Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 11

Dr Nick Hayward

Contents

Cordova app - Plugins

intro

- developing custom plugins for Cordova, and by association your apps
  - *a useful skill to learn and develop*
- it is not always necessary to develop a custom plugin
  - *to produce a successful project or application*
  - *dependent upon the requirements and constraints of the project itself*
- use and development of Cordova plugins is not a recent addition
- with the advent of Cordova 3 plugins have started to change
  - *introduction of Plugman and the Cordova CLI helped this change*
- plugins are now more prevalent in their usage and scope
  - *their overall implementation has become more standardised*

Cordova app - Plugins

structure and design - part 1

- as we start developing our custom plugins
  - *makes sense to understand the structure and design of a plugin*
- what makes a collection of files a plugin for use within our applications
- we can think of a plugin as a set of files
  - *as a group extend or enhance the capabilities of a Cordova application*
- already seen a number of examples of working with plugins
  - *each one installed using the CLI*
  - *its functionality exposed by a JavaScript interface*
- a plugin could interact with the host application without developer input
- majority of plugin designs provide access to the underlying API
  - *provide additional functionality for an application*

Cordova app - Plugins

structure and design - part 2

- a plugin is, therefore, a collection of contiguous files
  - *packaged together to provide additional functionality and options for a given application*
- a plugin includes a `plugin.xml` file
  - *describes the plugin*
  - *informs the CLI of installation directories for the host application*
  - *where to copy and install the plugin's components*
  - *includes option to specify files per installation platform*
- a plugin also needs at least one JavaScript source file
  - *file is used within the plugin*
  - *helps define methods, objects, and properties required by the plugin*
  - *source file is used to help expose the plugins API*

Cordova app - Plugins

structure and design - part 3

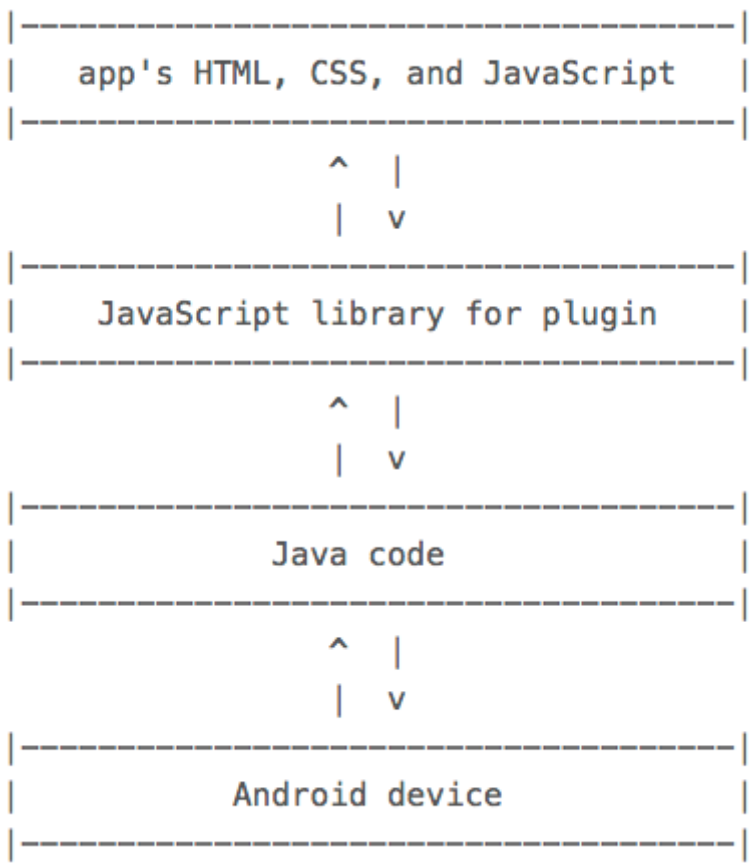
- within our plugin structure
  - *easily contain all of the required JS code in one file*
  - *divide logic and requirements into multiple files...*
- structure depends on plugin complexity and dependencies
- eg: we could bundle other jQuery plugins, handlebars.js. maps functionality...
- beyond the requirement for a `plugin.xml` and plugin JS source file
  - *plugin's structure can be developer specific*
- for most plugins, we will add
  - *native source code files for each supported mobile platform*
  - *may also include additional native libraries*
  - *any required content such as stylesheets, images, media...*

Cordova app - Plugins

architecture - Android

- we can choose to support one or multiple platforms for an application
- consider a plugin for Android
  - *we can follow a useful, set pattern for its development*
- android plugin pattern
  - *application's code makes a call to the specific JS library, API*
  - *plugin's JS then sends a request down the chain*
  - *request sent to specific Java code written for supported versions of Android*
  - *Java code communicates with the native device*
  - *upon success, any return is then handled*
  - *return passed up the plugin chain to the app's code for Cordova*
- bi-directional flow from the Cordova app to the native device, and back again

Image - Cordova Plugin Architecture - Android



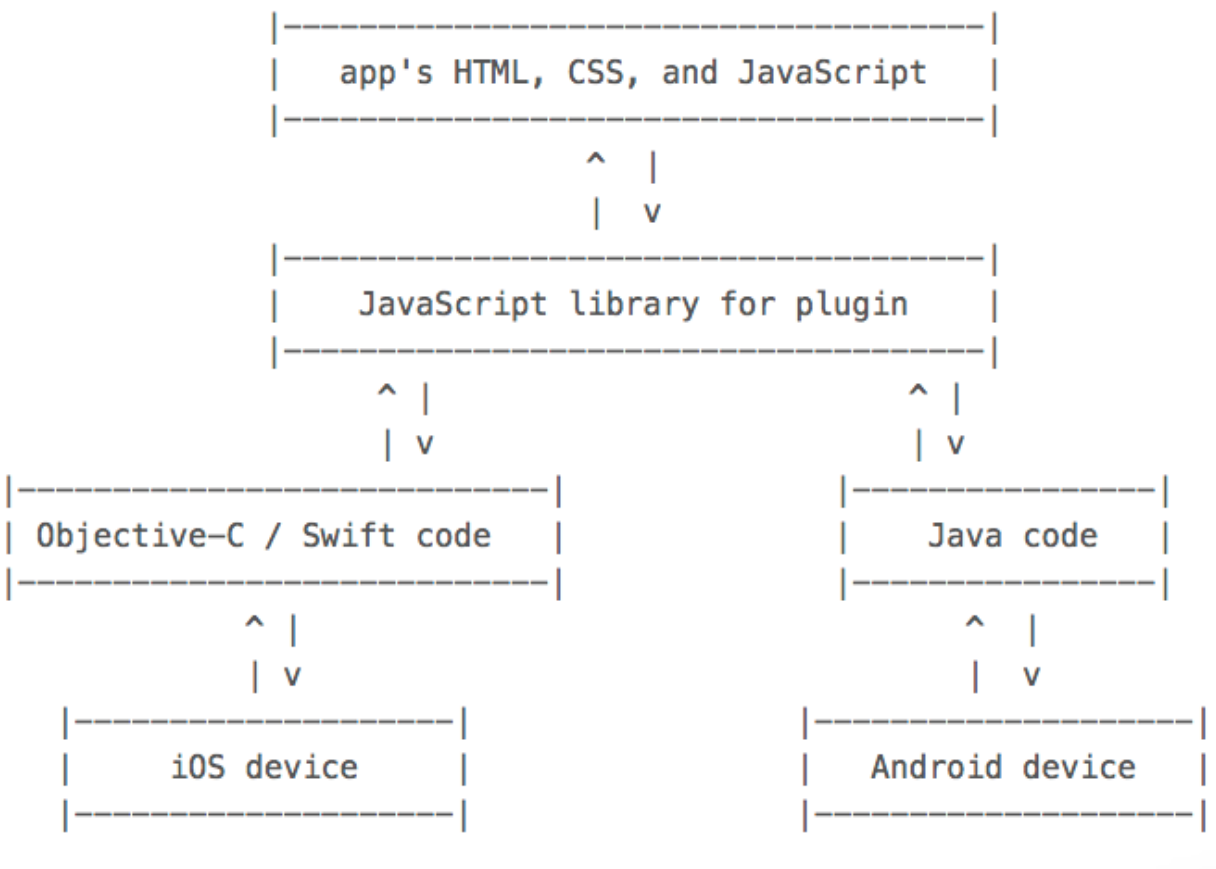
Cordova Plugin Architecture - Android

Cordova app - Plugins

architecture - cross-platform

- update our architecture to support multiple platforms within our plugin design
- maintain the same exposed app content
  - *again using HTML, CSS, and JavaScript*
- maintain the same JavaScript library, API for our plugin
- add some platform specific code and logic for iOS devices
  - *add native Objective-C/Swift code and logic*
- inherent benefit of this type of plugin architecture
  - *the plugin's JavaScript library*
- as we support further platforms
  - *plugin's JavaScript library should not need to change per platform*

Image - Cordova Plugin Architecture - Cross-platform



Cordova Plugin Architecture - Cross-platform

Cordova app - Plugins

Plugman utility - part 1

- for many plugin tasks in Cordova we can simply use the CLI tool
- we can also use the recent *Plugman* tool
  - *useful for the platform-centric workflow*
- *Plugman* tool helps us develop custom plugins
  - *helps create simple, initial template for building plugins*
  - *add or remove a platform from a custom plugin*
  - *add users to the Cordova plugin registry*
  - *publish our custom plugin to the Cordova plugin registry*
  - *likewise, unpublish our custom plugin from the Cordova plugin registry*
  - *search for plugins in the Cordova plugin registry*

Cordova app - Plugins

Plugman utility - part 2

- need to install *Plugman* for use with Cordova
  - *use NPM to install this tool*

```
npm install -g plugman
```

- OS X may need `sudo` to install
- `cd` to working directory for our new custom plugin
  - *now create the initial template*

```
plugman create --name cordova-plugin-test --plugin_id org.csteach.plugin.Test --plugin_version 0.0
```

- with this command, we are setting the following parameters for our plugin
  - *`--name` = the name of our new plugin*
  - *`--plugin_id` = sets an ID for the plugin*
  - *`--plugin_version` = sets the version number for the plugin*
- also add optional metadata, such as author or description, and path to the plugin...
- new plugin directory containing
  - *`plugin.xml`, `www` directory, `src` directory*

Cordova app - Plugins

Plugman utility - part 3

- using `plugman`, we can also add any supported platforms to our custom plugin

```
// add android
plugman platform add --platform_name android
// add ios
plugman platform add --platform_name ios
```

- command needs to run from the working directory for the custom plugin
- template creates plugin directories

```
|- plugin.xml
|- src
   |- android
      |- Test.java
|- www
   |- test.js
```

- three important files that will help us develop our custom plugin
  - *`plugin.xml` file for general definition, settings...*
  - *`Test.java` contains the initial Android code for the plugin*
  - *`test.js` contains the plugin's initial JS API*

Cordova app - Plugins

Plugman utility - part 4

- now update plugin's definition, settings in `plugin.xml` file
  - *helps us define the general structure of our plugin*
- within the `<plugin>` element, we can identify our plugin's metadata
  - *`<name>`, `<description>`, `<licence>`, and `<keywords`*
- need to clearly define and structure our JS module
  - *corresponds to a JS file for our plugin*
  - *helps expose the plugin's underlying JS API*
- `<clobbers>` element is a sub-element of `<js-module>`
  - *inserts JS object for plugin's JS API into application's window*
- update `target` attribute for `<clobbers>` adding required window value

```
<clobbers target="window.test" />
```

- now corresponds to object defined in `www/test.js` file
- exported into app's window object as `window.test`
  - *access underlying plugin API using this `window.test` object*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 1

- majority of Cordova plugins include native code
  - *for platforms such as Android, iOS, Windows Phone...*
  - *not a formal requirement for plugins*
- start by developing our custom plugin using JavaScript
  - *eg: create a custom plugin to package a JavaScript library*
  - *or a combination of libraries*
  - *create a structured JS plugin for our application*
- start by creating a simple JavaScript only plugin
  - *helps demonstrate plugin development*
  - *general preparation and usage*
- need to quickly update our `plugin.xml` file
  - *correctly describe our new plugin*

```
<description>output a daily random travel note</description>
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 2

- now start to modify our plugin's main JS file, `www/test.js`
- use this JS file to help describe the plugin's primary JS interface
  - *developer can call within their Cordova application*
  - *helps them leverage the options for the installed plugin*
- by default, when Plugman creates a template for our custom plugin
  - *includes the following JS code for `test.js` file*

```
var exec = require('cordova/exec');

exports.coolMethod = function(arg0, success, error) {
    exec(success, error, "test", "coolMethod", [arg0]);
};
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 3

- part of the default JS code
  - *created based upon the assumption we are creating a native plugin*
  - *eg: for Android, iOS platforms...*
- loads the `exec` library
  - *then defines an export for a JS method called `coolMethod`*
- as we develop a native code based plugin for Cordova
  - *need to provide this method for each target platform*
- working with a JS-only plugin, simply export a function for our own plugin
- now update this JS file for our custom plugin

```
module.exports.dailyNote = function() {
return "a daily travel note to inspire a holiday...";
}
```

- to be able to use this plugin
  - *a Cordova application simply calls `test.dailyNote()`*
  - *the note string will be returned*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 4

- simply exposing one test method through the available custom plugin
- easily build this out
  - *expose more by simply adding extra exports to the `test.js` file*
- also add further JS files to the project
  - *also export functions for plugin functionality*
- need to update our plugin to work in an asynchronous manner
  - *a more Cordova like request pattern for a plugin*
- when the API is called
  - *at least one callback function needs to be passed*
  - *then the function can be executed*
  - *then passed the resulting value*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 5

```
module.exports = {

  // get daily note
  dailyNote: function() {
      return "a daily travel note to inspire a holiday...";
  },

  // get daily note via the callback function
  dailyNoteCall: function (noteCall) {
    noteCall("a daily travel note to inspire a holiday...");
  }
};
```

- exposing a couple of options for requests to the plugin
- now call `dailyNote()`
  - *get the return result immediately*
- call `dailyNoteCall()`
  - *get the result passed to the callback function*

Cordova app - Plugins

Test plugin 1 - JS plugin - part 6

- now need to test this plugin, and make sure that it actually works as planned
- first thing we need to do is create a simple test application
  - *follow the usual pattern for creating our app using the CLI*
  - *add our default template files*
  - *then start to add and test the plugin files*

```
cordova create customplugintest1 com.example.customplugintest1 customplugintest1
```

- also add our required platforms,

```
cordova platform add android
```

Cordova app - Plugins

Test plugin 1 - JS plugin - part 7

- we can then add our new custom plugin

```
cordova plugin add ../custom-plugins/cordova-plugin-test
```

- currently installing this plugin from a relative local directory
- when we publish a plugin to the Cordova plugin registry
  - *install custom plugin using the familiar pattern for standard plugins*
- we can now check the installed plugins for our custom plugin

```
cordova plugins
```

Image - Cordova Custom Plugin

```
Drs-MacBook-Air-2:customplugintest1 ancientlives$ cordova plugins
cordova-plugin-whitelist 1.0.0 "Whitelist"
org.csteach.plugin.Test 1.0.0 "Test"
Drs-MacBook-Air-2:customplugintest1 ancientlives$
```

Cordova Installed Plugins

Cordova app - Plugins

Test plugin 1 - JS plugin - part 8

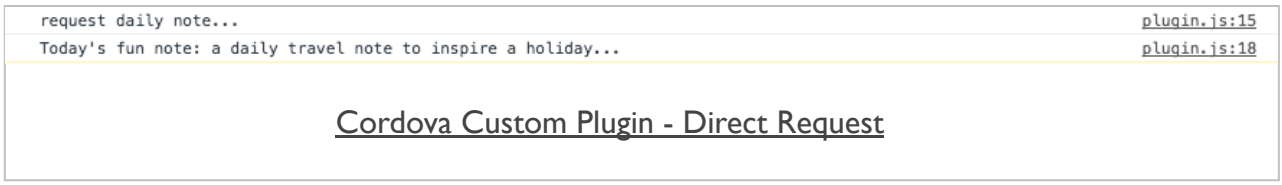
- now need to setup our home page,
- add some jQuery to handle events
- then call the exposed functions from our plugin
- start by adding some buttons to the home page

```
<button id="dayNote">Daily Note</button>
<button id="dayNoteSync">Daily Note Async</button>
```

- then update our app's `plugin.js` file
  - *include the logic for responding to button events*
  - *then call plugin's exposed functions relative to requested button*

```
//handle button tap for daily note - direct
$("#dayNote").on("tap", function(e) {
  e.preventDefault();
  console.log("request daily note...");
  var note = test.dailyNote();
  var noteOutput = "Today's fun note: "+note;
  console.log(noteOutput);
});
```

Image - Cordova Custom Plugin



Cordova app - Plugins

Test plugin 1 - JS plugin - part 9

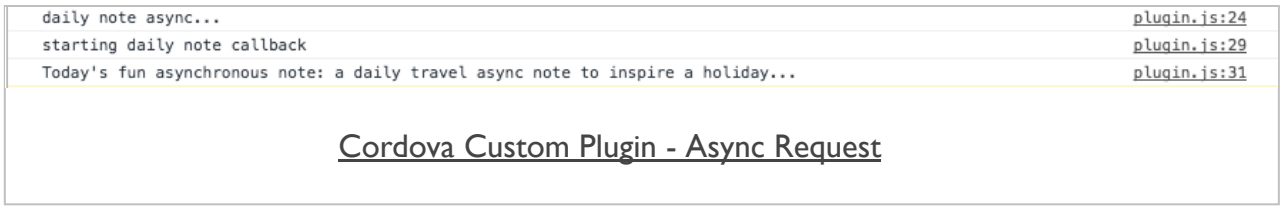
- request asynchronous version of daily note function from plugin's exposed API
- add an event handler to our `plugin.js` file
  - *responds to the request for this type of daily note*

```
//handle button press for daily note - async
$("#dayNoteSync").on("tap", function(e) {
  e.preventDefault();
  console.log("daily note async...");
  var noteSync = test.dailyNoteCall(noteCallback);
});
```

- then add the callback function

```
function noteCallback(res) {
  console.log("starting daily note callback");
  var noteOutput = "Today's fun asynchronous note: "+ res;
  console.log(noteOutput);
}
```

Image - Cordova Custom Plugin



Cordova Custom Plugin - Async Request

Cordova app - Plugins

Test plugin 2 - Android plugin - part 1

- now setup and tested our initial JS only plugin application
- JS only can be a particularly useful way to develop a custom plugin
- often necessary to create one using the native SDK for a chosen platform
  - *eg: a custom Android plugin*
- now create a second test application
  - *then start building our test custom Android plugin*

```
cordova create customplugintest2 com.example.customplugintest2 customplugintest2
```

- add test template to application

Cordova app - Plugins

Test plugin 2 - Android plugin - part 2

- start to consider developing our custom Android plugin
- Android plugins are written in Java for the native SDK
- build a test plugin to help us understand process for working with native SDK
- test a few initial concepts for our plugin
  - *processing user input,*
  - *returning some output to the user*
  - *some initial error handling*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 3

- now consider setup of our application to help us develop a native Android plugin
- three parts to a plugin that need concern us as developers

```
|- plugin.xml
|- src
   |- android
      |- Test2.java
|- www
   |- test2.js
```

- then add our required platforms for development

```
// add android
plugman platform add --platform_name android
```

- focus on the Android platform for the plugin

Cordova app - Plugins

Test plugin 2 - Android plugin - part 4

- start to build our native Android plugin
- begin by modifying the `Test2.java` file
- Cordova Android plugins require some default classes

```
import org.apache.cordova.CordovaPlugin;
import org.apache.cordova.CallbackContext;
```

- our Java code begins importing required classes for a standard plugin
- these include Cordova required classes
  - *required for general Android plugin development*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 5

- now start to build our plugin's class
- start by creating our class, which will extend CordovaPlugin

```
public class Test2 extends CordovaPlugin {
  ...do something useful...
}
```

- then start to consider the internal logic for the plugin
- each Android based Cordova plugin requires an `execute()` method
- this method is run
  - *whenever our Cordova application requires interaction or communication with a plugin*
  - *this is where all of our logic will be run*

```
@Override
public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
throws JSONException {
    if (action.equals("coolMethod")) {
        String message = args.getString(0);
        this.coolMethod(message, callbackContext);
        return true;
    }
    return false;
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 6

- for the execute method
  - *passing an action string*
  - *tells plugin what is being requested*
- plugin uses this requested action
  - *checks which action is being used at a given time*
  - *eg: plugins will often have many different features*
- code within `execute()` method needs to be able to check the required action
- now update our `execute()` method,

```
@Override
public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
throws JSONException {
    if (ACTION_GET_NOTE.equals(action)) {
        JSONObject arg_object = args.getJSONObject(0);
        String note = arg_object.getString("note");
    }
    String result = "Your daily note: "+note;
    callbackContext.success(result);
    return true;
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 7

- with our updated `execute()` method
  - *if the request action is `getNote`*
  - *our Java code grabs requested input from JSON data structure*
- current test plugin has a single input value
- if we started to build out the plugin
  - *eg: requiring additional inputs*
  - *we could grab them from the JSON as well*
- we've also added some basic error handling
- able to leverage the default `callbackContext` object
  - *provided by the standard Cordova plugin API*
- able to simply return an error to the caller
  - *if an invalid action is requested*
- one of the good things about developing an Android plugin for Cordova
  - *majority of plugins follow a similar pattern*
  - *main differences will be seen within the `execute()` method*

Cordova app - Plugins

Test plugin 2 - Android plugin - part 8

```
package org.csteach.plugin;
import org.apache.cordova.CallbackContext;
import org.apache.cordova.CordovaPlugin;
import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

public class Test2 extends CordovaPlugin {

  public static final String ACTION_GET_NOTE = "dailyNote";

    @Override
    public boolean execute(String action, JSONArray args, CallbackContext callbackContext)
    throws JSONException {
        if (ACTION_GET_NOTE.equals(action)) {
            JSONObject arg_object = args.getJSONObject(0);
            String note = arg_object.getString("note");
        String result = "Your daily note: "+note;
        callbackContext.success(result);
        return true;
    }
    callbackContext.error("Invalid action requested");
    return false;
    }
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 9

- need to update the JavaScript for our plugin
  - *helps us expose the API for the plugin itself*
- first thing we need to do is create a primary object for our plugin
- then use this to store the APIs needed to be able to request and use our plugin

```
var noteplugin = {
... do something useful...
}

module.exports = noteplugin;
```

- current API will support one action, our `getNote` action

```
getNote:function(note, successCallback, errorCallback) {
...again, do something useful...
}
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 10

- communication between JavaScript and the native code in the Android plugin
  - *performed using the `cordova.exec` method*
- method is not explicitly defined within our application or plugin
- when this code is run within the context of our Cordova application
  - *the `cordova` object and the required `exec()` method become available*
  - *they are part of the default structure of a Cordova application and plugin*
- now add our `cordova.exec()` method

```
cordov.exec(
...add something useful...
);
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part I I

- now pass our `exec()` method two required argument
  - *represents necessary code for success and failure*
- basically telling Cordova how to react to a given user action
- then tell Cordova which plugin is required
  - *and associated action to pass to the plugin*
- also need to pass any input to the plugin
- updated `exec()` method is as follows

```
cordova.exec(
  successCallback,
  errorCallback,
  'Test2',
  'getNote',
  [{
  "note": note
  }]
);
```

Cordova app - Plugins

Test plugin 2 - Android plugin - part 12

- plugin's JavaScript code should now look as follows

```
var noteplugin = {

  getNote:function(note, successCallback, errorCallback) {

    cordova.exec(
      successCallback,
      errorCallback,
      'Test2',
      'getNote',
      [{
        "note": note
      }]
    );

  }
}

module.exports = noteplugin;
```

Cordova app - Plugins

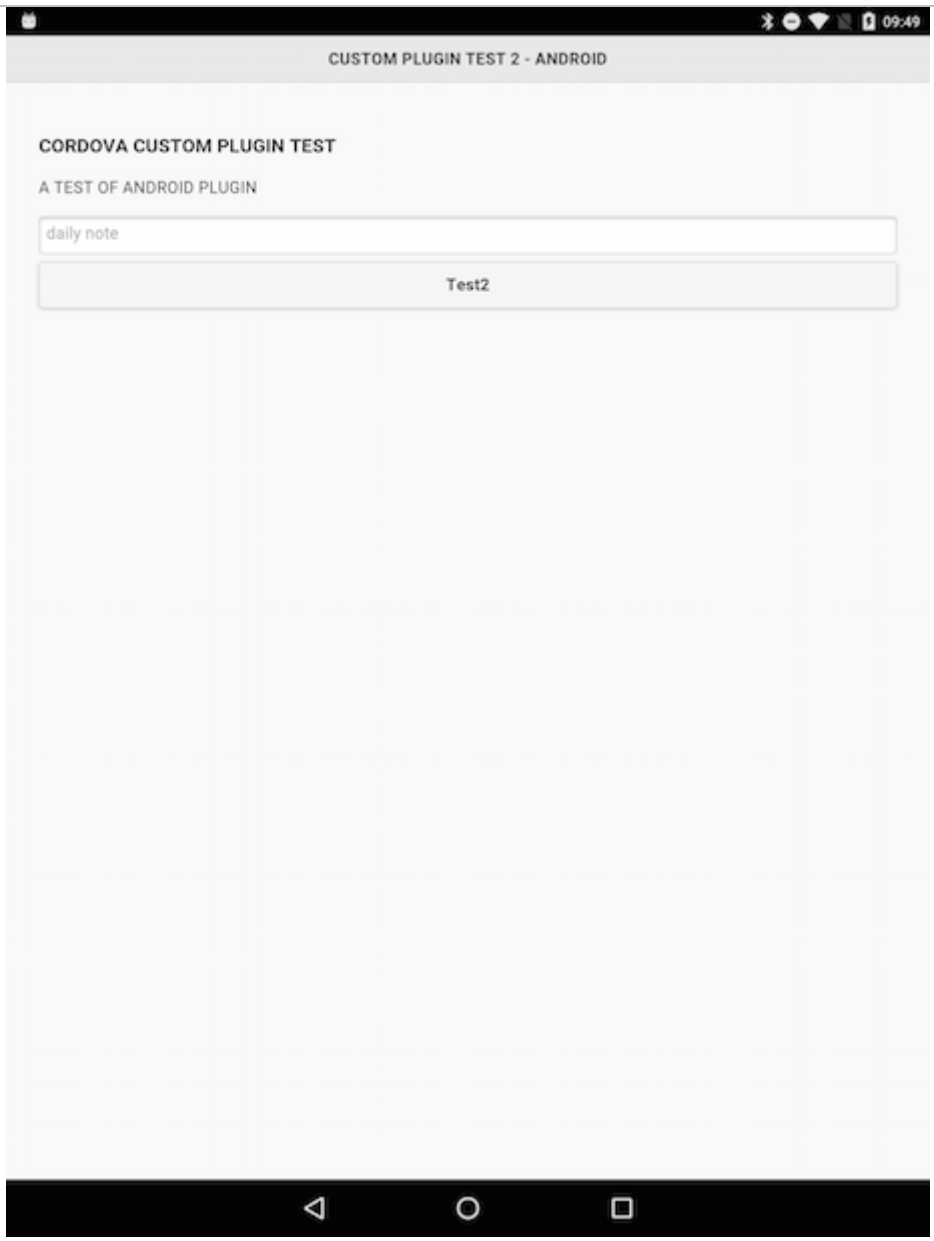
Test plugin 2 - Android plugin - part 13

- now need to test our plugin with our application
- update our home page to allow a user to interact with our new custom plugin
- add an input field for the user requested note
- add a button to submit the request itself

```
<input type="text" id="noteField" placeHolder="daily note">
<button id="testButton">Test2</button>
```

- exposed plugin API will be able to respond
  - *use the input data from the user*
  - *then pass to the native Android plugin*

Image - Cordova Custom Plugin 2



Cordova Custom Plugin 2 - HTML Update

Cordova app - Plugins

Test plugin 2 - Android plugin - part 14

- update app's `plugin.js` to handle user input
  - *then process for use with our custom plugin*
- still need to wait for the `deviceready` event to return successfully
- then we can start to work with our user input and custom plugin
- our native Android plugin's API is similarly exposed using the window object

```
window.test2
```

- we can then execute it from our application's JS

```
windows.test2.getNote
```

- then pass the requested note data to the API
- define how we're going to work with success and error handlers
  - *render the returned value to the application's home page*

```
window.test2.getNote(note,
  function(result) {
    console.log("result = "+result);
    $("#note-output").html(result);
  },
  function(error) {
    console.log("error = "+error);
    $("#note-output").html("Note error: "+error);
  }
);
```

Cordova app - Plugins

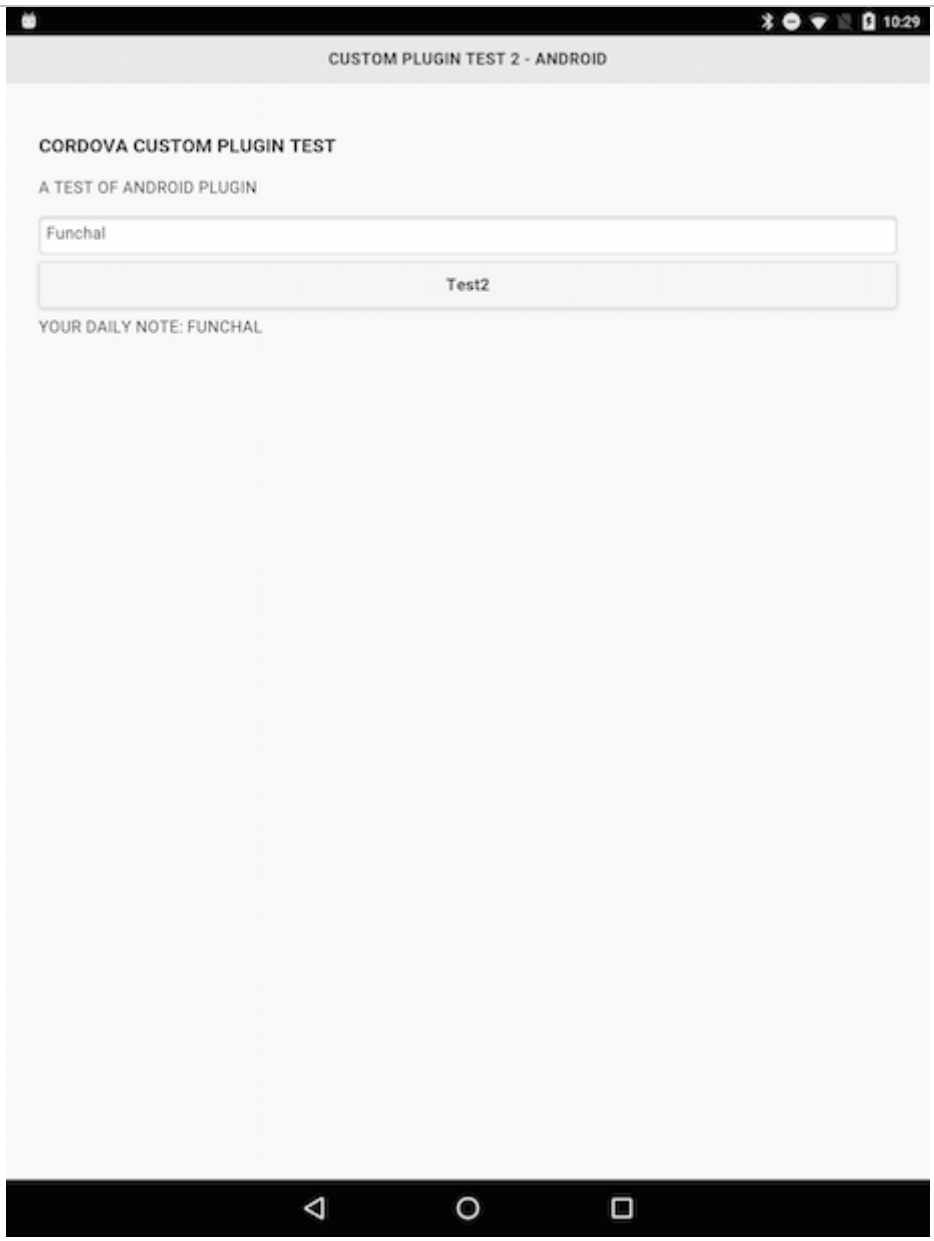
Test plugin 2 - Android plugin - part 15

```
function onDeviceReady() {

//handle button press for daily note - direct
$("#testButton").on("tap", function(e) {
  e.preventDefault();
  console.log("request daily note...");
  var note = $("#noteField").val();
  console.log("requested note = "+note);
  if (note === "") {
    return;
  }
  window.test2.getNote(note,
    function(result) {
      console.log("result = "+result);
      $("#note-output").html(result);
    },
    function(error) {
      console.log("error = "+error);
      $("#note-output").html("Note error: "+error);
    }
  );
});

}
```

Image - Cordova Custom Plugin 2



Cordova Custom Plugin 2 - Android plugin output

Cordova app - Plugins

Summary of custom plugin development

- an initial template for a custom plugin can be created using the *Plugman* tool
- create JS only custom plugins
- create native SDK plugins
  - *eg: Android, iOS, Windows Phone...*
- custom plugin consists of
  - *plugin.xml*
  - *JavaScript API*
  - *native code*
- create the plugin separate from the application
  - *then add to an application for testing*
  - *remove to make changes, then add again...*

References

- Cordova API
  - *Plugin Development Guide*
  - *Plugin.xml*
- Cordova Plugins