UDK : 004.738.12 ; 004.438"JAVA" ; 004.439"HTML" ID:196828428 **Review Article**

# BUILD A MOBILE APP WITH HTML5 AND JAVASCRIPT

***Miljan Jeremić[1] , Zvonko Damjanović[2] , Đorđe Kostadinović[3], Dušan Jeremić[4]***
*[1] Knjževačka Gimnazija knjaževac,*
*[2] University of Belgrade, Technical Faculty in Bor*
*[3] Nis Petrol, Niš,*
*[4]Faculty of Information Technology – FIT*

***Abstract:*** *You don't need separate platform-specific technologies to develop powerful, useful apps. And you'd be surprised with what you can build with HTML5 and popular JavaScript libraries. The app will be called "Where You parked your car", and we'll build it with jQuery Mobile and Cordova (also known as PhoneGap). To create this application, I'll also use the new HTML5 API, specifically the local storage and the geolocation (actually not really part of the HTML5 specs), as well as the Google Maps API. A hybrid app is built using web technologies, and then wrapped in a platform-specific shell that allows it to be installed just like a native app.*

***Key words:*** *1. Web app.; 2.HTML 5; 3. Java Script, 4. Google Maps API, 5. JQuery Mobile*

## 1. INTRODUCTION

The aim of "Where You parked your car" is to let you bookmark where you parked your car on a map using the GPS function of your smartphone, and then—after you have finished your walk—find a route to return to it. This kind of application is useful if you don't have a good memory or if you're in a foreign city. In addition, "Where I parked my car" will also save a log of your saved positions (up to 50), so that you can retrieve a location for future use. Since scrolling a list of 50 items can be really annoying, the app will have a search filter so you can easily achieve the task. The list allows you to see the position on a map, and you can also to delete one or more old locations. Finally, I'll also create a small credits page within the app.

The Ingredients

To fully understand what's going to be developed, you must have at least a basic knowledge of the following languages and frameworks:

HTML: It will be used to create the markup of the pages. Where appropriate, I'll use some new HTML5 tags.

CSS: Since most of the CSS enhancements will be done by jQuery Mobile, it isn't really needed extensively. However, I'll use few rules to correct or adjust some styles as you'll see later.

JavaScript: The business logic is entirely written in JavaScript so if you don't know some key concepts like what's a callback or a closure, you'll need to study it before proceeding.

jQuery: I'll use jQuery mainly to select elements and attach event handlers, so there's nothing special here.

jQuery Mobile: It'll be used to automatically enhance the application interface. I'll use the page loading widget, the listview for the positions' log, and I'll put some buttons inside the header and the footer of the pages to to build a toolbar.

Cordova (PhoneGap): As said before, Cordova will be used to wrap the files so you can compile them as if you built a native app. Of course, I'll take advantage of some of the APIs offered by the framework, such as notifications, geolocation, and connection

The version of jQuery Mobile used will be 1.2.0, so if you need a refresher on what's new in this version, you can read What's New in jQuery Mobile 1.2.0? and Build Lists and Popups in Minutes Using jQuery Mobile. The Cordova version used will be the 2.0.0. I choose this rather than the newer versions, because in my tests it has performance better in conjunction with jQuery Mobile.

Apart from the above list, I'll use also these APIs:

Geolocation API: It provides location information for the device in the form of latitude and longitude. Since many operating systems already support it, Cordova will use the native interface rather than its own implementation. However, for the platforms that don't have this API, Cordova uses its implementation that adheres to the W3C specification. If you need an introduction to this API, you can read Using the HTML5 Geolocation API.

Google Maps API: The Google Maps API lets you integrate the Google Maps service with your website. The service is completely free, but you should subscribe to the APIs console website to obtain your own API key, and use it when you send a request to the service. Since this API will still work without the key, I'll use it without a unique API key.

Web Storage API: This provides access to the devices storage options. If you need a starting point, you can take a look at An Overview of the Web Storage API.

Remember that Cordova uses a different JavaScript file for every operating system, so if you'd like to compile the application on your own, you need to use a specific IDE-SDK bundle like Eclipse and the Android SDK, Visual Studio and the Windows Phone SDK, and so on, for every platform. This could be a real pain, so "Where I parked my car" will be developed with the assumption that the compilation will be done using the Adobe cloud service, called Adobe PhoneGap Build. It'll be the service itself that includes the correct Cordova library for each platform at compiling time. So, to follow this tutorial, you can use a simple text editor or the IDE that you prefer to use, for example NetBeans, without installing any other SDK. I'll release the final code on my bitbucket repository, and I'll include the Android JavaScript file.

The Project's Structure

The structure of the project will be quite straightforward. In fact, I'll create the HTML files, the Adobe PhoneGap Build configuration file, the standard splashscreen, and the app's icon in the root of the project. Then, I'll divide the other files into three categories (actually folders): CSS, images, and JS. To develop "Where I parked my car", in addition to the jQuery, jQuery Mobile, and Cordova files, I'll create the following documents:

index.html: This is the entry point and will include all the libraries' files used throughout the application. It'll also have the buttons to set and get the current position and to view the positions' log.
map.html: This is where you'll see the map that will show you your position, your car position, and the route to reach it.
positions.html: This page will show the positions' log as a list.
style.css: As I said before, I'll write few style changes that deviate from the jQuery Mobile standard style.
jquery.mobile.config.js: It'll contain a configuration for jQuery Mobile
functions.js: This file will contain the function to initialize the application and some other utility functions.
maps.js: This file will have the functions that use the Google Maps API to draw the map and the route to the car.
positions.js: This file will save and load the previous positions using the Local Storage API.
config.xml: This XML file will contain the metadata of the applications and will be used by the Adobe cloud service to store settings like the app version number and the package contents.
At this point, you might argue that since we're building a small app, I could use a single file that has all the pages combined. You're absolutely right, but I prefer to keep things separated for organization.
Setting Up the Project
First of all, create the project folder, for example "where-did-i-park" and create the three folders mentioned above: CSS, images, and JS. Now, you need to download the jQuery, jQuery Mobilem and Codova files. Place all the JavaScript files of these frameworks inside the "JS" folder, put the jQuery Mobile bundled images in the "images" folder, and finally put the jQuery Mobile CSS file inside the "CSS" folder. If you've done everything right, the project should have the following structure:

```
\---css
|   \---jquery.mobile-1.2.0.min.css
|   \---style.css
\---images
|   \---ajax-loader.gif
|   \---ajax-loader.png
|   \---icons-18-black.png
|   \---icons-18-white.png
|   \---icons-36-black.png
|   \---icons-36-white.png
\---js
|   \---cordova-2.0.0.js
|   \---jquery-1.8.3.min.js
|   \---jquery.mobile-1.2.0.min.js
```

As said before, a reference to the Google Maps API is required. Including the file that you need is very simple and consists of just adding the following line to your HTML.
<script src="http://maps.google.com/maps/api/js?sensor=true"></script>

Please note that this line contains a sensor parameter. It must be included in the URL and set to true if you're using a sensor (like a GPS), or false otherwise. Since we're using the Maps API to create a mobile app, we'll set it to true.

In this first and introductory article, I showed the features of "Where I parked my car" and outlined an overview of the technologies that will be used for its implementation. I have structured the files that comprise the application in an organized fashion so that we can move straight toward developing.

## 2. THE HOMEPAGE

In this part we'll show you the source of all the HTML pages that make up the application, highlighting the key points of each one. We'll also explain the short stylesheet that's included. Unless you build explicit custom configurations, jQuery Mobile will load all of the pages using AJAX so, all the files needed by "Where You parked your car", such as the CSS and the JavaScript files, have to be loaded by the entry point, that is index.html.

The index page will show the title in the header and the credits (actually my name) in the footer, with a button to see additional details about the author (again, me). As explained in the first part, the index has three buttons: one to see and save the current position, one to see your current position (and to view the route to the car), one to look at the previously saved positions. Having said that, the code of the homepage will result in the following source.

```
<!DOCTYPE html>
<html>
  <head>
     <meta charset="UTF-8" />
     <meta name="viewport" content="width=device-width,
initial-scale=1">
     <title>Where I Parked my Car</title>
     <link rel="stylesheet" href="css/jquery.mobile-
1.2.0.min.css" type="text/css" media="all" />
     <link rel="stylesheet" href="css/style.css"
type="text/css" media="all" />
     <script src="js/jquery-1.8.3.min.js"></script>
     <script src="js/jquery.mobile.config.js"></script>
     <script src="js/jquery.mobile-1.2.0.min.js"></script>
     <script src="js/cordova-2.0.0.js"></script>
     <script
src="http://maps.google.com/maps/api/js?sensor=true">
</script>
```

```
    <script src="js/functions.js"></script>
    <script src="js/maps.js"></script>
    <script src="js/positions.js"></script>
    <script>
       $(document).one('deviceready', initApplication);
    </script>
  </head>
  <body>
    <div id="welcome-page" data-role="page">
       <header data-role="header">
          <h1>Where I parked my car</h1>
       </header>
       <div id="content" data-role="content">
          <p>
             Where You parked your car lets you bookmark
where you parked your car on a map
             and then find a route when you want to return
to it. The app will also
             save a log of your saved positions (up to 50).
          </p>
          <a href="map.html?requestType=set" id="set-car-
position" data-role="button">Set position</a>
          <a href="map.html?requestType=get" id="find-car"
data-role="button">Find car</a>
          <a href="positions.html" id="positions-history"
data-role="button">Positions history</a>
       </div>
       <footer data-role="footer">

       </footer>
    </div>
  </body>
</html>
```

You might note that, for the header and the footer of the application, I've used the new HTML5 <header> and <footer> tags instead of the generic . Moreover, all the buttons inside the <header> and the <footer> tags, use the attribute data-iconpos="notext" that tells to jQuery Mobile to hide the text of the link. This attribute is very useful for small screens. However, as you'll see later, I'll attach an handler to the pagebeforecreate and the orientationchange events, so programmatically I'll test for the size of the screen and, if a larger screen is found, the attribute will be removed and the text will be shown in full.

The Map Page

The map page (map.html) is probably the most important page of the application and, quite surprisingly, it's also the simplest one. In fact, apart from the header containing the app's title, it'll have a single empty  where the map will be shown. (The map will be generated on-the-fly by the Google Maps API so you won't have to worry about the markup.)

In conclusion, the full code of the map page is the following:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8" />
      <meta name="viewport" content="width=device-width,
initial-scale=1">
      <title>Where I parked my car</title>
   </head>
   <body>
      <div id="map-page" data-role="page">
         <header data-role="header">
            <a href="#" data-icon="back" data-rel="back"
data-iconpos="notext" title="Go back">Back</a>
            <h1>Where I parked my car</h1>
         </header>
         <div id="content" data-role="content">
            <div id="map">
            </div>
         </div>
      </div>
   </body>
</html>
```

In the <head> section of this page, I haven't put in the CSS and the JavaDcript files. I'll rely on the standard AJAX mechanism of jQuery Mobile, so there's no need to add them into the pages manually other than the entry point of "Where You parked your car". As you may have noticed, in the <header> of the page, I put a "go back" button. There are two reasons for this, one less obvious than the other. The first and more straightforward is to enhance the usability. Regarding the second reason, remember that an app developed using jQuery Mobile and Cordova can be deployed on many operating systems. However, while the Cordova APIs work in almost the same way across all platforms, there are some substantial differences among physical devices. For example, the Android devices have a physical "go back" button, so if you want to go to the previous page, you can use that. Conversely, the iPhone does not have a physical "go back" button (other than the "home screen" button, which would exit the app entirely), so if you don't build one into your interface, your iPhone users will be highly penalized.

Position History

The position history page (positions.html) will show you the list of the previously-saved positions, giving you the ability to search for a specific location and even delete one or more items. The page has an empty list that will be filled programmatically, as you'll see later. The list has two key attributes to analyze that are data-filter="true" to notify jQuery Mobile that you want a search filter above the list and data-split-icon="delete" to set the icon on the right of the text shown in the list item (the address). The latter is used in the split button lists, a type of list where each item has two links that (should) execute different actions: a main action activated when the user clicks on the list item text (the first link text), and a secondary one activated through the button that replaces the secondary link text, positioned after the item text.
The complete source of positions.html is the following:

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="UTF-8" />
      <meta name="viewport" content="width=device-width,
initial-scale=1">
      <title>Positions' history</title>
   </head>
   <body>
      <div id="positions-page" data-role="page">
         <header data-role="header">
            <a href="#" data-icon="back" data-rel="back"
data-iconpos="notext" title="Go back">Back</a>
            <h1>Positions' history</h1>
         </header>
         <div id="content" data-role="content">
            <ul id="positions-list" data-role="listview"
data-inset="true" data-split-icon="delete" data-
filter="true">
            </ul>
         </div>
         <footer data-role="footer">
            <h3>Created by Aurelio De Rosa</h3>
            <a href="aurelio.html" data-icon="info" data-
iconpos="notext" class="ui-btn-right">Credits</a>
         </footer>
      </div>
   </body>
</html>
```

Style Tuning

jQuery Mobile does a very good job enhancing the pages' interface, nonetheless there's always room for some style tuning. By default, even if you don't have buttons within the header or the footer of a page, the framework still reserves some space on both side of the elements. So, if the text inside the header or the footer—usually a title as you've seen in the sources listed—exceeds this space, jQuery Mobile cuts the text and appends an ellipsis. However, this happens to other elements as well, and I don't always want it to happen. So, in the very short stylesheet that I mentioned in the previous part called style.css, I changed it by applying the rule white-space: normal !important;. Another relevant point to discuss involves the  having the id of map. As said before, it'll be filled programmatically by the Google Maps API, so at the beginning it doesn't have its own height (it's just an empty ). This means that if you don't set a height, you won't see the map, even if the Google Maps API is working. I set the height to 600px, but you can change it to fit your taste.

The source of the discussed stylesheet is listed below:

```
.ui-header .ui-title,
.ui-footer .ui-title,
.ui-btn-inner *
{
   white-space: normal !important;
}
.photo
{
   display: block;
   margin: 0px auto;
}
dl.informations dt
{
   font-weight: bold;
}
#map
{
   width: 100%;
   height: 600px;
}
```

In this part, We've shown all the HTML pages of "Where You parked your car", so now you're able to navigate across them. However, at the moment, the pages are missing the business logic that will govern the functionality and interaction. In the next part, We'll start explaining some of the JavaScript files that power the application and I'll be going into deep of the Cordova APIs.

## 3. JQUERY MOBILE CONFIGURATION

jQuery Mobile does a lot for you just by adding it to your project without any special settings. However, there will be times when you want to modify or to take control of some default behavior. You can achieve this writing a configuration file. As stated in the first part of the series, the app has a file called jquery.mobile.config.js that, as you might guess, contains a configuration for jQuery Mobile. The most observant of you might have noticed that in the index.html file, the configuration is loaded before the library. You have to do so because when jQuery Mobile starts, it fires an event called mobileinit, and this is also the event you have to bind to override the default settings. Hence, you have to attach the handler before jQuery Mobile starts, but after the jQuery library.
You can override the default configuration in two ways. The first is using the jQuery extend() method as you can see in the following example.

```
1    $(document).bind('mobileinit', function() {
2      $.extend($.mobile , {

3        property1: value1,
4        property2: value2

5        // and so on...
6      });

7    });
```

The second way is to set the properties using the $.mobile object.

```
1    $(document).bind('mobileinit', function() {
2      $.mobile.property1 = value1;

3      $.mobile.property2 = value2;
4      // and so on...

5    });
```

Explaining all the jQuery Mobile properties is outside the scope of this article, however you can study them in deep detail by reading the jQuery Mobile configuration docs.

In the app configuration file, I won't change a lot of properties. I'll focus on changing the pages' transition, the option to show the text when showing the page loader widget, and the theme. The full source of the file (that uses the second method to set jQuery Mobile properties) is listed below.

```
01   $(document).on(
02     'mobileinit',
```

```
03     function()
04     {

05       // Default pages' transition effect
06       $.mobile.defaultPageTransition = 'slide';
07       // Page Loader Widget
08       $.mobile.loader.prototype.options.textVisible =
true;

09       // Theme
10       $.mobile.page.prototype.options.theme  = 'b';

11       $.mobile.page.prototype.options.headerTheme = 'b';
12       $.mobile.page.prototype.options.contentTheme = 'b';

13       $.mobile.page.prototype.options.footerTheme = 'b';
14       $.mobile.page.prototype.options.backBtnTheme = 'b';

15     }
16   );
```

The APIs

From this section on, I'll start describing the APIs used throughout the project. Apart from the Google Maps API, the other two—that is the Geolocation API and the Web Storage API—are W3C specifications. We'll use this APIs through the Cordova framework, however some devices may already provide an implementation of these APIs. For those devices, we used their built-in support instead of the Cordova's implementation, but for those that don't have storage support, the Cordova implementation has been built to remain compatible with the W3C specs.

Managing the Locations

In this section I'll show you the class called Position, which is responsible for managing the locations. It's aim is to create, delete, and load the locations using the Web Storage API that is composed by two areas: the Session and the Local. Cordova uses the local storage because the storage is always at app level. All the locations will be stored in an item called "positions."

Before going on, I want to highlight two facts. The first is that to store complex data (like objects and arrays) using the Web Storage API, you have to use the JSON format. So, in the Position class methods, you'll see the use of the JSON class and its parse() and stringify() methods. The second is that Windows Phone 7 doesn't support the dot notation so, you must use the setItem() and getItem() methods to ensure the compatibility for all devices. This is just the first of a lot of quirks that you have to face while developing with Cordova. Since with "Where I parked my car" we're not targeting a

specific platform, you have to take into account the different support and the quirks of the Cordova API. Luckily, the first one we encountered wasn't so hard to deal with.
For the purpose of the app, we need to save the user's and car's latitude and longitude, but we also need the accuracy of the measurement to give the user an idea of how precise our locations are. I'll wrap the first three data into a class called Coords.

Reading coordinates isn't very user-friendly, so to enhance the user experience, we'll use the Google Maps API to retrieve the address of the current location. The app will also store the date and time associated with that GPS location. Remember that the app will store up to 50 locations, so if the limits is reached, the extra positions (the older ones) will be eliminated using the JavaScript slice() method.

For what discussed so far, the starting point of the Position class is implemented by the following code.

```
01   Function Position(position, address, datetime)
02   {

03     var _db = window.localStorage;
04     var MAX_POSITIONS = 50;

05     this.getMaxPositions = function()
06     {

07       return MAX_POSITIONS;
08     }

09     this.position = position;
10     this.address = address;

11     this.datetime = datetime;
12   }

13   Function Coords(latitude, longitude, accuracy)
14   {

15     this.latitude = latitude;
16     this.longitude = longitude;

17     this.accuracy = accuracy;
18   }
```

As you may have guessed, in the JavaScript sense of the object-oriented approach, the _db property is private and MAX_POSITIONS is a constant. With the given skeleton we can't do much, in fact, we need methods to let the application interface with the Web Storage API. These methods are
savePosition() to store the car's position

updatePosition() to update the last position retrieved with the address if the callback to the Google Maps API succeed
deletePosition() to allow the user to remove a previously saved position
getPositions() to load all the stored locations

In all the cited methods, I'll test if the database var (_db) is null and if it is, I'll show an error message to the user. I test for this state because I like to try to anticipate and manage unexpected interface problems. To show the message, I'll take advantage of the alert() method of the Cordova Notification API. Most of the supported platforms use a native dialog box, however, others (like Bada 2.X) use the well-known browser's alert() function's look and feel, which is less customizable. The Cordova alert() function accepts up to four parameters:
message: The string that contains the message to show
alertCallback: A callback to invoke when the alert dialog is dismissed
title: The title of the dialog (by default is "Alert")
buttonName: The text of the button included in the dialog (by default is "OK")
There's another quirk that we have to consider while developing with PhoneGap. Windows Phone 7 ignores the button name and always uses the default. And, there isn't a built-in browser alert, so if you want to use alert('message'); you have to assign window.alert = navigator.notification.alert. For a complete list of the differences between mobile operating systems, refer to the Cordova Notification alert() documentation.

## 4. CONCLUSION

In this third part, you've seen how to configure "Where you parked your car" to change the pages' transition, the option to show the text when showing the page loader widget, and the theme. You also learned how to store the car's positions using the Web Storage API. We'll show you the last two JavaScript files: maps.js, and functions.js. The former contains the functions to initialize the application and some other utility functions, while the latter has the functions that use the Google Maps API to draw the map and the route to the car.

## LITERATURE:

[1] Milosavljević M., Kreiranje aplikacija za PocketPC, Internet ogledalo, 2005. Beograd
[2] Milosavljević M., Svojstva, metode, događaji..., Internet ogledalo, 2005. Beograd
[3] Nikitović B., PocketPC povezivanje, PC PRESS, 2005. Beograd
[4] Nikitović B., PocketPC računari, PC PRESS, 2005. Beograd
[5] Siler B. i Spotts J., Razvoj Web aplikacija: VB.NET i VC#.NET, QUE i CET, 2003.Beograd
[6] www.sitepoint.com,, Where I parked my car, 2012.
[7] www. jquery.com
[8] http://www.popwebdesign.net/popart_blog/2010/06/jquery-sta-je-i-cemu-sluzi/