Volume 3 No. 3, March 2013 ISSN 2223-4985

International Journal of Information and Communication Technology Research

http://www.esjournals.org

Online Map Application Development Using Google Maps API, SQL Database, and ASP.NET

[1] Shunfu Hu, [2] Ting Dai

[1] Department of Geography, Southern Illinois University Edwardsville, Edwardsville, IL 62026, USA
[2] Farm Service Agency, Unites States Department of Agriculture, Washington, DC 20250, USA

ABSTRACT

Recently, there has seen increasing interest in developing online map services using Google Maps Application Programming Interface (API), Yahoo! Maps API, Microsoft Bing Maps API, Nokia Ovi Maps API, and ESRI ArcGIS API. Application developers utilize Maps API as a platform and combine spatial data from multiple sources to create new customized services – a buzzword commonly called map "mashups". The use of Maps API has revolutionized online mapping applications on the Internet. However, there are two major drawbacks in the map "mashups". First, the application developer utilizes open source methods such as XML, Fusion Tables, CSV, or KML for the preparation of limited amount of usually non-secured spatial data, which are not suitable for data sources in the format of a commercial database stored on a secure data server. Second, map "mashups" is focused on the use of the Maps API platform for the fast delivery of the customized services or data, so they usually lack of sophisticated functionalities and intuitive user interfaces that can offer the user the capability to manipulate the data. The objective of this paper is to demonstrate an online mapping application that requires the access to data sources in the format of a commercial database stored on a secure data server and that offers sophisticated functionalities for the user to manipulate the data. A case study of developing an online map service to display tens of thousands gardens on the Internet for the United States Department of Agriculture (USDA) People's Garden Initiative is presented. Google Maps API, Google Geocoderer, Microsoft SQL database, Microsoft aspx.NET, and Spry Framework for Ajax are employed to develop this online map application. It is also anticipated that the online map application can be used in major web browsers such as Microsoft Internet Explorer (IE) 7.0+, Google Chrome, Mozilla Firefox, and Apple Safari.

Keywords: *Online Mapping, Google Maps API, SQL Database, ASP.NET, USDA*

1. INTRODUCTION

The Google Maps launched in 2005 has revolutionized online mapping service applications on the World Wide Web. Based on Asynchronous JavaScript and XML (AJAX), a new type of client/server interaction was introduced in Google Maps to maintain a continuous connection between the client and the server for immediate downloading of additional map information [1]. In addition, Google also provides programmers its extensive sources of code called the Application Programming Interface (API). The API consists of a set of data structures, object classes or functions that can be used by a programmer using JavaScript, PHP or other scripting language [2]. With the current version 3, it is not required to register the API key to use the Google Maps. The new version supports both traditional web browsers such as Internet Explorer 7.0+, Firefox 3.0+, Safari 4+, Chrome, Android, BlackBerry, and Dolfin as well as web browsers such as the Apple iPad and iPhone on mobile devices. all of which having a full JavaScript implementation. These features make Google Maps JavaScript API the most commonly used Maps API for online mapping [3]. Other Maps APIs are also available for online mapping, including Yahoo! Maps API, Microsoft Bing Maps API, Nokia Ovi Maps API, and ESRI ArcGIS API.

Recently there has seen increasing interest in utilizing Google Maps API to implement web-based mapping services, ranging from simple applications to display just a few points of interest with information window to sophisticated map mashups [4] [5] [6] [7] [8]. Scholefield [9] developed a web-based map service for tourism of eighteenth and nineteenth century Edinburgh using Google Map API, Oracle RDMS (Relational Database Management System), Microsoft SQL (Structured Query Language), Perl, eXtensible Markup Language (XML), JavaScript, Hypertext Markup Language (HTML), eXtensible HTML (XHML), and Cascade Style Sheet (CSS). Similarly, Pejic et al. [10] developed an eTourism application using Google Map API to present prominent points of tourist destinations. Bildirici and Ulugtekin [11] demonstrates a web mapping service with Google Maps (API V2) mashups in which points, polylines and polygons from the data stored in Keyhole Markup Language (KML), XML and Geodatabase format are overlaid with Google Maps through JavaScript code. Liu and Palen [12] examine the use of Google Maps mashups in the crisis management for nine natural disasters such as earthquakes, fires, sea level rise, and so on using near real-time and publicly available data feeds. Hu [13] discusses a new approach of mashups in multimedia mapping that utilizes Google Maps API, Yahoo! Flickr API, and YouTube API to combine spatial data, multimedia information and functionality from multiple sources to create the online visitor guide for the Southern Illinois University Edwardsville campus. Hu [14] also uses Google Maps JavaScript API, and other JavaScript libraries such as jQuery, XML, and MarkerClusterer to develop an online map service to display and search over 600 locations of the gardens from the United States Department of Agriculture (USDA) People's Garden Initiative. However, there are two major drawbacks in the map "mashups". First, the application developer utilizes open source methods in the preparation of spatial data, including XML files, Google Fusion Tables, comma-separated values (CSV) files, or Keyhole Markup Language (KML) files. The drawback of such

methods is that it took efforts to reformat the original data and the data is not in its original database that can be updated in real time. Second, map "mashups" is focused on the use of the Maps API platform for the fast delivery of the customized services or data, so they usually lack of sophisticated functionalities and intuitive user interfaces that can offer the user the capability to manipulate the data. The objective of this paper is to demonstrate an online mapping application that requires the access to live data sources in the format of a commercial database stored on a secure data server and that offers sophisticated functionalities that allow the user to manipulate the data. A case study of developing an online map service to display tens of thousands gardens on the Internet for the United States Department of Agriculture (USDA) People's Garden Initiative is presented. Google Maps API, Google Geocoderer, Microsoft SQL database, Microsoft aspx.NET, and Spry Framework for Ajax are employed to develop this online map application. It is also anticipated that the online map application can be used in major brands of web browsers such as Microsoft Internet Explorer (IE) 7.0+, Google Chrome, Mozilla Firefox, and Apple Safari.

2. METHODOLOGY

2.1. Data Set

The USDA People's Garden Initiative is an effort to challenge its employees to establish People's Gardens at USDA facilities worldwide or help communities create gardens [15]. The garden information is collected initially through the USDA People's Garden online registration process and is in a Microsoft SQL Server 2005 database stored on a USDA's secure server. The data set for this project contains thousands of gardens, including the name of each garden, the street address, city, state, and zip code of the garden, the type of the garden (1 - At USDA Facilities; 2 - At Schools; 3 - At Other Places Within the Community; 4 - At Faith-based Centers; and 5 - At Other Federal Agencies), the geographic location (i.e., latitude and longitude in decimal degrees) of each garden, and more importantly what are planted in each garden.

2.2. "Mashup" of Google Maps API and SQL Database through JavaScript and XHTML

Since the online map application is a web application, every Maps API implementation is based on a web page. JavaScript is the native language of Google Maps. In addition, Google Maps is built of XHTML (Extensible HTML), formatted with CSS (Cascading Style Sheet) [16]. Therefore, both JavaScript and XHTML are used for developing the USDA online garden map application. Figure 1 illustrates a conceptual framework for the integration of the major components. The implementation is done using the Microsoft Visual Studio Express, which is a light-weight version of the Microsoft's Visual Studio (VS) and provides a set of free software programs in the form of Microsoft's integrated development environment (IDE). The VS Express includes Visual Web Developer Express (VWDE), Visual Basic Express, Visual C++ Express, Visual C# Express and SQL Server Express. The VWDE allows application developers to create ASP websites with the programming language either Visual C# or Visual Basic. It has a very user friendly interface for web design. The SQL Server Management Studio Express provides the capability to edit, update, and manage SQL Server Express database. For instance, the developer can create a new SQL database for a web site but import the data from Oracle, Microsoft ACCESS, Excel or ODBC sources. For this project, Visual Web Developer 2010 Express was chosen to develop the USDA online map application, and SQL Server Management Studio 2008 Express was selected to import the USDA garden database and to create a new SQL database called *PeoplesGarden* for the application development environment. In *PeoplesGarden* database, two views (virtual tables) were created to dynamically summarize the total garden counts based on the state names. Using the standard Structured Query Language (SQL), the first view called *Gardens* was created to tally the number of gardens for each US state, and the second view called *State_Centerpoints* that contains the center latitude and longitude of each state was appended to the first view so that the summary data can be displayed in the map window.

Microsoft aspx.NET technology was employed to connect to and retrieve data from the database. An aspx page contains scripts written in Microsoft .NET languages such as Visual Basic.NET and C#.NET. In this application, Visual Basic.NET was used to perform all the database tasks.

As shown in Figure 1, when the *garden.htm* is accessed by a web user, it first determines the user's web browser type (see section 2.4) and a browser-specific html file calls an *initialize* function from the *Main.js* JavaScript file. When the *Main.js* initializes the online map page, it sends a request to an aspx named *NewGetData.aspx* (see below). The aspx script retrieves garden data either at the national level or at the state level from *PeoplesGarden* database depending on users' requests.

```
function initialize()
{
   geocoder = new google.maps.Geocoder();

//For centering map on contiguous USA
  var myLatlng = new google.maps.LatLng(38, -95);
  var myOptions = {
            zoom: 4,
            center: myLatlng,
           scaleControl: true,
           mapTypeId:
                google.maps.MapTypeId.ROADMAP }

map = new google.maps.Map(document.getElementById
              ("map_canvas"), myOptions);

//Calls an aspx script to retrieve data from SQL Server database
$.get("newGetData.aspx?level=national", displaynational);
}
```

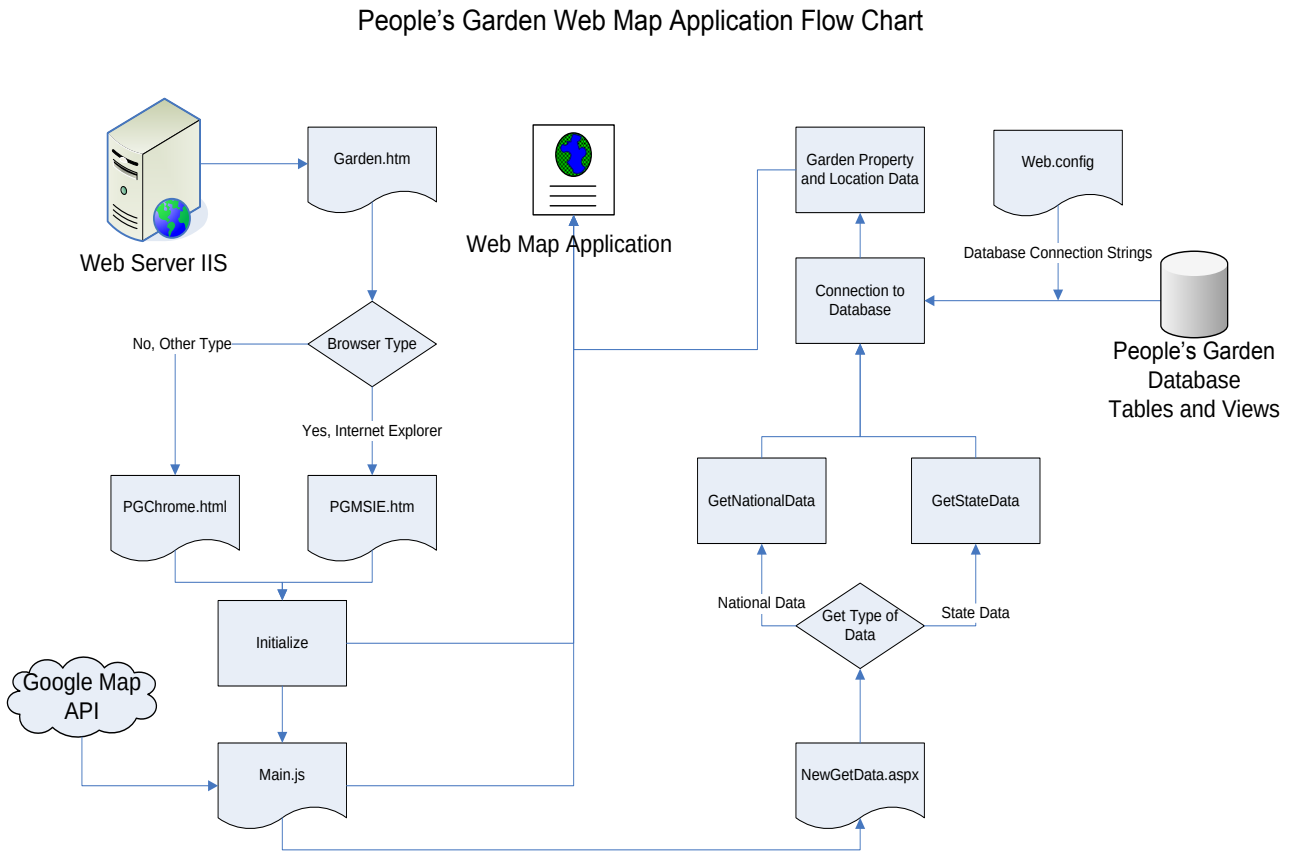

Figure 1. A conceptual framework of developing the People's Garden online map application.

The aspx script then connects to the database using a connection string defined in the website configuration file *web.config*.

```
<connectionStrings>
  <add name="MyDatabaseConnectionString"
  connectionString="Data Source=SERVERNAME;
  Initial Catalog=PeoplesGarden; Integrated
     Security=True" />
</connectionStrings>
```

Once the database is connected, the aspx uses two member functions to retrieve the garden data. The first function *getNationalData()* retrieves summary data for each US state including state name, total garden count by state, and state centric latitude and longitude (Figure 2). The second function *getStateData()* retrieves individual garden information in each selected state from the *PeoplesGarden* database, including Garden Name, Garden City, Garden State, Garden Zip Code, Garden Street Address, Garden Type, Latitude, Longitude, Photo Name, and Photo Location (Figure 3). Below is the Visual Basic.NET code that works with the *web.config* to get the individual garden information from the database.

```
Imports Microsoft.VisualBasic
Imports System.Data.SqlClient
Imports System.Configuration

Public Class NewDataLogic
Dim separator As String = ";;"
Dim ConnectionString As String =ConfigurationManager.
  ConnectionStrings("MyDatabaseConnectionString").
  ConnectionString

Public Function getStateData(state As String) As String
Dim myConnection As New SqlConnection
Dim myCommand As New SqlCommand
Dim myDataReader As SqlDataReader
Dim result As String = ""

If (state = "") Then state = "DC"
Try
   myConnection = New
   SqlConnection(ConnectionString)
   myConnection.Open()

 Dim mysql As String = "select Garden_Name as name,
Garden_city as city, Garden_State as state,
Garden_Zip_code as zipcode, Garden_Street_Address_1
as address, Garden_Type as gtype, Type_of_Garden as
CIndex, Lat as y, Lon as x, Photo_Name as Photo_Name,
Location as Location, Garden_Street_Address_2 as
address2 from Gardens where Garden_State = '" & state
```

```
& """

myCommand = New SqlCommand(mysql,
 myDataReader = myCommand.ExecuteReader()
        Dim x As String
        Dim y As String
        Dim name As String
        Dim st As String
        Dim city As String
        Dim zip As String
        Dim addr As String
        Dim cindex As String
        Dim gtype As String
        Dim Photo_Name As String
        Dim Location As String
        Dim address2 As String

 While myDataReader.Read()
     If (IsDBNull(myDataReader("x"))) Then
            x = ""
     Else
            x = CStr(myDataReader("x"))
     End If
     If (IsDBNull(myDataReader("y"))) Then
            y = ""
     Else
            y = CStr(myDataReader("y"))
     End If
     If (IsDBNull(myDataReader("name"))) Then
            name = ""
     Else
            name = CStr(myDataReader("name"))
     End If
     If (IsDBNull(myDataReader("State"))) Then
            st = ""
     Else
            st = CStr(myDataReader("State"))
     End If
     If (IsDBNull(myDataReader("city"))) Then
            city = ""
     Else
            city = CStr(myDataReader("city"))
     End If
     If (IsDBNull(myDataReader("zipcode"))) Then
            zip = ""
     Else
            zip = CStr(myDataReader("zipcode"))
     End If
     If (IsDBNull(myDataReader("address"))) Then
            addr = ""
 Else
       addr = CStr(myDataReader("address"))
 End If
 If (IsDBNull(myDataReader("CIndex"))) Then
       cindex = ""
 Else
       cindex = CStr(myDataReader("CIndex"))
       myConnection)

 End If
 If (IsDBNull(myDataReader("gtype"))) Then
       gtype = ""
 Else
       gtype = CStr(myDataReader("gtype"))
 End If
 If (IsDBNull(myDataReader("Photo_Name")))
 Then
       Photo_Name = ""
 Else
       Photo_Name =
         CStr(myDataReader("Photo_Name"))
 End If
 If (IsDBNull(myDataReader("Location"))) Then
       Location = ""
 Else
       Location = CStr(myDataReader("Location"))
 End If
 If (IsDBNull(myDataReader("address2"))) Then
       address2 = ""
 Else
       address2 = CStr(myDataReader("address2"))
 End If
 If (x <> "") And (y <> "") Then
       result = result + y + separator + x
       result = result + separator + name + separator + st
       result = result + separator + city + separator + zip
       result = result + separator + addr + separator +
            cindex
       result = result + separator + gtype + separator +
            Photo_Name
       result = result + separator + Location + separator +
            address2
       result = result + "|"
 End If
 End While
       myDataReader.Close()
       myDataReader = Nothing

 Catch ex As Exception
 Finally
       myConnection.Close()
       myConnection = Nothing
 End Try

 Return result
 End Function
```

The *result* string generated from the above code that contains individual garden information is passed back to *Main.js* and then processed to a point on the online map.

2.3. Development of Searching and Filtering Functions

It is required for this project to provide the user with a web interface for search functions such as Find Gardens by Location (e.g., address or zip code) and Find Gardens by State (Figure 2) so the user can have the option to see only selected gardens around that address or zipcode, or within that state. Furthermore, search functions need to be accompanied by another function called Filter by Type. The first two functions, Find Gardens by Location and Find Gardens by State, were accomplished by using Google Maps API's Geocoding process, which converts addresses (e.g., "8008 Davis Dr., St. Louis, MO") into geographic coordinates (e.g., latitude 38.64005705 and longitude -90.3373296). With this pair of latitude and longitude, the programmer can place a marker onto the map. This can be done using the geocoder function (i.e., class) from the Google Maps API. However, the programmer has to create a new geocoder object within the *Main.js function initialize()* as follows:

```
var geocoder = new google.maps.Geocoder();
```

and then create a new *codeaddress ()* function (refer to https://developers.google.com/maps/documentation/javascript/geocoding).

2.4. Web Browsers Compatibility

It is anticipated that the USDA online map service needs to be used in most of the web browsers such as Microsoft Internet Explorer (IE) 7.0+, Google Chrome, Mozilla Firefox, and Apple Safari. The initial testing of the USDA online map application indicated that there were compatibility issues. For instance, Microsoft IE (7, 8 and 9) and Mozilla Firefox did not support rounded corners for tabbed panels. Apple Safari for iPad and iPhone did not support flash movies. Only Google Chrome did not have any problem. Those issues were resolved by offering a main page *Garden.htm* to detect whether the browser is IE or Google Chrome, Mozilla Firefox, and Apple Safari. If it is IE, the web page is automatically redirected to *PGMSIE.htm*; otherwise, it is redirected to *PGchrome.html*. Below is the code for *Garden.htm*.

```
<html>
<head>
   <meta name="viewport" content="initial-scale=1.0, user-scalable=no" />
   <meta http-equiv="content-type" content="text/html; charset=UTF-8" />
   <title>USDA People's Garden Initiative</title>
   <script type="text/javascript" language="JavaScript">

   function detectbrowser() {
    var browserName = navigator.appName;
    switch (browserName) {
     case ("Microsoft Internet Explorer"):
      document.write('<META HTTP-EQUIV="REFRESH" CONTENT="1;URL=PGMSIE.htm">'); break;
     case ("Chrome"):
     case ("Firefox"):
     case ("Safari"):
     case ("Mozilla"):
     case ("Netscape"):
      document.write('<META HTTP-EQUIV="REFRESH" CONTENT="1;URL=PGchrome.html">');
      break;
     default: alert("You browser might not be supported");
        }
      }
   </script>
</head>
<body>
 <p> People's Garden</p>
 <script type="text/javascript" language="JavaScript">
      detectbrowser();
 </script>
</body>
</html>
```

2.5. Use of JavaScript and CSS to Design the Layout of the Online Map Application

In the design of the online map application, a HTML table with three-row layout design was adopted, including first row for a tabbed interface, second row for the map container, and third row for the garden type legend. In the tabbed interface, two tabs are provided, including one for Find Gardens by Location, and the other for Find Gardens by State (Figure 2). The tabbed interface was developed using the Adobe's Spry 1.6 framework for Ajax, which is a JavaScript library for the development of interactive web pages ([17]. To do so, the programmer has to download SpryTabbedPanels.css and SpryTabbedPanels.js (all open sources) from the Adobe Labs. The former links the CSS for the tabbed panel; the latter links the Spry *TabbedPanels* JavaScript library with the search functions. Both of them need to be placed in the head section of the web page (i.e., *PGMSIE.htm* or *PGChrome.html*) as follows:

```
<link href="SpryTabbedPanels.css" rel="stylesheet" type="text/css" />

<script src="SpryTabbedPanels.js" type="text/javascript">
</script>
```

Working in conjunction with the tab Find Gardens by Location is the HTML <input> tag and input field that allow the user to type in address or zip code from the keyboard. The HTML code looks as below:

```
<input id="address"  size="40" type="textbox"
value="Please type your address or zipcode here">
```

Then, the <select> tag is used to create a drop-down list for the user to select an item from a list and the <option> tags inside the select element specify the available items (i.e., options) in the list. For instance, Filter by Type allows the user to choose one of the five garden types with 0 as a default value for no filtering. It was implemented with the following HTML code:

```
<select id = "select_type" name="Garden_Type">
<option value="0" selected>Filter by Type</option>
<option value="1">At USDA Facilities</option>
```

<option value="2">At Schools</option>
<option value="3">At Other Places within the Community
</option>
<option value="4">At Faith-based Centers</option>
<option value="5">At Other Federal Agencies</option>
</select>

Similarly, the tab Find Gardens by State allows the user to choose one state from the list with "All States" as the default value to show all the gardens in the entire United States and its territories. It is worthy of note that each tab is linked to the search function and filter function described in the Section 2.3.

In the map container, the Google map or satellite imagery is displayed, and points of interest (e.g., gardens) are marked with the customized marker icons – the green shovel for gardens at USDA facilities, the yellow shovel for gardens at schools, and so on. The garden type legend is placed at the bottom of the map contains that match the marker icons displayed in the map container.
In order to provide the user with interaction with the map, a few standard Google Maps controls are added, such as Pan and Zoom controls; Map Scale control; and Map Type control– Roadmap and Satellite. In addition, tooltips (e.g., garden name) to the markers are provided, along with clickable marker icons with Google Maps API's standard Infowindow, which displays the information about each garden (i.e., garden name, address, city, state, zip code, a link to garden pictures, etc.).

3. RESULTS

The use of Google Maps API V3 provides a very efficient mechanism to deliver digital cartographic information to the Internet users with fast response time and user friendly interaction. Using Google Maps standard Map Type control, the user is able to choose one of the two map types: roadmap or satellite imagery. Figure 2 shows the startup view of the People's Garden online mapping application in Google Chrome. At the initial launch of the web page is the display of Google Maps with the number of gardens in each state. This gives the user a clear idea where most of the gardens are concentrated. Notice that in Figure 2 there are two tabs, Find Gardens by Location and Find Gardens by State. The user can click one of them to perform the search functions.

Figure 3 shows the result of a search function, Find Gardens by State. In this case, the user selects the state of Illinois. All of the individual gardens in the state are displayed on the map with the customized marker icons. The map legend at the bottom of the screen provides the five types of gardens shown on the map. This feature provides the user with a clear understanding about the gardens he/she is looking at.

Figure 4 shows the result a Filter by Type function. In this case, the user first select all the gardens in the state of Illinois and then filer the gardens by a type (e.g., At USDA Faculties). Figure 5 shows the result of another search function, Find Gardens by Location, using addresses or zip codes. The user can first type in the street
address, city, and state, or a zip code, and the map window will zoom into that address or zip code and display all the gardens around it. The user can then filer the gardens by a type (e.g., At USDA Facilities). The user can click on an icon at any time to get specific information about that garden in the standard Google Infowindow.

4. CONCLUSION AND DISCUSSION

This paper has demonstrated an online mapping application that was successfully developed using Google Maps API v3, Google Geocoding, Microsoft SQL Server Express database, and Spry Framework for Ajax. The case study presented in this article provides the advanced functionality to display the locations and state-based summary counts of USDA's thousands of peoples' gardens on the Internet with customized icons and map legend. It also provides the sophisticated functionalities for searching, filtering, and tabbed interface that offer the user the capability to manipulate the data.

Online mapping from a database being updated in real time can be very useful for many purposes. First, the database can be collected from an online registration process that, along with other information contains locational information such as latitude and longitude or street addresses. This is the way how the USDA people's garden information has been gathered. Such database can be stored on a secure server inside a firewall. Second, once the data has been collected and stored, it can be easily and directly retrieved in full or partially for online mapping application without going through a data format transformation as it has been done in the past (e.g., XML). Third, the backend database can be updated through the database interface and the resulting data changes will be reflected immediately on the web interface. Fourth, complex data manipulation can be carried out using the powerful SQL scripts in the backend databases.

Publishing and sharing geo-spatial data are becoming important and popular tasks in various applications. One particular sector is in the public health. Doctors at different offices across a region or a country can report certain type of disease (e.g., West Nile Virus, SARS) in real time to a centralize database, and such information can be delivered to an online map immediately so the health officials and the general public can quickly take preventive actions. The project described in this paper can be easily modified to meet the requirements of such important tasks.

ACKNOWLEDGEMENTS

We thank USDA People's Garden Initiative director Livia Marques for her leadership in this project, USDA Annie Ceccarini for her initial web interface design, USDA NRCS Tianpu Liang for his ASP.NET coding and database connection support, USDA OCIO John Roccaforte for providing database access, and Acacia Dai of the Thomas Jefferson High School for Science and Technology for graphics design for the web page.

International Journal of Information and Communication Technology Research

http://www.esjournals.org

Figure 2. The initial view of the USDA People's Garden online map application with the number of gardens for each state.

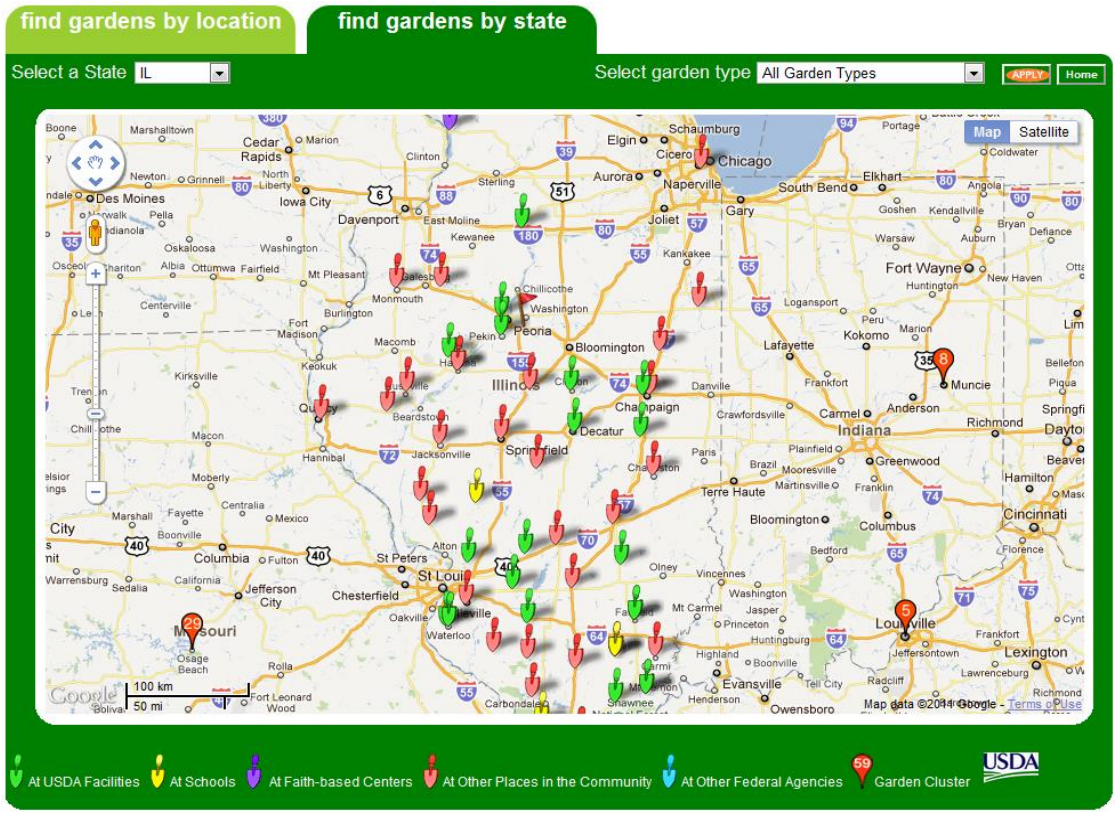

Figure 3. Find Gardens by State displays only the gardens in a selected state. The map legend at the bottom indicates the type of the gardens.

International Journal of Information and Communication Technology Research

http://www.esjournals.org

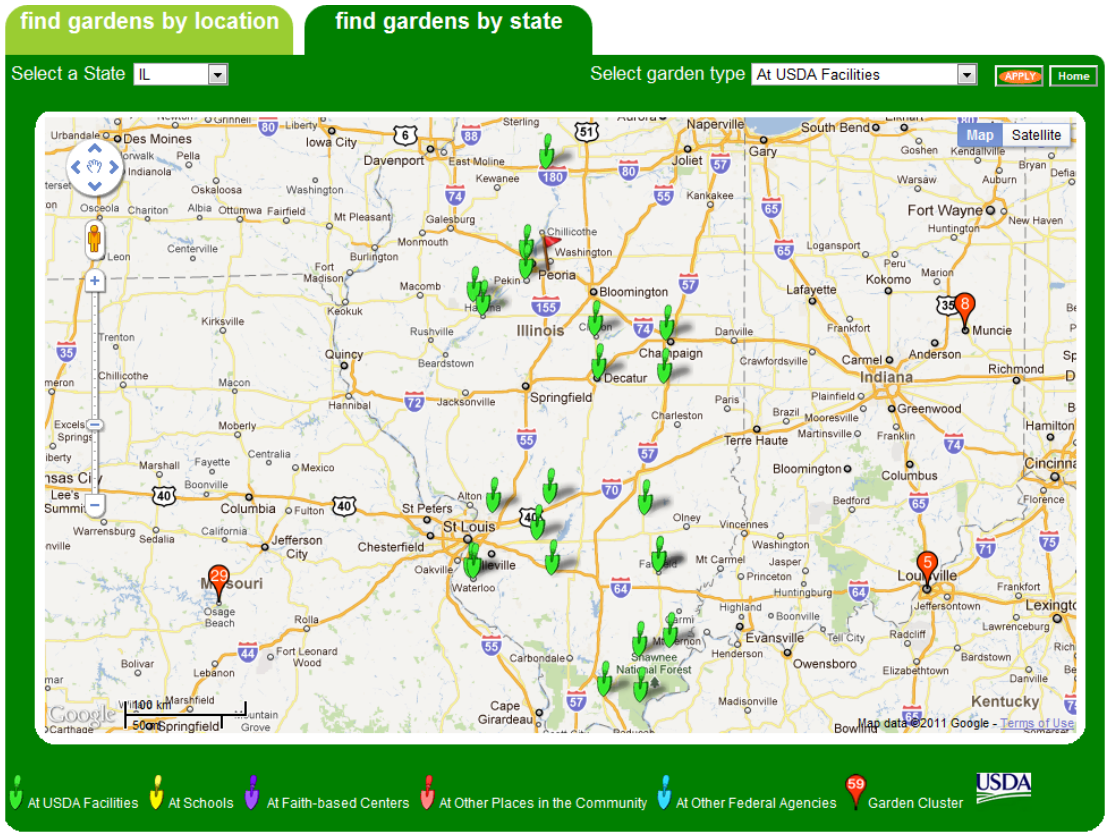

Figure 4. Filter by Type (in this case, At USDA Facilities) displays only the gardens in that category.

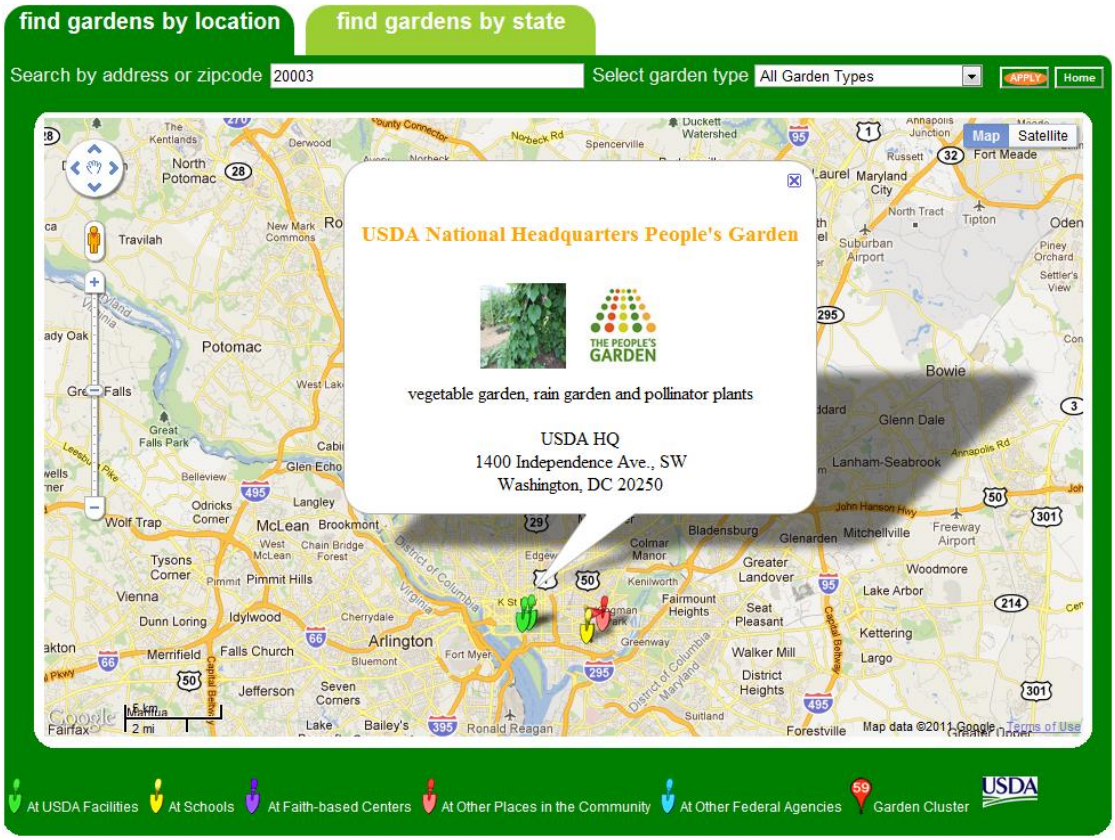

Figure 5. Find Gardens by Location (address or zipcode) will zoom to that location. If the user clicks a garden icon on the map, the information about that garden is displayed in the Infowindow.

REFERENCES

[1]. Peterson, M. P. (2008). International Perspectives on Maps and the Internet: An Introduction, In M. P. Peterson (Ed.), *International Perspectives on Maps and the Internet (pp. 3-10)*, Springer.

[2]. Udell, S. (2009). Beginning Google Maps Mashups with Mapplets, KML, and GeoRSS. New York, NY: Apress.

[3]. Peterson, M. P. (2012). Online Mapping with APIs, *Online Maps with APIs and Mapservices* (M. P. Peterson, ed.), Springer, pp. 3-12.

[4]. Johnston, L. R. and K. L. Jensen. (2009). MapHappy: A user-centered interface to library map collections via a Google maps "Mashup". *Journal of Map and Geography Libraries*, 5(2): 114-130.

[5]. Peng, X. and X. Wu. (2010). Digital campus map publishing based on Google Map API *Journal of Geomatics*, 35(1), pp. 25-27,

[6]. Roth, R. E. and K. S. Ross. (2009). Extending the Google maps API for event animation mashups. *Cartographic Perspectives*, 64, pp. 21-31.

[7]. Chow, T. E. (2008). The potential of Maps APIs for Internet GIS Applications, *Transactions in GIS*, 12(2), pp. 179-191.

[8]. Pan, B., J. C. Crottsa, and B. Mullerb. (2010). Developing Web-Based Tourist Information Tools Using Google Map. http://www.ota.cofc.edu/pan/PanCrottsMullerDeveloping GoogleMap.pdf. Last accessed on May 7, 2011.

[9]. Scholefield, K. (2008). Web based map services for scientific tourism: a case study of eighteenth and nineteenth century Edinburgh. Master of Science Thesis, http://hdl.handle.net/1842/2475.

[10]. Pejic, A., S. Pletl, and B. Pejic. (2009). An Expert System for Tourists Using Google Maps API, *7th International Symposium on Intelligent Systems and Informatics*, SISY '09.

[11]. Bildirici, I. O. and N. N. Ulugtekin. (2010). Web Mapping with Google Maps Mashups**:** Overlaying Geodata. A Special Joint Symposium of ISPRS Technical Commission IV & AutoCarto in Conjunction With ASPRS/CaGIS 2010 Fall Specialty Conference, November 15-19, Orlando, Florida.

[12]. Liu, S. B., and L. Palen. (2010). The New cartographers: Crisis Map Mashups and the Emergence of Neogeographic Practice, Cartography and geographic Information System, Vol. 37, No. 1, pp. 69-90.

[13]. Hu, S. (2012). Multimedia Mapping on the Internet Using Commercial APIs, *Online Maps with APIs and Mapservices* (M. P. Peterson, ed.), Springer, pp. 61-71.

[14]. Hu, S. (2012). Online Map Service Using Google Maps API and Other JavaScript Libraries: An Open Source Method, *Online Maps with APIs and Mapservices* (M. P. Peterson, ed.), Springer, pp. 265-278.

[15]. United States Department of Agriculture (USDA). (2011). People's Garden Initiative. http://www.usda.gov/wps/portal/usda/usdahome?navid= PEOPLES_GARDEN. Last accessed on February 9, 2011.

[16]. Udell, S. (2009). Beginning Google Maps Mashups with Mapplets, KML, and GeoRSS. New York, NY: Apress.

[17]. Adobe. (2011). Working with Spry 1.6. http://livedocs.adobe.com/en_US/Spry/SDG/help.html. Last accessed on May 10, 2011.