Park Social: The Essential Urban Parking Assistant

by

Timothy Pollack

Submitted to
the Faculty of the Department of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor of Science
in Information Technology

University of Cincinnati
College of Education, Criminal Justice, and Human Services

April 2013
Park Social: The Essential Urban Parking Assistant

by

Timothy Pollack

Submitted to
the Faculty of the Department of Information Technology
in Partial Fulfillment of the Requirements for
the Degree of Bachelor in Science
in Information Technology

© Copyright 2013 Timothy Pollack

The author grants to the Department of Information Technology permission
to reproduce and distribute copies of this document in whole or in part.

Insert Your Name Here

Date

Anna Prabhakar

04/22/2013

Insert Faculty Advisor's Name Here, Faculty Advisor

Date
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem Need</td>
<td>4</td>
</tr>
<tr>
<td>Technical Areas</td>
<td>5</td>
</tr>
<tr>
<td>Database</td>
<td>6</td>
</tr>
<tr>
<td>User Profile</td>
<td>7</td>
</tr>
<tr>
<td>Design Protocol</td>
<td>10</td>
</tr>
<tr>
<td>Resources and Logistics</td>
<td>13</td>
</tr>
<tr>
<td>Conclusion</td>
<td>16</td>
</tr>
<tr>
<td>Sources</td>
<td>17</td>
</tr>
<tr>
<td>Appendix A</td>
<td>18</td>
</tr>
<tr>
<td>Appendix B</td>
<td>39</td>
</tr>
<tr>
<td>Appendix C</td>
<td>50</td>
</tr>
<tr>
<td>Appendix D</td>
<td>51</td>
</tr>
<tr>
<td>Appendix E</td>
<td>55</td>
</tr>
<tr>
<td>Appendix F</td>
<td>59</td>
</tr>
<tr>
<td>Appendix G</td>
<td>61</td>
</tr>
<tr>
<td>Appendix H</td>
<td>67</td>
</tr>
</tbody>
</table>
Park Social

*The essential urban parking assistant*

**Problem Need**

Population growth is an area of concern that largely impacts our economy. Local governments not only have to worry about where more people will live, but also how having more people changes traffic patterns and parking habits. Individuals, on the other hand, only care how these people will impact their every-day decisions. Traffic is an aspect of driving that gauges the roads used to commute between places, and has dozens of mobile applications and Websites dedicated to monitoring it and displaying current conditions. However, driving consists of another aspect that is sometimes just as frustrating, but largely forgotten until it is needed. That aspect is parking.

The lack of available parking acts like a plague in hot-spot communities. Roads frequently undergo construction, and stores and houses sprout like wildflowers in high-traffic areas to the point that there are always more cars than available parking spaces. Parking from businesses clashes with the needs of the surrounding residential areas. Popular special events bring spikes of interest to certain areas, while decreasing the amount of available parking spaces with street closures. It would seem that parking would be amazingly easier and less stressful if real-time information could be collected and displayed about the availability of parking. Currently available applications show only the garages and lots downtown and their normal prices. This data is largely static and occasionally out-of-date.
This paper outlines a crowd-source-based mobile Web application that will keep track of available parking areas, including both on- and off-street, in hot-spot areas around any densely populated area. The intent for this application is to allow users to help maintain a map of parking areas by entering information about where they park their cars. Users are able to view or note if an area is full or has open parking spaces, as well as the cost of parking there. Users can also learn the status of street closures or limited traffic patterns. The information provided by users will be helpful in determining where to park in hot-spot areas or for crowded events.

**Technical Areas**

The Park Social application focuses on two main areas of Information Technology: software development and databases.

*Software Development*

Park Social is a Web application designed for use with mobile devices. It responds to the need for people to be able to find parking on-the-fly in heavily populated urban areas or for special events. The application will require using current client-side and server-side scripting languages to create an application that will help end users find parking areas in an urban environment. The application uses JavaScript and JQuery Mobile, along with HTML 5 and CSS3 standards to provide an interactive front end for users that can be used in any Web browser on either mobile devices or personal computers. The application also uses the Google Maps API to display an interactive map and work with the GPS location of the device. A Web service API was developed using PHP in order
to interact easily, efficiently, and securely with back-end data stored in a MySQL database. Because of the complexity of the Park Social application and the functionality it provides, it was necessary to have a good user interface design that maintains simplicity while providing adequate information to the user.

**Database**

The Park Social application required the creation of a MySQL database. This database holds location-specific information about parking areas and locations, along with keeping track of usage data for those locations based on where users park. This location data is taken entirely from user data and changes frequently. Therefore, the efficiency of code and reliable database design is essential to both performance and security.

**Solution Definition**

The solution for the shortage of available parking information around urban areas is a Web application to be used on mobile devices that will provide valuable parking information to its users. This application, Park Social, offers up-to-date information about available parking area locations. A large part of the data for the application is managed by its users. By allowing users to input information about parking areas and current availability, the application is able to offer real-time information with regards to parking during peak times or for special events.

Park Social can easily become an important application in anyone’s life because of its simplistic design as well as its unique data that it offer its customers. As part of a busy and ever-changing environment, Uptown Cincinnati can be a confusing area
through which to drive, let alone find a parking space near one of its many crowded retail or restaurant areas, hospitals, university buildings, or residences. Whether it is used frequently or rarely, Park Social is an application that provides information that currently does not exist all in one location or, in some cases, does not exist at all. While using this application, users can expect to feel less stressed about their commute to any destination in Uptown, be prepared for the extreme crowding of special events, and save valuable time by not having to search for available parking.

**User Profile**

The primary users of the Park Social application fall into one of two categories: end users (Users) or parking lot owners (Proprietors). The Users will be those who use the application for the primary purpose of finding parking. The Proprietors will have a special account that binds them to the parking area(s) over which they have control or ownership.

Either type of user will have the same technical qualities. As a whole, users can range from those who are very technically savvy with their smartphone devices, and those who only use the device as a phone. With this kind of application, however, it is most likely that those who download it will be at least moderately familiar with other applications that display maps and use GPS, such as navigation applications. Those who do not use similar applications already would have no reason to start now. Even with the similarities between Park Social and similar map applications, there will be guides and tips available to explain functionality to the users. The User will be examined first.
The largest group of potential users of the application will primarily consist of students, faculty, and staff of the University of Cincinnati, or others who are employed nearby. This category of User needs to park around the Uptown area almost every day, and some even multiple times per day. They have some knowledge of the area from what they have driven in the past, but they may or may not know the intricacies of the surrounding neighborhoods and busy streets. They also have some existing knowledge of common parking areas, but may or may not know of nearby parking areas or structures, or be aware of on-street parking or the laws and times regarding those areas at various times of day. This type of User might have frequent contact with the application, which would make him or her most likely user to add or update parking availability or vital information for the area.

The smallest group of potential users of the application will primarily consist of residents of the Uptown area. This category of User will have an intimate knowledge of the streets and neighboring areas, as well as local business hot-spots. This User will need to park several times on a daily basis, but, because of the environment, the User may not be able to easily find available parking on the street or even block on which he or she lives. This type of User might have frequent contact with the application, making him or her likely to add or update parking information for the area. This User may also take advantage of the last parked location feature if an abnormal parking location was utilized that day. This User may also use the broadcast feature to alert friends where parking is available next to a certain location.

The second largest group of potential users of the application primarily consists of those who have little-to-no working knowledge of the Uptown area. This User could
be a parent of a student being delivered to campus, a spectator of a sporting event or concert, a hospital visitor, or just a stranger to the area. This type of User may need to park once per week, once per month, or even only once per year. This User is unlikely to enter any additional data, but will find the information entered by others to be very helpful. Without much knowledge of the area, this User is the most likely to try to simply navigate to the nearest available parking structure or area to avoid waiting too long in traffic.

The second type of user is the Proprietor. The Proprietor is a person who owns a parking area or structure, or who works for such a business. The Proprietor and Park Social will have a symbiotic relationship. For the application, the Proprietor will be able to update and maintain the information for his or her parking area. This information will need to be verified against fraudulent claims so that the Proprietor is unable to take advantage of Park Social End Users. For the Proprietor, having this information up-to-date and displayed on the map will attract customers that may not otherwise be aware of the parking area. The Proprietor will also be able to pay to have their parking area favored over others. This advertising might be a discount for a specific day or event, a coupon for the customer to show the attendant, or simply having the parking area pop up on the map in a way it would not normally. Through the application, the Proprietor may also see what other parking locations are doing for their advertisements to attract customers and put out competing ads accordingly. Depending on the amount of advertising wanted and special event prices, an individual Proprietor may use the application either frequently or infrequently, though the possibility exists for either.
Design Protocol

The Park Social application caters to two specific types of actors already mentioned: the End User and the Proprietor. These actors will have different subsets of functionality based on their specific needs for using the application; however, both actors consume a limited common core set of functionality. The complete list of functionality for both actors can be seen illustrated in Figure 1 below.

Figure 1: Use case diagram
The End User, being the primary consumer of the Park Social application, has the larger set of functionality. While some instances of the End User will not use very many features, there may be others who will use everything and crave even more. Because of the wide range of customers for the End User, the application will default to displaying as few buttons and text as would be needed. Should the End User feel so inclined, there exists a configuration settings menu that allows the customization of what controls and information will be displayed on both the map screen and the parking area information screen.

As the End User enters the application for the first time, they should see only a map centered on their GPS location and showing tags of any nearby parking areas. Upon clicking a tag once, the name and location of the parking area will appear, along with a special event price, if applicable. If clicked a second time, the parking area information screen will appear with pertinent information about the location. Some of the information displayed by default will be an hourly price (or special event price if applicable), hours available for parking, a contact number, and an estimated status of how busy the parking area might be for that day and time or for a special event. If the parking area tag is long-clicked on the map screen, a navigation button will appear. This button will open the native navigation application for the user set to navigate to the selected location. The navigation application to be opened will be able to be configured through the configuration screen if multiple navigation applications are installed.

For the more advanced End User, other features can be added to the aforementioned screens. A “Park Here” tag can be added to the map in such a way that when the End User stops moving, a button will appear for parking. When clicked, a
marker is placed at the current GPS location so that it is easy to return to the exact location parked by seeing it on the map. For those who use metered street parking, a meter timer will also be available so that there is no worry about getting towed for not knowing how much time is left on the meter.

The End User will also be able to provide credentials to log in to the system. These credentials may or may not be tied to social media accounts. After logging in, the End User will be able to edit the parking area that is selected in order to help keep parking information up-to-date. Adding locations will also be available when an empty location on the map is long-clicked. Other options will also be made available at this point, such as being able to leave comments at a parking area location for others to see, and being able to broadcast a parking area to selected friends, alerting them of open parking or current parking locations.

The Proprietor has a different set of options. When logged in, the information screen for the owned parking area will appear. If no parking areas are owned, the map will appear to select an existing location or create a new location. These new locations will have to be verified and tied to the Proprietor account. If multiple areas are owned, a list will appear showing all available parking areas. From this parking area information screen, the Proprietor will be allowed to update or add information pertaining to their parking area and add a picture of the area. The Proprietor will be enticed to add this information, as it will help advertise the parking area to the End User. The Proprietor will also have buttons to advertise, and be able to select which type of advertisement they would like to purchase.
Resources and Logistics

Software and Hardware Needs

Application development for the Web, more specifically for use by mobile Web browsers, has certain requirements from both software and hardware perspectives. These requirements allow the developer to develop code both for the mobile device and Web services for back-end communication, create and maintain application data, deploy the application to a device, and test the application in a real-world environment.

Software

JavaScript. JavaScript is a client-side scripting language that allows for the manipulation of elements on a Web page. It is a necessary component of Web applications where any user interaction or dynamic content is to be utilized.

jQuery Mobile. jQuery is a complex JavaScript library which simplifies many activities that can be accomplished with JavaScript but are very complicated, such as animation or controlling groups of HTML controls. jQuery Mobile is an extension the jQuery library that focuses on creating crisp, smooth user interfaces and controls that are specifically designed for use with mobile devices.

Google Maps API. Google Maps is an open-source mapping Web service technology provided by Google. Not only does the API provide map data for display in the Web application, but it also allows for addresses to be pin-pointed using GPS coordinates and the ability for both the user and
Web application to interact with the displayed map and display appropriate data.

**PHP.** PHP: Hypertext Preprocessor is a dynamic, open-source server-side scripting language. Its inclusion in the design of this application is to create a back-end application to take Web service calls, allowing the users to send data to and retrieve data from a central data repository.

**MySQL.** MySQL is an open-source relational database management system which is widely used to hold data for Web applications. This application will need to use databases to store user information and parking location data.

**Hardware**

**Web Server.** The back-end Web application and databases need to reside on a Web server in order to provide calls to the application and necessary Web services over the internet. These can be set up and maintained fairly cheaply depending on usage.

**Smartphone Device.** Being the target device for the application, a mobile device is needed to test the deployment of the application in a real-world environment. The phone used for the development of the application was an iPhone 4s already owned by the development team.

**Project Cost**

Due to the nature of Web application development, all open-source technologies are free to use, and there are no major hardware requirements. The low costs of software development allow the developer to cover all of the costs while the application
remains small. Once the application grows to a size requiring dedicated resources and increased data transfer capabilities, alternate funding options should be readily available through merchant advertising.

<table>
<thead>
<tr>
<th></th>
<th>Actual Cost</th>
<th>Start-up Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Software</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JavaScript / jQuery Mobile</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Google Maps API</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PHP</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MySQL Databases</td>
<td>0</td>
<td>0+ (can grow with app usage)</td>
</tr>
<tr>
<td><strong>Hardware</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Web Server</td>
<td>0</td>
<td>8.95+ / mo (grows with app usage)</td>
</tr>
<tr>
<td>Test Device(s)</td>
<td>0</td>
<td>varies by device</td>
</tr>
</tbody>
</table>

**Figure 2:** Park Social cost structure

As seen in Figure 2, the cost structure decreases dramatically because of products already owned (iPhone test device) and services provided by the University of Cincinnati (Web hosting services). Some additional costs will be needed after the project is stored elsewhere, such as third-party Web server hosting. Additional testing devices may be needed as the technology changes to make certain that all functionality displays correctly and with appropriate spacing on any screen size and browser.

**Timeline**

Due to a late start on this particular project, the activities required for completing the application spanned from December 2012 to April 2013. There were three target dates for the completion of various project deliverables. All preliminary project documents were submitted for grading on December 14th, 2012; the working prototype of the application had to be completed for progress presentation on January 7th, 2013;
and the final project was exhibited on April 2\textsuperscript{nd}, 2013, at the University of Cincinnati’s Information Technology Tech Expo. All other target dates for the project timeline are merely for the development team to set goals and track development progress.

\textbf{Conclusion}

Being one of the most commonly overlooked activities of our days, parking is one that frequently causes headaches and frustration. In a busy, fast-paced world, the last thing we want to do when we go somewhere is waste valuable time trying to complete such a simple activity. While some of the larger landmark cities are finally seeing the need to handle parking in a user-friendly way, smaller cities, such as Cincinnati, are still floundering around with no quick solution.

Being the most densely populated area of the city, the Uptown area of Cincinnati faces daily traffic from a huge student population, in addition to the University of Cincinnati’s commuting faculty and staff, thousands of medical employees, and normal residents. The diversity of density of this population makes the Uptown area a prime candidate to test such a parking solution. The goal of Park Social is for Cincinnatians to take charge of parking around the Uptown area. Park Social is an iPhone application that will allow users to keep track of real-time parking information and view a map of parking structures and street availability.

If approved, the Park Social application will utilize modern development techniques and technologies to help solve a problem that plagues the people of Cincinnati. If successful, the application could very easily support other populated areas across the country.
Sources


Appendix A - index.html javascript

```javascript
var PSID;
var DEFAULT_ZOOM = 16;
var geocoder;
var map;
var marker;
var tempMarker = new google.maps.Marker;
var carMarker = new google.maps.Marker({ icon: 'img/car-park.png' });
var otherCarLocationMarkers = [];
var parkingLocationMarkers = [];
var userIP;
var infobox;
var centerLatLng;
var timerVar;
var flashTimerVar;
var lastSelectedParkingLocationId = 0;
var isAtParkingLocation = false;
var PreviouslyGeodedLocations_LatLng = [];
var PreviouslyGeodedLocations_Text = [];

// initializes the Mobiscroll window, used for setting meter timer
function initMobiscroll() {
    $('#timeScroller').mobiscroll().time({
        headerText: 'Parking Meter Time',
        theme: 'jqm',
        display: 'modal',
        animate: 'slidevertical',
        mode: 'scroller',
        timeFormat: 'HH:ii',
        timeWheels: 'HHii',
        width: 100,
        onSelect: function() {
            var val = $(this).mobiscroll('getValue');
            var hours = setLeadingZeros(val[0] + '', 2);
            var minutes = setLeadingZeros(val[1] + '', 2);
```
    parkMeHere(hours + minutes); // parameter in format of HHMM
    }
}});

// sets the user's parking location and sets the timer if necessary
// param: meterLength - expecting string in the format of HHMM
function parkMeHere(meterLength) {
    var lat;
    var lon;
    var locId = 0;

    // if there is a custom marker on the map, use its location
    if (tempMarker.getMap() != null && tempMarker.position != null) {
        lat = tempMarker.position.lat();
        lon = tempMarker.position.lng();
    } // if there is a previously selected parking location, use it
    else if (lastSelectedParkingLocationId > 0) {
        var x = getParkingLocationIndexById(lastSelectedParkingLocationId);
        if (x > -1) {
            locId = LocalParkingLocations_Id[x];
            lat = LocalParkingLocations_Latitude[x];
            lon = LocalParkingLocations_Longitude[x];
            isAtParkingLocation = true;
        }
    } else { // default use current position
        lat = marker.position.lat();
        lon = marker.position.lng();
    }

    // call dataService to park user
    // if successful call, will show parked car and re-init mobiscroll
    if (parkUser(PSID, lat, lon, meterLength, locId)) {
        $('#timeScroller').mobiscroll('setValue', '0', false, 0);
        $('#btnPark .ui-btn-text').text('Leave');
        showParkedCarInfo();
    }
// returns the index of the parking location using the given ID
// returns -1 if not found
function getParkingLocationIndexById(id) {
    for (var x = 0; x < LocalParkingLocations_Id.length; x++) {
        if (id == LocalParkingLocations_Id[x]) {
            return x;
        }
    }
    return -1;
}

// displays the parked car icon
function showParkedCarInfo() {
    carMarker.setPosition(new google.maps.LatLng(
        ParkedCarInfoLatitude, ParkedCarInfoLongitude));
    carMarker.setMap(map);

    // removes infobox and marker if not at a parking area
    if (!isAtParkingLocation) {
        infobox.setVisible(false);
        tempMarker.setMap(null);
    }

    // if meter length is valid, run the timer() method every 1 second
    if (!isNaN(ParkedCarInfoMeterLength) && parseInt(ParkedCarInfoMeterLength) > 0) {
        timerVar = setInterval(function() { timer() }, 1000);
    }
}

// initializing toolbar and form buttons
function initButtons() {
    // TOOLBAR - Add location
    $('#btnAdd').click(function () {
        // code
if (tempMarker.getMap() && tempMarker.getPosition()) {
    // show form and initialize fields
    flipToDialog('addFormDialog', 'Add Location');
    clearFormFields();
    $('#locationName').val($('#infobox').html());
    $('#locationLatitude').val(tempMarker.getPosition().lat());
    $('#locationLongitude').val(tempMarker.getPosition().lng());
    disableFormFields(false); // can edit fields
    $('#formRemove').hide(); // hide delete button
}
});

// TOOLBAR - Edit location
$('#btnEdit').click(function () {
    if (lastSelectedParkingLocationId) {
        // can edit fields
        setFormFieldValues('addFormDialog', 'Edit Location', false)
        $('#formRemove').show(); // show delete button
    }
});

// TOOLBAR - Show location info
$('#btnInfo').click(function () {
    if (lastSelectedParkingLocationId) {
        // cannot edit fields
        setFormFieldValues('addFormDialog', 'Location Info', true);
        $('#formRemove').hide(); // hide delete button
    }
});

// before init, calls dataService to get existing parking info
// also sets isCarParked as a result
if (getParkedCarInfoIfExists(PSID)) {
    setParkButtonText('Leave');
    showParkedCarInfo();
}
// TOOLBAR - Park
$('#btnPark').click(function () {
    if (isCarParked) {
        unparkUser(PSID);
        stopTimer(false);
        carMarker.setMap(null); // clear parked marker
        isAtParkingLocation = false;
        infobox.setVisible(false);
        setParkButtonText('Park');
    } else {
        handlePark();
    }
});

// DIALOG FORM - Park
$('#formPark').click(function () {
    handlePark();
});

// DIALOG FORM - Delete location
$('#formRemove').click(function () {
    var locId = $('#locationId').val();
    var x = getParkingLocationIndexById(locId);

    // call dataService to delete location;
    if (softDeleteLocation(PSID, x)) {
        //if successful, remove from cached array as well
        lastSelectedParkingLocationId = 0;
        infobox.setMap(null);
        LocalParkingLocations_Id.splice(x, 1);
        LocalParkingLocations_Latitude.splice(x, 1);
        LocalParkingLocations_Longitude.splice(x, 1);
        LocalParkingLocations_Name.splice(x, 1);
        LocalParkingLocations_Type.splice(x, 1);
        LocalParkingLocations_Icon.splice(x, 1);
LocalParkingLocations_Comments.splice(x, 1);
LocalParkingLocations_NumSpaces.splice(x, 1);

// remove parking location marker
for (var z = 0; z < parkingLocationMarkers.length; z++) {
    if (parkingLocationMarkers[z].getTitle() == locId) {
        parkingLocationMarkers[z].setMap(null);
        parkingLocationMarkers.splice(z, 1);
        break;
    }
}

return true; // successful
}
return false; // failure
});

// DIALOG FORM - Submit info for new location
$('#formSubmit').click(function () {
    // get variable data from form fields
    var locId = $('#locationId').val();
    var locLat = $('#locationLatitude').val();
    var locLon = $('#locationLongitude').val();
    var locName = $('#locationName').val();
    var locType = $('.ui-icon-radio-on').parent().text().trim();
    var locIcon = null;
    var locSpaces = $('#locationSize').val();
    var locComments = $('#locationComments').val();
    var x;

    // get correct icon info by type
    for (var ico = 0; ico < ParkingLocationTypes.length; ico++) {
        if (locType == ParkingLocationTypes[ico]) {
            locIcon = ParkingLocationIcons[ico];
            break;
        }
    }
});
// if new location
if (!locId || locId == '') {
    // add to arrays for dataService to access
    x = LocalParkingLocations_Id.length; // x is last index
    LocalParkingLocations_Latitude.push(locLat);
    LocalParkingLocations_Longitude.push(locLon);
    LocalParkingLocations_Name.push(locName);
    LocalParkingLocations_Type.push(locType);
    LocalParkingLocations_Icon.push(locIcon);
    LocalParkingLocations_Comments.push(locComments);
    LocalParkingLocations_NumSpaces.push(locSpaces);

    // if dataService call successful
    if (addUpdateLocation(PSID, x)) {
        // set a new map marker and select it
        setupNewParkingLocationMarker(x, new
            google.maps.LatLng(locLat, locLon));
        lastSelectedParkingLocationId =
            LocalParkingLocations_Id[x];

        tempMarker.setMap(null); // remove old marker
        return true; // success
    }
} else { // old location
    // update location info in arrays
    x = getParkingLocationIndexById(locId);

    oldIcon = LocalParkingLocations_Icon[x];

    LocalParkingLocations_Name[x] = locName;
    LocalParkingLocations_Type[x] = locType;
    LocalParkingLocations_Icon[x] = locIcon;
    LocalParkingLocations_Comments[x] = locComments;
    LocalParkingLocations_NumSpaces[x] = locSpaces;

    // if dataService call successful
    if (addUpdateLocation(PSID, x)) {
// change icon if needed
if (oldIcon != locIcon) {
    for (var z = 0; z < parkingLocationMarkers.length; z++) {
        if (parkingLocationMarkers[z].getTitle() == locId) {
            parkingLocationMarkers[z].setIcon('img/' + locIcon);
            break;
        }
    }
}
return true; // success
}
return false; // failure
});

// Park buttons lead here
function handlePark() {
    // shows mobiscroll to allow timer set
    $('#timeScroller').mobiscroll('show');
}

// shows dialog using flip animation
// param: formId - DOM id of form; allows for more than 1 if needed
// param: headerText - display text for title of form
function flipToDialog(formId, headerText) {
    formId = '#' + formId;
    $.mobile.changePage(formId, { transition: 'flip' });
    $(formId + ' h1').html(headerText);
}

// resets the form to default state
function clearFormFields() {
    toggleOffAllRadios();
}
$('#locationName').val('');
$('#locationSize').val(1).slider('refresh');
$('#locationComments').val('');
$('#locationId').val('');
$('#locationLatitude').val('');
$('#locationLongitude').val('');

// fills field values for form
// param: formId - DOM id of form
// param: headerText - display text for title of form
// param: disableVal - if true cannot edit fields
function setFormFieldValues(formId, headerText, disableVal) {
    // have to flip to dialog before setting or will error
    flipToDialog(formId, headerText);
    formId = $('formId').val();
    toggleOffAllRadios();
    var x = getParkingLocationIndexById(lastSelectedParkingLocationId);

    // selects appropriate radio button
    for (var i = 1; i <= $(formId + ' input[type="radio"]').length; i++) {
        if (LocalParkingLocations_Type[x] ==
            $('#locationType' + i).parent().text().trim()) {
            $('#locationType' + i).prop('checked',
                true).checkboxradio('refresh');
            break;
        }
    }

    // prefill fields from cached data
    $('#locationId').val(LocalParkingLocations_Id[x]);
    $('#locationLatitude').val(LocalParkingLocations_Latitude[x]);
    $('#locationLongitude').val(LocalParkingLocations_Longitude[x]);
    $('#locationName').val(LocalParkingLocations_Name[x]);
    $('#locationSize').val(LocalParkingLocations_NumSpaces[x]).slider('refresh');
```javascript
$("#locationComments").val(LocalParkingLocations_Comments[x]);

// disable / enable fields
disableFormFields(LocalParkingLocations_Id[x], disableVal);
}

// sets fields to disable or enable
// param: locId - cached index of parking location
// param: disabled - if true then cannot edit
function disableFormFields(locId, disabled) {
    var status = 'enable';
    if (disabled) {
        status = 'disable';
        // only disabled on show info, so get cars parked at location
        var numParked = fetchNumCarsAtParkingLocation(locId);
        var str = numParked + ' user';
        if (numParked != 1) {
            str += 's';
        }
        str += ' currently parked here.';

        $('#addFormDialog input[type="text"], #addFormDialog textarea').attr('readonly','readonly');
        $('#formSubmit').closest('.ui-btn').hide();
        if (isCarParked) {
            $('#formPark').closest('.ui-btn').hide();
        } else {
            $('#formPark').closest('.ui-btn').show();
        }
        $('#numParkedUsers').show();
        $('numParkedUsers').html(str);
    } else {
        $('#addFormDialog input[type="text"], #addFormDialog textarea').removeAttr('readonly');
        $('#formSubmit').closest('.ui-btn').show();
        if (isCarParked) {
            $('#formPark').closest('.ui-btn').hide();
        } else {
            $('#formPark').closest('.ui-btn').show();
        }
        $('#numParkedUsers').hide();
    }
```
// radios and slider use "enable" and "disable" as param
$('#addFormDialog input[type="radio"]').checkboxradio(status);
$('#locationSize').slider(status);
}

// builds html for form radio buttons after DataService call
function initFormRadioButtons() {
  fetchParkingLocationTypes();
  for (var x = 0; x < ParkingLocationTypes.length; x++) {
    var id = x + 1;
    $('#locationTypeControlGroup').append(
      '<input type="radio" name="locationType' + id +"
      id="locationType' + id +"
      value="locationType' + id +"
      mini="true" />
      <label for="locationType' + id +'">' + ParkingLocationTypes[x] + '</label>'
    );
  }
  $('#locationTypeControlGroup').trigger('create');

  // sets click event listener for all radio buttons
  $('input[type="radio"]').livequery('click', function() {
    toggleOffAllRadios();
    $(this).prop('checked', true).checkboxradio('refresh');
  });
}

// deselect radios - this is needed otherwise can't uncheck
function toggleOffAllRadios() {
  $('input[type="radio"]').each(function() {
    $(this).prop('checked', false).checkboxradio('refresh');
  });
}

// param: text - display text for Park button
function setParkButtonText(text) {

$("#btnPark .ui-btn-text").text(text);
}

// called on page load, fires initialization for app
// REQUIRED BEFORE INITIALIZE
function preInit() {
  PSID = getSetCookie(); // find existing or create new id for user
  userIP = getIP(); // dataService call to get IP address for logs

  initMobiscroll();
  // trigger GPS, try to get new position every 5 seconds as needed
  navigator.geolocation.getCurrentPosition(locSuccess, locError,
  {maximumAge:5000});
}

// initialize app
// param: lat - user latitude to center map
// param: lon - user longitude to center map
function initialize(lat, lon) {
  geocoder = new google.maps.Geocoder();
  centerLatLng = new google.maps.LatLng(lat, lon);
  // run through init methods IN ORDER
  mapInit(centerLatLng); // init map with center
  infoBoxInit();
  initButtons();
  initFormRadioButtons();

  // creates LongClick event handler
  // has to be set after map init
  new LongClick(map, 500); // hold 1/2 second

  // set user location marker
  marker = new google.maps.Marker({
    icon: 'img/blu-star.png',
    position: centerLatLng,
    map: map
  });
// allow tempMarker to be dragged; have to set after init
tempMarker.setDraggable(true);

// call to listeners
setEventListener();

// fires click event on user marker to eliminate an
// error wherein the first click on map would not
// register properly in some browsers
google.maps.event.trigger(marker, 'click');

// return true if toolbars are hidden;
function isToolbarHidden() {
    var classes = $('[data-position="fixed"]').attr('class');
    return (classes.indexOf('ui-fixed-hidden') >= 0);
}

// will hide toolbars if shown
function hideToolbarsIfShown() {
    if (!isToolbarHidden()) {
        $('[data-position="fixed"]').fixedtoolbar('hide');
    }
}

// will show toolbars if hidden
function showToolbarsIfHidden() {
    if (isToolbarHidden()) {
        $('[data-position="fixed"]').fixedtoolbar('show');
    }
}

// called in pre-init if unable to determine location
function locError(error) {
    // initialize map with a static predefined latitude, longitude for UC
    initialize(39.131083, -84.517766);
function locSuccess(position) {
    initialize(position.coords.latitude, position.coords.longitude);
}

function placeMarker(latlng) {
    lastSelectedParkingLocationId = 0; // no location selected
    updateInfoBoxGeoCode(infobox, latlng); // get geocode info for infobox
    tempMarker.setMap(map);
    tempMarker.setPosition(latlng); // place marker
    openInfoboxOnLocation(tempMarker, latlng); // show infobox for marker
}

function updateInfoBoxGeoCode(window, latLng) {
    // use cached info if able to reduce geocode calls
    for (var x = 0; x < PreviouslyGeocodedLocations_LatLng.length; x++) {
      if (latLng == PreviouslyGeocodedLocations_LatLng[x]) {
        setInfoBoxText(PreviouslyGeocodedLocations_Text[x]);
        return;
      }
    }
    // no cached info found, must get new info
    geocodeLatLng(window, latLng, function(geocodeText) {
      setInfoBoxText(geocodeText);
    });
}

// calls geocoding for infobox
// param: window - infobox window specified
// param: latlng - google maps latlng for geocoded position
// param: callback - anon func to call AFTER receiving async data
// this prevents old text from showing before call completes
function geocodeLatLng(window, latlng, callback) {
    geocoder.geocode({'latLng': latlng}, function(results, status) {
        if (status == google.maps.GeocoderStatus.OK) {
            var result = results[0];
            if (result) { // success
                // text will be number and street name
                var text = result.address_components[0].long_name + ' ' + result.address_components[1].short_name;
                //add to cached data
                PreviouslyGeocodedLocations_LatLng.push(latlng);
                PreviouslyGeocodedLocations_Text.push(text);
                callback(text);
            } else {
                callback('No results found');
            }
        } else {
            callback('Geocoder failed due to: ' + status);
        }
    });
}

// sets up long click event on map
// param: map - specified map to add listener
// param: length - how long user must hold before registered
// used to differentiate between normal click on map
function LongClick(map, length) {
    this.length_ = length;
    var me = this;
    me.map_ = map;
    google.maps.event.addListener(map, 'mousedown', function(e) {
        me.onMouseDown_(e);
    });

google.maps.event.addListener(map, 'mouseup', function(e)
{
    me.onMouseUp_(e);
});

// when click released, determines if target
// length of time has been met to fire action
LongClick.prototype.onMouseUp_ = function(e) {
    var now = +new Date;
    var curLength = now - this.down_;
    if (curLength > this.length_) {
        google.maps.event.trigger(this.map_, 'longpress', e);
    }
}

// sets time on click to determine the start of the click
LongClick.prototype.onMouseDown_ = function() {
    this.down_ = +new Date;
}

// setting new infobox text can be tedious; method varies
// param: str - text to display
function setInfoBoxText(str) {
    $('#infobox').expire(); // removes old livequery on infobox
    // if infobox already exists on page, normal dom
    if (document.getElementById('infobox')) {
        $('#infobox').html(str);
    } else { // for brief time between disappear and reappear
        // use livequery function to fire when reappear
        $('#infobox').livequery(function() {
            $(this).html(str);
        });
    }
}

// attaches infobox to map marker
// param: marker - existing map marker
// param: latlng - google maps latlng coords
function openInfoboxOnLocation(marker, latlng) {
    if (infobox.map == null) { // infobox off map
        infobox.open(map, marker);
    } else { // on map just not visible
        infobox.setPosition(latlng);
        infobox.setVisible(true);
    }
}

// initialize infobox
function infoBoxInit() {
    infobox = new InfoBox({
        map: null, // starts off map
        // tied to hiddentext element
        content: document.getElementById('infobox'),
        disableAutoPan: false,
        maxWidth: 150,
        pixelOffset: new google.maps.Size(-140, 0),
        zIndex: null,
        boxStyle: {
            background: "js/images/tipbox.gif' no-repeat",
            opacity: 0.85,
            width: '250px'
        },
        closeBoxMargin: '12px 4px 2px 2px',
        closeBoxURL: 'js/images/close.gif',
        infoBoxClearance: new google.maps.Size(1, 1)
    });
}

// init map size to map is not affected by header/footer toolbars
function mapSizeInit() {
    var winSize = window.innerHeight;
    var headSize = document.getElementById('header').offsetHeight;
    var footSize = document.getElementById('footer').offsetHeight;
    var mapSize = winSize - headSize - footSize - 200; // 200 = header + footer + margin
    // adjust map size
    map.setMapSize(new google.maps.Size(mapSize, mapSize));
    // adjust map size
    mapSize = winSize - headSize - footSize - 200; // 200 = header + footer + margin
    // adjust map size
    map.setMapSize(new google.maps.Size(mapSize, mapSize));
}
var newSize = winSize - headSize - footSize;
document.getElementById('map_canvas').style.height = winSize + 'px';
}

// initialize map
// param: centerLatLng - user's initial location
function mapInit(centerLatLng) {
    var mapOptions = {
        zoom: DEFAULT_ZOOM,
        center: centerLatLng,
        mapTypeId: google.maps.MapTypeId.ROADMAP
    }
    // set map to html div element
    map = new google.maps.Map(document.getElementById('map_canvas'),
        mapOptions);
}

// event handler for when map is moved or zoomed
google.maps.event.addListener(map, 'bounds_changed', function() {
    handleLocationChange(map);
    google.maps.event.clearListeners(map, 'bounds_changed');
});

// displays other parked cars on map if not at a parking location
// param: lat - map center latitude
// param: lon - map center longitude
// param: radius - desired radius to get info
function handleLocalParkedCarLocations(lat, lon, radius) {
    // dataService call loads data to cached arrays
    fetchLocalParkedCarLocationsByRadius(PSID, lat, lon, radius);
    // cycle through results to add to map
    for (var x = 0; x < OtherLocalParkedCars_Latitude.length; x++) {
        var latlng = new google.maps.LatLng(
            OtherLocalParkedCars_Latitude[x],
            OtherLocalParkedCars_Longitude[x]);
        var isNew = true;
for (var i = 0; i < otherCarLocationMarkers.length; i++) {
    // will be equal if existing on map
    if (latlng == otherCarLocationMarkers[i].getPosition()) {
        isNew = false;
        break;
    }
}

// if it doesn't exist on map, add marker
if (isNew) {
    var otherCarLocationMarker = new google.maps.Marker({
        map: map,
        position: latlng,
        icon: 'img/car-other.png',
        clickable: false
    });
    otherCarLocationMarkers.push(otherCarLocationMarker);
}
}

// displays parking locations on map
// param: lat - map center latitude
// param: lon - map center longitude
// param: radius - desired radius to get info
function handleLocalParkingLocations(lat, lon, radius) {
    // dataService call loads data to cached arrays
    fetchLocalParkingLocationsByRadius(lat, lon, radius);

    // gets list of location ids where markers already exist
    // so we don't duplicate when adding to map
    var existingIds = [];
    for (var z = 0; z < parkingLocationMarkers.length; z++) {
        existingIds.push(parkingLocationMarkers[z].getTitle());
    }

    // cycles through all locations, add to map if necessary
for (var x = 0; x < LocalParkingLocations_Id.length; x++) {
    var isNew = true;
    for (var i = 0; i < existingIds.length; i++) {
        // when a match is found, since ids are unique, remove
        // id from array for increased performance
        if (LocalParkingLocations_Id[x] == existingIds[i]) {
            isNew = false;
            existingIds.splice(i, 1);
            break;
        }
    }
    if (isNew) {
        // need to add to map
        setupNewParkingLocationMarker(x, new google.maps.LatLng(
            LocalParkingLocations_Latitude[x],
            LocalParkingLocations_Longitude[x]));
    }
}

// create a parking location marker
// param: x - index of location info in cached arrays
// param: latlng - google maps latlng for marker position
function setupNewParkingLocationMarker(x, latlng) {
    if (!isNaN(x) && latlng) { // if valid
        var parkingLocationMarker = new google.maps.Marker({
            map: map,
            position: latlng,
            icon: 'img/' + LocalParkingLocations_Icon[x],
            // set title to location id so we can search on it
            title: LocalParkingLocations_Id[x],
            infoWindow: new google.maps.InfoWindow({
                content: LocalParkingLocations_Id[x],
                boxClass: 'parkingLocationInfoWindow'
            })
        });
    }
}
// add listeners
setParkingLocationMarkerListeners(parkingLocationMarker);
// add to marker array
parkingLocationMarkers.push(parkingLocationMarker);
Appendix B - dataService.js

```javascript
var PAGE_PARK = 'parkUser';
var FUNCTION_PARK = 'PARK';
var FUNCTION_UNPARK = 'UNPARK';
var FUNCTION_GET_PARKED_LOCATION_INFO = 'INFO';
var FUNCTION_GET_OTHER_PARKED_CARS_WITHIN_RADIUS = 'OTHERS';
var FUNCTION_GET_USERS_AT_LOCATION = 'CARCOUNT';

var PAGE_LOCATIONS = 'locations';
var FUNCTION_GET_LOCATIONS_WITHIN_RADIUS = 'LOCQUERY';
var FUNCTION_GET_LOCATION_TYPES = 'TYPES';
var FUNCTION_ADD_UPDATE_LOCATION = 'ADDUPDATE';
var FUNCTION_DELETE_LOCATION = 'DELETE';

// currently parked info
var ParkedCarInfoLatitude = 0;
var ParkedCarInfoLongitude = 0;
var ParkedCarInfoParkTime = 0;
var ParkedCarInfoMeterLength = '';
var ParkedCarInfoParkingLocation = 0;
var isCarParked = false;

// other cars cached data
var OtherLocalParkedCars_Latitude = [];
var OtherLocalParkedCars_Longitude = [];

// parking location type cached data
var ParkingLocationTypes = [];
var ParkingLocationIcons = [];

// parking locations cached data
var LocalParkingLocations_Id = [];
var LocalParkingLocations_Latitude = [];
var LocalParkingLocations_Longitude = [];
var LocalParkingLocations_Name = [];
var LocalParkingLocations_Type = [];
```
var LocalParkingLocations_Icon = [];
var LocalParkingLocations_Comments = [];
var LocalParkingLocations_NumSpaces = [];

var lastParkUserCallTime = 0;

// returns the ip address of the user
function getIP() {
    var urlStr = "ipfinder.php?callback=getip";
    var response = executeRequest(urlStr);

    return response.replace(/"/g, '');
}

// used to execute http request to send/receive data to Web services
// returns raw response text
function executeRequest(urlStr) {
    if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
    } else {
        xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
    }

    xmlhttp.open("GET", urlStr, false);
    xmlhttp.send();

    return xmlhttp.responseText
}

// sets the parked car info for user
// param: psid - unique user id
// param: lat - latitude position
// param: lon - longitude position
// param: meterLength - length of meter time; format HHMM
// param: locationId - id of parking location used, if there is one
// return boolean for success / failure
function parkUser(psid, lat, lon, meterLength, locationId) {

// due to an issue with mobiscroll, this method gets called
twice in a row; inserted a 5-second gap to ignore duplicate calls

var newParkUserCallTime = new Date().getTime();
if (newParkUserCallTime - lastParkUserCallTime > 5000) {
  lastParkUserCallTime = newParkUserCallTime;
  if (!locationId) {
    // not at a parking location
    locationId = 0;
  }
}

var urlStr = PAGE_PARK + ".php?"
  + "function=" + FUNCTION_PARK
  + ";id=" + psid
  + ";latitude=" + lat
  + ";longitude=" + lon
  + ";meterlength=" + meterLength
  + ";locid=" + locationId
;

var response = executeRequest(urlStr);
if (response != null && response != '') {
  alert(response);
  return false; // failure
}

// store park info
ParkedCarInfoLatitude = lat;
ParkedCarInfoLongitude = lon;
// round time to nearest second
ParkedCarInfoParkTime = Math.floor(newParkUserCallTime / 1000);
ParkedCarInfoMeterLength = "+ meterLength;
ParkedCarInfoParkingLocation = locationId;
isCarParked = true;
return true; // success
}
return false; // failure
// removes parked car info for user
// param: psid - unique user id
function unparkUser(psid) {
    var urlStr = PAGE_PARK + ".php?"
        + "function=" + FUNCTION_UNPARK
        + "+id=" + psid

    var response = executeRequest(urlStr);
    if (response != null && response != '') {
        alert(response);
        return false; // failure
    }
    isCarParked = false;
}

// find parked car info for user if they are parked
// param: psid - unique user id
// return boolean for success / failure
function getParkedCarInfoIfExists(psid) {
    var urlStr = PAGE_PARK + ".php?"
        + "function=" + FUNCTION_GET_PARKED_LOCATION_INFO
        + "+id=" + psid

    var response = executeRequest(urlStr);
    if (response != null && response != '') {
        var rArray = response.split('|');
        // set current parking info
        ParkedCarInfoLatitudem = rArray[0];
        ParkedCarInfoLongitude = rArray[1];
        ParkedCarInfoParkTime = Math.floor(new Date(rArray[2]).getTime() / 1000);
        ParkedCarInfoMeterLength = '' + rArray[3];
        ParkedCarInfoParkingLocation = rArray[4];
        isCarParked = true;
    }
return true; // success
}

isCarParked = false;
return false; // failure
}

// gets the number of users currently parked at a location
// param: locationId - id of parking location
// return a number
function fetchNumCarsAtParkingLocation(locationId) {
    var urlStr = PAGE_PARK + ".php?"
        + "function=\" + FUNCTION_GET_USERS_AT_LOCATION
        + "&locid=\" + locationId
    ;

    var response = executeRequest(urlStr);
    if (response != null && response != '') {
        return response; // success
    }

    return false; // failure
}

// gets other parked car locations in the area
// param: psid - unique user id
// param: lat - latitude to search by
// param: lon - longitude to search by
// return result from fetch by radius of default miles
function fetchLocalParkedCarLocations(psid, lat, lon) {
    return fetchLocalParkedCarLocationsByRadius(psid, lat, lon, 3);
}

// gets other parked car locations in the area
// param: psid - unique user id
// param: lat - latitude to search by
// param: lon - longitude to search by
// param: radius - radius to search by
// return boolean for success / failure
function fetchLocalParkedCarLocationsByRadius(psid, lat, lon, radius) {
    if (psid && lat && lon && radius) {
        // clear cached data
        OtherLocalParkedCars_Latitude = [];
        OtherLocalParkedCars_Longitude = [];

        var urlStr = PAGE_PARK + ".php?"
            + "function=" +
            FUNCTION_GET_OTHER_PARKED_CARS_WITHIN_RADIUS
            + ";id=" + psid
            + ";latitude=" + lat
            + ";longitude=" + lon
            + ";radius=" + radius
        ;

        var response = executeRequest(urlStr);
        if (response != null && response != '') {
            var rArray = response.split('|');
            var i = 1;
            while (i < rArray.length) {
                OtherLocalParkedCars_Latitude.push(rArray[i++]);
                OtherLocalParkedCars_Longitude.push(rArray[i++]);
                i++; // extra comma moves to next row
            }
            return true; // success
        }
    }
    return false; // failure
}

// get local parking locations
// param: lat - latitude to search by
// param: lon - longitude to search by
// return result from fetch by radius of default miles
function fetchLocalParkingLocations(lat, lon) {
    return fetchLocalParkedCarLocationsByRadius(lat, lon, 3); // default 3 miles
function fetchLocalParkingLocationsByRadius(lat, lon, radius) {
    if (lat && lon && radius) {
        var urlStr = PAGE_LOCATIONS + ".php?"
            + "function=" +
            FUNCTION_GET_LOCATIONS_WITHIN_RADIUS
                + "&latitude=" + lat
                + "&longitude=" + lon
                + "&radius=" + radius
            ;

        var response = executeRequest(urlStr);
        if (response != null && response != '') {
            var rArray = response.split('|');
            var i = 1;
            while (i < rArray.length) {
                var curid = rArray[i++];
                var newLocation = true;
                for (var x = 0; x < LocalParkingLocations_Id.length; x++) {
                    if (curid == LocalParkingLocations_Id[x]) {
                        newLocation = false;
                        break;
                    }
                }

                if (newLocation) {
                    // set cached data for new location
                    LocalParkingLocations_Id.push(curid);
                    LocalParkingLocations_Latitude.push(rArray[i++]);
                }
            }
        }
    }
}
LocalParkingLocations_Longitude.push(rArray[i++]);
    if (rArray[i] != 0) {

LocalParkingLocations_Name.push(rArray[i++]);
    } else { // name is blank
        i++;
        LocalParkingLocations_Name.push('');
    }
LocalParkingLocations_Type.push(rArray[i++]);
LocalParkingLocations_Icon.push(rArray[i++]);
    if (rArray[i] != 0) {

LocalParkingLocations_Comments.push(''+rArray[i++]);
    } else { // comments are blank
        i++;
        LocalParkingLocations_Comments.push('');
    }

LocalParkingLocations_NumSpaces.push(rArray[i++]);
    } else { // not a new location
        i += 7; // so skip over those entries
    }

    i++; // extra moves to next row
}

return true; // success
}

} // get parking location types and init arrays
function fetchParkingLocationTypes() {
    var urlStr = PAGE_LOCATIONS + ".php?" + "function=" + FUNCTION_GET_LOCATION_TYPES;

    }
```javascript
var response = executeRequest(urlStr);
if (response != null && response != '') {
    var rArray = response.split('|
');
    var i = 0;
    while (i < rArray.length) {
        ParkingLocationTypes.push(rArray[i++]);
        ParkingLocationIcons.push(rArray[i++]);
    }
    return null;
}

// either add or update location info
// param: psid - unique user id
// param: x - array index of cached data
// return boolean for success / failure
function addUpdateLocation(psid, x) {
    if (psid && !isNaN(x)) {
        var locid = 0;
        // get location id if there is one
        if (x < LocalParkingLocations_Id.length) {
            locid = LocalParkingLocations_Id[x];
        }

        // find location type
        var locTypeNum = null;
        for (var i = 0; i < ParkingLocationTypes.length; i++) {
            if (LocalParkingLocations_Type[x].trim() == ParkingLocationTypes[i].trim()) {
                locTypeNum = i + 1;
                break;
            }
        }

        var urlStr = PAGE_LOCATIONS + ".php?" + 
                        "function=" + FUNCTION_ADD_UPDATE_LOCATION
```

47

---
var response = executeRequest(urlStr);
if (response != null && (response.trim() == '' || isNaN(response.trim()))) {
    alert(response);
    return false; // failure
}

// if id is 0 (invalid) then it's a new location, so add it
if (locid == 0) {
    LocalParkingLocations_Id.push(response.trim());
}
return true; // success
}
return false; // failure

// delete location info
// param: psid - unique user id
// param: x - array index of cached data
// return boolean for success / failure
function softDeleteLocation(psid, x) {
    if (psid && !isNaN(x)) {
        if (x >= LocalParkingLocations_Id.length) {
            return false; // not within array bounds
        }
    }
var locid = LocalParkingLocations_Id[x];

var urlStr = PAGE_LOCATIONS + ".php?"
    + "function=" + FUNCTION_DELETE_LOCATION
    + "&psid=" + psid
    + "&locid=" + locid
;

var response = executeRequest(urlStr);
if (response == null || response.trim() == '') {
    return true; // success
}
alert(response);
}
return false; // failure
Appendix C - ipfinder.php

```php
<?php
    header('content-type: application/json; charset=utf-8');

    $data = json_encode($_SERVER['REMOTE_ADDR']);
    echo $data;
?>
```
Appendix D - locations.php

```php
<?php

$FUNCTION_GET_LOCATIONS_WITHIN_RADIUS = "LOCQUERY";
$FUNCTION_GET_LOCATION_TYPES = "TYPES";
$FUNCTION_ADD_UPDATE_LOCATION = 'ADDUPDATE';
$FUNCTION_DELETE_LOCATION = 'DELETE';

$con = mysqli_connect("ucfsb.ucfilespace.uc.edu:3306", "pollactd", "neocow45", "pollactd");
if (mysqli_connect_errno()) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

if (isset($_GET['function'])) {
    switch ($_GET['function']) {
        case $FUNCTION_GET_LOCATIONS_WITHIN_RADIUS:
            getLocationsWithinRadius($con);
            break;
        case $FUNCTION_GET_LOCATION_TYPES:
            getLocationTypes($con);
            break;
        case $FUNCTION_ADD_UPDATE_LOCATION:
            addUpdateLocation($con);
            break;
        case $FUNCTION_DELETE_LOCATION:
            softDeleteLocationById($con);
            break;
        default:
            echo "Hello.";
            break;
    }
} else {
    echo "Hello.";
}

mysqli_close();
```
function getLocationsWithinRadius($con) {
    if (isset($_GET['latitude'], $_GET['longitude'],
        $_GET['radius'])) {
        $latitude  = $_GET['latitude'];
        $longitude = $_GET['longitude'];
        $radius    = $_GET['radius'];

        $sql = "SELECT ((ACOS(SIN($latitude * PI() / 180) * 
            SIN(latitude * PI() / 180) +
            COS($latitude * PI() / 180) * COS(latitude * PI() /
            180) * COS((longitude - longitude) *
            PI() / 180)) * 180 / PI()) * 60 * 1.1515) AS
dataDistance,locationId,latitude,longitude,locationName,plt.description,plt.iconFile,comments,totalSpaces
        FROM ParkingLocations pl
        JOIN ParkingLocationTypes plt ON plt.plt_id =
        pl.locationType
        WHERE pl.softDelete <> 1
        HAVING distance<=$radius ORDER BY distance ASC";

        $result = mysqli_query($con, $sql);
        if (mysqli_num_rows($result) > 0) {
            while($resArray = mysqli_fetch_array($result,
                MYSQLI_NUM)) {
                echo implode('|', $resArray);
                echo '|' ;
            }
        }
    }
}

function getLocationTypes($con) {
    $sql = "SELECT description,iconFile FROM ParkingLocationTypes";

    $result = mysqli_query($con, $sql);
    $numRows = mysqli_num_rows($result);
for ($x = 0; $x < $numRows; $x++) {
    echo implode('|', mysqli_fetch_array($result, MYSQLI_NUM));
    if ($x < $numRows - 1) {
        echo '|';
    }
}

function addUpdateLocation($con) {
    if (isset($_GET['psid'], $_GET['locid'], $_GET['latitude'],
            $_GET['longitude'], $_GET['name'], $_GET['type'], $_GET['comments'],
            $_GET['spaces'])) {
        $locid = $_GET['locid'];
        $latitude = $_GET['latitude'];
        $longitude = $_GET['longitude'];
        $name = $_GET['name'];
        $type = $_GET['type'];
        $comments = $_GET['comments'];
        $spaces = $_GET['spaces'];

        $sql = null;
        if ($locid == 0) {
            $sql = "INSERT INTO ParkingLocations
            (latitude,longitude,locationName,locationType,comments,totalSpaces) VALUES
            ('$latitude','$longitude','$name',$type,'$comments',$spaces)
            ";
        } else {
            $sql = "UPDATE ParkingLocations
            SET
            latitude='$latitude',longitude='$longitude',locationName='$name',
            locationType=$type,comments='$comments',totalSpaces=$spaces
            WHERE locationId='$locid'
            ";
        }
    }
}
if (!mysqli_query($con,$sql)) {
    die('Error: ' . mysqli_error($con) . '.');
}

$sql = "SELECT locationId
    FROM ParkingLocations
    WHERE latitude='$latitude' AND longitude='$longitude'";
$result = mysqli_query($con, $sql);
if (mysqli_num_rows($result) > 0) {
    $resArray = mysqli_fetch_array($result, MYSQLI_NUM);
    echo implode('|', $resArray);
}

function softDeleteLocationById($con) {
    if (isset($_GET['psid'], $_GET['locid'])) {
        $locid = $_GET['locid'];

        $sql = "UPDATE ParkingLocations
            SET softDelete=1
            WHERE locationId='$locid'
        ";

        if (!mysqli_query($con,$sql)) {
            die('Error: ' . mysqli_error($con) . '.');
        }
    }
}
?>
<?php

$FUNCTION_PARK = "PARK";
$FUNCTION_UNPARK = "UNPARK";
$FUNCTION_GET_PARKED_LOCATION_INFO = "INFO";
$FUNCTION_GET_OTHER_PARKED_CARS_WITHIN_RADIUS = "OTHERS";
$FUNCTION_GET_USERS_AT_LOCATION = "CARCOUNT";

$con = mysqli_connect("ucfsb.ucfilespace.uc.edu:3306", "pollactd", "neocow45", "pollactd");
if (mysqli_connect_errno()) {
    echo "Failed to connect to MySQL: " . mysqli_connect_error();
}

if (isset($_GET['function'])) {
    switch ($_GET['function']) {
    case $FUNCTION_PARK:
        park($con);
        break;
    case $FUNCTION_UNPARK:
        unpark($con);
        break;
    case $FUNCTION_GET_PARKED_LOCATION_INFO:
        getParkingInfo($con);
        break;
    case $FUNCTION_GET_OTHER_PARKED_CARS_WITHIN_RADIUS:
        getParkedCarsWithinRadius($con);
        break;
    case $FUNCTION_GET_USERS_AT_LOCATION:
        getNumCarsAtLocation($con);
        break;
    default:
        echo "Hello.";
        break;
    }
} else {

}
```
echo "Hello."
}

mysqli_close();

function park($con) {
    if (isset($_GET['id'], $_GET['latitude'], $_GET['longitude'])) {
        $id = $_GET['id'];
        $lat = $_GET['latitude'];
        $lon = $_GET['longitude'];
        $meterLen = 'NULL';
        $parkingLocationId = 'NULL';

        if (isset($_GET['meterlength'])) {
            $meterLen = intval($_GET['meterlength']);
        }

        if (isset($_GET['locid'])) {
            $parkingLocationId = $_GET['locid'];
        }

        $sql = null;
        if (!isParked($con, $id)) {
            $sql = "INSERT INTO ParkedCars  
            (cookieId,latitude,longitude,meter_length,parkingLocationId)  
            VALUES ('$id','$lat','$lon', $meterLen,$parkingLocationId)";
        } else {
            $sql = "UPDATE ParkedCars  
            SET  
            latitude='$lat',longitude='$lon',park_time=CURRENT_TIMESTAMP,meter_length=$meterLen,parkingLocationId=$parkingLocationId  
            WHERE cookieId='$id'";
        }

        if (!mysqli_query($con,$sql)) {
            die('Error: ' . mysqli_error($con) . '.
```
function isParked($con, $id) {
    $sql = "SELECT 1 FROM ParkedCars WHERE cookieId='$id'";
    $result = mysqli_query($con, $sql);
    return mysqli_num_rows($result) > 0;
}

function getNumCarsAtLocation($con) {
    if (isset($_GET['locid'])) {
        $locId = $_GET['locid'];
        $sql = "SELECT 1 FROM ParkedCars WHERE parkingLocationId='$locId'";
        $result = mysqli_query($con, $sql);
        echo mysqli_num_rows($result);
    }
}

function getParkingInfo($con) {
    if (isset($_GET['id'])) {
        $id = $_GET['id'];
        $sql = "SELECT latitude,longitude,park_time,meter_length,parkingLocationId FROM ParkedCars WHERE cookieId='$id'";
        $result = mysqli_query($con, $sql);
        if (mysqli_num_rows($result) > 0) {
            $resArray = mysqli_fetch_array($result, MYSQLI_NUM);
            echo implode('|', $resArray);
        }
    }
}

function unpark($con) {
    if (isset($_GET['id'])) {
        $id = $_GET['id'];
    }
$sql = "DELETE FROM ParkedCars WHERE cookieId='$id'";
if (!mysqli_query($con,$sql)) {
    die('Error: ' . mysqli_error($con) . '.');
}
}

function getParkedCarsWithinRadius($con) {
    if (isset($_GET['id'], $_GET['latitude'], $_GET['longitude'],
$_GET['radius'])) {
        $id = $_GET['id'];
        $latitude = $_GET['latitude'];
        $longitude = $_GET['longitude'];
        $radius = $_GET['radius'];

        $sql = "SELECT ((ACOS(SIN($latitude * PI() / 180) * 
SIN(latitude * PI() / 180) +
        COS($latitude * PI() / 180) * COS(latitude * PI() / 
180) * COS($longitude - longitude) *
        PI() / 180)) * 180 / PI()) * 60 * 1.1515) AS
        distance,
        latitude,longitud
        FROM ParkedCars
        WHERE cookieId <> '$id' AND parkingLocationId < 1
        HAVING distance<=$radius ORDER BY distance ASC";

        $result = mysqli_query($con, $sql);
        if (mysqli_num_rows($result) > 0) {
            while($resArray = mysqli_fetch_array($result, MYSQLI_NUM)) {
                echo implode('|', $resArray);
                echo '|';
            }
        }
    }
}
Appendix F - jscookie.js

var COOKIE_NAME = "park_social_id";

// adds leading zeros to a number to match a specified string length
// param: str - number to add zeros to
// param: num - target length of string
// return number with leading zeros of string length num
function setLeadingZeros(str, num) {
    while (str.length < num) {
        str = "0" + str;
    }
    return str;
}

// gets the cookie on the user's device, if one exists
// otherwise, creates a new cookie and stores it
// return psid - unique user id - from cookie
function getSetCookie() {
    var psid = getCookie(COOKIE_NAME);
    if (psid == null || psid == '') {
        psid = getNewPsid();
        setCookie(COOKIE_NAME, psid, 365);
    }

    return psid;
}

// create a new id by using the current datetime followed
// by a very large random number
function getNewPsid() {
    var time = "" + new Date().getTime();
    var randomness = "" + Math.floor(Math.random() * 999999999);

    return time + setLeadingZeros(randomness, 9);
}
// look for existing cookie
// param: c_name - cookie name argument
// return psid if cookie exists
function getCookie(c_name) { // from w3schools
    var i, x, y, ARRcookies=document.cookie.split(';');
    for (i=0; i<ARRcookies.length; i++) {
        x=ARRcookies[i].substr(0, ARRcookies[i].indexOf('='));
        y=ARRcookies[i].substr(ARRcookies[i].indexOf('=') + 1);
        x=x.replace(/\s+/g, "");
        if (x==c_name) {
            return unescape(y);
        }
    }
}

// set a new cookie
// param: c_name - cookie name argument
// param: value - psid for user
// param: exdays - amount of days until expiration
function setCookie(c_name, value, exdays) { // from w3schools
    var exdate=new Date();
    exdate.setDate(exdate.getDate() + exdays);
    var c_value=escape(value) + ((exdays==null) ? "" : ";");
    expires=""+exdate.toUTCString();
    document.cookie=c_name + "=" + c_value;
}
Appendix G - listeners.js

var listeners_previousZoomLevel;
var listeners_previousCenter;

// used to tell if we need to fetch new data because of map scroll
// param: map - google map
// return bool for whether the previous center of the map has been
// moved off the map
function prevCenterOffMap(map) {
    var prevLat = listeners_previousCenter.lat();
    var prevLon = listeners_previousCenter.lng();
    var newMapBounds = map.getBounds();
    var newNEcorner = newMapBounds.getNorthEast();
    var newSWcorner = newMapBounds.getSouthWest();

    var leftLat  = newSWcorner.lat();
    var rightLat = newNEcorner.lat();
    var upperLon = newNEcorner.lng();
    var lowerLon = newSWcorner.lng();

    return (prevLat < leftLat  || prevLat > rightLat
    || prevLon < lowerLon || prevLon > upperLon);
}

// determine radius of displayed map
// param: map - google map
// return radius in miles
function getMapRadius(map) {
    var center = map.getCenter();
    var centerLat = center.lat();
    var centerLon = center.lng();
    // using one corner point to be farthest from the center
    var NEpoint = map.getBounds().getNorthEast();
    var cornerLat = NEpoint.lat();
    var cornerLon = NEpoint.lng();
    var PI = Math.PI;
var radius =
  (Math.acos
   (Math.sin
    (centerLat * PI / 180)
    * Math.sin(cornerLat * PI / 180)
    + Math.cos(centerLat * PI / 180)
    * Math.cos(cornerLat * PI / 180)
    * Math.cos((centerLon - cornerLon) * PI / 180)
   ) * 180 / PI
  ) * 60 * 1.1515
;

return radius;
}

// pass location change parameters to other methods
// param: map - google map
function handleLocationChange(map) {
  var radius = getMapRadius(map);
  var center = map.getCenter();
  var lat = center.lat();
  var lon = center.lng();

  handleLocalParkedCarLocations(lat, lon, radius);
  handleLocalParkingLocations(lat, lon, radius);
}

// call groups of listener inits
function setEventListeners() {
  setMapListeners();
  setCurrentLocationMarkerListeners();
  setParkedLocationMarkerListeners();
  setTempLocationMarkerListeners();
}

// set event listeners for map
function setMapListeners() {
    listeners_previousZoomLevel = map.getZoom();
    listeners_previousCenter = map.getCenter();

    // if map zooms out, need to fetch more data for wider zoom
    google.maps.event.addListener(map, 'zoom_changed', function() {
        var newZoomLevel = map.getZoom();
        if (newZoomLevel < listeners_previousZoomLevel) {
            handleLocationChange(map);
        }
        listeners_previousZoomLevel = newZoomLevel;
    });

    google.maps.event.addListener(map, 'click', function() {
        //
    });

    // place temp marker and read location on long click
    google.maps.event.addListener(map, 'longpress', function(event) {
        placeMarker(event.latLng);
    });

    // clear listener on drag so it does not register as a long click
    google.maps.event.addListener(map, 'dragstart', function() {
        google.maps.event.clearListeners(map, 'longpress');
    });

    google.maps.event.addListener(map, 'dragend', function() {
        // add the long click listener back after dragging
        google.maps.event.addListener(map, 'longpress',
            function(event) {
                placeMarker(event.latLng);
            });

        // fetch new map data if map moved too far
        if (prevCenterOffMap(map)) {
            handleLocationChange(map);
        }
    });
listeners_previousCenter = map.getCenter();
}
});
}

function setCurrentLocationMarkerListeners() {
google.maps.event.addListener(marker, 'click', function() {
    // no parking location currently selected
    lastSelectedParkingLocationId = 0;

    // update infobox if needed
    if (infobox.getPosition() != this.getPosition()) {
        updateInfoBoxGeoCode(infobox, this.getPosition());
    }
    openInfoboxOnLocation(this, this.getPosition());

    // center map on location
    map.panTo(this.getPosition());
});
}

function setParkedLocationMarkerListeners() {
google.maps.event.addListener(carMarker, 'click', function() {
    // no parking location currently selected
    lastSelectedParkingLocationId = 0;

    // update infobox if needed
    if (infobox.getPosition() != this.getPosition()) {
        updateInfoBoxGeoCode(infobox, this.getPosition());
    }
    openInfoboxOnLocation(this, this.getPosition());

    // center map on location
    map.panTo(this.getPosition());
});
}
function setTempLocationMarkerListeners() {
    google.maps.event.addListener(tempMarker, 'click', function() {
        // update infobox if needed
        if (infobox.getPosition() != this.getPosition()) {
            updateInfoBoxGeoCode(infobox, this.getPosition());
        }
        openInfoboxOnLocation(this, this.getPosition());
    });

    google.maps.event.addListener(tempMarker, 'dragstart', function() {
        openInfoboxOnLocation(this, this.getPosition());
    });

    google.maps.event.addListener(tempMarker, 'drag', function() {
        infobox.setVisible(false);
    });

    google.maps.event.addListener(tempMarker, 'dragend', function() {
        updateInfoBoxGeoCode(infobox, this.getPosition());
        openInfoboxOnLocation(this, this.getPosition());
    });
}

function setParkingLocationMarkerListeners(parkingLocationMarker) {
    google.maps.event.addListener(parkingLocationMarker, 'click', function() {
        // clear custom marker
        tempMarker.setMap(null);

        // set selected parking location id
        lastSelectedParkingLocationId = parkingLocationMarker.getTitle();

        // update infobox if needed
        if (infobox.getPosition() != this.getPosition()) {
            updateInfoBoxGeoCode(infobox, this.getPosition());
        }
        openInfoboxOnLocation(this, this.getPosition());
    });
}
updateInfoBoxGeoCode(infobox, this.getPosition());
}
openInfoboxOnLocation(this, this.getPosition());
});
}
// adds leading zeros to a number to match a specified string length
// param: str - number to add zeros to
// param: num - target length of string
// return number with leading zeros of string length num
function setLeadingZeros(str, num) {
    while (str.length < num) {
        str = "0" + str;
    }
    return str;
}

// shows how much time is left on parking meter timer
function timer() {
    // have to recalculate each time the method is called
    // because waiting 1000 milliseconds wasn't always an exact
    // second,
    // causing the countdown to be off
    var hours = parseInt(ParkedCarInfoMeterLength.substring(0, 2));
    var minutes = parseInt(ParkedCarInfoMeterLength.substring(2));
    var endTime = ParkedCarInfoParkTime + (hours * 3600) + (minutes * 60);
    var parkTimeRemaining = endTime - Math.floor(new Date().getTime() / 1000);
    if (parkTimeRemaining > 0) {
        var hoursLeft = setLeadingZeros('' + Math.floor(parkTimeRemaining / 3600), 2);
        var minutesLeft = setLeadingZeros('' + Math.floor((parkTimeRemaining - (hoursLeft * 3600)) / 60), 2);
        var secondsLeft = setLeadingZeros('' + (parkTimeRemaining - hoursLeft * 3600 - minutesLeft * 60), 2);
        $('#meterTimer').html(hoursLeft + ':' + minutesLeft + ':' + secondsLeft);
    }
}
// start flashing if fewer than 5 minutes remain
if (parkTimeRemaining <= 300) {
    flashColor('#meterTimer', '#C00');
}

// stop timer at 0
if (parkTimeRemaining <= 0) {
    stopTimer(true);
}
}

// stop the timer
// param: shouldFlash - timer will flash red if true
function stopTimer(shouldFlash) {
    // stop counting down
    window.clearTimeout(timerVar);

    // set to 0 to resolve issue where incorrect numbers displayed
    $('#meterTimer').html('00:00:00');

    if (shouldFlash) { // start flashing
        flashTimerVar = setInterval(
            function() {
                flashColor('#meterTimer', '#C00')
            },
            1000
        );
    } else { // stop flashing
        window.clearTimeout(flashTimerVar);
    }
}

// make the background flash a color
// param: id - html id of an element
// param: hex - hex code for color flash should be
function flashColor(id, hex) {
    var container = $(id);
    var flashColor = function() {
        container.css('background-color', hex);
        container.css('color', hex);
    };
    flashColor();
    flashColor();
    flashColor();
if (container.length) { // if element exists
    // get original color to flash back
    var originalColor = container.css('backgroundColor');

    // start flashing
    container.animate({
        backgroundColor: hex
    },'normal','linear',function(){
        $(this).animate({
            backgroundColor:originalColor
        });
    });
}