While this button is selected, the user cannot to input an address to zoom to or get directions from. Any node can be clicked on to view its information. When the “Nearby” option is selected, then the user has the option to enter an address and select a radius distance. When these inputs are given and a search is initiated, then the map will zoom to the given location and display only the nodes within the given radius of the given address. Clicking on any of these nodes will display a route from the user’s address to the marker that was clicked on. In addition, the node’s information such as name, longitude, latitude, provider, street address, city, state, zipcode, if available, will be displayed.

Shown here is a shot of the user entering their location (College Station) in the search box. The map is then zoomed in with the user location as the center and the nearby stations within a 5 mile radius as selected are shown on the map.

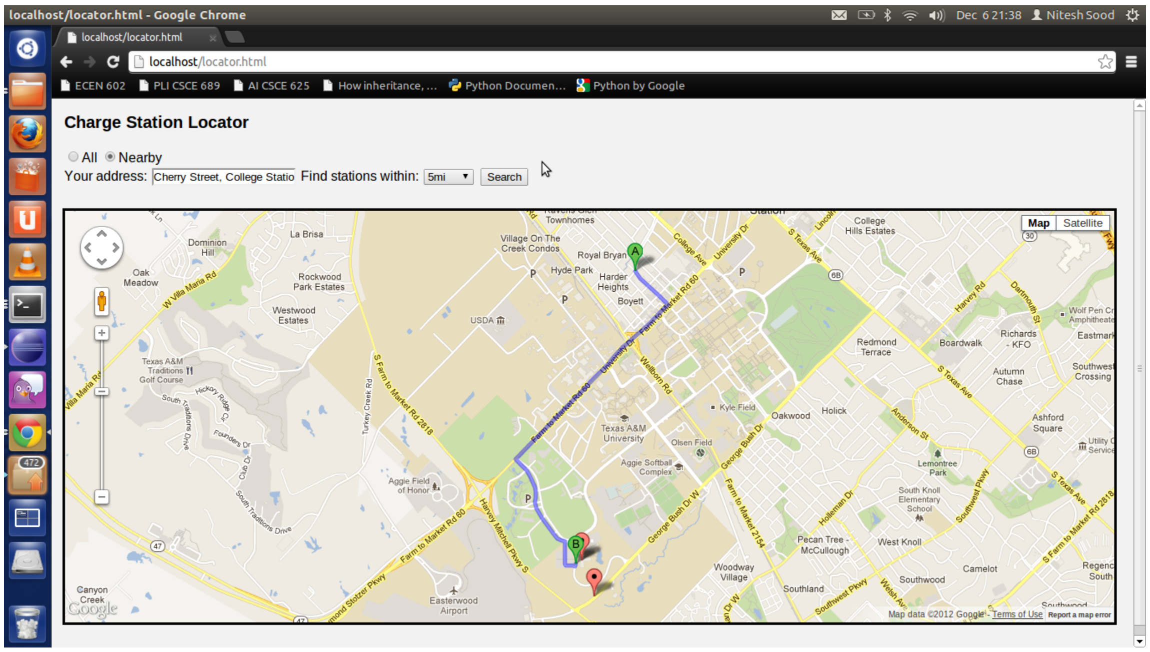


Charge Station Locator

All [✓ Nearby]

Your address: Cherry Street, College Statio Find stations within: 5mi Search

Shown below are shots of the user selecting one of the stations, and the map showing up the path to the location along with the station information, including its distance from the user location.

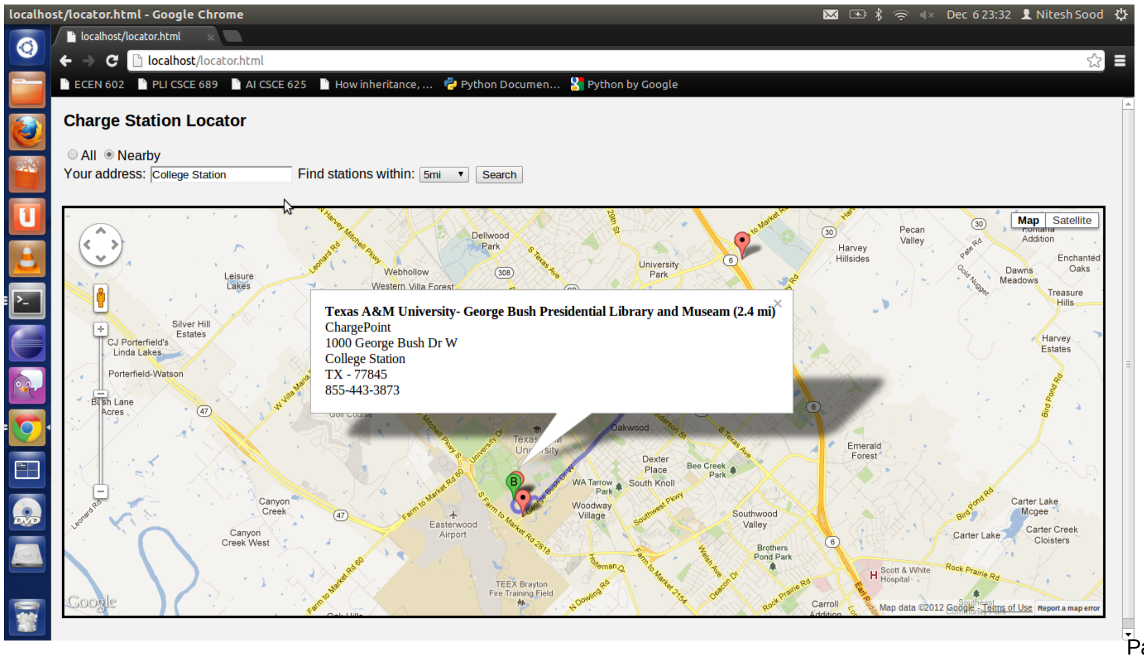

localhost/locator.html - Google Chrome
Dec 6 21:38 Nitesh Sood
localhost/locator.html
ECEN 602 PLI CSCE 689 AI CSCE 625 How inheritance, ... Python Documen... Python by Google
Search
Map Satellite
Texas A&M University
College Station
Map data ©2012 Google - Terms of Use Report a map error

Charge Station Locator

All [✓ Nearby]

Your address: Cherry Street, College Statio Find stations within: 5mi

Shown below is the shot of the information for a particular station being displayed, i.e. its name, provider, address and the distance to the station from the current user location.


localhost/locator.html - Google Chrome
Dec 6 23:32 Nitesh Sood
localhost/locator.html
ECEN 602 PLI CSCE 689 AI CSCE 625 How inheritance, ... Python Documen... Python by Google
Search
Map Satellite
Map data ©2012 Google - Terms of Use Report a map error

Charge Station Locator

All [✓ Nearby]

Your address: College Station Find stations within: 5mi

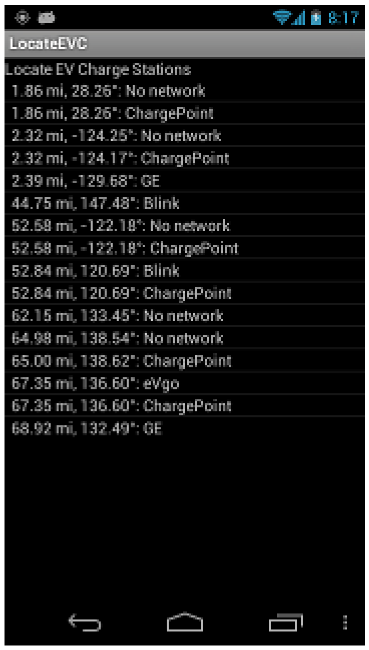
Texas A&M University- George Bush Presidential Library and Museam (2.4 mi)
ChargePoint
1000 George Bush Dr W
College Station
TX - 77845
855-443-3873

The website intelligently and dynamically zooms in the exact amount that is required to display all the returned stations within a single frame. Markers can be clicked one after the other in quick succession to display the paths leading to them, no page refreshing is needed for this.

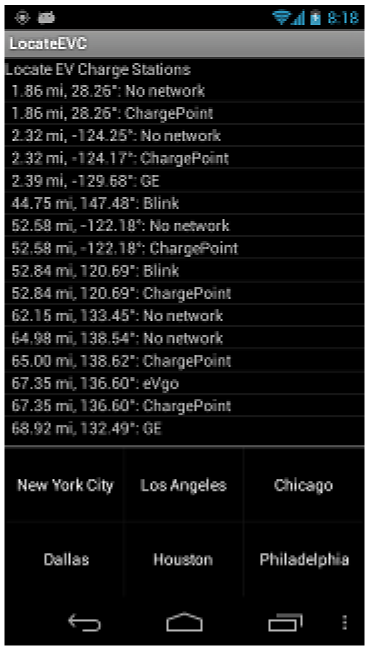
Scope for further work:

Further improvements to this website could be made, including but not limited to, displaying directions in a side panel, advanced search with filtering options available etc.

Android App development

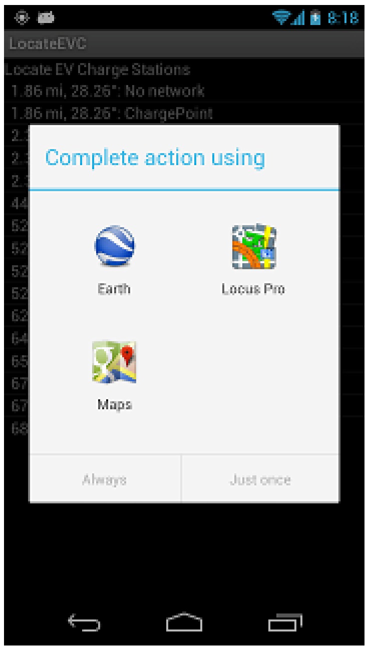


Locate EV Charge Stations
1.86 mi, 28.26°: No network
1.86 mi, 28.26°: ChargePoint
2.32 mi, -124.25°: No network
2.32 mi, -124.17°: ChargePoint
2.39 mi, -129.68°: GE
44.75 mi, 147.48°: Blink
52.58 mi, -122.18°: No network
52.58 mi, -122.18°: ChargePoint
52.84 mi, 120.69°: Blink
52.84 mi, 120.69°: ChargePoint
62.15 mi, 133.45°: No network
64.98 mi, 138.54°: No network
65.00 mi, 138.62°: ChargePoint
67.35 mi, 136.60°: eVgo
67.35 mi, 136.60°: ChargePoint
68.92 mi, 132.49°: GE



Locate EV Charge Stations
1.86 mi, 28.26°: No network
1.86 mi, 28.26°: ChargePoint
2.32 mi, -124.25°: No network
2.32 mi, -124.17°: ChargePoint
2.39 mi, -129.68°: GE
44.75 mi, 147.48°: Blink
52.58 mi, -122.18°: No network
52.58 mi, -122.18°: ChargePoint
52.84 mi, 120.69°: Blink
52.84 mi, 120.69°: ChargePoint
62.15 mi, 133.45°: No network
64.98 mi, 138.54°: No network
65.00 mi, 138.62°: ChargePoint
67.35 mi, 136.60°: eVgo
67.35 mi, 136.60°: ChargePoint
68.92 mi, 132.49°: GE

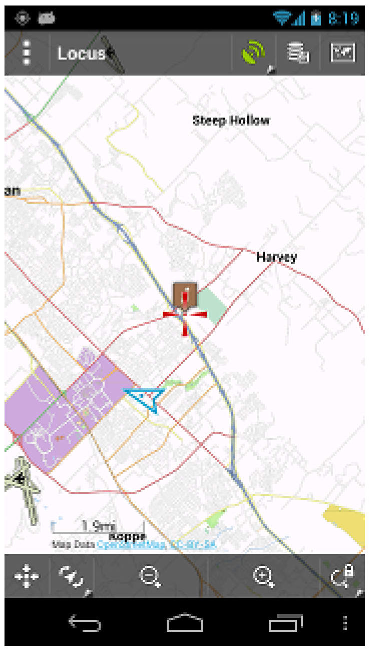
New York City	Los Angeles	Chicago
Dallas	Houston	Philadelphia



Locate EV Charge Stations
1.86 mi, 28.26°: No network
1.86 mi, 28.26°: ChargePoint

Complete action using

Earth
Locus Pro
Maps



The first screenshot (left) shows the list of charge stations near College station. Each entry lists the distance from the current location in miles, the bearing to the charge station, and the network of the charge station. The second screenshot (middle) shows the menu of 6 large cities which have preset locations for testing the app on devices without a location service such as a GPS. The third screenshot (right) shows the Android 4.2 launcher screen which pops up when you click a location on the list. This screen only shows up if the user has more than one app which can open a URI to latitude longitude coordinates and no default is set. If only one app is available or a default is set that app will be opened to the coordinates of the charge station. This allows the user to use whichever map or navigation software they are already familiar with, and decreases the development necessary to maintain the app. The last screenshot is not from our app but shows one coordinate which was opened with Locus Pro. The source files written for this app, “DatabaseAbstraction.java” and “LocateEVC.java”, are included in the appendix.

The Android app API includes support for SQLite databases. A script called “mysql2sqlite.sh” was used to create a SQLite friendly dump of the database. This script can be found at http://code.google.com/p/mindcore/source/browse/trunk/data/devel/mysql2sqlite.sh?r=2 as of December 1st, 2012. This was then imported into an SQLite database which was included in the assets folder of the Android app. When the app is first opened it checks for the database in the data folder where Android expects it. If it isn’t there it copies it from the assets folder so it can be accessed through the Android API.

Because SQLite doesn’t include trigonometric functions to implement the haversine a simple algorithm was developed which sets limits on latitude and longitude based on a predefined radius. Some stations outside the radius may be included in the search results in addition to those in the radius, but this is acceptable as the app could be modified to truncate these results before displaying them if so desired.

<u>**Appendix**</u>

Please note that many of the sources could not be included because of the length constraints for the report document.

<u>**Source Code for Data Import:**</u> (Both teams)

EVCharge.sql.gz: database schema installation and seeding script.

dbStructure.php: Views the structure of the database.

include.db.php: File that sets up the connection to the mysql server.

importer.php: This php source contains the code to import the data into the mysql database.

parse.py: parses the xml dump got from DoE site and generates insert sqls.

update_schema.py: updates the stations already in the database with additional information from the DoE xml dump.

Note: the 2 scripts above, parse.py and update_schema.py were only used during development. They don’t play a part in the final installation script of the database as the inserts were generated accordingly by PhpMyAdmin based on the final required database state.

The below mentioned import files set the attributes in the mysql database for the appropriate vendor:

import_shorepower_data.php, import_charge_point.php, import_eVgo_data.php ,import_GE_data.php

The below javascript files are meant to be executed using nodejs. They either download the html content from the webpage or allowing us to segregate the data easily and then use the google's geocoder API to convert the addresses into latitudes and longitudes.

import_shorepower_data.js, import_ge_data.js

The json files that supply the data to the mysql database:

GE_data.json, Shorepower_data.json

A duplicate removal PHP script, HandleCollisions.php, was written to remove the collisions between data from Charge Point and the Department of Energy data sets. It deletes the entries from Charge Point because they didn't have addresses listed and then updates the Provider_ID for the ones listed as "No Network" to be listed as Charge Point.

EVCharge_structure.sql

```
SET FOREIGN_KEY_CHECKS=0;
SET SQL_MODE="NO_AUTO_VALUE_ON_ZERO";
SET time_zone = "+00:00";

DROP TABLE IF EXISTS `ChargingStations`;
CREATE TABLE IF NOT EXISTS `ChargingStations` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `Name` text,
  `Provider_ID` int(11) NOT NULL,
  `lat` double NOT NULL,
  `lon` double NOT NULL,
  `Street_Address` text,
  `City` text,
  `State` text,
  `Phone` text,
  `Zip` text,
  `metadata` text NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `ChargingStations_u1` (`Provider_ID`,`lat`,`lon`),
  KEY `ChargingStations_i1` (`lat`,`lon`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `Providers`;
CREATE TABLE IF NOT EXISTS `Providers` (
  `ID` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(64) CHARACTER NOT NULL,
  `URL` text CHARACTER NOT NULL,
  PRIMARY KEY (`ID`),
  UNIQUE KEY `Providers_u1` (`name`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8 ;

ALTER TABLE `ChargingStations`
  ADD CONSTRAINT `ChargingStations_ibfk_1` FOREIGN KEY (`Provider_ID`) REFERENCES
`Providers` (`ID`) ON DELETE CASCADE ON UPDATE CASCADE;
SET FOREIGN_KEY_CHECKS=1;
```

<u>Source Code for Website:</u> (Nitesh and Nathaniel)

ajax_load_all.php: loads all stations from the database
ajax_search_near.php: searches for stations near a given location
locator.html: web page
markerclusterer.js: marker clustering library for Google Maps API v3
phpsqlsearch_dbinfo.php: database info for the website

locator.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
    <style type="text/css">
      html
      {
        height: 100%
      }
      body
      {
        height: 100%;
        margin: 1%;
        padding: 0;
        background-color: #F2F2F2;
      }
      #header
      {
        font-family: Arial, Helvetica, sans-serif;
        font-size: 20px;
        font-weight: bold;
      }
      #inputs
      {
        padding: 2px;
        font-family: Arial, Helvetica, sans-serif;
      }
      #empty-space1
      {
        width: 100%;
        height: 2px;
      }
      #empty-space2
      {
        width: 100%;
        height: 10px;
      }
      #map
      {
        width: 99%;
        height: 75%;
        border: 3px solid black;
      }
    </style>
    <script src="https://maps.google.com/maps/api/js?sensor=false"></script>
    <script type="text/javascript" src="markerclusterer.js"></script>

    <script type="text/javascript">
      var map;
      var infoWindow;
      var userLoc;
      var marker;
      var markers = [];
      var markerCluster;
      var alreadySearched;

      var directionsDisplay;
      var directionsService = new google.maps.DirectionsService();

      function loadMap()
      {
        document.getElementById("addressInput").setAttribute("disabled", "disabled");
             directionsDisplay = new google.maps.DirectionsRenderer();
        map = new google.maps.Map(document.getElementById('map'), {
          zoom: 4,
          center: new google.maps.LatLng(39.5, -98.35),
          mapTypeId: google.maps.MapTypeId.ROADMAP
        });
             directionsDisplay.setMap(map);

       var searchUrl = 'ajax_load_all.php';
        callServer(searchUrl, function(data) {
          var xml = parseXml(data);
          var markerNodes = xml.documentElement.getElementsByTagName("marker");
          for(var i = 0; i<markerNodes.length; i++)
          {
            var name = markerNodes[i].getAttribute("name");
            var lat = markerNodes[i].getAttribute("lat");
            var lon = markerNodes[i].getAttribute("lon");
            var address = createMarkerAddress(markerNodes[i]);
            var latlng = new google.maps.LatLng(
              parseFloat(lat),
              parseFloat(lon));

            marker = createMarker(latlng, name, address, -1);
           markers.push(marker);
          }

          markerCluster = new MarkerClusterer(map, markers);
        });
      }

      function createMarkerAddress(markerXmlNode)
      {
        var provider = markerXmlNode.getAttribute("provider");
        var street_address = markerXmlNode.getAttribute("street_address");
        var city = markerXmlNode.getAttribute("city");
        var state = markerXmlNode.getAttribute("state");
        var zipcode = markerXmlNode.getAttribute("zipcode");
        var phone = markerXmlNode.getAttribute("phone");

        var html = provider+"<br>"+street_address+"<br>"+city+"<br>"+state+" -
"+zipcode+"<br>"+phone;
        return html;
      }

      function searchLocations()
      {
        var address = document.getElementById("addressInput").value;
        if(address == null || address == '')
          alert('Please enter a valid address');

        var geocoder = new google.maps.Geocoder();
        geocoder.geocode({address: address}, function(results, status) {
               userLoc = results[0].geometry.location; //assign the global variable
               if(status == google.maps.GeocoderStatus.OK)
               {
            searchNear(userLoc);
          }
               else
               {
            alert('Please enter a valid address');
          }
        });
      }
      function searchNear(userLoc)
      {
        clearLocations();

        var radius = document.getElementById('radiusSelect').value;
        var searchUrl = 'ajax_search_near.php?lat=' + userLoc.lat() + '&lon=' +
userLoc.lng() + '&radius=' + radius;

        callServer(searchUrl, function(data) {
          var xml = parseXml(data);
          var markerNodes = xml.documentElement.getElementsByTagName("marker");
          var bounds = new google.maps.LatLngBounds();

          for(var i=0; i<markerNodes.length; i++)
          {
            var name = markerNodes[i].getAttribute("name");
            var address = createMarkerAddress(markerNodes[i]);
            var distance = parseFloat(markerNodes[i].getAttribute("distance"));
            var latlng = new google.maps.LatLng(
              parseFloat(markerNodes[i].getAttribute("lat")),
              parseFloat(markerNodes[i].getAttribute("lon")));

            createMarker(latlng, name, address, distance);
            bounds.extend(latlng);
          }

          map.fitBounds(bounds);
        });
      }

      function clearLocations()
      {
        infoWindow.close();
        for(var i = 0; i<markers.length; i++)
          markers[i].setMap(null);

        markerCluster.clearMarkers();
        markers.length = 0;
      }

      function parseXml(str)
      {
        if(window.ActiveXObject)
        {
          var doc = new ActiveXObject('Microsoft.XMLDOM');
          doc.loadXML(str);
          return doc;
        }
        else if(window.DOMParser)
        {
          return (new DOMParser).parseFromString(str, 'text/xml');
        }
      }

      function callServer(url, callback)
           {
        var request = window.ActiveXObject ? new ActiveXObject('Microsoft.XMLHTTP') : new
XMLHttpRequest();

        request.onreadystatechange = function() {
          if(request.readyState == 4) {
            request.onreadystatechange = doNothing;
            callback(request.responseText, request.status);
          }
        };

        request.open('GET', url, true);
        request.send(null);
      }

           function doNothing() {}
```

```
     function createMarker(latlng, name, address, distance)
     {
       var html;
       if(distance != -1)
       {
         var dist = distance.toString();
         html = "<b>"+name+" ("+dist.substring(0,3)+" mi)</b> <br>"+address;
  }
  else
  {
    html = "<b>"+name+"</b> <br>"+address;
  }

  var m = new google.maps.Marker({
    map: map,
    position: latlng
  });

  infoWindow = new google.maps.InfoWindow();
  google.maps.event.addListener(m, 'click', function() {
    infoWindow.setContent(html);
    infoWindow.open(map, m);
    if(document.getElementById("nearYou").checked)
    {
      if(userLoc != null)
      {
        calcRoute(userLoc, latlng)
      }
    }
  });

  return m;
}

     function roundNumber(number, digits)
     {
  var multiple = Math.pow(10, digits);
  var rndedNum = Math.round(number * multiple) / multiple;
  return rndedNum;
}

     function searchKeyPress(e)
{
  if(typeof e == 'undefined' && window.event) { e = window.event; }
  if(e.keyCode == 13)
    document.getElementById('btnSearch').onclick();
}

function calcRoute(userLoc, destLoc)
{
  if(alreadySearched != null && alreadySearched == destLoc)
  {
    return;
  }

  var request = {
      origin: userLoc,
      destination: destLoc,
      travelMode: google.maps.DirectionsTravelMode.DRIVING
  };

  directionsService.route(request, function(response, status) {
    if(status == google.maps.DirectionsStatus.OK)
    {
      directionsDisplay.setDirections(response);
      alreadySearched = destLoc;
    }
              else
    {
                           alert('Please enter a valid addresses.');
                    }
  });
```

```
      }

           function switchInput()
      {
             if(!document.getElementById("nearYou").checked)
        {
               document.getElementById("addressInput").setAttribute("disabled",
"disabled");
                    for(i = 0; i<markers.length; i++)
            markers[i].setMap(null);

                    markers = [];
                    loadMap();
             }
             else
        {
                    document.getElementById("addressInput").removeAttribute("disabled");
             }
           }
    </script>

  </head>
  <body onload="loadMap()">
  <div id="inputs">
  <div id="empty-space1"></div>
  <font id="header">Charge Station Locator</font><br><br>
  <form name="form1" action="" method="GET">
         <input type="radio" name="type" id="all" checked="checked"
onclick="switchInput()">All
         <input type="radio" name="type" id="nearYou" onclick="switchInput()">Nearby<br>
    Your address: <input type="text" id="addressInput"
onkeypress="searchKeyPress(event)"/>
         Find stations within:
         <select id="radiusSelect">
      <option value="5" selected>5mi</option>
      <option value="10">10mi</option>
      <option value="25">25mi</option>
      <option value="100">100mi</option>
      <option value="200">200mi</option>
    </select>
    <input type="button" id ="btnSearch" onclick="searchLocations()" value="Search"/>
         <input type="text" style="visibility:hidden">
         <br>
  </form>
  </div>
  <div><select id="locationSelect" style="width:100%;visibility:hidden"></select></div>
  <div id="map"></div>
  <div id="empty-space2"></div>
  </body>

</html>
```

<u>Source Code for Android App</u>:(Allen and Jeyenth)

LocateEVC.java

```
package ecen602.LocateEVC;

import android.app.Activity;
import android.app.AlertDialog;
import android.content.ActivityNotFoundException;
import android.content.Context;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationManager;
import android.net.Uri;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import java.io.IOException;
import java.text.DecimalFormat;
import java.util.Arrays;

public class LocateEVC extends Activity
{
    ListView listOfChargeStations;
    ChargeStation[] values; //Contains the results of the database search
    //These are the current location and search radius
    double lat=34.0522;
    double lon=-118.2428;
    int mile_radius=50;

    public static final double mile2lat=1/69.047;

    /** This function is used to create pop-up messages*/
    static final private int MENU_NYC = Menu.FIRST;
    static final private int MENU_LA = Menu.FIRST+1;
    static final private int MENU_Chi = Menu.FIRST+2;
    static final private int MENU_Dal = Menu.FIRST+3;
    static final private int MENU_Hou = Menu.FIRST+4;
    static final private int MENU_Phi = Menu.FIRST+5;

    /** This function is used to create pop-up messages*/
    public void showMessage(String msg)
    {
        AlertDialog.Builder dlgAlert  = new AlertDialog.Builder(this);
        dlgAlert.setMessage(msg);
        dlgAlert.setTitle("Locate EVC");
        dlgAlert.create().show();
    }

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Context context = getApplicationContext();
        LocationManager locationManager = (LocationManager)
this.getSystemService(Context.LOCATION_SERVICE);
        listOfChargeStations = (ListView)findViewById(R.id.list);

        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.NO_REQUIREMENT);
        criteria.setPowerRequirement(Criteria.NO_REQUIREMENT);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(false);
        criteria.setSpeedRequired(false);
        String providerName = locationManager.getBestProvider(criteria,true);
        if (providerName != null) {
            Location loc = locationManager.getLastKnownLocation(providerName);
            if(loc!=null){
                Log.e("LocateEVC", "Location: "+loc.toString());
                lat=loc.getLatitude();
                lon=loc.getLongitude();
            } else {
                showMessage("No last location found!");
            }
        } else {
            showMessage("Unable to find a location service!");
        }
        try{
            DatabaseAbstraction.checkCopyFromAssets(context);
        } catch(IOException ioe) {
            showMessage("Unable to import database from assets!");
            Log.e("LocateEVC", "exception", ioe);
        }
        update_list();
    }

    /** This function searches the database for all the charging stations within
     * the radius around the coordinates, lat and lon, and displays the results*/
    public void update_list(){
        Context context = getApplicationContext();
        //caculate the search filter
        double lat_dist=mile_radius*mile2lat;
        double min_lat=Math.max(lat-lat_dist,-90), max_lat=Math.min(lat+lat_dist,90);
//assuming the app doesn't need to cross the poles
        double div=Math.cos(Math.max(Math.abs(min_lat), Math.abs(max_lat))*Math.PI/180);
        double lon_dist=mile_radius*mile2lat/Math.max(div,mile_radius*mile2lat/179.9);
        double min_lon=lon-lon_dist, max_lon=lon+lon_dist;
        String where_string="lat BETWEEN "+min_lat+" AND "+max_lat;
        if(min_lon<-180){
            min_lon+=360;
            where_string+=" AND (lon BETWEEN -180 AND "+max_lon+" OR lon BETWEEN
"+min_lon+" AND 180)";
        } else if(max_lon>180){
            max_lon-=360;
            where_string+=" AND (lon BETWEEN -180 AND "+max_lon+" OR lon BETWEEN
"+min_lon+" AND 180)";
        } else {
            where_string+=" AND lon BETWEEN "+min_lon+" AND "+max_lon;
        }

        //open the database
        DatabaseAbstraction database = new DatabaseAbstraction(context);
        SQLiteDatabase db = database.getReadableDatabase();

        //process the search results
        Cursor results = db.rawQuery("SELECT cs.lat, cs.lon, p.name"
                + " FROM ChargingStations cs JOIN Providers p ON p.ID=cs.Provider_ID
WHERE "+where_string, null);
        Log.e("LocateEVC", "SELECT cs.lat, cs.lon, p.name"
                + " FROM ChargingStations cs JOIN Providers p ON p.ID=cs.Provider_ID
WHERE "+where_string);
        int x=0, size=results.getCount();
        values = new ChargeStation[size];
        ChargeStation.setLocation(lat,lon);

        if(size>0){ //if there are search results
            results.moveToFirst();
            while(x<size){
                values[x++]=new
ChargeStation(Double.parseDouble(results.getString(0)),Double.parseDouble(results.getStri
ng(1)),results.getString(2));
                results.moveToNext();
            }
            Arrays.sort(values);

            ArrayAdapter<ChargeStation> adapter = new ArrayAdapter<ChargeStation>(this,
                    R.layout.my_list_item_1, values);
            listOfChargeStations.setAdapter(adapter);
            listOfChargeStations.setOnItemClickListener(new OnItemClickListener(){
                public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
                    try{
                        //handle one of the locations being clicked
                        startActivity(new
Intent(Intent.ACTION_VIEW,Uri.parse("geo:"+LocateEVC.this.values[position].getLatLon()+"?
z=14")));
                    } catch(ActivityNotFoundException anfe) {
                        showMessage("Unable to find a map app to open the location!");
                    }
                }
            });
        }
    }

    /** Called when your activity's options menu needs to be created. */
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_NYC, 0, R.string.menu01).setShortcut('0', 'n');
        menu.add(0, MENU_LA, 0, R.string.menu02).setShortcut('1', 'l');
        menu.add(0, MENU_Chi, 0, R.string.menu03).setShortcut('2', 'c');
        menu.add(0, MENU_Dal, 0, R.string.menu04).setShortcut('3', 'd');
        menu.add(0, MENU_Hou, 0, R.string.menu05).setShortcut('4', 'h');
        menu.add(0, MENU_Phi, 0, R.string.menu06).setShortcut('5', 'p');
        return true;
    }

    /** Called right before your activity's option menu is displayed. */
    @Override
    public boolean onPrepareOptionsMenu(Menu menu) {
        super.onPrepareOptionsMenu(menu);
        return true;
    }

    /** Called when a menu item is selected. */
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case MENU_NYC:
                lat=40.7142;
                lon=-74.0064;
                update_list();
                return true;
            case MENU_LA:
                lat=34.0522;
                lon=-118.2428;
                update_list();
                return true;
            case MENU_Chi:
                lat=41.8500;
                lon=87.6500;
                update_list();
                return true;
            case MENU_Dal:
                lat=32.7828;
                lon=-96.8039;
                update_list();
                return true;
            case MENU_Hou:
                lat=29.7631;
                lon=-95.3631;
                update_list();
                return true;
            case MENU_Phi:
                lat=39.9522;
                lon=-75.1642;
                update_list();
                return true;
        }

        return super.onOptionsItemSelected(item);
    }
}

/** helper class written to manage the search results from the database*/
class ChargeStation implements Comparable<ChargeStation>{
    private double lat, lon;
    private String network;
    private static double at_lat, at_lon;
    public static final double meter2mile=1/1609.34;
    public static final DecimalFormat format = new DecimalFormat("0.00");

    public ChargeStation(double lat, double lon, String network) {
        this.lat = lat;
        this.lon = lon;
        this.network = network;
    }

    public float[] getDistance(){
        float[] dist=new float[2];
        Location.distanceBetween(at_lat,at_lon,lat,lon,dist);
        return dist;
    }

    public int compareTo(ChargeStation another) {
        float diff = getDistance()[0]-another.getDistance()[0];
        return (diff==0)?0:(diff>0?1:-1);
    }

    public static void setLocation(double at_lat, double at_lon){
        ChargeStation.at_lat=at_lat;
        ChargeStation.at_lon=at_lon;
    }

    @Override
    public String toString() {
        float[] dist=getDistance();
        return format.format(dist[0]*meter2mile)+" mi, "+format.format(dist[1])+"°:
"+network;
    }

    public String getLatLon(){
        return lat+","+lon;
    }
}
```