Building 100% Native Mobile Apps with ASP.NET MVC

by Daniel Jebaraj

Syncfusion
Deliver innovation with ease

Contents

Native Mobile Apps with ASP.NET MVC

Mobile devices are everywhere

Over a decade ago, while attending a Microsoft Professional Developers Conference, we were shown a video on the coming mobile future. The video showcased futuristic-looking, Windows-powered phones being used for tasks such as locating the closest doctor's office. In an era when the Palm VII (http://en.wikipedia.org/wiki/Palm_VII) was the closest thing to a wireless smartphone, the video showcased an impressive future.

Fast forward to 2012: There is no mistake that we are living this future. Smartphones and other mobile devices such as tablets are everywhere. They are available at multiple price points and are increasingly affordable. In fact, for many in the developing world, their only computer is the powerful smartphone they own.

Mobile application development: growth predictions from Gartner

Gartner predicts[i] that by 2016, at least 50 percent of enterprise e-mail users will rely primarily on a browser, tablet, or mobile client instead of a desktop client. Given the increase in the adoption of mobile devices, it is also expected that software application development targeting these devices will also dramatically increase in the coming years. Again, Gartner predicts that by 2015 mobile application development projects targeting smartphones and tablets will outnumber native PC projects by a ratio of 4 to 1. Gartner further says that smartphones and tablets will represent more than 90 percent of the new net growth in device adoption in the coming four years.

The Apple App Store now boasts over 500,000 apps. Android has close to the same number and the Windows Phone marketplace, a much more recent contender, recently crossed 50,000 and is growing at a fast pace.

Line of business mobile applications: the challenge posed by fragmentation

Given this rather exciting backdrop, we can be certain that most line-of-business applications will be made available on mobile platforms in the immediate future. As with any other opportunity, mobile application development with all its promises comes with its own set of challenges.

One of the primary challenges is the issue of fragmentation. Estimates from the third quarter of 2011 indicated that the mobile operating system market is very fragmented. The variants of Android accounted for around 50% of devices sold during this quarter. Symbian and iOS accounted for about 17% each. Research in Motion (RIM/Blackberry) accounted for about 11% and the Windows Phone platform for around 2%.

Developing a line-of-business application that will function on all these devices involves working with the following vastly different technologies:

Mobile platform	Primary development platform	Primary development language	Primary IDE	Development platforms
Android	Java based	Java	Eclipse	Windows, Mac OSX, Linux
iOS	Cocoa Touch framework	Objective C	Xcode	Mac OSX
RIM	Java ME	Java	Eclipse	Windows, Mac OSX
Windows Phone 7	Silverlight on the .NET platform	C#	Visual Studio	Windows

The platforms, languages, and tools involved are substantially different, and the effort involved in producing a solution that will work on every platform is substantial.

It is also worth noting that there is substantial fragmentation even within some of the platforms. This is especially true with the currently dominant Android platform. Given that Android is open and vendors are free to make changes, there are literally hundreds of Android-based devices available on the market today. Many of them work only with specific levels of the Android API. Some of them even have issues with applications that target certain features within a supported API level. In summary, there is no shortage of fragmentation in the mobile market. This makes the implementation of a native solution on multiple platforms quite daunting.

Mobile Web applications: the solution for fragmentation?

Web applications are an alternative to native applications. All the major mobile platforms offer very capable browsers. In addition, with the exception of the Windows Phone browser, most other platform browsers are based on the open source WebKit browser platform that powers the desktop versions of Apple Safari and Google Chrome. There is excellent support for JavaScript on these browser platforms; jQuery is fully supported on most current mobile devices. Also, increasing compliance with HTML 5 and related Web standards is making the browsers even more attractive as a development platform. It is possible to build very functional Web sites that work very well on mobile devices with technology available today.

Mobile Web applications: additional considerations

Building a mobile Web site does not offer the same experience as a native application. Users on specific hardware platforms are accustomed to the enhanced experience offered by native applications. Such applications are installed natively and are always available on the launcher surface of the device. Native applications also obey user interface contracts on the device. For instance, on Android the left Menu button usually displays a context menu. Users expect this. Web applications can be installed as shortcuts on the launcher surface for most devices, but they do not obey specific user-interface expectations on the deployed device. Another disadvantage Web applications have is that they have no native access to hardware beyond what is exposed by HTML and related Web standards. For instance, there is no direct access to contacts, images, or the camera on the device. For many applications, access to key elements of device hardware is important.

Hybrid applications: the best of Web and native applications

Hybrid applications are completely native applications that embed a platform-specific Web browser control. All major mobile platforms including Android, iOS, Windows Phone 7, and Blackberry/RIM support embedding Web browser controls as implemented on their platform. Since the wrapper is completely native, users are not often even aware they are interacting with a Web application. It is quite possible for the native application to provide a seamless navigation experience.

It is also possible for Web pages displayed in the browser to interact with the native hardware through a JavaScript bridge, a form of which is available on every major platform. Using such callbacks to the native platform make it possible to access contacts, capture or select images, and play media. In fact, anything you can accomplish through native code can be accomplished through the bridge. The bridge code will of course have to be re-written for every target platform, but this is usually a small fraction of your total application code.

Also, several JavaScript bridge frameworks exist; the most popular is the open source PhoneGap platform which provides a substantial part of this plumbing. We will not be using any frameworks for this purpose. We will instead illustrate the concept with a simple Android wrapper.

ASP.NET MVC: an elegant framework for your back end

Hybrid applications can of course be built with any Web back end, but we firmly believe that ASP.NET MVC is ideally suited[ii] for the implementation of hybrid applications. Below are some aspects that make ASP.NET MVC a good choice for such applications.

Clear separation of responsibilities

The clear separation of responsibilities afforded by the MVC environment makes it possible to have very precise control over HTML output. This makes it very easy to generate mobile-friendly HTML. There is no built-in, self-contained control model that makes it hard to control the markup that is produced.

Sharing most code with desktop or tablet Web clients

If you have an existing ASP.NET MVC Web application that targets desktop browsers, much of the code can be shared with your mobile application. The controller and model code can be shared almost as is. Only the view needs to be changed. It is not difficult to specify a custom view for mobile clients even with the current version of ASP.NET MVC, but the next version of ASP.NET MVC makes it even simpler. For additional details on mobile-friendly features in the upcoming version of ASP.NET MVC, please refer to http://www.asp.net/mvc/tutorials/mvc-4/aspnet-mvc-4-mobile-features.

Minimal friction with underlying Web development model

ASP.NET MVC does not build several layers of abstraction over stateless Web applications. Instead it offers a very simple model that works in alignment with the underlying platform. This makes it very easy to make AJAX calls or use jQuery on the client. There is no complex abstraction such as ASP.NET Web Forms' ViewState to worry about.

In addition to the above, it is also worth pointing out that the business and database layers that already exist in your current .NET applications can be effectively reused with ASP.NET MVC applications. ASP.NET MVC is completely agnostic about the business and database layers and can work effectively with any system that is currently in place.

Sample hybrid application

We will now walk through a very simple sample that will illustrate the development of a hybrid application end-to-end using the ASP.NET MVC platform. The sample displays information on students attending a fictional university named Contoso University. There are a couple of general information links as well as access to a student directory where students can be looked up by name. The sample does not implement any security or error handling in order to keep the code clear. There is no complex code since the objective of the sample is not to showcase the

power of the ASP.NET MVC platform, but to showcase its suitability as a back-end platform for the development of hybrid, native mobile applications.

The complete code for this sample is available at http://bit.ly/mvc-native-mobile-apps.

Prerequisites to work with the sample code:

- ASP.NET MVC 3 with Visual Studio 2010 (any version including the Express Edition).
- Functional installation of the Android SDK and the Android Development Tools plugin for Eclipse.

Detailed instructions and requirements are available here:
http://developer.android.com/sdk/requirements.html

- jQuery and jQuery Mobile. A local copy is not required since the sample code will simply reference the jQuery CDN.

The ASP.NET MVC back end

In the sample code, the provided _Layout.cshtml contains script references to the jQuery and jQuery Mobile libraries. They are not required to build an ASP.NET MVC mobile application, but they do handle much of the grunt work. We use jQuery Mobile in our sample since our purpose is not to illustrate the nuances of formatting content on mobile devices.

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css" />
<link href="@Url.Content("~/Content/Site.css")" rel="stylesheet" type="text/css" />

<script type="text/javascript" src="http://code.jquery.com/jquery-1.6.4.min.js"></script>
<script type="text/javascript" src="http://code.jquery.com/mobile/1.0/jquery.mobile-
1.0.min.js"></script>
```

Most mobile Web clients assume that a Web page is sized at about 900 pixels and will automatically scale to display the entire page on the device. With a mobile site that is optimized for a smaller device, we can provide a hint to the device that it should not scale but should instead use the width of the device. This is accomplished via the use of the viewport meta tag as shown below.

```
<meta name="viewport" content="width=device-width, initial-scale=1.0 ">
```

The default index action method on the home controller is mapped to the following view markup.

```
<nav >
    <ul id="menu" data-role="listview">
        <li>@Html.ActionLink("About Us", "AboutUs", "Home")</li>
        <li>@Html.ActionLink("Contact Us", "ContactUs", "Home")</li>
        <li>@Html.ActionLink("Student Directory", "StudentDirectory", "Home")</li>
    </ul>
</nav>
```

We have a simple unordered list with three action links. We specify that the list should be automatically formatted as a list view by the jQuery Mobile runtime through the use of the "`data-role=listview`" attribute setting. This is all that is required to display the following initial UI on a mobile device.

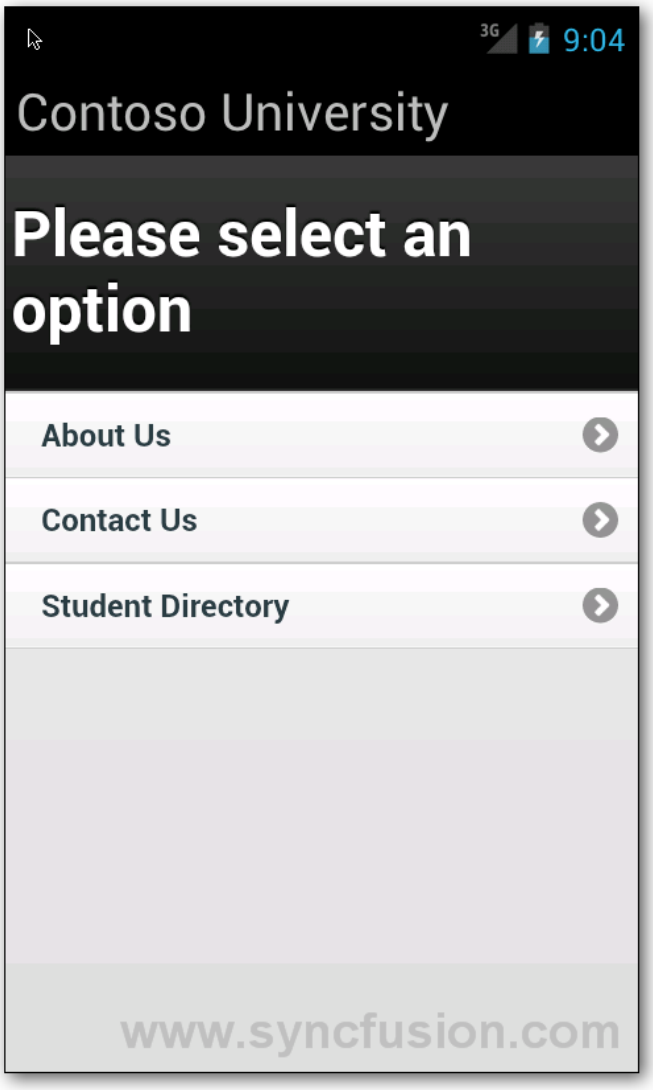

Figure 1: Initial screen

The jQuery Mobile runtime takes care of formatting it as a list view. As mentioned previously, jQuery Mobile is certainly not needed, and indeed it is fairly simple to accomplish this task without the use of jQuery Mobile. However, jQuery Mobile handles this in a seamless manner and works on a large number of mobile devices. You can pick the formatting and scripting approach that suits your needs best.

The sample contains views that are displayed when the About Us and Contact Us options are invoked. These screens are straightforward and do not require any further explanation.

The **Student Directory** link displays a page with student names grouped by starting letter. The page also displays the number of students listed under each letter.

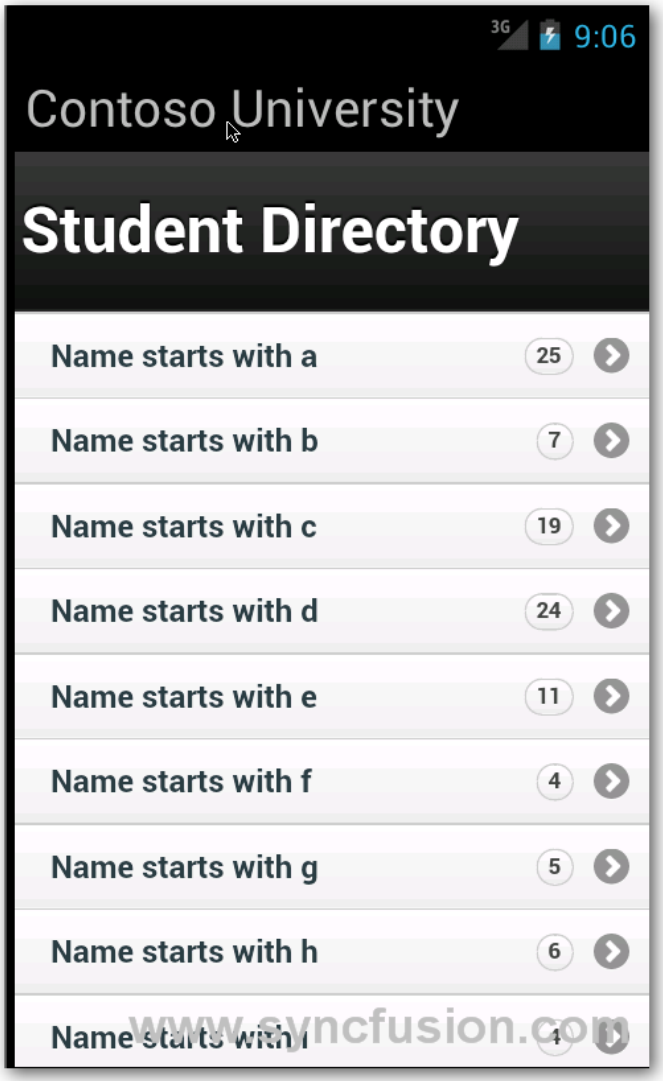

Figure 2: Student directory initial screen

Clicking on any option displays a list of students as shown in the following figure.

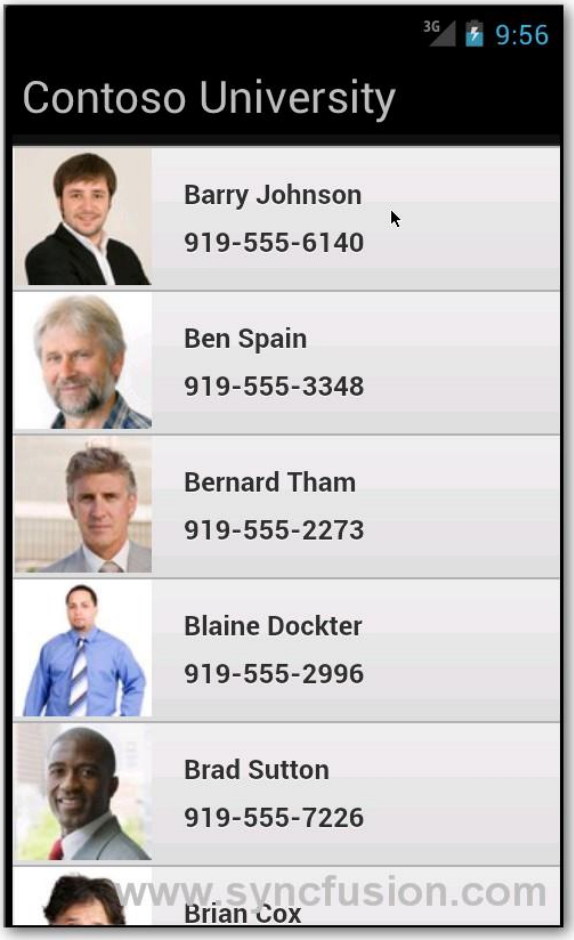

Figure 3: Student directory

The student directory views are also fairly simple. They iterate through and display data in a list. The view that displays student details is shown below.

```
@{
    ViewBag.Title = "Student Directory";
    Layout =
    "~/Views/Shared/_Layout.cshtml"; var
    random = new Random();
}

<ul data-role="listview">
@foreach (string student in ViewBag.Students)
{
    <li>

        @{var number = random.Next(1000, 9999); }

        <img src="@Url.Content("~/Content/images/UserImages/80-80/" + student + ".jpg")" alt="@student"/>
        <h3>@student</h3>
        <h4>919-555-@number</h4>
    </li>
}
</ul>
```

It is a good idea to run the ASP.NET MVC back end in a desktop browser and test it out before proceeding to review the Android wrapper that we will work with next.

You can also directly test on a mobile browser provided the test site is accessible from your test device. If both the development PC and your test device are on the same network, it is possible to make setting changes to the ASP.NET development browser or IIS Express to allow access to the Web application from your test device. Such access is blocked by default.

An easier, alternate approach is to use a proxy which simply redirects traffic on an external port to the internal server. This is the approach we often use. The proxy that we use is available for download from https://github.com/jocull/SharpProxy[iii].

Android wrapper

The code for the Android wrapper that hosts the Web application inside a native Android application is reproduced below.

```
 package com.syncfusion.contoso;
 import android.app.Activity;
 import android.os.Bundle;
 import android.util.Log;
 import android.view.KeyEvent;
 import android.view.View;
 import android.webkit.WebView;
 import android.webkit.WebViewClient;

public class ContosoActivity extends Activity {

       WebView mWebView;

     private class ContosoWebViewClient extends WebViewClient {
         @Override
         public boolean shouldOverrideUrlLoading(WebView view, String url) {
             view.loadUrl(url);
             return true;
         }
     }

     /** Called when the activity is first created. */
     @Override
public void onCreate(Bundle savedInstanceState) {

         super.onCreate(savedInstanceState);
         setContentView(R.layout.main);

          mWebView = (WebView) this.findViewById(R.id.webview);

         // Disable scrollbars
         mWebView.setVerticalScrollBarEnabled(false);
         mWebView.setHorizontalScrollBarEnabled(false);

         // Scrollbar Overlay Content
         mWebView.setScrollBarStyle(View.SCROLLBARS_INSIDE_OVERLAY);

     mWebView.getSettings().setJavaScriptEnabled(true);
     mWebView.getSettings().setAppCacheEnabled(false);
     mWebView.loadUrl("http://your-web-link");
     mWebView.setWebViewClient(new ContosoWebViewClient() );
```

```
    }

    @Override
    public boolean onKeyDown(int keyCode, KeyEvent event) {
        if ((keyCode == KeyEvent.KEYCODE_BACK) && mWebView.canGoBack()) {
            mWebView.goBack();
            return true;
        }
        return super.onKeyDown(keyCode, event);
    }
}
```

The code is quite simple to follow.

1. WebView is the Android equivalent of the WebBrowser control. It is a wrapper around the default WebKit-based Android browser.
2. We obtain access to an instance of the Android WebView control (defined in an XML layout file and instantiated by the Android runtime at execution).
3. We enable the use of JavaScript on this WebView instance since JavaScript is disabled by default with the WebView control.
4. We then make a few adjustments to the display of the scrollbar—basically turning it off to mimic the look and feel of a native application.
5. We then load the actual Web application using a call to the loadUrl API on the WebView instance. `your-web-link` should be changed to point to your Web application.
6. The last section of the code handles the invocation of the hardware **Back** button and causes the embedded WebView to navigate to the previous page.

As you can see this code is not tied to the Web application in any direct manner and will not change substantially from application to application. You will only need to add additional code when you require access to specific hardware functionality on the device. We do not delve deeper into this topic here but if you are interested in investigating this further, please look up information on the addJavascriptInterface[iv] method of the WebView.

For simplicity, we have described just the Android wrapper. Similar wrappers and extension mechanisms exist on all other major mobile platforms.

Figure 4: Contact Us page displayed on Android 4.0 Emulator inside a native application shell

Conclusion

Hybrid applications are a very promising solution worth looking into for any line-of-business mobile application. They are not suited for scenarios where extensive access to native hardware is required (such as with games) but will work very well in most other scenarios. Any solution implemented with a Web back end is also more likely to be future-proof. The HTML standard has evolved slowly over the years and is unlikely to dramatically change as proprietary solutions often tend to do. It offers a stable base on which applications can be built with the certainty that they will continue to work for the foreseeable future. Mobile platform vendors are putting an extraordinary amount of effort into the implementation of HTML 5 and related standards. This will also serve to make Web applications more powerful and able to accomplish a substantial subset of what is possible with native applications.

You can leverage your existing .NET Web development skills and produce powerful, 100% native solutions that work on a broad cross section of devices. At Syncfusion, we are excited by the immense potential offered by hybrid applications. We currently offer a wide set of mobile controls for use under the ASP.NET MVC platform and have many more exciting offerings on the way. We hope you too are excited by the potential offered by hybrid mobile applications.

[i] http://bit.ly/uIkqKR—Gartner Reveals Top Predictions for IT Organizations and Users for 2012 and Beyond.

[ii] Ignore ASP.NET MVC at Your Own Peril: Lessons Learned from the Trenches available from http://www.syncfusion.com/downloads/resources/whitepaper/aspnet-mvc.

[iii] For further details, please refer to http://www.codefromjames.com/wordpress/?p=97.

[iv] http://developer.android.com/reference/android/webkit/WebView.html#addJavascriptInterface(java.lang.Object, java.lang.String).