

Web on Mobile

Hybrid Apps:
HTML5 Webapps on
Android Mobile Devices

Motivation

- Can't we just point our mobile browser at a remote web server that has a responsive HTML5 web application?
 - Yes, but:
 - » what if you want a self-contained local app?
 - » Or an app with a WebView activity alongside other activities?
- Why would you want to do either of those?
 - Technical reasons:
 - » HTML5 can provide a true cross-platform solution, instead of going back to the old-school cross-platform days
 - » Many apps are content-driven (weather, news, social media) and HTML5 is an excellent platform for content-driven apps
 - Non-technical (software engineering):
 - » Write once, run everywhere is more manageable for your dev organization

Hybrid Applications on Android

Cross platform development for the mobile platform:

- Most mobile apps still written natively and ported to multiple OS platforms
- Some use toolkits (Xamarin, Titanium, PG/Cordova) with mixed results
- Smaller but growing number look to HTML5/JS stack → *vm - JS engine*
- Issues are: performance vs. UX quality vs. cost to dev vs. cost to maintain

Hybrid applications attempt to get the best of both worlds

- Performance by using native APIs for device-specific features
- Better UX, portability, and maintainability through embedded RWD for hypermedia content consumption

Challenges:

- Hybrid means more technologies in your stack
 - Burden on developers, processes, toolchains
 - Introduces additional technology dependencies => risk
- How do you communicate in/out of the WebView?



Apps with WebViews

Step 1: Create a WebView

R: java

See sampleapp1webview project

WebView browser = (WebView) findViewById(R.id.webView); // presumes
browser.loadUrl(getString(R.string.wvURL)); // webView, wvURL exist

Step 2: Set your WebViewClient to handle URLs

- if needed create your own custom one to control page loading

```
private class MyWebViewClient extends WebViewClient {  
    public boolean shouldOverrideUrlLoading(WebView wv, String u) {  
        // in here goes logic to determine whether I want my  
        // WebView should load the URL - return false for yes  
        // since we specified above we want to load. If I don't  
        // want WebView to load it, create another Intent so we  
        // can have another Activity do it (i.e. default browser)  
    }
```

```
webView.setWebViewClient(new MyWebViewClient());  
// if I don't want a custom WebView I can use some builtins like  
// the generic WebViewcClient directly
```

Apps with WebViews (cont.)

Step 3: Enable Javascript

```
WebSettings ws = webView.getSettings();  
ws.setJavascriptEnabled(true);
```

See sampleapp1webview project

Step 4: Decide how to handle page navigation

- We have a “back” button in Android that pops the Activity stack
- We have a back button in the browser that goes back in the history
- By default, the Android back button will pop the WebView
- To change this, we have to capture the event and check the history

```
public boolean onKeyDown(int keyCode, KeyEvent event) {  
    if ((keyCode == KeyEvent.KEYCODE_BACK) && myWebView.canGoBack()) {  
        myWebView.goBack(); ← Browser back button  
        return true;  
    }  
    // If it wasn't the Back key or there's no web history, use the default  
    // system behavior (probably exit the activity)  
    return super.onKeyDown(keyCode, event);  
}
```

Hybrid Apps with WebViews

To enable HTML5 content and interaction, we need an Android widget of some kind to render and run the content and scripts

- The Android widget is called a WebView
- Introduced in API level 1, but has evolved significantly over time
- You have to program what capabilities this “chromeless browser widget” has and how it should communicate to other native Android widgets

NOTE: First do steps 1-3 from non-hybrid app

See js-interface project

Step 5: Create a Java method annotated with @JavascriptInterface

```
public class Tweek {    // Note it doesn't extend anything
    @JavascriptInterface
    public void collectUnderpants(String foo) { ... }
    @JavascriptInterface
    public void profit(String bar) { ... }
}
```

Hybrid Apps with WebViews (cont.)

Step 6: Add an interface to the method to the WebView

```
wv.addJavascriptInterface(new Tweek(this), "Gnome")
```

Step 7: Invoke method through interface from Javascript

```
<input type="button" value="OK" onClick="show('you killed Kenny!')"/>
<script type="text/javascript">
  function sMsg(msg) {
    Gnome.collectUnderpants(msg);
    Gnome.profit(msg);
  }
</script>
```

Note:

- You do not reference Tweek, you use dot notation on Gnome
- Methods in Java were not static, so there is a Tweek object created
- Tweek object runs a different thread than the one in which it was constructed

Hybrid Apps with WebViews (cont.)

- This makes Java “callbacks” available to Javascript
- What about passing information into the Javascript?
 - Bug in 3.x versions that prevents query parameters on URL
 - file:/// URLs do not support them (no HTTP, so no headers)
 - Better just calling back to Java via event handler at beginning

request headers

Calling Javascript from Java

- You can tell the WebView to load a javascript URL

```
wv.loadUrl('javascript:myFunc()');
```

file:/// <<<let>>>
?foo=bar

- But this is consider bad form as it blocks the main UI thread
- Instead use evaluateJavascript to execute JS asynchronously

```
wv.evaluateJavascript(<myScript>, <myValueCallback>);
```

ⓐ

4.4

```
// ValueCallback<String> has 1 method, onReceiveValue
```


Hybrid Apps Summary

- For the use case where you prefer native for highly interactive and HTML5 for content
- I (Dr. Gary) personally believe HTML5 will start invading the interactive space as well and hybrid apps will decline as a result – but I've been wrong before!
 - The risk won't be performance but security!
- The mechanisms for achieving data and control integration across the WebView boundary have been changing, buggy, and frustrating.
 - And just when you figure it out, they change the web rendering engine in Android!

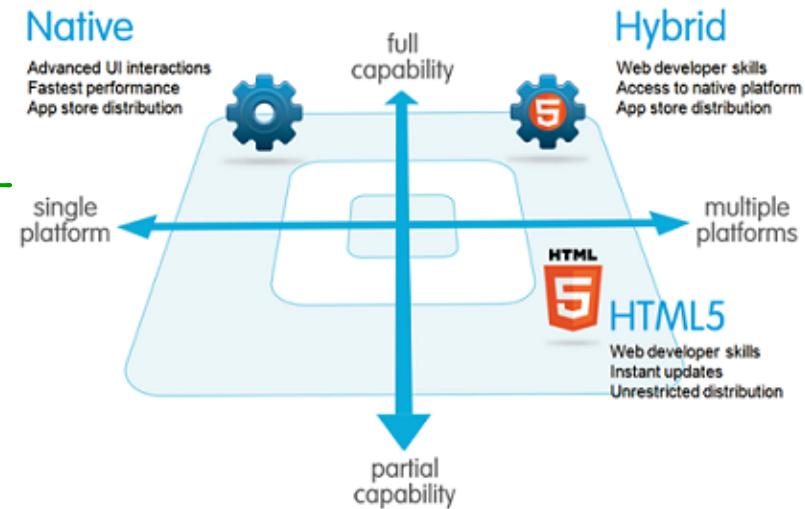


Image from <https://goo.gl/l1qFGL>



U.4 A Pika Kitkat