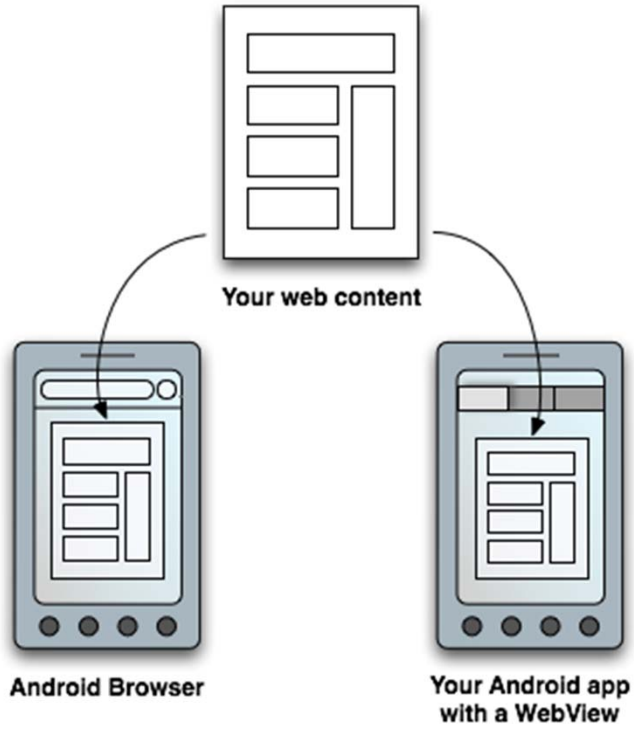
Mobile Application Development

Building Web Apps in WebView

Web Apps

- There are essentially two ways to deliver Web Apps on Android:
  - As an Android Application (Using WebView)
  - As online Web Application (Using Web Browser installed on your device)



WebView

- The **WebView** class is an extension of Android's View class that allows you to display web pages as a part of your activity layout.
- It does not include any features of a fully developed web browser, such as navigation controls or an address bar.
- All that WebView does, by default, is show a web page.

Why Use WebView?

- A common scenario in which using WebView is helpful is when you want to provide information in your application that you might need to update, such as an **end-user agreement** or a **user guide**.

- Another scenario in which WebView can help is if your application provides data to the user that **always requires an Internet connection to retrieve data**.

  - In this case, you might find that it's easier to build a WebView in your Android application that shows a web page with all the user data, rather than performing a network request, then parsing the data and rendering it in an Android layout.

  - Instead, you can design a web page that's tailored for Android devices and then implement a WebView in your Android application that loads the web page.

Adding WebView

- To add a WebView to your Application, simply include the <WebView> element in your activity layout.

```
<RelativeLayout ...>

    <WebView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/webView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

Load URL in WebView

- To load a web page in the WebView, use **loadUrl()**. For example:

```
WebView wView;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        wView=(WebView)findViewById(R.id.webView);
        wView.loadUrl("http://w3.org");

    ...
}
```

Get Internet Access

- Before this will work, however, your application must have access to the Internet. To get Internet access, **request the INTERNET permission in your manifest file**. For example:

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

Enabling JavaScript

- **JavaScript is disabled** in a WebView by default.
- You can enable it through the **WebSettings** attached to your WebView.
- You can retrieve WebSettings with **getSettings()**, then enable JavaScript with **setJavaScriptEnabled()**.

```
...
wView=(WebView)findViewById(R.id.webView);
WebSettings webSettings = wView.getSettings();
webSettings.setJavaScriptEnabled(true);
wView.loadUrl("http://w3.org");
...
```

Handling Page Navigation

- When the **user clicks a link** from a web page in your WebView, the default behavior is for Android to launch an application that handles URLs.
- Usually, the **default web browser opens and loads the destination URL**.
- However, **you can override this behavior** for your WebView, so links open within your WebView.
- You can **then allow the user to navigate backward and forward** through their web page history that's maintained by your WebView.

Handling Page Navigation

- To open links clicked by the user, simply provide a **WebViewClient** for your WebView, using **setWebViewClient()**. For example:

```
...
wView=(WebView)findViewById(R.id.webView);
WebSettings webSettings = wView.getSettings();
webSettings.setJavaScriptEnabled(true);
wView.setWebViewClient(new WebViewClient());
wView.loadUrl("http://w3.org");
...
```

Override URL Loading

- If you want more control over where a clicked link load, **create your own WebViewClient that overrides the shouldOverrideUrlLoading() method**. For example:

```
private class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        if (Uri.parse(url).getHost().endsWith("w3.org")) {
            return false;
        }

        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        view.getContext().startActivity(intent);
        return true;
    }
}
```

Override URL Loading

- Then create an instance of this new WebViewClient for the WebView:

```
...
wView=(WebView)findViewById(R.id.webView);
WebSettings webSettings = wView.getSettings();
webSettings.setJavaScriptEnabled(true);
// wView.setWebViewClient(new WebViewClient());
wView.setWebViewClient(new MyWebViewClient());
wView.loadUrl("http://w3.org");
...
```

Handling the Android Back Button

- As you navigate around the webpages, hitting the back button on Android exits the application, even though you've explored a few pages of the site. To overcome this we can override onBackPressed() method in our activity using WebView's canGoBack() & goBack() methods .

```
@Override
public void onBackPressed() {
    if (wView.canGoBack()) {
            wView.goBack();
    } else {
            super.onBackPressed();
    }
}
```

USING HTML FROM FILE SYSTEM

Loading Local Pages

- In case you want to store a copy of a webpage locally to be loaded into a WebView, you can put it in the android **assets** folder.
- If you do not find one under your main/ directory, then you can create one. Place the html, css, js, etc in this folder.
- For example, if you wanted to load a page index.html, you can place this file in this location: {**ProjectName}/app/src/main/assets/index.html**

Adding WebView

- To add a WebView to your Application, simply include the <WebView> element in your activity layout.

```
<RelativeLayout ...>

    <WebView
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:id="@+id/localWebView"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

Load URL in WebView

- To load a web page in the WebView, use **loadUrl()**. For example:

```
WebView localWeb;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        localWeb=(WebView)findViewById(R.id.localWebView);
        localWeb.loadUrl("file:///android_asset/index.html");

    ...
}
```

Get Internet Access

- If your app is only using WebView for local files then you don't need to make the following entry in AndroidManfest.xml file:

```
<manifest ... >
    <uses-permission android:name="android.permission.INTERNET" />
    ...
</manifest>
```

Enabling JavaScript

- **JavaScript is disabled** in a WebView by default.
- You can enable it through the **WebSettings** attached to your WebView.
- You can retrieve WebSettings with **getSettings()**, then enable JavaScript with **setJavaScriptEnabled()**.

```
...
localWeb=(WebView) findViewById(R.id.localWebView);
WebSettings webSettings = localWeb.getSettings();
webSettings.setJavaScriptEnabled(true);
localWeb.loadUrl("file:///android_asset/index.html");
...
```

Override URL Loading

- If you want more control over where a clicked link load, **create your own WebViewClient that overrides the shouldOverrideUrlLoading() method**. For example:

```
private class MyWebViewClient extends WebViewClient {
    @Override
    public boolean shouldOverrideUrlLoading(WebView view, String url) {
        if (Uri.parse(url).getHost().endsWith("w3.org")) {
             return false;
        }
        else if (Uri.parse(url).getHost().length()==0) {
            return false;
        }
        Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(url));
        view.getContext().startActivity(intent);
        return true;
    }
}
```

Override URL Loading

- Then create an instance of this new WebViewClient for the WebView:

```
...
localWeb=(WebView) findViewById(R.id.localWebView);
WebSettings webSettings=localWeb.getSettings();
webSettings.setJavaScriptEnabled(true);
localWeb.setWebViewClient(new MyWebViewClient());
localWeb.loadUrl("file:///android_asset/index.html");
...
```

Handling the Android Back Button

- As you navigate around the webpages, hitting the back button on Android exits the application, even though you've explored a few pages of the site. To overcome this we can override onBackPressed() method in our activity using WebView's canGoBack() & goBack() methods .

```
@Override
public void onBackPressed() {
    if (localWeb.canGoBack()) {
            localWeb.goBack();
    } else {
            super.onBackPressed();
    }
}
```

BINDING JAVASCRIPT CODE TO ANDROID CODE

Binding JavaScript with Android

- When developing a web application that's designed specifically for the WebView in your Android application, you can create interfaces between your JavaScript code and client-side Android code.

- For example, your JavaScript code can call a method in your Android code to display a Toast, instead of using JavaScript's alert() function.

- To bind a new interface between your JavaScript and Android code, call **addJavascriptInterface()**, passing it a class instance to bind to your JavaScript and an interface name that your JavaScript can call to access the class.

Create Interface Class

- Create a new java class:

```
public class WebAppInterface {
    Context activity;

    WebAppInterface(Context c) {
        activity=c;
    }

    @JavascriptInterface
    public void showToast(String s) {
       Toast.makeText(activity, s, Toast.LENGTH_SHORT).show();
    }
}
```

Bind Class to the JavaScript

- You can bind this class to the JavaScript that runs in your WebView with **addJavascriptInterface()** and name the interface Android. For example: This creates an interface called Android for JavaScript running in the WebView.

```
...
localWeb=(WebView) findViewById(R.id.localWebView);
WebSettings webSettings=localWeb.getSettings();
webSettings.setJavaScriptEnabled(true);
localWeb.setWebViewClient(new MyWebViewClient());

localWeb.addJavascriptInterface(new WebAppInterface(this),
"Android");

localWeb.loadUrl("file:///android_asset/index.html");
...
```

Bind Class to the JavaScript

- At this point, your web application has access to the WebAppInterface class. For example, here's some HTML and JavaScript that creates a toast message using the new interface when the user clicks a button:

```
...
<input type="button" value="Show Toast"
onclick="showToast('Message From JavaScript');" />
...
function showAndroidToast(toast) {
    Android.showToast(toast);
}
...
```

Start Activity From JavaScript

- In you Interface class:

```
public class WebAppInterface {
    Context activity;
    WebAppInterface(Context c) {
        activity=c;
    }
     @JavascriptInterface
    public void showToast(String s) {
       Toast.makeText(activity, s, Toast.LENGTH_SHORT).show();
    }

    @JavascriptInterface
     public void startActivity() {
            Intent i=new Intent(activity, MainActivity.class);
            activity.startActivity(i);
     }
}
```

References

- http://developer.android.com/guide/webapps/index.html
- https://developer.chrome.com/multidevice/webview/overview

Q & A