Creating hybrid mobile versions of the application with Apache Cordova

What will you learn here?

You finished creating the front-end for your application, and it has mobile support. You would now like to provide native client applications that your users can download from an application store. After reading this tutorial, you will understand how to reuse the existing HTML5 code for create native mobile clients for each target platform with Apache Cordova.

You will learn how to:

- make changes to an existing web application to allow it to be deployed as a hybrid mobile application
- create a native application for Android and iOS with Apache Cordova

What are hybrid mobile applications?

Hybrid mobile applications are developed in HTML5 - unlike native applications that are compiled to platform-specific binaries. The client code - which consists exclusively of HTML, CSS, and JavaScript - is packaged and installed on the client device just as any native application, and executes in a browser process created by a surrounding native shell.

Besides wrapping the browser process, the native shell also allows access to native device capabilities, such as the accelerometer, GPS, contact list, etc., made available to the application through JavaScript libraries.

In this example, we use Apache Cordova to implement a hybrid application using the existing HTML5 mobile front-end for TicketMonster, interacting with the RESTful services of a TicketMonster deployment running on JBoss A7 or JBoss EAP.

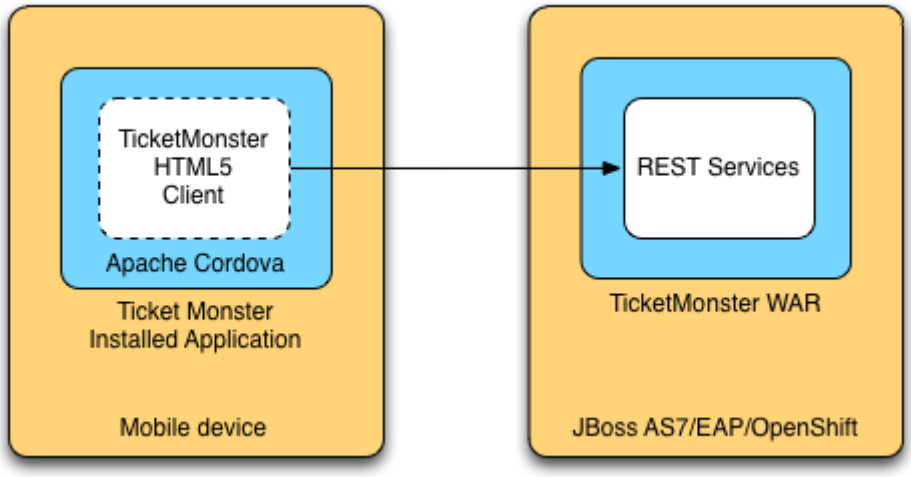

Figure 1. Architecture of hybrid TicketMonster

Tweak your application for remote access

Before we make the application hybrid, we need to make some changes in the way in which it accesses remote services. Note that the changes have already been implemented in the user front end, here we show you the code that we needed to modify.

In the web version of the application the client code is deployed together with the server-side code, so the models and collections (and generally any piece of code that will perform REST service invocations) can use URLs relative to the root of the application: all resources are serviced from the same server, so the browser will do the correct invocation. This also respects the same origin policy enforced by default by browsers, to prevent cross-site scripting attacks.

If the client code is deployed separately from the services, the REST invocations must use absolute URLs (we will cover the impact on the same-origin policy later). Furthermore, since we want to be able to deploy the application to different hosts without rebuilding the source, it must be configurable.

You already caught a glimpse of this in the user front end chapter, where we defined the `configuration` module for the mobile version of the application.

src/main/webapp/resources/js/configurations/mobile.js

```
...
define("configuration", function() {
    if (window.TicketMonster != undefined && TicketMonster.config != undefined) {
        return {
            baseUrl: TicketMonster.config.baseRESTUrl
        };
    } else {
        return {
            baseUrl: ""
        };
    }
});
```

```
...
```

This module has a `baseURL` property that is either set to an empty string for relative URLs or to a prefix, such as a domain name, depending on whether a global variable named `TicketMonster` has already been defined, and it has a `baseRESTUrl` property.

All our code that performs REST services invocations depends on this module, thus the base REST URL can be configured in a single place and injected throughout the code, as in the following code example:

src/main/webapp/resources/js/app/models/event.js

```
/**
 * Module for the Event model
 */
define([
    'configuration',
    'backbone'
], function (config) {
    /**
     * The Event model class definition
     * Used for CRUD operations against individual events
     */
    var Event = Backbone.Model.extend({
        urlRoot: config.baseUrl + 'rest/events' // the URL for performing CRUD
operations
    });
    // export the Event class
    return Event;
});
```

The prefix is used in a similar fashion by all the other modules that perform REST service invocations. You don't need to do anything right now, because the code we created in the user front end tutorial was written like this originally. Be warned, if you have a mobile web application that uses any relative URLs, you will need to refactor them to include some form of URL configuration.

Install Hybrid Mobile Tools and CordovaSim

Hybrid Mobile Tools and CordovaSim are not installed as part of JBoss Developer Studio yet. They can be installed from JBoss Central as shown below:

1. To install these plug-ins, drag the following link into JBoss Central: https://devstudio.jboss.com/central/install?connectors=org.jboss.tools.aerogear.hybrid. Alternatively, in JBoss Central select the **Software/Update** tab. In the **Find** field, type **JBoss Hybrid Mobile Tools** or scroll through the list to locate **JBoss Hybrid Mobile Tools + CordovaSim**. Select the corresponding check box and click **Install**.

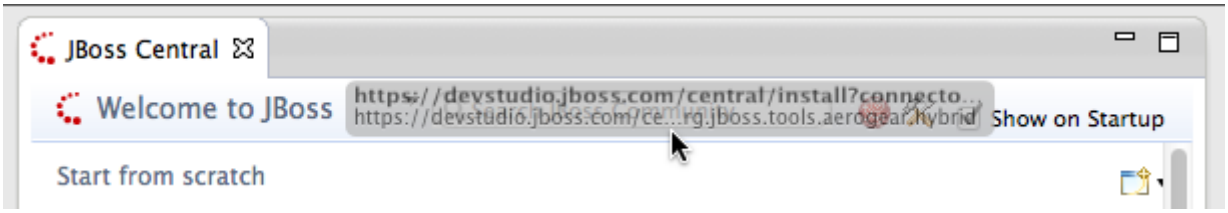

Figure 2. Start the Hybrid Mobile Tools and CordovaSim Installation Process with the Link

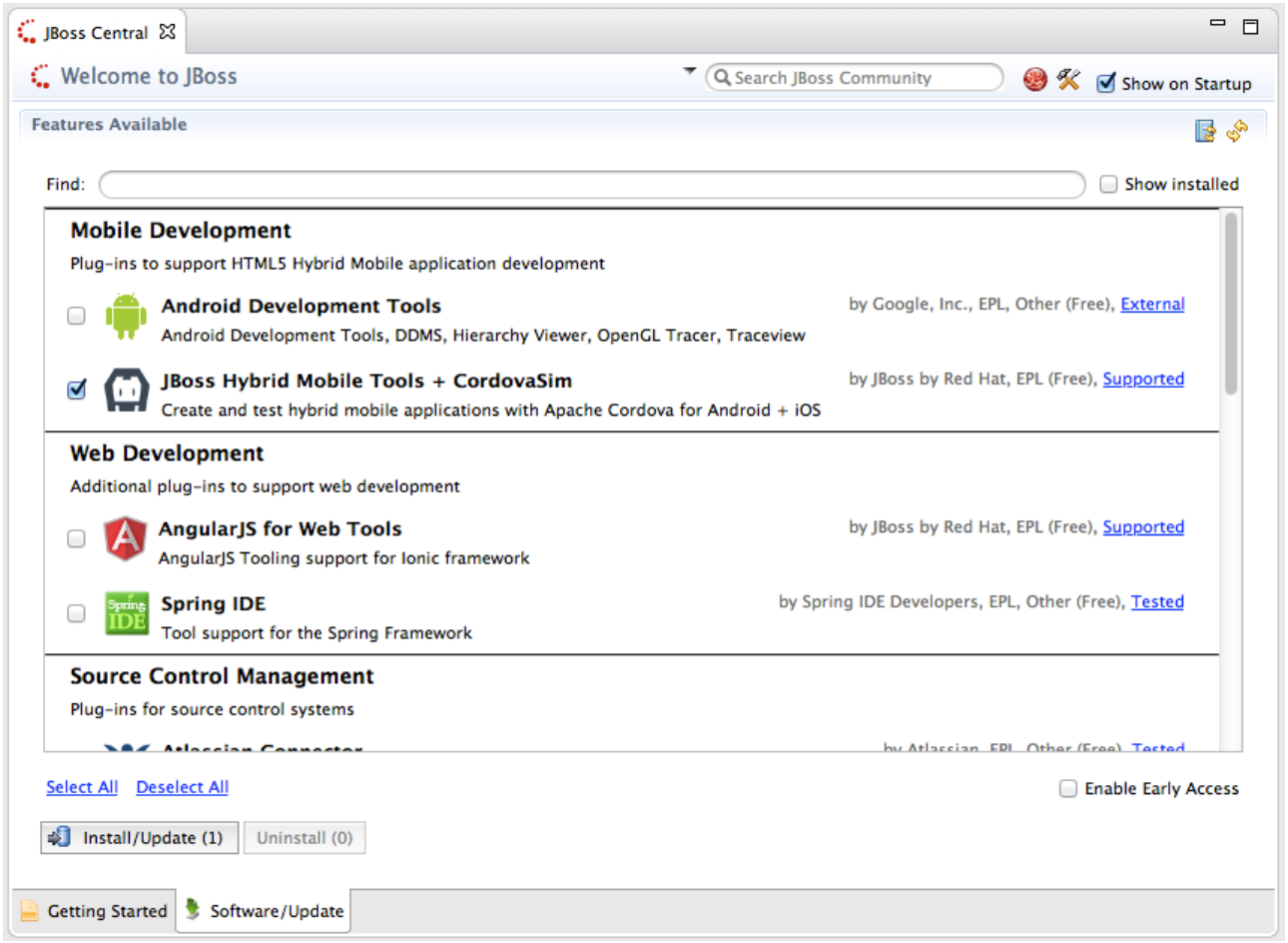

Figure 3. Find Hybrid Mobile Tools and CordovaSim in JBoss Central Software/Update Tab

2. In the **Install** wizard, ensure the check boxes are selected for the software you want to install and click **Next**. It is recommended that you install all of the selected components.
3. Review the details of the items listed for install and click Next. After reading and agreeing to the license(s), click **I accept the terms of the license agreement(s)** and click **Finish**. The **Installing Software** window opens and reports the progress of the installation.
4. During the installation process you may receive warnings about installing unsigned content. If this is the case, check the details of the content and if satisfied click **OK** to continue with the installation.

Figure 4. Warning Prompt for Installing Unsigned Content

5. Once the installation is complete, you will be prompted to restart the IDE. Click **Yes** to restart now and **No** if you need to save any unsaved changes to open projects. Note that changes do not take effect until the IDE is restarted.

Once installed, you must inform Hybrid Mobile Tools of the Android SDK location before you can use Hybrid Mobile Tools actions involving Android.

To set the Android SDK location, click **Window** → **Preferences** and select **Hybrid Mobile**. In the **Android SDK Directory** field, type the path of the installed SDK or click **Browse** to navigate to the location. Click **Apply** and click **OK** to close the **Preferences** window.

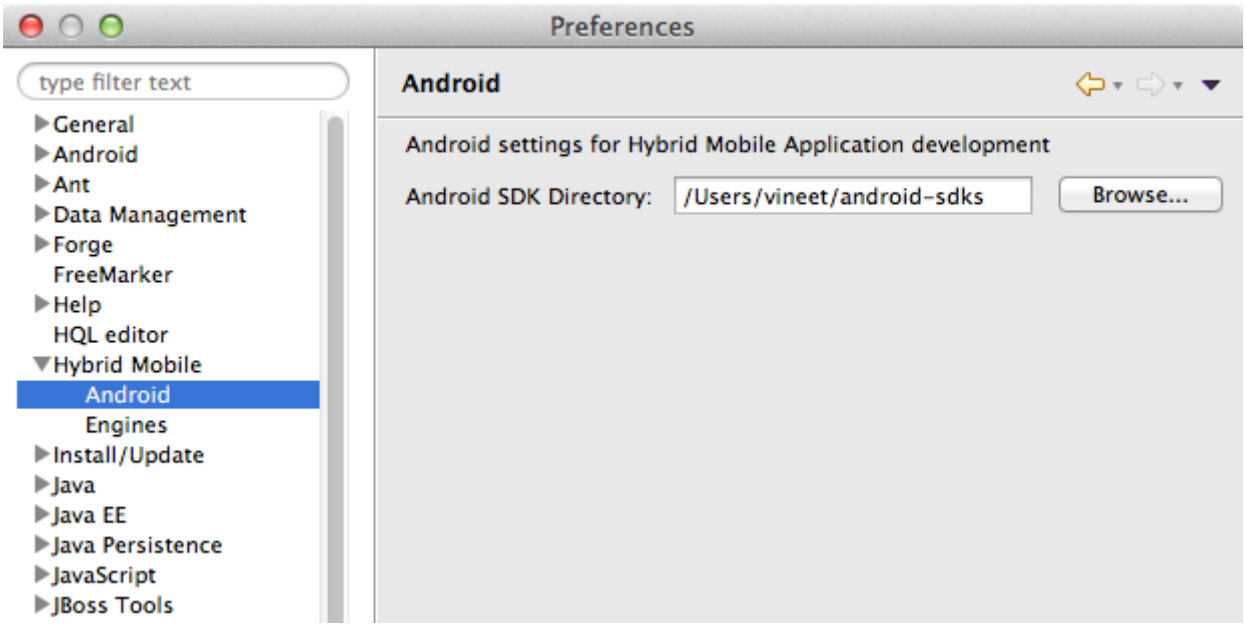

Figure 5. Hybrid Mobile Pane of Preferences Window

Creating a Hybrid Mobile project

1. To create a new Hybrid Mobile Project, click *File* → *New* → *Other* and select "Hybrid Mobile (Cordova) Application Project".

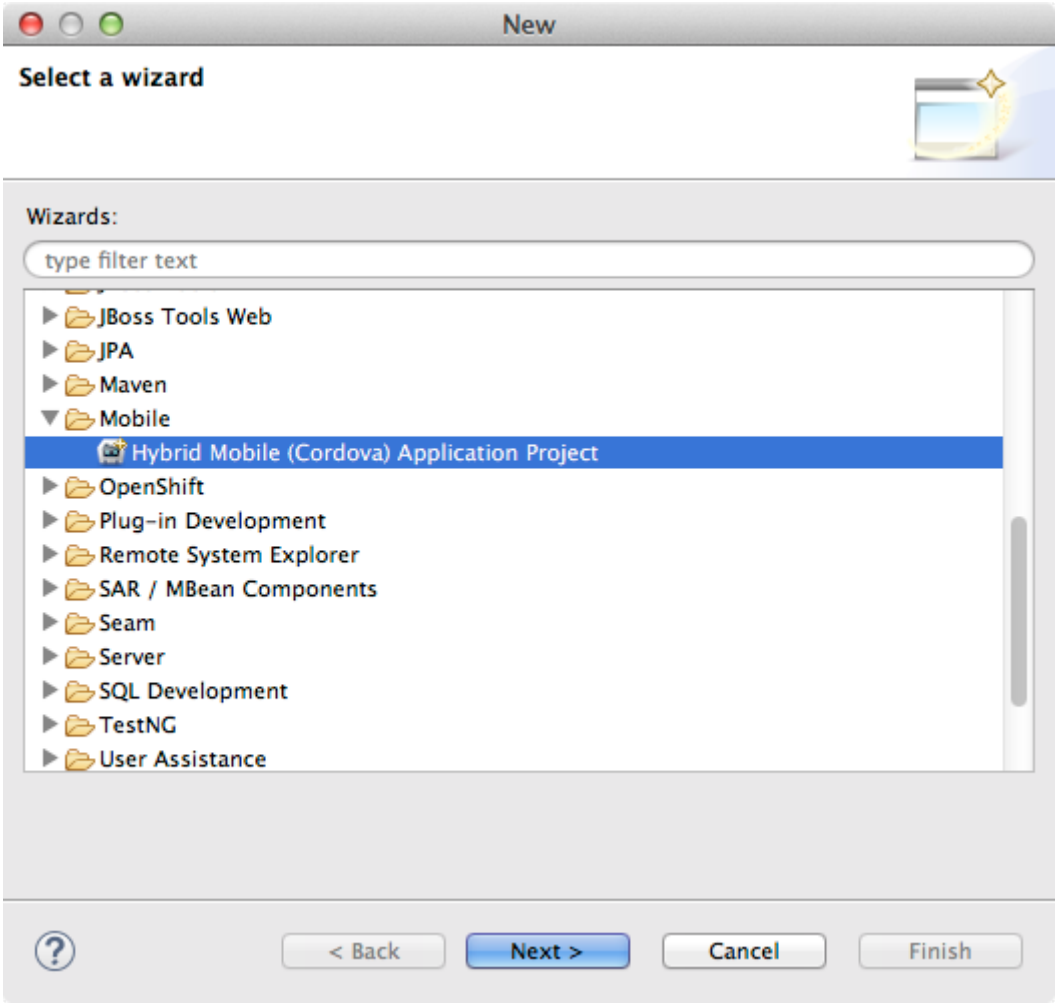

Figure 6. Starting a new Hybrid Mobile Application project

2. Enter the project information: application name, project name, package.

Project Name

TicketMonster-Cordova

Name

TicketMonster-Cordova

ID

org.jboss.examples.ticketmonster.cordova

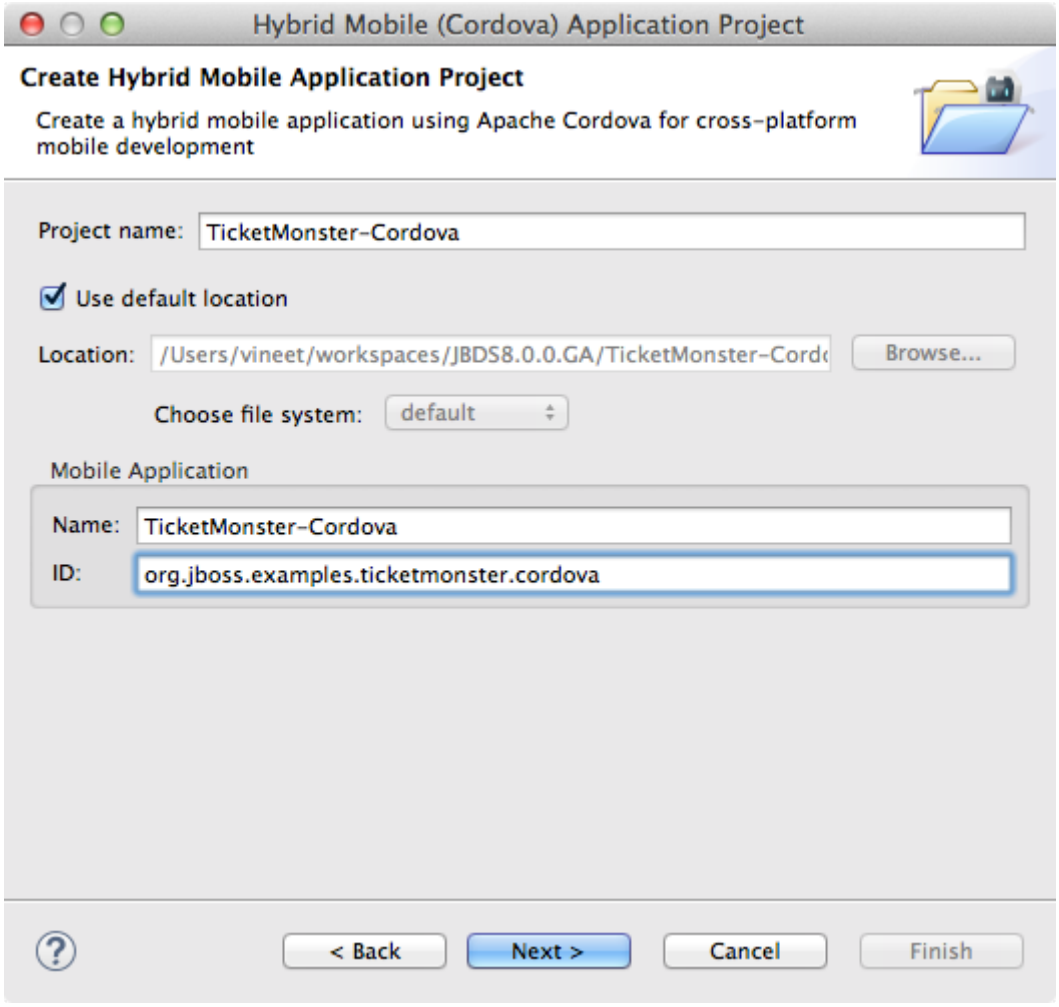

Figure 7. Creating a new Hybrid Mobile Application project

Click `Next` to choose the Hybrid Mobile engine for the project. If you have never setup a Hybrid Mobile engine in JBoss Developer Studio before, you will be prompted to download or search for engines to use. We'll click on the `Download` button to perform the former.

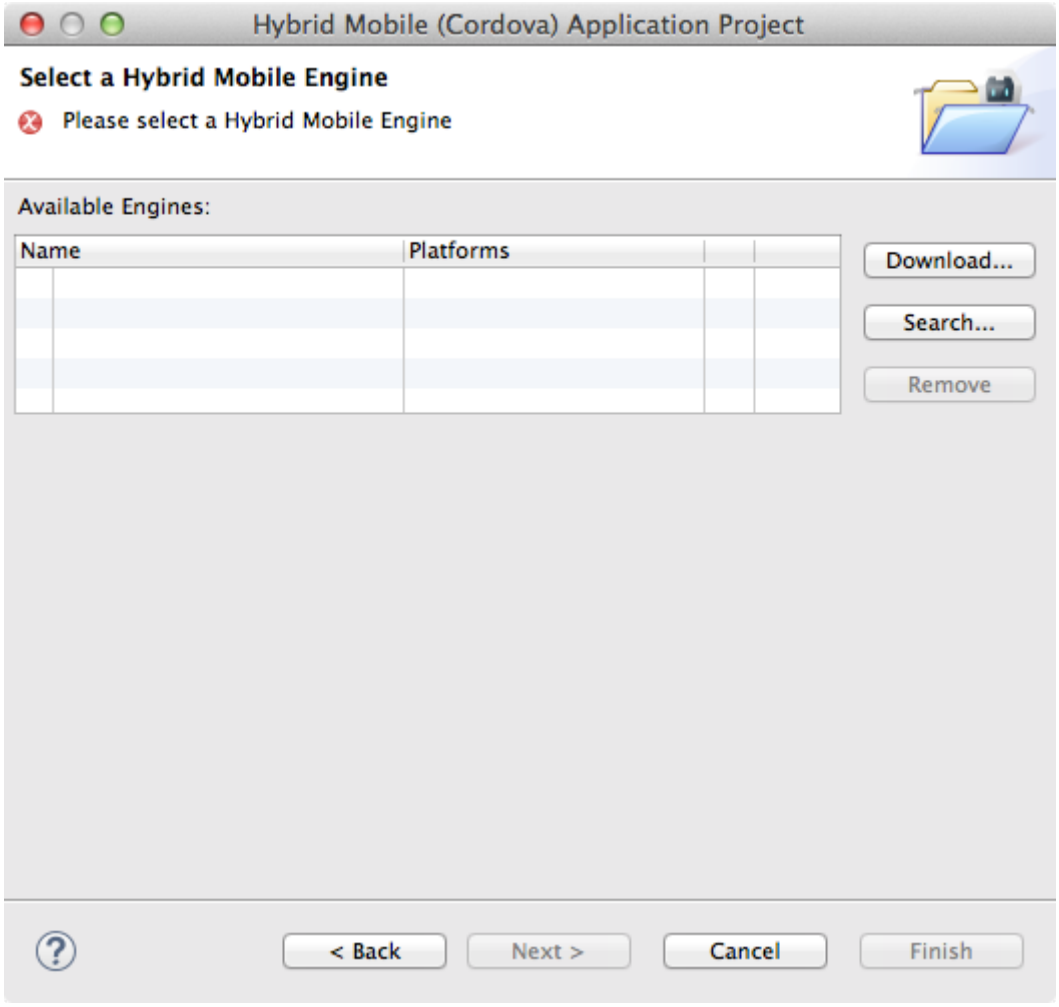

Figure 8. Setting up a Hybrid Mobile engine for the first time

You'll be prompted with a dialog where you can download all available hybrid mobile engines.

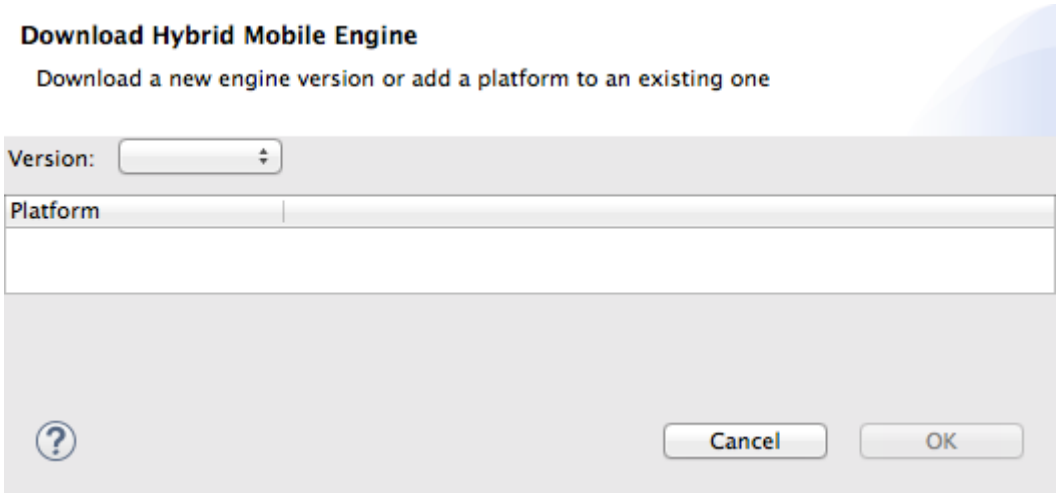

Figure 9. Choose the Hybrid Mobile engine to download

We'll choose Android and iOS variants of version 3.4.0.

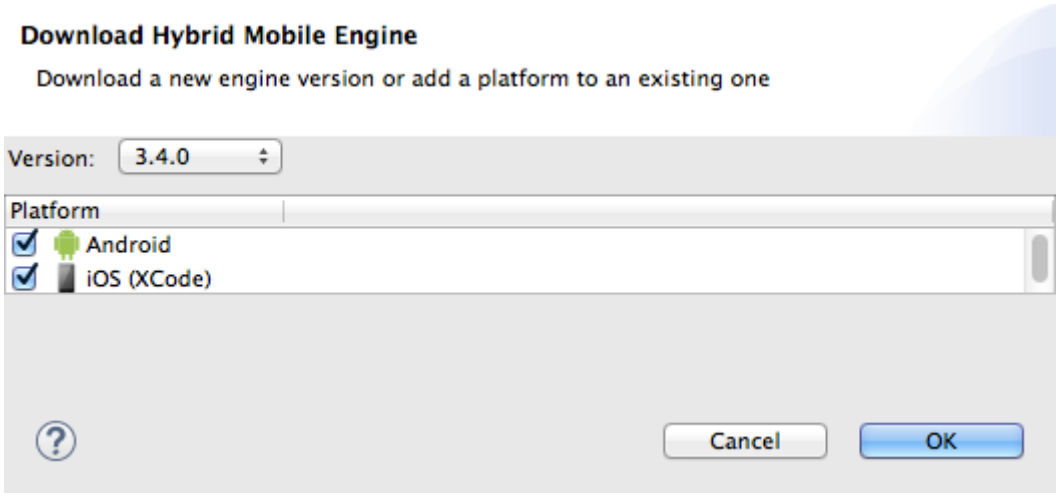

Figure 10. Select Android and iOS for 3.4.0

Now that we have downloaded and setup a hybrid mobile engine, let's use it in our project. Select the newly configured engine and click `Next`.

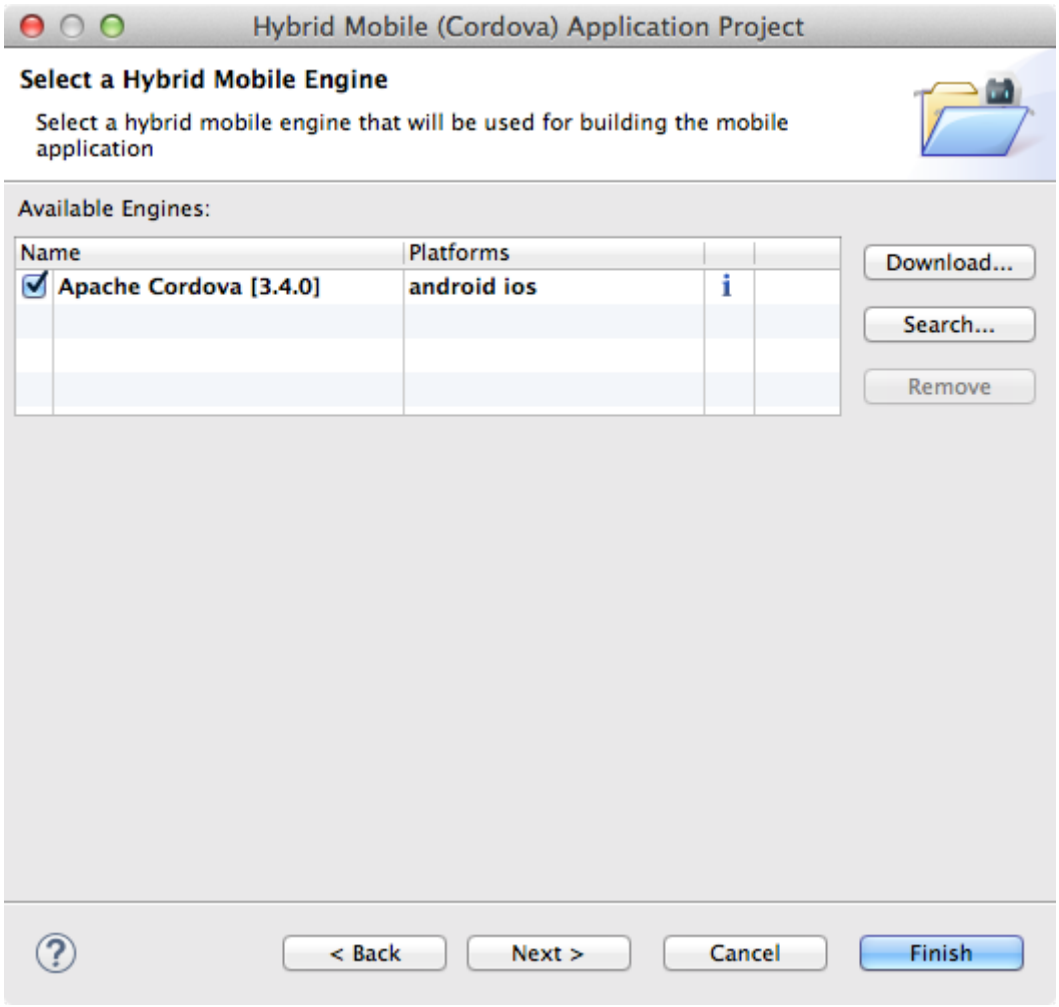

Figure 11. Creating a new Hybrid Mobile Application project

We will now be provided the opportunity to add Cordova plug-ins to our project.

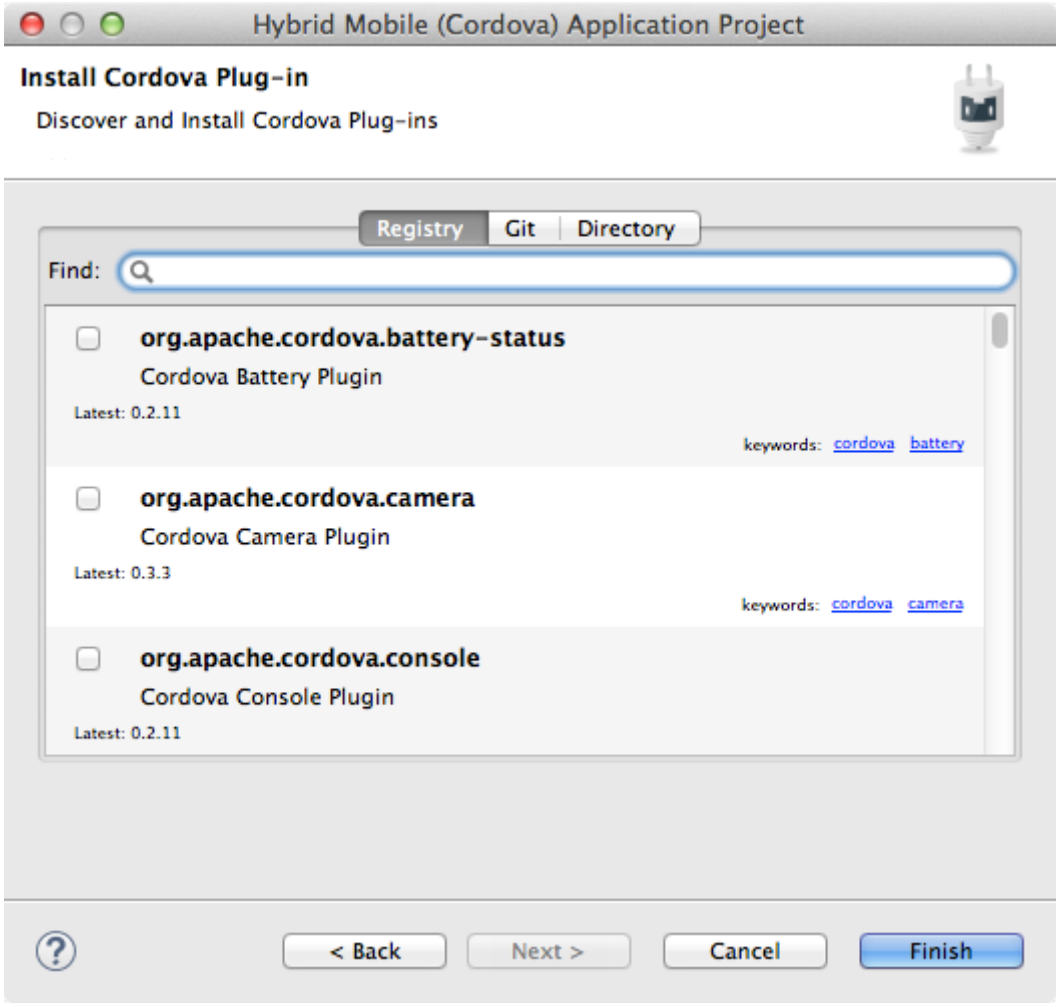

Figure 12. Adding Cordova plugins when a new Hybrid Mobile Application project

We will be using the Status Bar plugin from Cordova, to ensure that the status bar on iOS 7 does not overlap the UI. The Device plugin will be used to obtain device information for use in device detection. We'll also use the Notification plugin to display alerts and notifications to the end-user using the native mobile UI. We'll proceed to add the required Cordova plugins to the project.

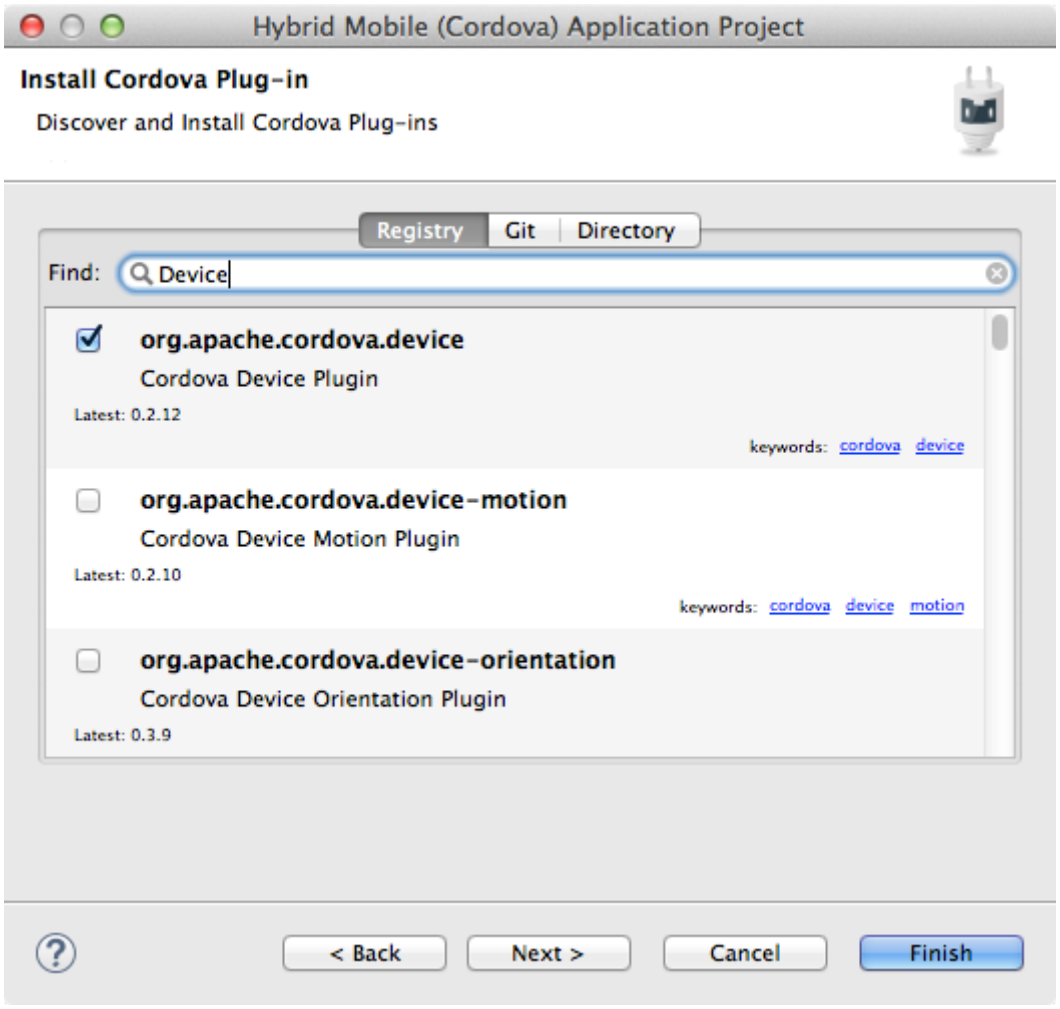

Figure 13. Add Cordova Device plugin

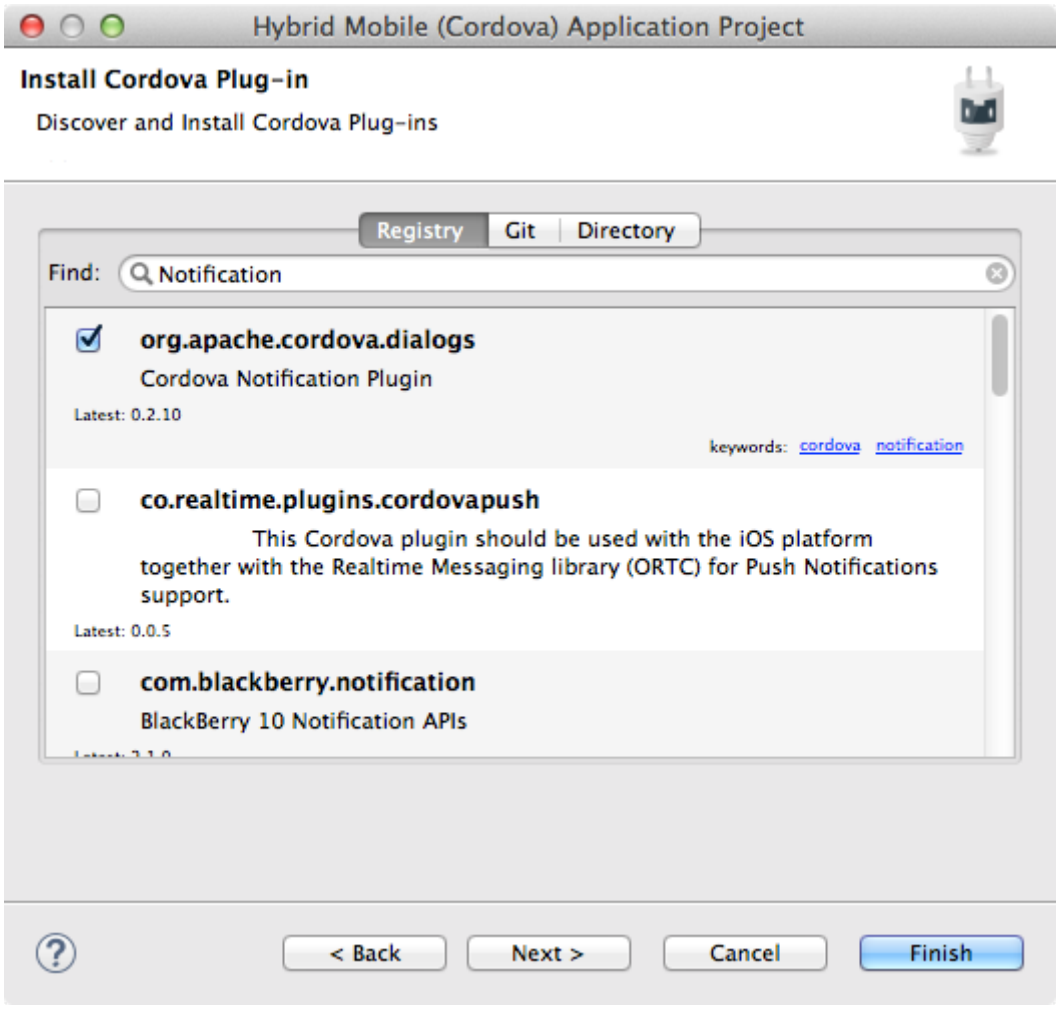

Figure 14. Add Cordova Notification plugin

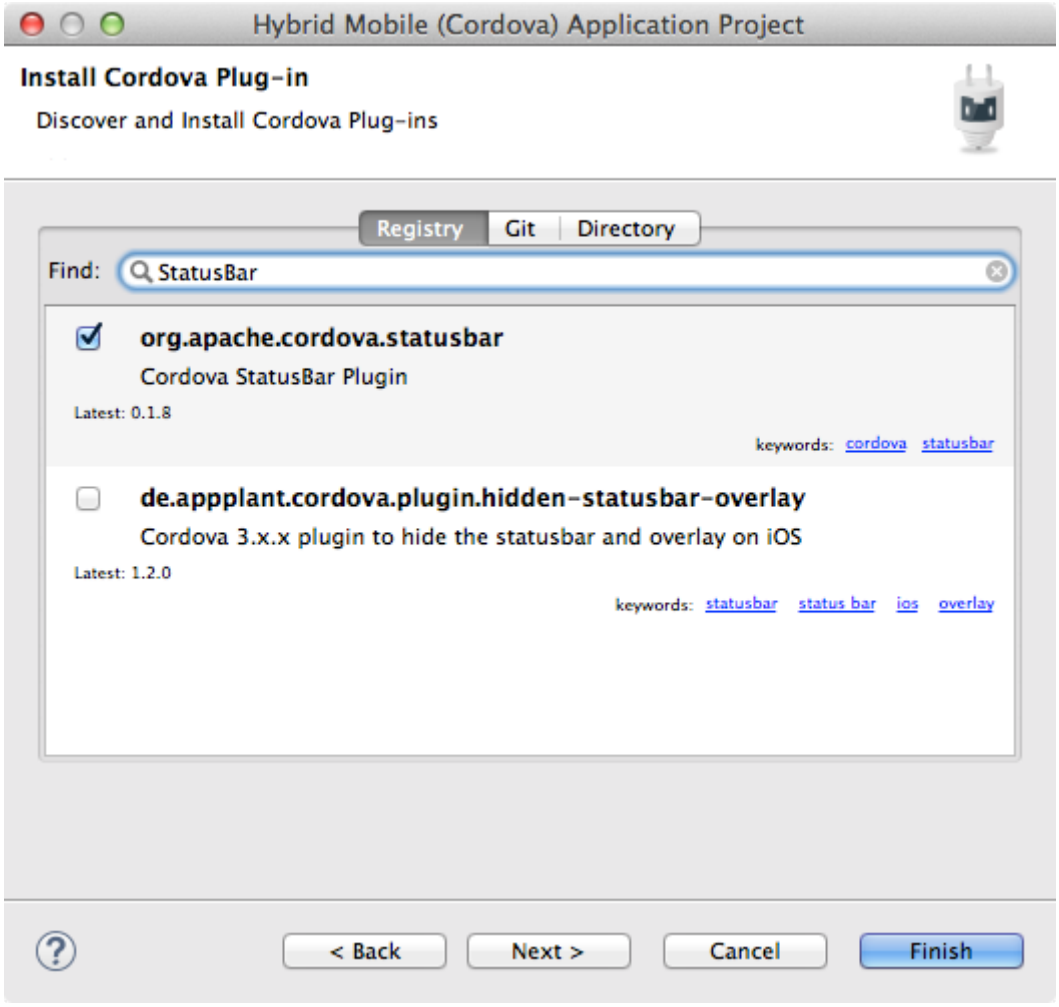

Figure 15. Add Cordova StatusBar plugin

Let's proceed to add these, by searching for them and selecting them. Click `Next` once you have finished selecting the necessary plug-ins. We will now confirm the plugins to be added to the project. Click `Finish` to create the new Hybrid Mobile application project.

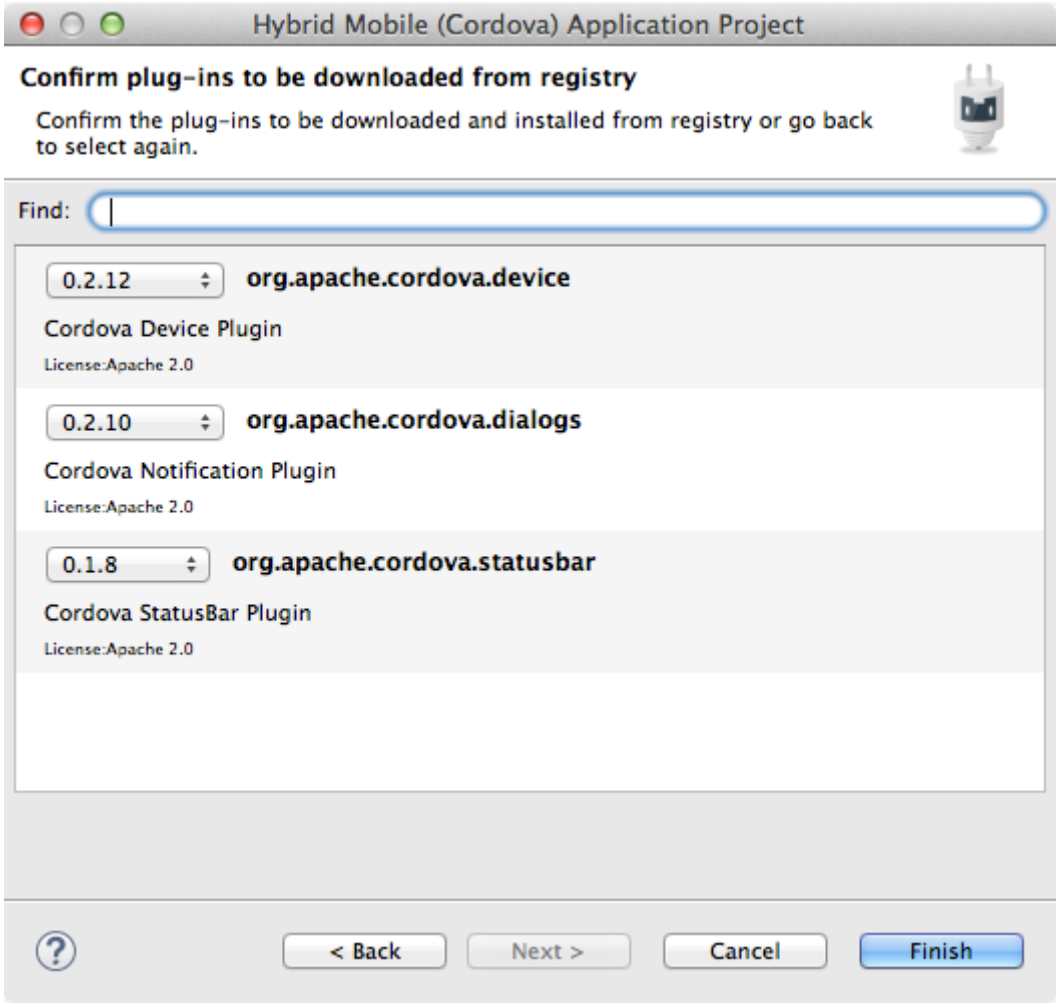

Figure 16. Confirm plugins to add

Once you have finished creating the project, navigate to the `www` directory, that will contain the HTML5 code of the application. Since we are reusing the TicketMonster code you can simply replace the `www` directory with a symbolic link to the `webapp` directory of TicketMonster; the `config.xml` file and `res` directory would need to be copied over to the `webapp` directory of TicketMonster. Alternatively, you can copy the code of TicketMonster and make all necessary changes there (however, in that case you will have to maintain the code of the application in both places); on Windows, it would be easier to do this.

```
$ cp config.xml $TICKET_MONSTER_HOME/demo/src/main/webapp
$ cp res $TICKET_MONSTER_HOME/demo/src/main/webapp
$ cd ..
$ rm -rf www
$ ln -s $TICKET_MONSTER_HOME/demo/src/main/webapp www
```

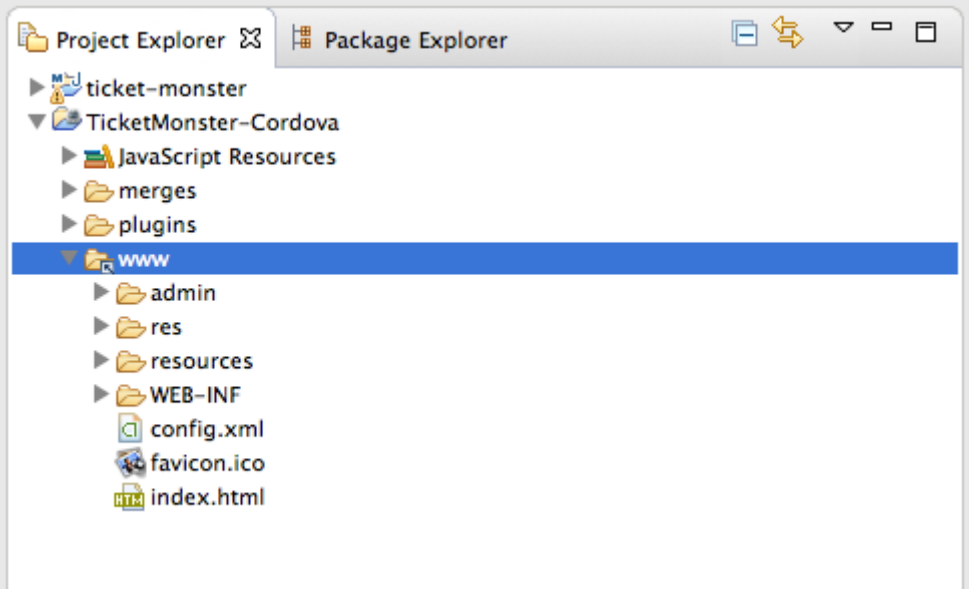

Figure 17. The result of linking www to the webapp directory

The Hybrid Mobile tooling requires that the cordova.js file be loaded in the application's start page. Since we do not want to load this file in the existing `index.html` file, we shall create a new start page to be used only by the Cordova app.

src/main/webapp/mobileapp.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Ticket Monster</title>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1, user-
scalable=no"/>

    <script type="text/javascript" src="resources/js/libs/modernizr-
2.8.3.min.js"></script>
    <script type="text/javascript" src="resources/js/libs/require.js"
            data-main="resources/js/configurations/loader"></script>
</head>
<body>
</body>
</html>
```

Let's now modify the Hybrid Mobile project configuration to use this page as the application start page. Additionally, we will add our REST service URL to the domain whitelist in the config.xml file (you can use `"*"` too, for simplicity, during development) :

src/main/webapp/config.xml

```
<?xml version="1.0" encoding="utf-8"?>
<widget xmlns="http://www.w3.org/ns/widgets" xmlns:gap="http://phonegap.com/ns/1.0"
    id="org.jboss.examples.ticketmonster.cordova" version="2.0.0">
```

```
    ...

    <!-- The application start page -->
    <content src="mobileapp.html" />

    <!--
    Add the TicketMonster cloud app to the domain whitelist.
    Domains are assumed blocked unless set otherwise.
     -->
    <access origin="http://ticketmonster-jdf.rhcloud.com"/>

    ...

</widget>
```

Next, we need to load the library in the application. We will create a separate module, that will load the rest of the mobile application, as well as the Apache Cordova JavaScript library for Android. We also need to configure a base URL for the application. For this example, we will use the URL of the cloud deployment of TicketMonster.

src/main/webapp/resources/js/configurations/hybrid.js

```
// override configuration for RESTful services
var TicketMonster = {
    config:{
        baseRESTUrl:"http://ticketmonster-jdf.rhcloud.com/"
    }
};

require(['../../../cordova'], function() {

    var bootstrap = {
        initialize: function() {
            document.addEventListener('deviceready', this.onDeviceReady, false);
        },
        onDeviceReady: function() {
            // Detect if iOS 7 or higher and disable overlaying the status bar
            if(window.device && window.device.platform.toLowerCase() == "ios" &&
                parseFloat(window.device.version) >= 7.0) {
                StatusBar.overlaysWebView(false);
                StatusBar.styleDefault();
                StatusBar.backgroundColorByHexString("#e9e9e9");
            }
            // Load the mobile module
            require (["mobile"]);
        }
    };

    bootstrap.initialize();
});
```

NOTE	We'll use the OpenShift hosted version of the TicketMonster application because it is easier to access in all environments - the smartphone simulators and emulators can also access it with relatively little or no configuration. On the other hand, accessing the locally running JBoss EAP instance may require some complicated network configuration, especially if the instance needs to be opened up to the internet for access from smartphones through a mobile internet link.

The above snippet of code contains a device-specific check for iOS 7.

Finally, we'll configure the loader module launched from `mobileapp.html` to use the above defined `hybrid` module:

src/main/webapp/resources/js/configurations/loader.js

```
//detect the appropriate module to load
define(function () {

    /*
     A simple check on the client. For touch devices or small-resolution screens)
     show the mobile client. By enabling the mobile client on a small-resolution
screen
     we allow for testing outside a mobile device (like for example the Mobile Browser
     simulator in JBoss Tools and JBoss Developer Studio).
     */

    var environment;

    if (document.URL.indexOf("mobileapp.html") > -1) {
        environment = "hybrid";
    }
    else if (Modernizr.touch || Modernizr.mq("only all and (max-width: 768px)")) {
        environment = "mobile";
    } else {
        environment = "desktop";
    }

    require([environment]);
});
```

In the above code snippet, we detect if the URL of the page contains `mobileapp.html` or not, and then proceed to activate the `hybrid` module if so. Since Apache Cordova is configured to use `mobileapp.html` as the application start page, the desired objective is achieved. This way, we avoid loading the `mobile` or `desktop` modules that do not have any logic in them to detect the `deviceready` event of Cordova.

The final step will involve adjusting `src/main/webapp/resources/js/configurations/loader.js` to load this module when running on Android, using the query string we have already configured in the project. We'll also tweak `src/main/webapp/resources/js/app/utilities.js` to use the Notification plugin to display alerts in the context of a Hybrid Mobile app.

src/main/webapp/resources/js/configurations/loader.js

```
//detect the appropriate module to load
define(function () {

    /*
     A simple check on the client. For touch devices or small-resolution screens)
     show the mobile client. By enabling the mobile client on a small-resolution
screen
     we allow for testing outside a mobile device (like for example the Mobile Browser
     simulator in JBoss Tools and JBoss Developer Studio).
     */

    var environment;

    if (document.URL.indexOf("mobileapp.html") > -1) {
        environment = "hybrid";
    }
    else if (Modernizr.touch || Modernizr.mq("only all and (max-width: 768px)")) {
        environment = "mobile";
    } else {
        environment = "desktop";
    }

    require([environment]);
});
```

We'll now examine the `displayAlert` function in the utilities object. It is set to use the Notification plugin when available:

src/main/webapp/resources/js/app/utilities.js

```
...
    // utility functions for rendering templates
    var utilities = {
        ...
        applyTemplate:function (target, template, data) {
            return target.empty().append(this.renderTemplate(template, data));
        },
        displayAlert: function(msg) {
            if(navigator.notification) {
                navigator.notification.alert(msg);
            } else {
                alert(msg);
            }
        }
    };
...
```

The function automatically works in non-mobile environments due to the absence of the

`navigator.notification` object in such environments.

Run the hybrid mobile application

You are now ready to run the application. The hybrid mobile application can be run on devices and simulators using the Hybrid Mobile Tools.

Run on an Android device or emulator

NOTE

What do you need for Android?

For running on an Android device or emulator, you need to install the Android Developer Tools, which require an Eclipse instance (JBoss Developer Studio could be used), and can run on Windows (XP, Vista, 7), Mac OS X (10.5.8 or later), Linux (with GNU C Library - glibc 2.7 or later, 64-bit distributions having installed the libraries for running 32-bit applications).

You must have Android API 17 or later installed on your system to use the **Run on Android Emulator** action.

To run the project on a device, in the **Project Explorer** view, right-click the project name and click **Run As → Run on Android Device**. This option calls the external Android SDK to package the workspace project and run it on an Android device if one is attached. Note that the Android SDK must be installed and the IDE correctly configured to use the Android SDK for this option to execute successfully.

To run the project on an emulator, in the **Project Explorer** view, right-click the project name and click **Run As → Run on Android Emulator**.

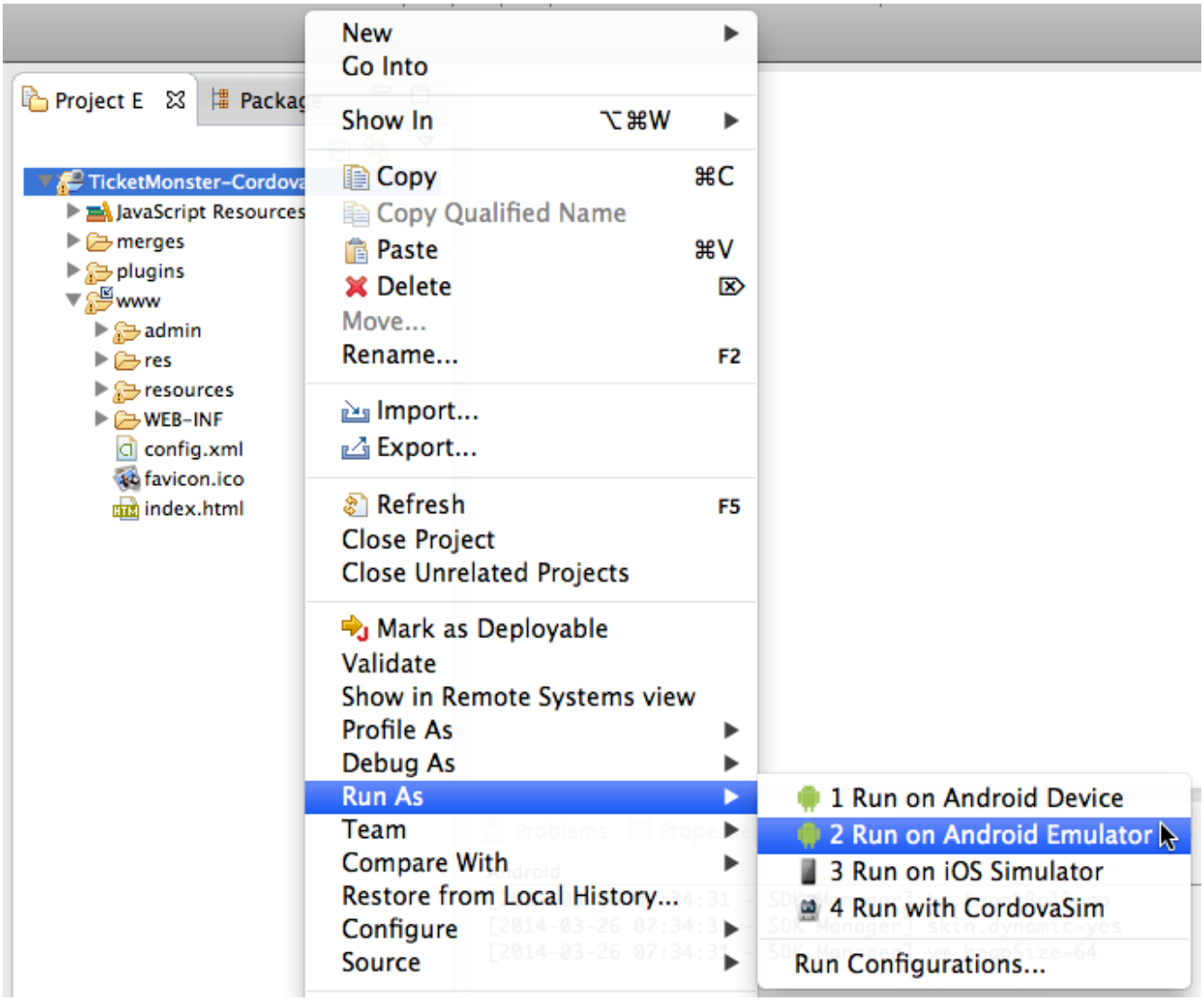

Figure 18. Running the application on an Android emulator

This requires that you create an Android AVD to run the application in a virtual device.

Once deployed, the application is now available for interaction in the emulator.

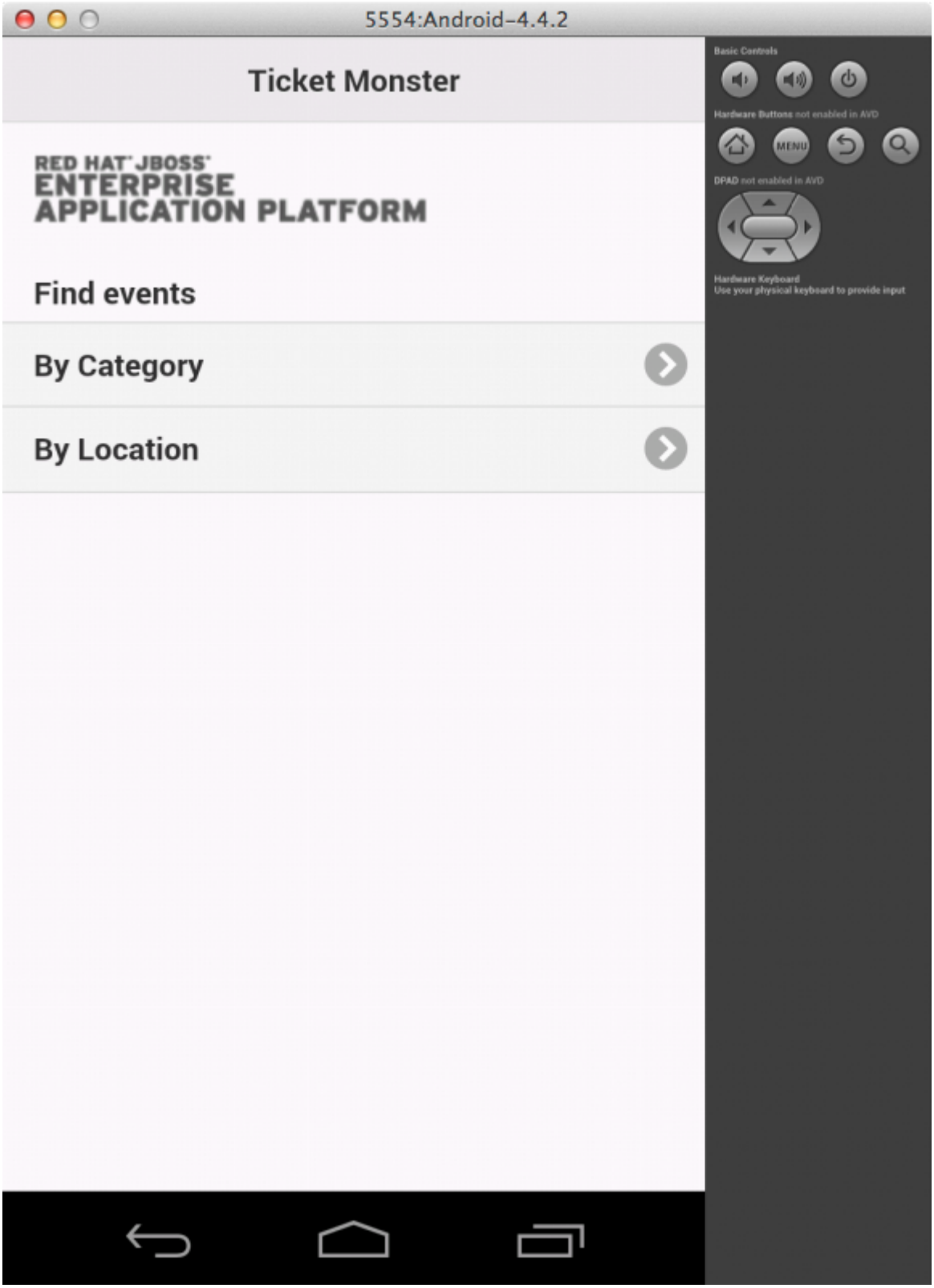

Figure 19. The app running on an Android AVD

Run on an iOS Simulator

NOTE

What do you need for iOS?

This option is only displayed when using OS X operating systems, for which the iOS Simulator is available. You must install **Xcode 4.5+** which includes the **iOS 6 SDK**. You must also install a Simulator for iOS 5.x or higher, to run the project on a simulator. Depending on various Cordova plugins that you may use, you may need

higher versions of simulators to run your applications.

In the Project Explorer view, right-click the project name and click **Run As → Run on iOS Emulator**.

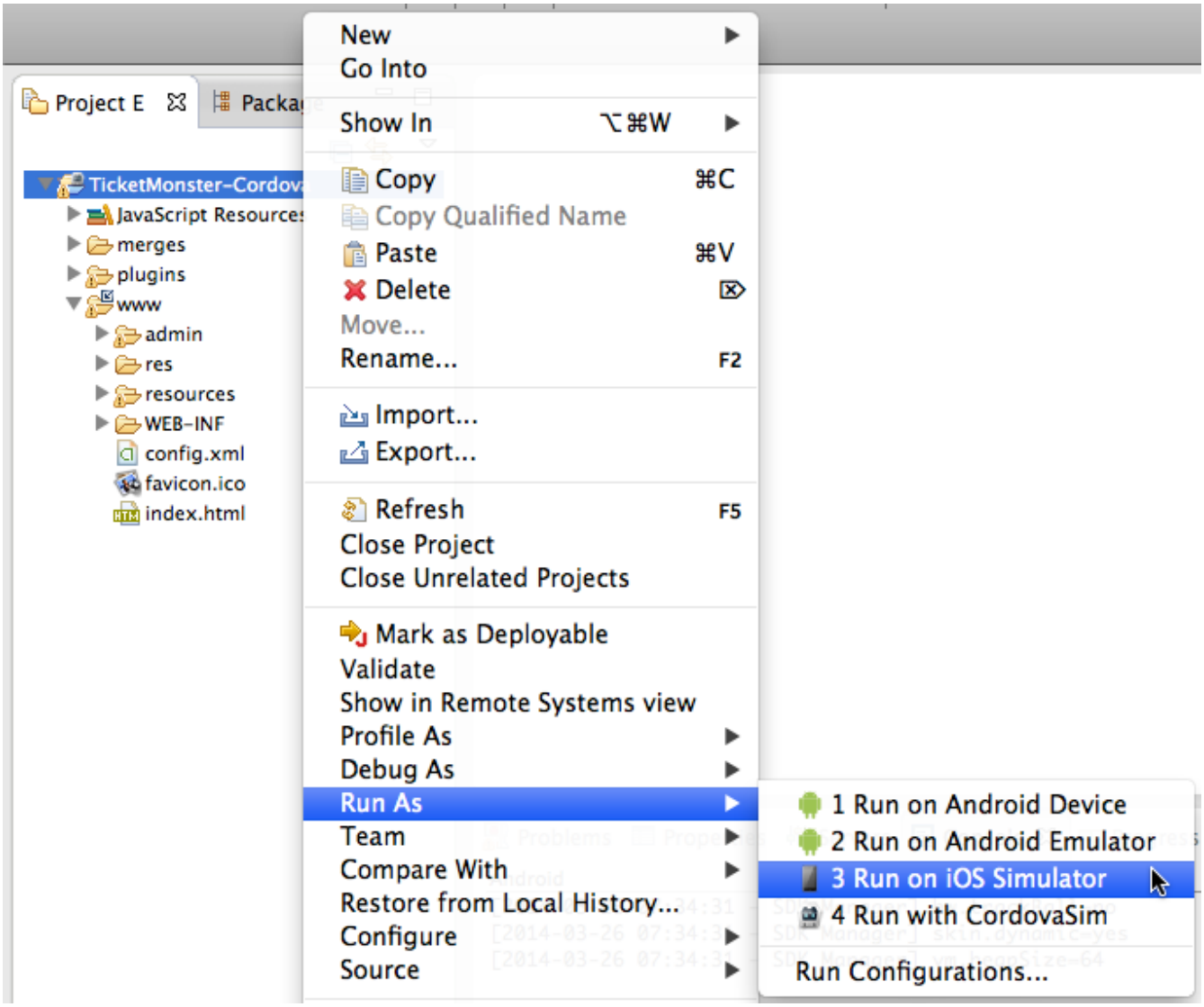

Figure 20. Running the application on an iOS simulator

This option calls the external iOS SDK to package the workspace project into an XCode project and run it on the iOS Simulator.

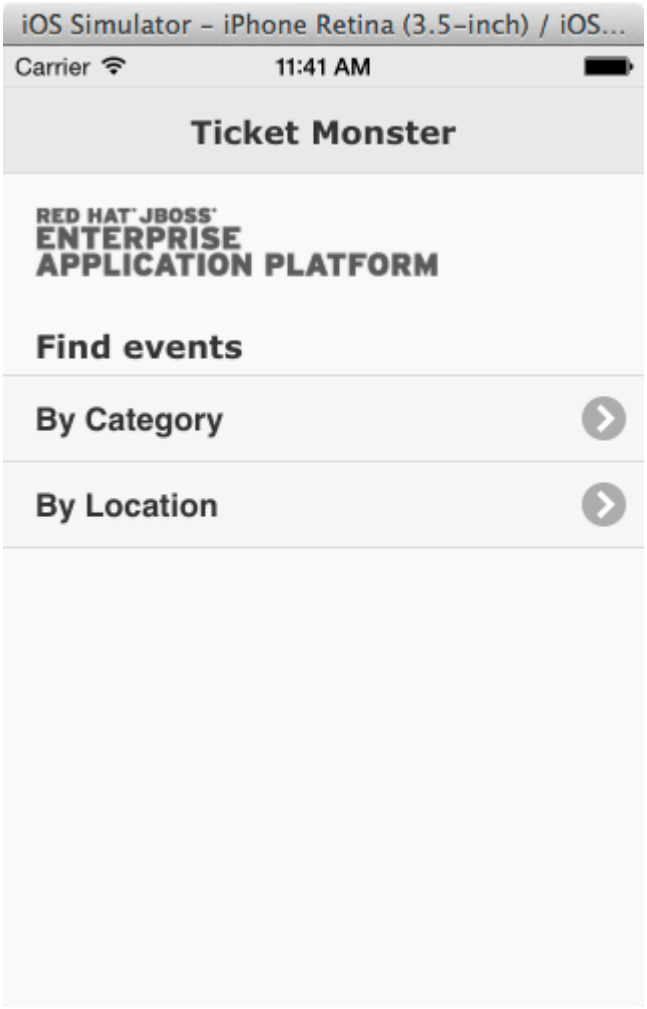

Figure 21. The app running on an iOS Simulator

Run on CordovaSim

CordovaSim allows you to run your hybrid mobile applications in your local workspace. You can develop the application without requiring a deployment to a real device or even to emulators and simulators to realize your application's behavior. There are some limitations on what you can achieve with CordovaSim, for instance, some Cordova plugins may not work with CordovaSim. But for the most part, you get to experience a faster development cycle.

In the Project Explorer view, right-click the project name and click **Run As → Run with CordovaSim**. This opens the application in CordovaSim, which is composed of a BrowserSim simulated device and a device input panel.

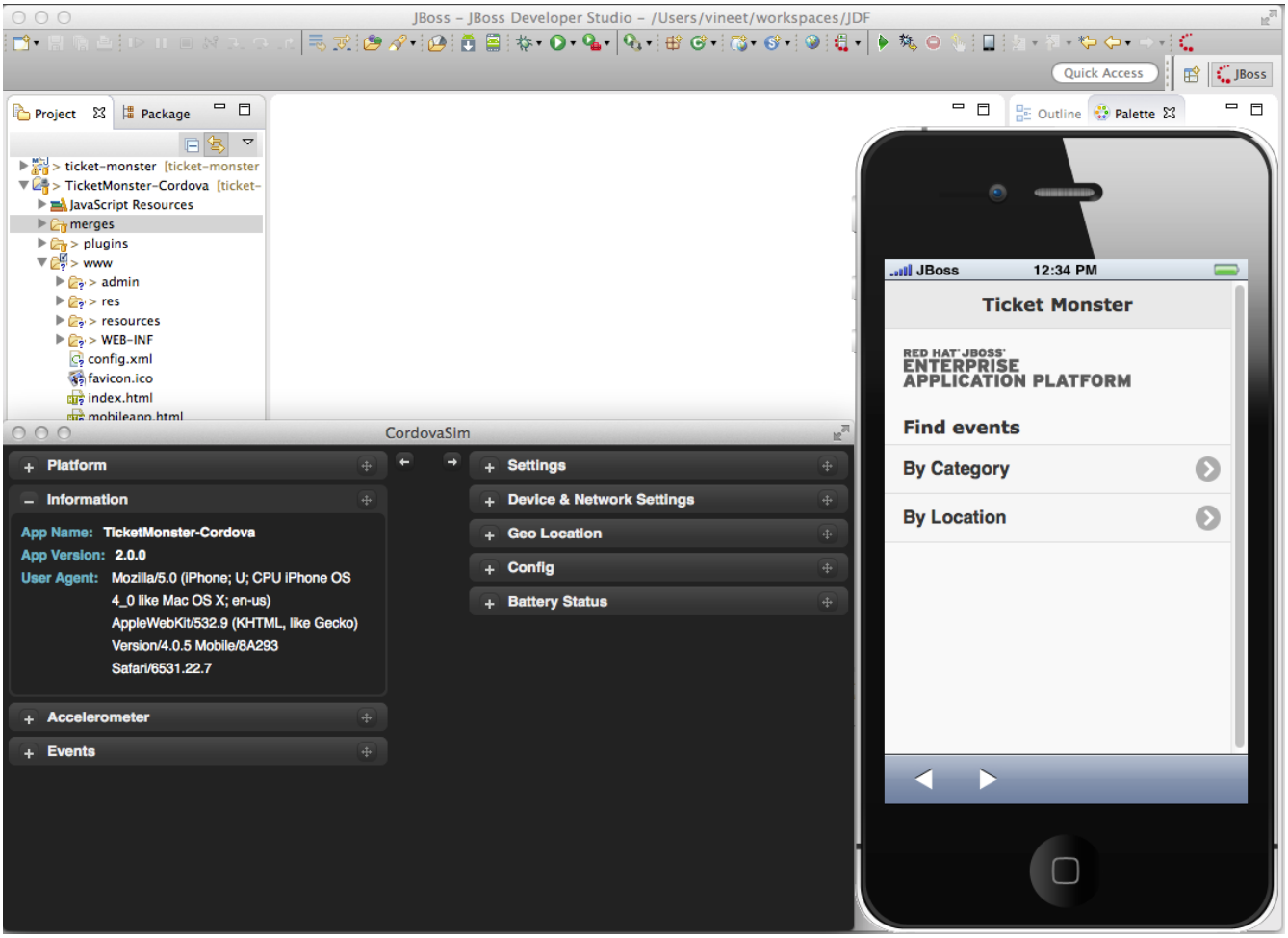

Figure 22. The app running on CordovaSim

Conclusion

This concludes our tutorial for building a hybrid application with Apache Cordova. You have seen how we have turned a working HTML5 web application into one that can run natively on Android and iOS.