Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 4

Dr Nick Hayward

Contents

- working with plugins
  - *update media playback*
  - *tests and output*
- splashscreens and icons
- API plugin examples
  - *custom templates*
  - *camera*
- extra notes

Cordova app - working with plugins - update media playback

- basic plugin test for media playback within an app
  - *user can play music in their app*
  - *user touch interaction with button*
  - *file loaded from local filesystem*
  - *device playback of selected audio file*
- leveraging native device functionality in app
  - *calling plugins for **device**, **file**, **media**...*
- basic app includes,
  - *user interaction in the UI*
  - *calls to the exposed JS API for the plugins*
  - *playback of audio by the native device*
- add further functionality
  - *stop, pause...*

Cordova app - working with plugins - stop button - part I

- consider how to **stop**, **pause** playback
  - *e.g. UI interaction, timer, event...*
- app logic is very similar
  - *respond to **stop** event*
  - *call method*
  - ...
- methods for **stop**, **pause**, &c. available in plugin API

```
media.pause

media.stop

media.release
```

Cordova app - working with plugins - stop button - part I

- start to update our existing app by adding a **stop** button to the UI
  - *allow our user to simply tap a button to stop playback*

```
<p>Stop playback...</p>
<input type="button" id="stopAudio" data-icon="delete" value="Stop" />
```

- update initial JS logic for the app
  - *listen for tap event on **stop** button*
  - *then call the stop method on the **media** object*

```
//button - stop audio
$("#stopAudio").on("tap", function(e) {
    //stop audio logic
    e.preventDefault();
  //call custom method to handle stopping audio...
    stopAudio();
});
```

Cordova app - working with plugins - stop button - part 2

- add the logic for our custom method to stop the audio
  - *call as `stopAudio()`*

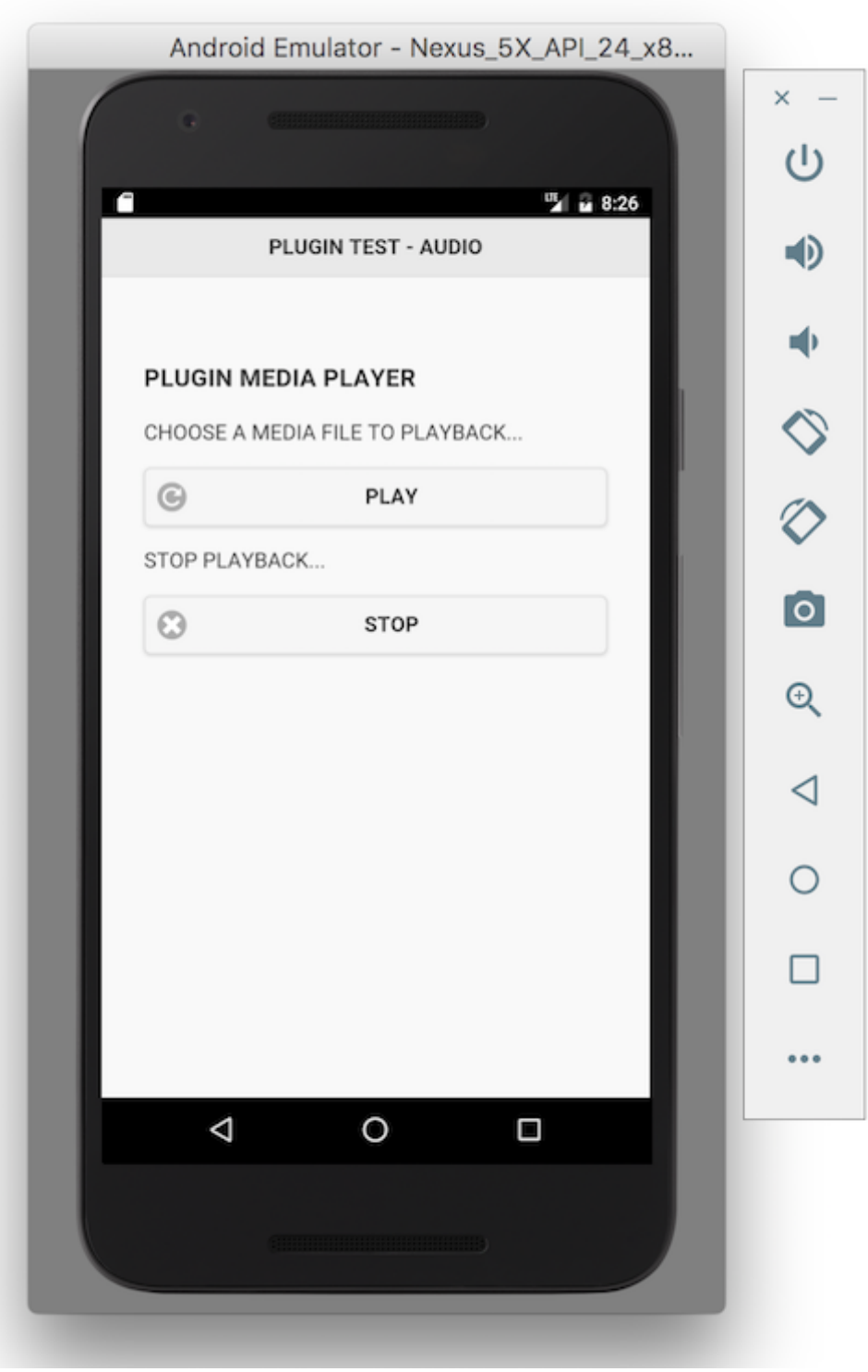
```
//stop audio file
function stopAudio() {
    //stop audio playback
    $audio.stop();
    //release audio - important for android resources...
    $audio.release();
    //just for testing
    alert("stop playing audio...& release!");
}
```

- logic still won't stop the audio playing
- issue is variable `$audio`
  - *currently restricted local scope to `playAudio()` method*
- initially alter scope of property for `$audio` itself
  - *now set in initial `onDeviceReady()` method*

```
function onDeviceReady() {
    //set initial properties
    var $audio;
...
}
```

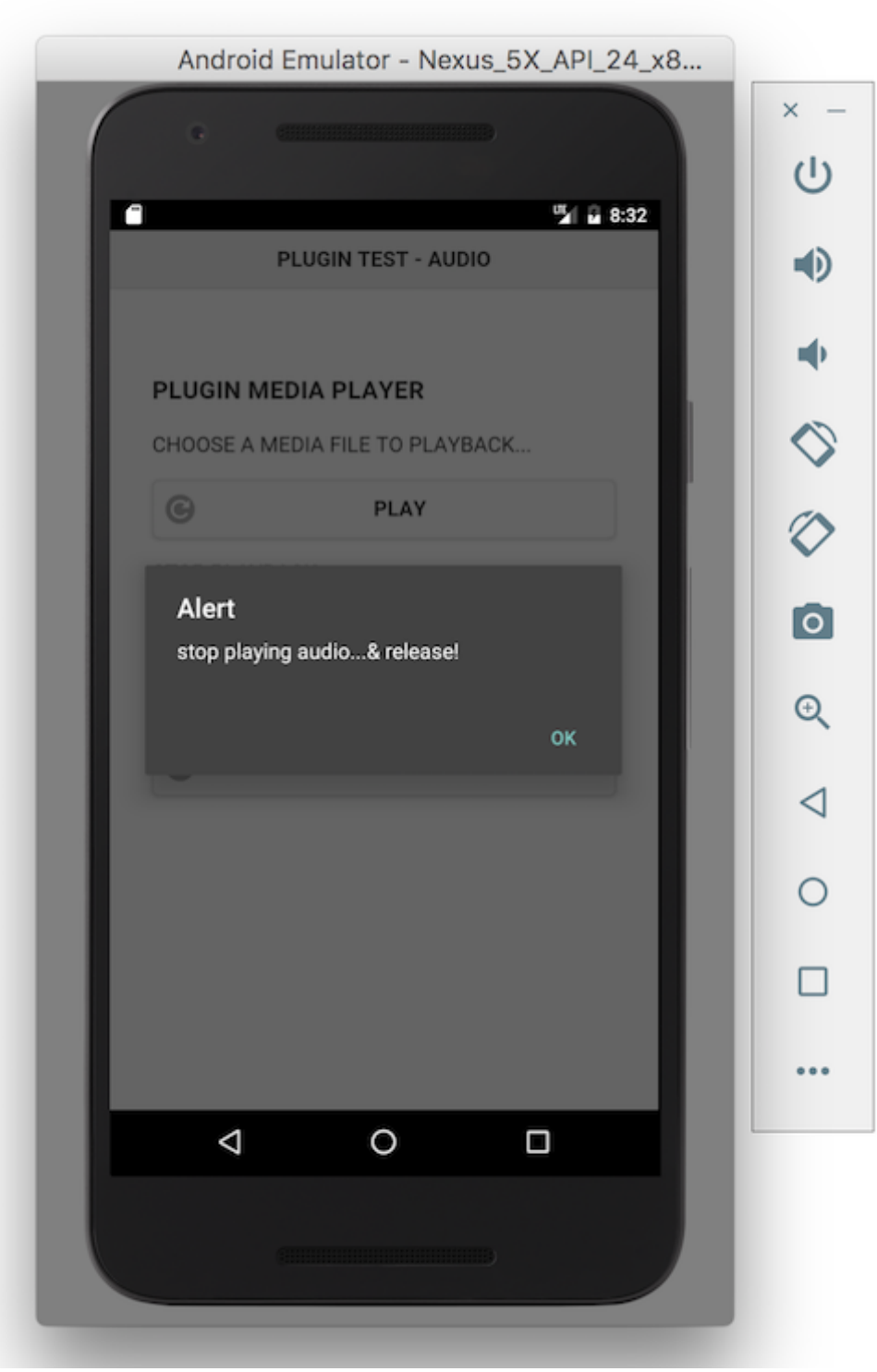
- logic will now stop audio playing
- call to `release()` method important for OS's audio resources
  - *particularly important to release unwanted resources on Android...*

Image - Cordova app - Plugin Test - stop audio playback



Cordova - Plugin Test - stop audio playback

Image - Cordova app - Plugin Test - stop audio playback 2



Cordova - Plugin Test - stop audio playback 2

Cordova app - working with plugins - pause button - part I

- follow similar pattern to add initial pause button to app's HTML

```
<p>Pause playback...</p>
<input type="button" id="pauseAudio" data-icon="bars" value="Pause" />
```

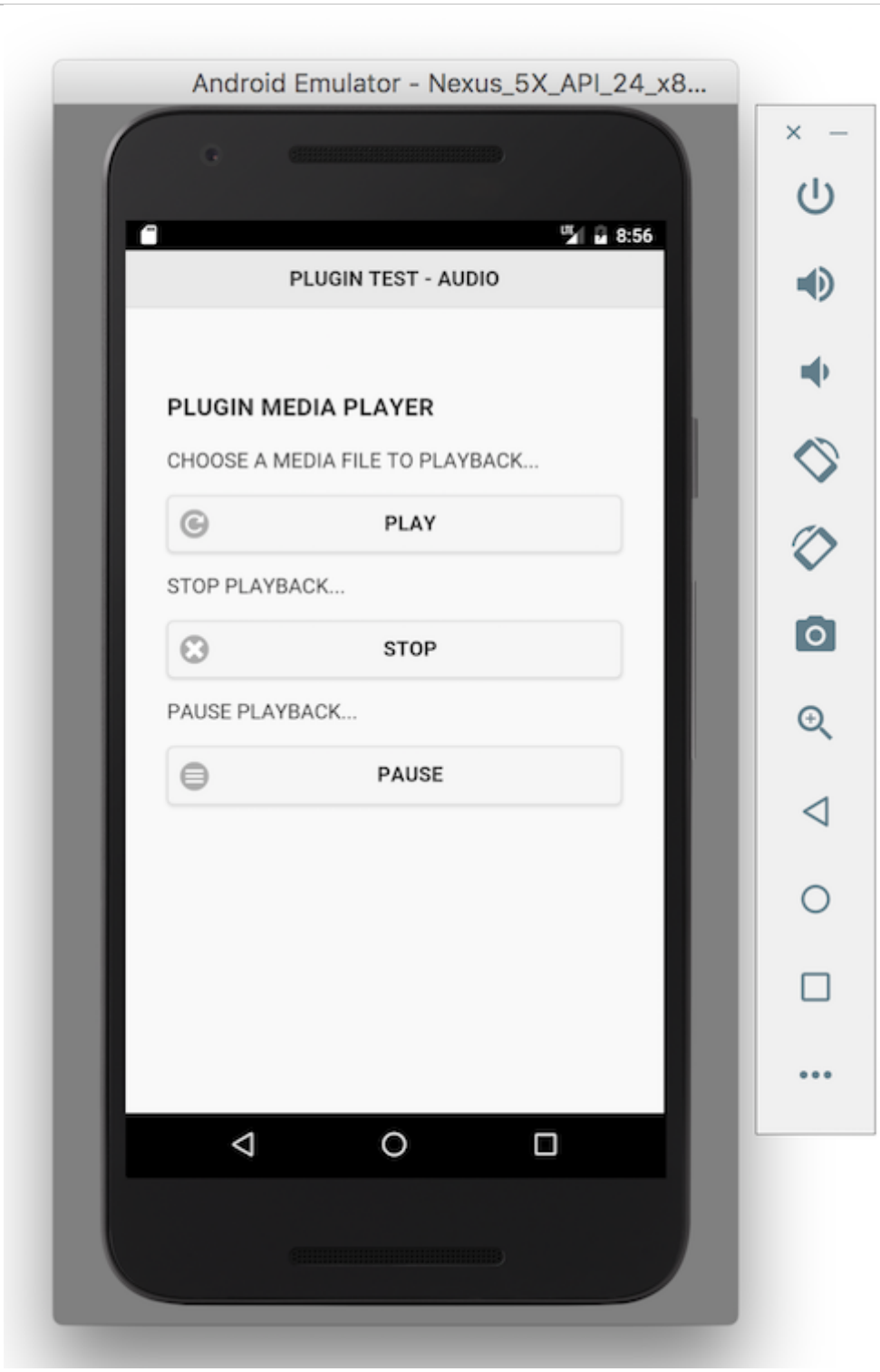
- then add basic listener for tap event on the pause button

```
//button - pause audio
$("#pauseAudio").on("tap", function(e) {
    //pause audio logic
    e.preventDefault();
    //call custom method to handle pausing audio...
    pauseAudio();
});
```

- then add our custom `pauseAudio()` method
  - *handles pausing of current media object*

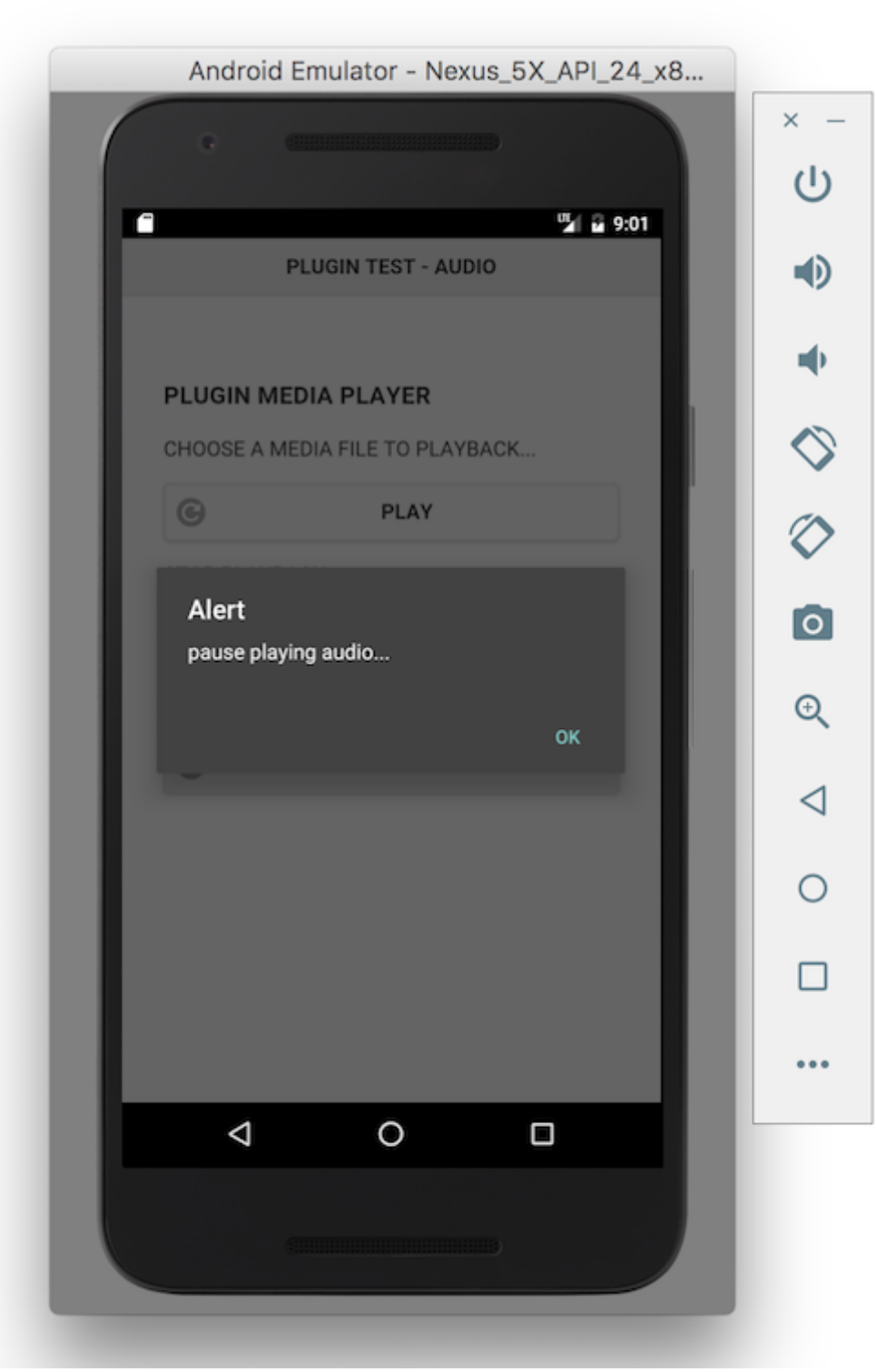
```
//pause audio file
function pauseAudio() {
    //pause audio playback
  $audio.pause();
}
```

Image - Cordova app - Plugin Test - pause audio playback



Cordova - Plugin Test - pause audio playback

Image - Cordova app - Plugin Test - pause audio playback 2

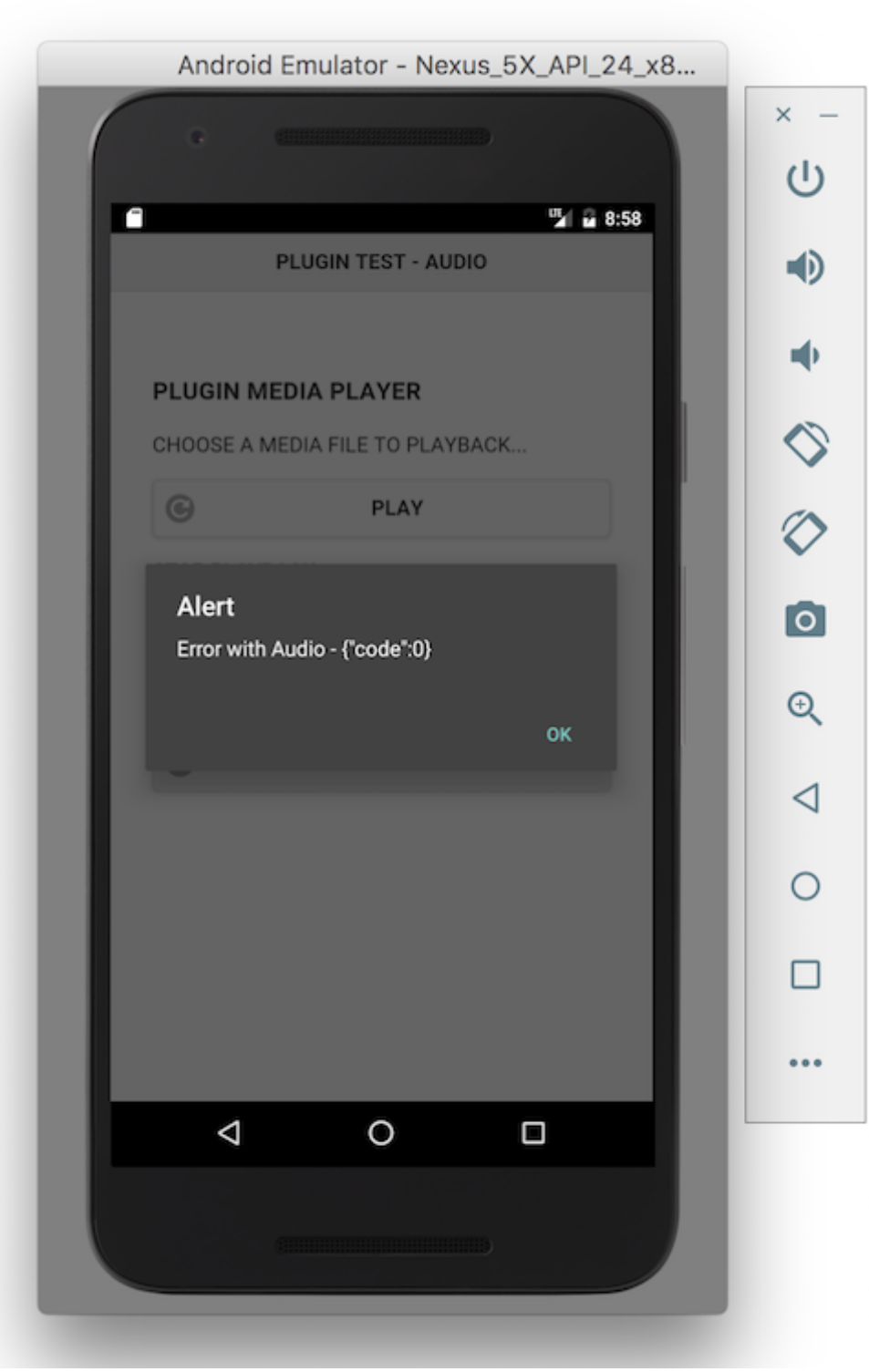


Cordova - Plugin Test - pause audio playback 2

Cordova app - working with plugins - pause button - part 2

- this logic works but it introduces issues and errors, e.g.
  - *start playback of audio and then pause*
  - *then touch play again*
  - *audio will restart from the start of the audio file*
  - *not ideal user experience...*
- an error will be thrown, e.g.
  - *press pause once, then twice...*
  - *error will be thrown for the call to the `pause()` method*

Image - Cordova app - Plugin Test - pause audio playback 3



Cordova - Plugin Test - pause audio playback 3

Cordova app - working with plugins - pause button - part 3

- we can monitor change in the playback with a simple property
  - *attached to scope for `onDeviceReady()` method*
  - *property available to `play()`, `pause()`, and `stop()` methods*

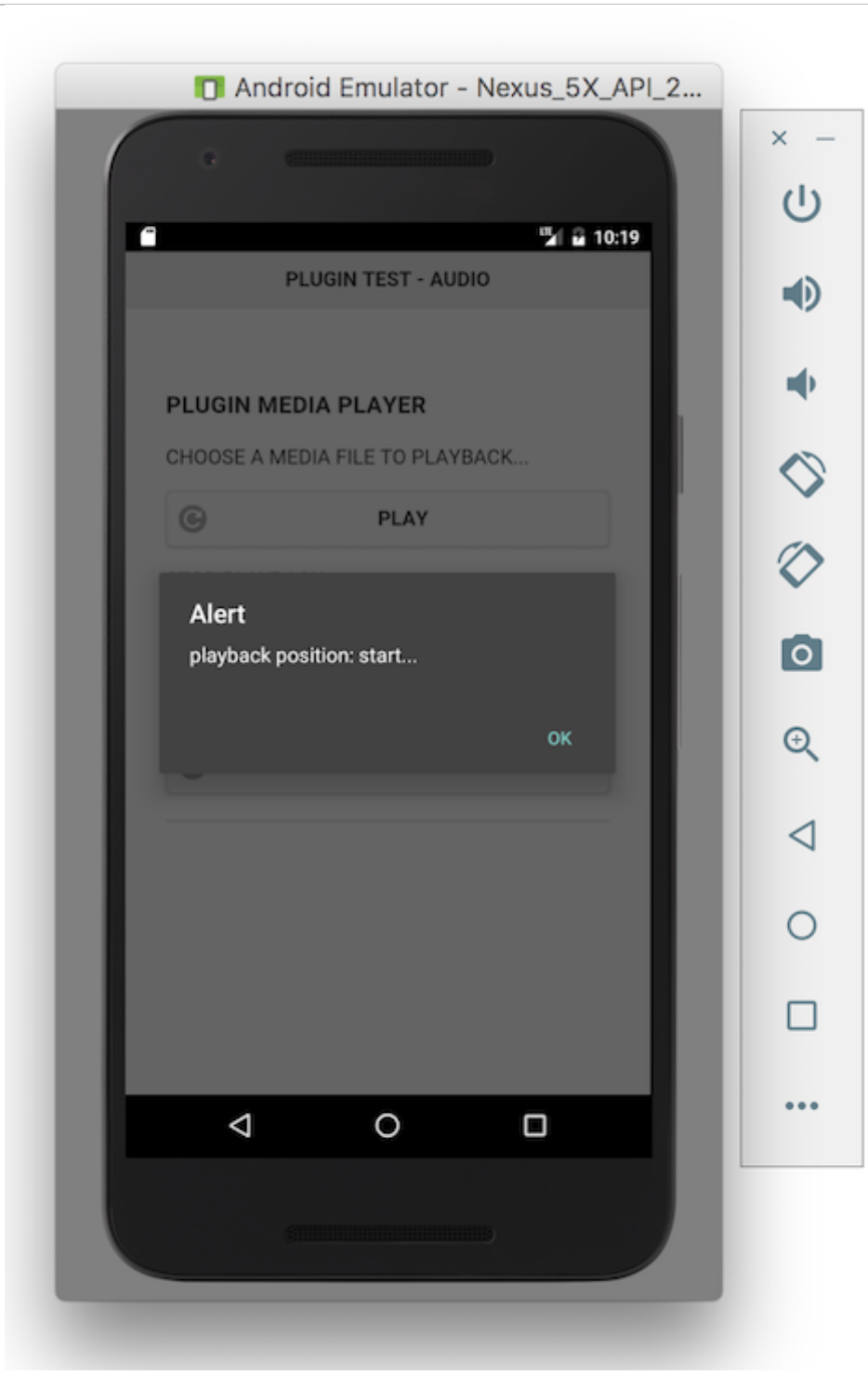
```
function onDeviceReady() {
  //set initial properties
    var $audio;
    var $audioPosn = 0;
...
}
```

- now have two properties we can monitor and update
  - *variable `$audioPosn` has been set to a default value of 0*
  - *we can check as we start to playback an audio file &c.*

```
//check current audio position
if ($audioPosn > 1) {
  $audio.play();
  alert("playback position: " + $audioPosn + " secs");
} else {
    $audio.play();
    alert("playback position: start...");
}
```

- also use property to output current playback position, reset for cancelling, &c.

Image - Cordova app - Plugin Test - update playback 1



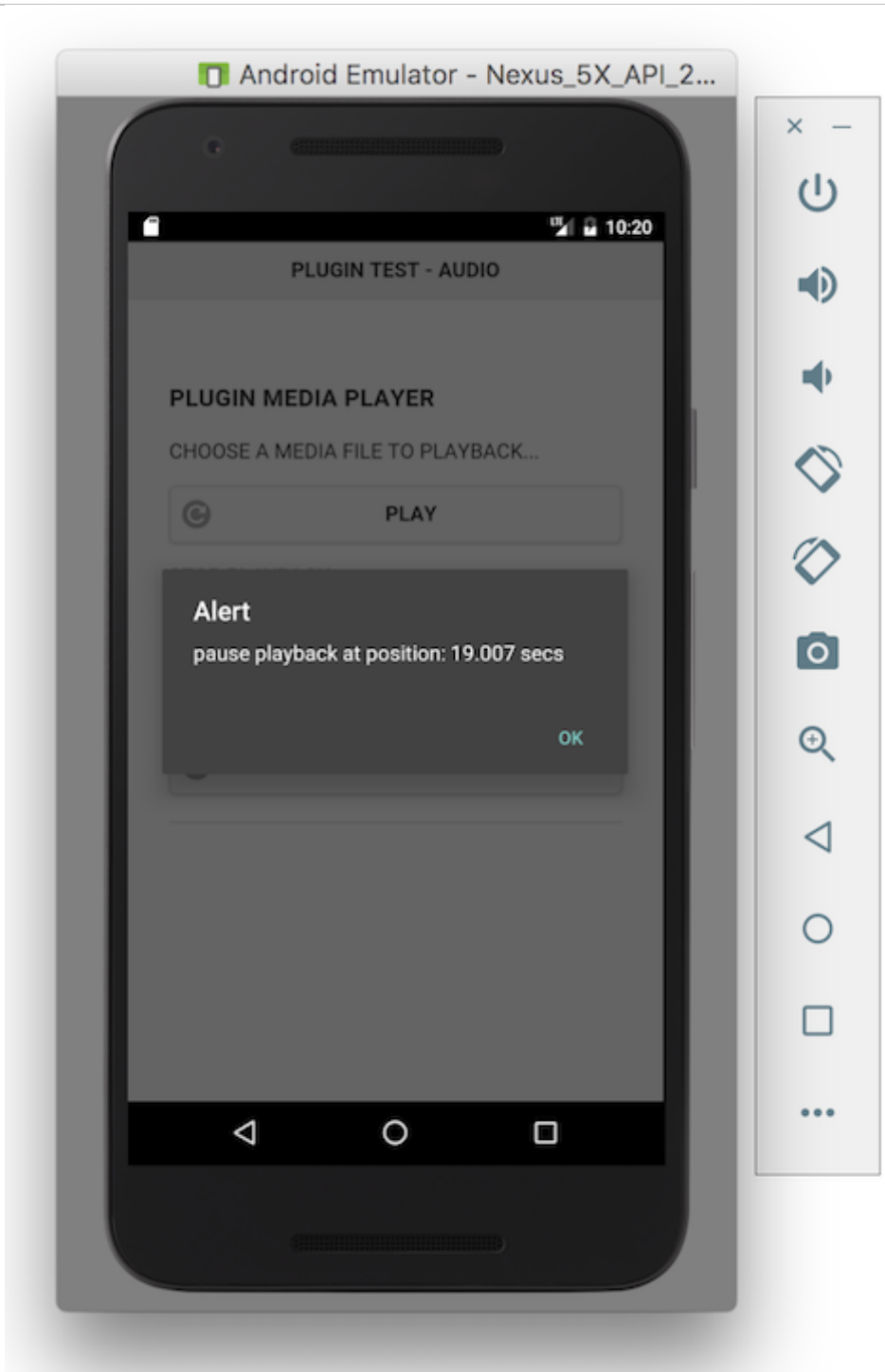
Cordova - Plugin Test - update playback 1

Cordova app - working with plugins - pause button - part 4

- pause a playing audio stream
  - *need to be able to get the current playback position for the audio file*
  - *then update our `$audioPosn` property.*
- check audio position in the `pauseAudio()` method
  - *use the `getCurrentPosition()` method*
  - *available on the `media` object...*

```
$audio.getCurrentPosition(
  // success callback
  function (position) {
    if (position > -1) {
            $audioPosn = position;
      alert("pause playback at position: " + position + " secs");
    }
  }, // error callback
     function (e) {
      ...
    }
);
```

Image - Cordova app - Plugin Test - update playback 2



Cordova - Plugin Test - update playback 2

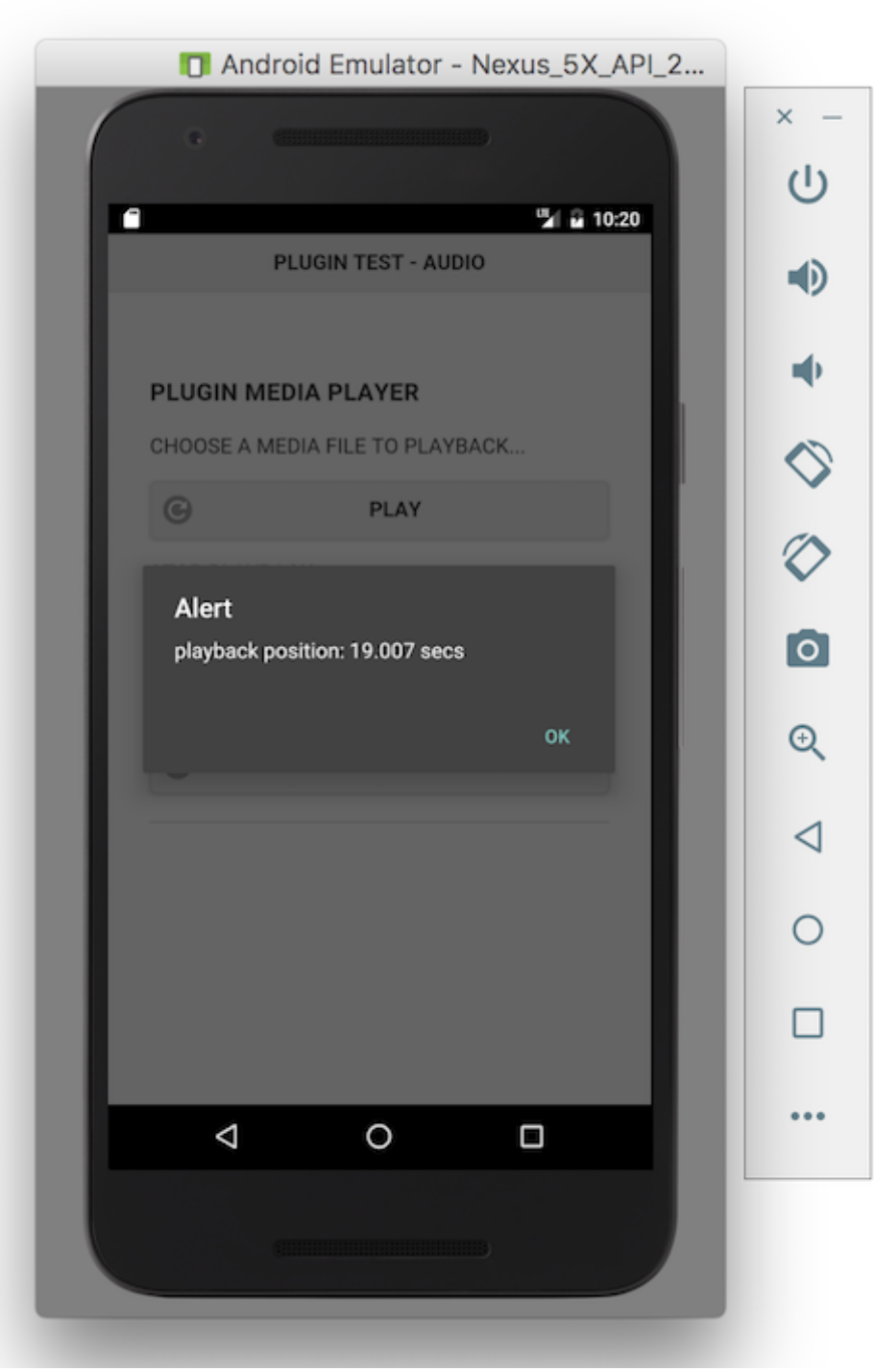
Cordova app - working with plugins - pause button - part 5

- we can now successfully pause our audio playback
  - *store value for current pause position in the audio stream*
- also need to update our audio playback
  - *need to check current position in audio stream*

```
//check current audio position
if ($audioPosn > 1) {
  $audio.seekTo($audioPosn*1000);
  $audio.play();
  alert("playback position: " + $audioPosn + " secs");
} else {
  $audio.play();
  alert("playback position: start...");
}
```

- we updated the `playAudio()` method to check value of `$audioPosn` property
- now use value to seek to current position in audio stream
  - *using `seekTo()` method exposed by `media` object itself...*
  - *method expects time in milliseconds*
  - *need to update value for our `$audioPosn` property, `$audioPosn*1000`*
- audio stream will now resume at correct position...

Image - Cordova app - Plugin Test - update playback 3



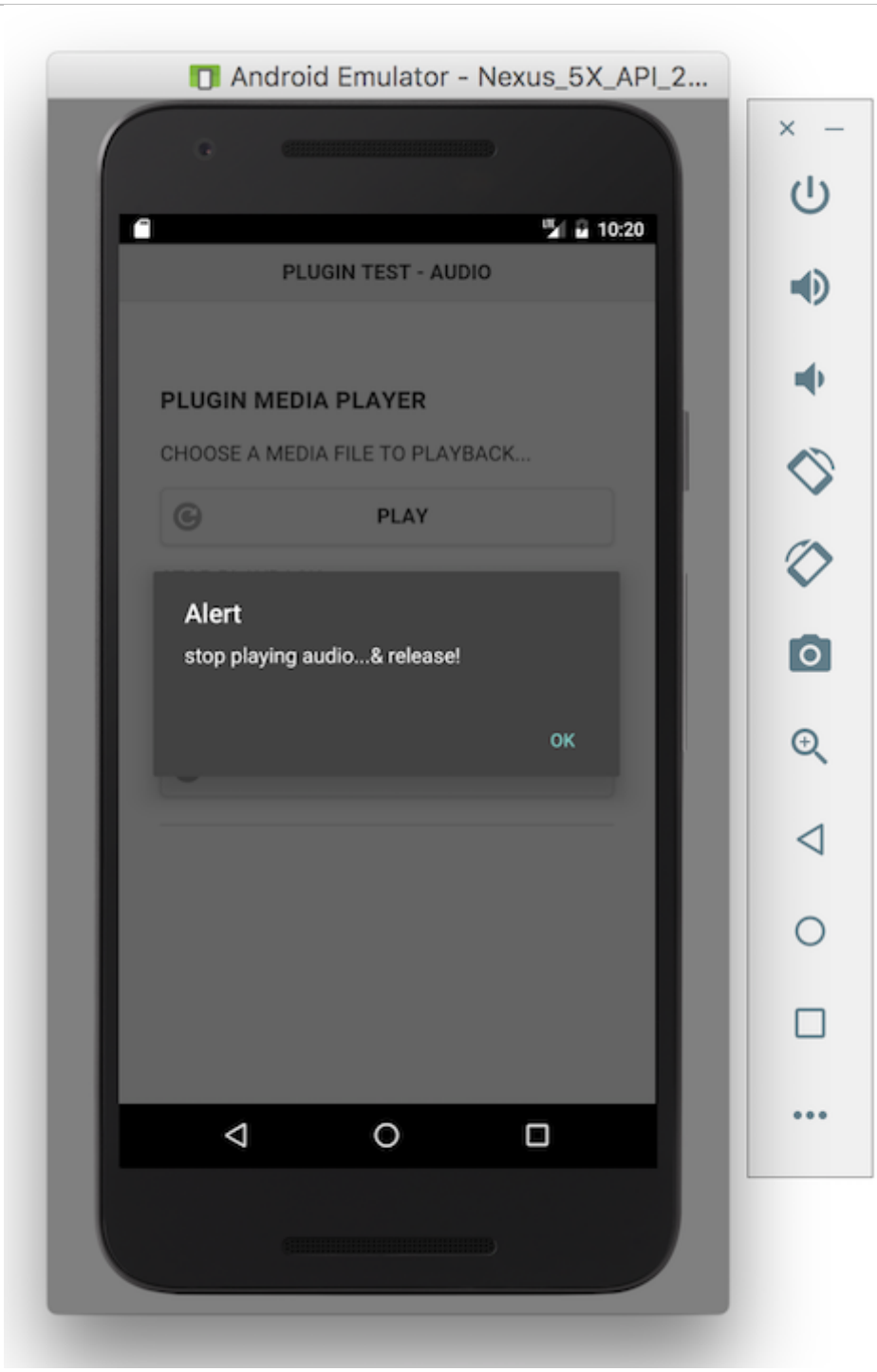
Cordova - Plugin Test - update playback 3

Cordova app - working with plugins - update stop button

- final touch for now, at least with the buttons
- need to update logic for app's **stop** button
- need to reset the value of the `$audioPosn` property
  - *if not, audio stream will always restart at set pause value*

```
//stop audio file
function stopAudio() {
  //stop audio playback
    $audio.stop();
    //reset $audioPosn
    $audioPosn = 0;
    //release audio - important for android resources...
    $audio.release();
    //just for testing
    alert("stop playing audio...& release!");
}
```

Image - Cordova app - Plugin Test - update playback 4



Cordova - Plugin Test - update playback 4

Cordova app - working with plugins - current playback position

- now seen how we can check the current position of a playing audio file
- many different options for outputting this value
  - *e.g. appending its value to the DOM, showing a dialogue, and so on...*
- how we use the value of this property is up to us as developers
  - *naturally informed by the requirements of the app*
- may only be necessary to use this value internally
  - *help with the app's logic*
- may need to output this result to the user

Cordova app - working with plugins - further considerations

A few updates and modifications for a media app

- update logic for app
  - *checks for event order, property values, &c.*
- indicate playback has started
  - ***without*** *alerts...*
- update state of buttons in response to app state
  - *highlights, colour updates...*
- inactive buttons and controls when not needed
  - *update state of buttons...*
- grouping of buttons to represent media player
  - *add correct icons, playback options...*
- metadata for audio file
  - *title, artist, length of track...*
- image for track playing
  - *thumbnail for track, album...*
- track description
- notification for track playing
- persist track data and choice in cache for reload...
- ...

Cordova app - working with plugins - add splashscreen

- add support for splashscreens in Cordova
  - *install splashscreen plugin in project*

```
cordova plugin add cordova-plugin-splashscreen
```

- then we need to return to our `config.xml` file
  - *set different splashscreens for different supported platforms*
  - *specify different images to use for given screen resolutions*
- Android example,

```
<platform name="android">
  <!-- splashscreens - you can use any density that exists in the Android project -->
  <!-- landscape splashscreens -->
  <splash src="res/screen/android/splash-land-hdpi.png" density="land-hdpi"/>
  <splash src="res/screen/android/splash-land-ldpi.png" density="land-ldpi"/>
  <splash src="res/screen/android/splash-land-mdpi.png" density="land-mdpi"/>
  <splash src="res/screen/android/splash-land-xhdpi.png" density="land-xhdpi"/>
  <!-- portrait splashscreens -->
  <splash src="res/screen/android/splash-port-hdpi.png" density="port-hdpi"/>
  <splash src="res/screen/android/splash-port-ldpi.png" density="port-ldpi"/>
  <splash src="res/screen/android/splash-port-mdpi.png" density="port-mdpi"/>
  <splash src="res/screen/android/splash-port-xhdpi.png" density="port-xhdpi"/>
</platform>
```

- specifying different images for each screen density
  - *then specify for portrait and landscape aspect ratios*
- URL for the `src` attribute is relative to the project's root directory
  - *not the customary www*

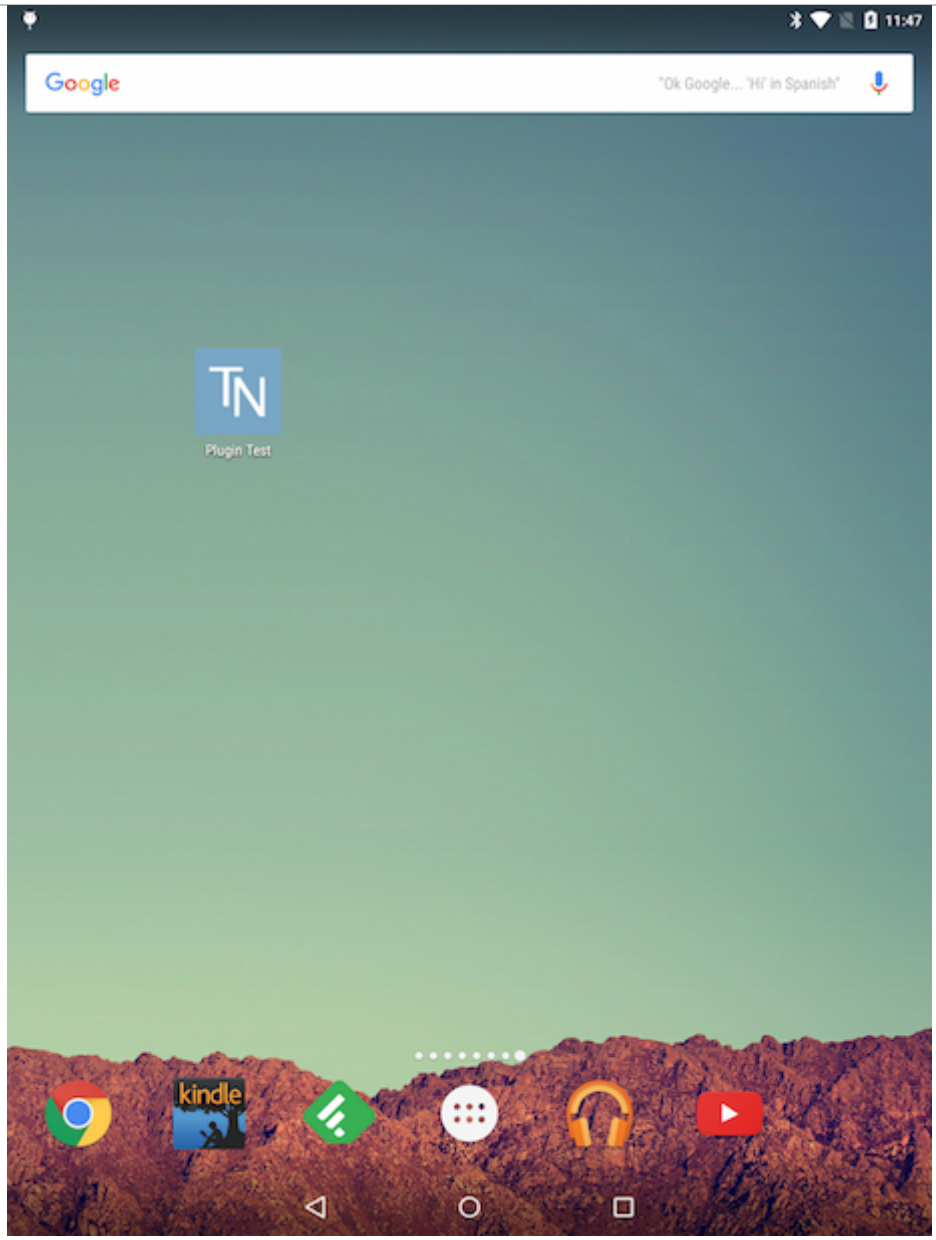
Cordova app - working with plugins - add an app icon

- also set our own app's icon
  - *again in the `config.xml` setting for the application*

```
<platform name="android">
  <icon src="res/icon/android/ldpi.png" density="ldpi" />
  <icon src="res/icon/android/icon/mdpi.png" density="mdpi" />
  <icon src="res/icon/android/icon/hdpi.png" density="hdpi" />
  <icon src="res/icon/android/icon/xhdpi.png" density="xhdpi" />
</platform>
```

- again, we can target specific platforms
  - *useful way to handle different screen resolutions and densities*
- icon's URL is specified relative to the project's root directory

Image - Cordova app - Plugin Test 1 - getting started



Cordova - Plugin Test - custom icon

Density	Launcher icon size

Cordova app - working with plugins - Android icon sizes for launcher

Density	Launcher icon size
ldpi	36 x 36 px
mdpi	48 x 48 px
hdpi	72 x 72 px
xhdpi	96 x 96 px

and so on...

Cordova app - API plugin examples

- a few API plugins to consider
  - *accelerometer*
  - *camera*
  - *connection*
  - *device*
  - *geolocation*
  - *InAppBrowser*
  - *media, file, and capture*
  - *notification*
  - *StatusBar*
  - ...

Cordova app - API plugin examples - plugin test 2

setup

- create our initial plugin test shell application

```
cordova create plugintest2 com.example.plugintest2 plugintest2
```

- add any required plaforms, eg: Android, iOS, Windows...

```
cordova platform add android --save
```

- then run an initial test to ensure the shell application loads correctly
  - *run in the Android emulator or*
  - *run on a connected Android device*

```
cordova emulate android
```

- or

```
cordova run android
```

- then start to update the default `www` directory
- modify the initial settings in our app's `config.xml` file

Cordova app - API plugin examples - plugin test 2

application structure

- might update our initial Cordova template
  - *better structure for plugin test application*
  - *structure might look as follows*

```
|- hooks
|- platforms
   |- android
   |- platforms.json
|- plugins
   |- cordova-plugin-whitelist
   |- android.json
   |- fetch.json
|- resources
   |- icon
   |- splash
|- www
   |- assets
      |- images
      |- scripts
      |- styles
   |- docs
      |- json
      |- txt
      |- xml
   |- media
      |- audio
      |- images
      |- video
   |- index.html
|- config.xml
```

Cordova app - templates - basic

- Cordova default template for project structure
  - `create` *command used for basic structure...*
- create custom, reusable template for a new project
  - *e.g. create starting template for tabs, menu &c. based app...*
- to create a custom template
  - *start with new project structure for Cordova*
  - *then modify to create and configure app structure*
  - *set required icons, splashscreens, designs &c. for template*
- then we can start to package a reusable template

Cordova app - templates - structure

- each template uses the following directory structure

```
|-- template_package
    |__ package.json
    |__ index.js
    |__ template_src
        |__ ... (app template contents...)
```

- template specific code is added to `template_src` directory
- `package.json` includes reference to template's `index.js` file
- `index.js` used to export reference to `template_src` directory

Cordova app - templates - `template_src`

- `template_src` usually includes the following structure

```
|-- hooks (add custom hooks for template, app &c...)
|-- www
    |__ css
        |__ index.css
    |__ img
        |__ logo.png
    |__ js
        |__ index.js
    |__ index.html
|-- config.xml
```

- add any custom scripts to the `hooks` directory
- design and build our template in the `www` directory
- `template_src/config.xml` will usually follow pattern of default Cordova config
- then add template customisations, e.g.
  - *name, description, icons, splashscreens...if necessary*

Cordova app - templates - `package.json`

- `package.json` includes template specific metadata
  - *add keyword `cordova:template` & `ecosystem:cordova`*
  - *used for package distribution, e.g. NPM*
- add reference to `index.js`

```
"main": "index.js"
```

- output will be similar to a standard NPM `package.json` file
  - *created for NPM package management*
  - *then initialised using the command,*

```
npm init
```

Cordova app - templates - template `index.js`

- then add necessary export reference for `template_src` to our template `index.js` file
  - *follows a standard pattern*

```
var path = require('path');


module.exports = {
    dirname : path.join(__dirname, 'template_src')
};
```

Cordova app - templates - finish & create

- template is now ready to be published and shared online
  - *use NPM, GitHub, &c.*
- use as the template for a new local project

```
cordova create basic com.example.basic BasicTemplate --template <path-to-template>
```

- add the local directory path for the custom template
  - *replace `<path-to-template` with local directory for template...*
- creates new Cordova project with custom template
  - *uses `template_src` for the project*

Cordova app - API plugin examples - plugin test 2

plugins - add camera plugin

- now add the camera plugin to our test application
- two ways we can add camera functionality to our application
  - *use the camera plugin*
  - *use the more generic Media Capture API*
- main differences include
  - ***camera*** *plugin focuses on camera capture and functionality*
  - ***media capture*** *includes additional options such as video and audio recording*
- add the camera plugin using the following Cordova CLI command

```
cordova plugin add cordova-plugin-camera
```

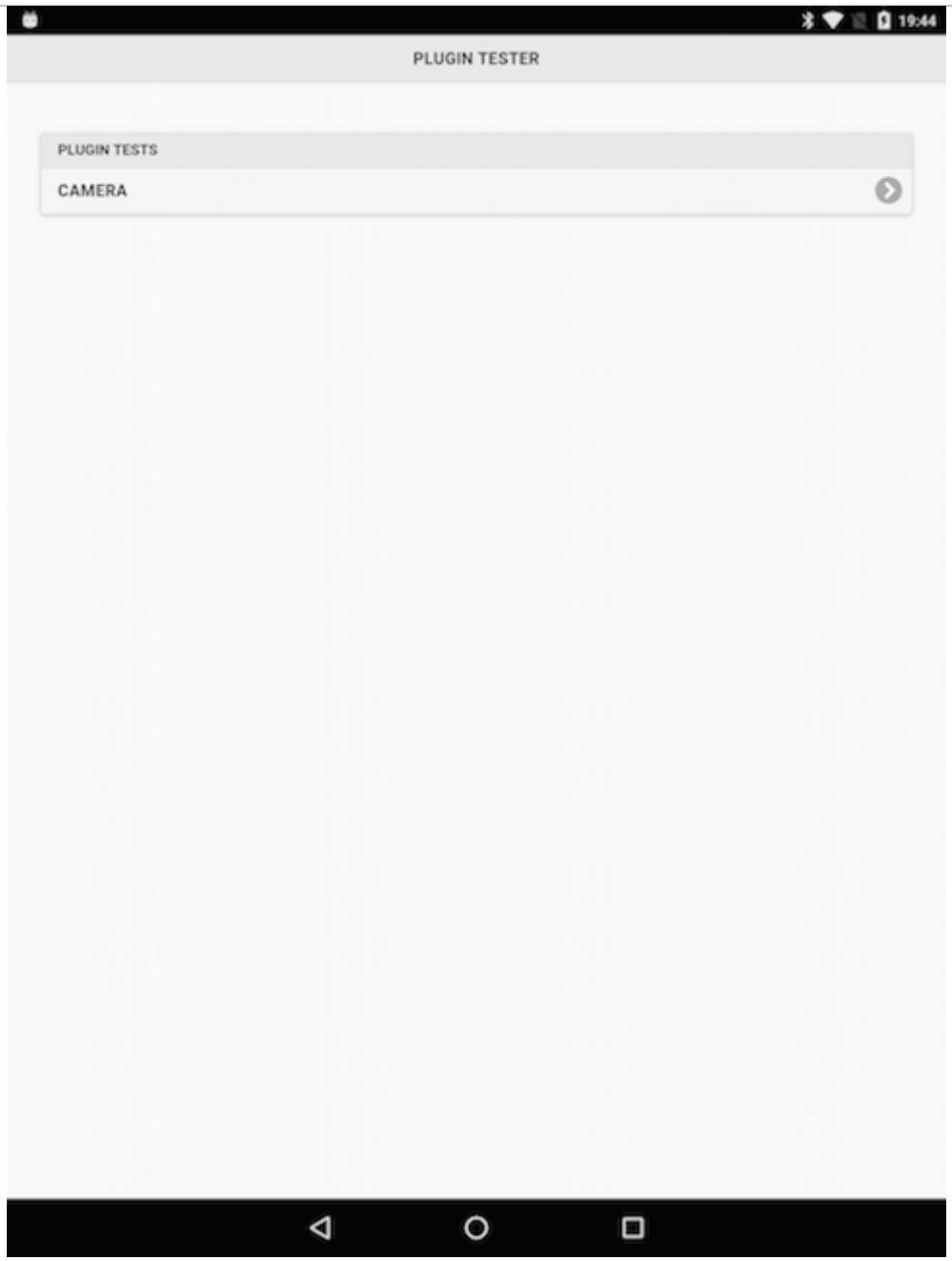
- provides standard `navigator` object
  - *enables taking pictures, and choose images from local image library*

Cordova app - API plugin examples - plugin test 2

plugins - add camera page

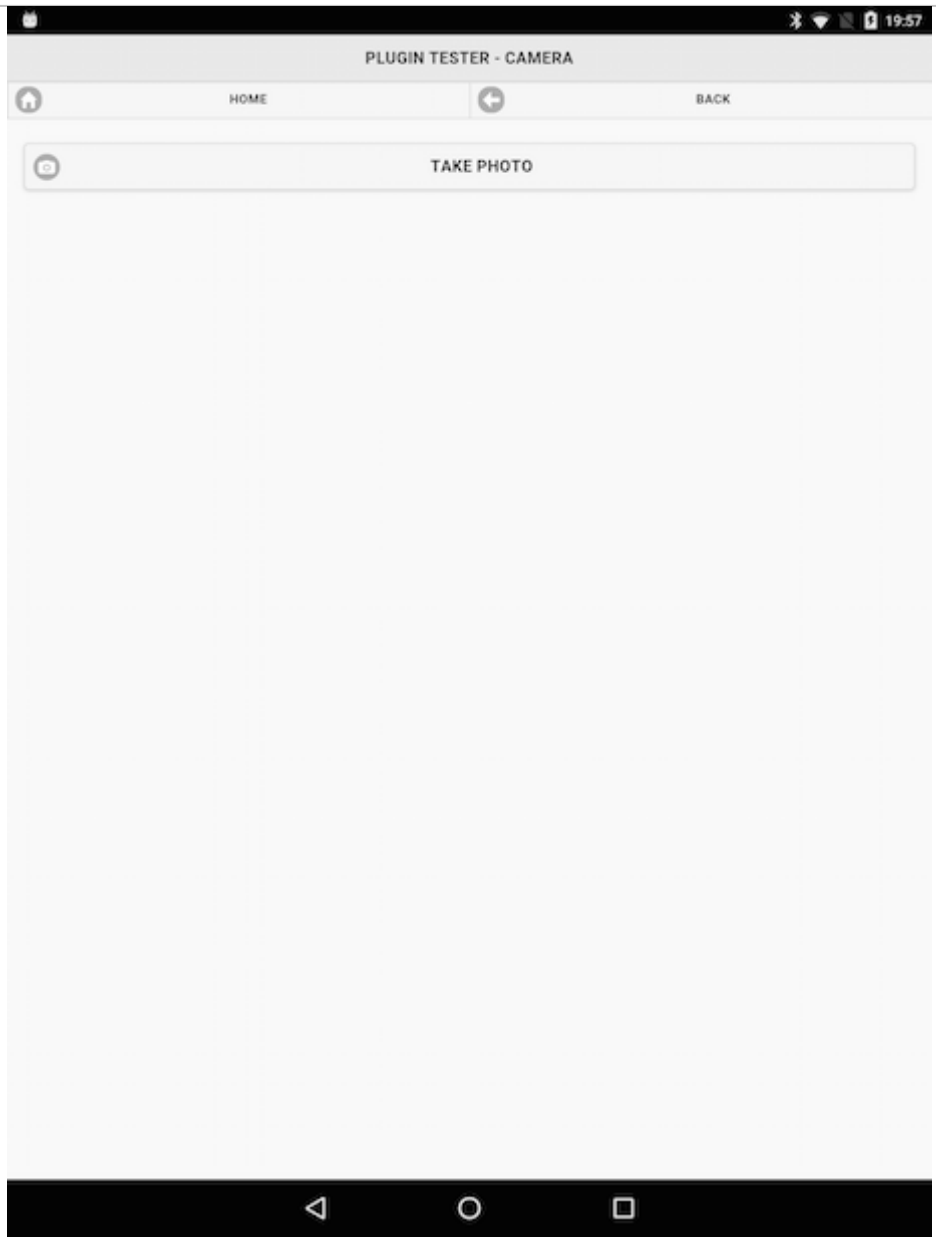
```
<!-- camera page -->
<div data-role="page" id="camera">
  <div data-role="header">
    <h3>plugin tester - camera</h3>
    </div><!-- /header -->
  <div data-role="navbar" data-iconpos="left">
    <ul>
      <li><a class="ui-btn" data-icon="home" data-transition="slide" href="#home">home</a></li>
      <li><a class="ui-btn" data-icon="arrow-l" data-rel="back">back</a></li>
    </ul>
  </div><!-- /navbar -->
  <div data-role="content">
    <input type="button" id="takePhoto" data-icon="camera" value="Take Photo" />
    <div id="photo">
      <img id="imageView" style="width: 100%;"></img>
    </div><!-- /photo -->
    <div data-role="popup" id="photoSelector" style="min-width: 250px;">
      <ul data-role="listview" data-inset="true">
        <li data-role="divider">Choose Photo</li>
        <li><a id="cameraPhoto" href="#">Take Photo with Camera</a></li>
        <li><a id="galleryPhoto" href="#">Get Photo from Gallery</a></li>
      </ul>
    </div><!-- /photoSelector -->
  </div><!-- /content -->
</div><!-- /camera page -->
```

Image - API Plugin Tester - Home



API Plugin Tester - initial home page

Image - API Plugin Tester - Camera



API Plugin Tester - initial camera page

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

- basic UI is now in place
- start to add some logic for taking photos with the device's camera
- need to be able to get photos from the device's image gallery
- app's logic in initial `plugin.js` file
- handlers for the tap events
  - *a user tapping on the **takePhoto** button*
  - *then the options in the **photoSelector***
  - *take a photo with the camera*
  - *get an existing photo from the gallery*
- use the `onDeviceReady()` function
  - *add our handlers and processors for both requirements*
  - *add functionality for camera and gallery components*

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

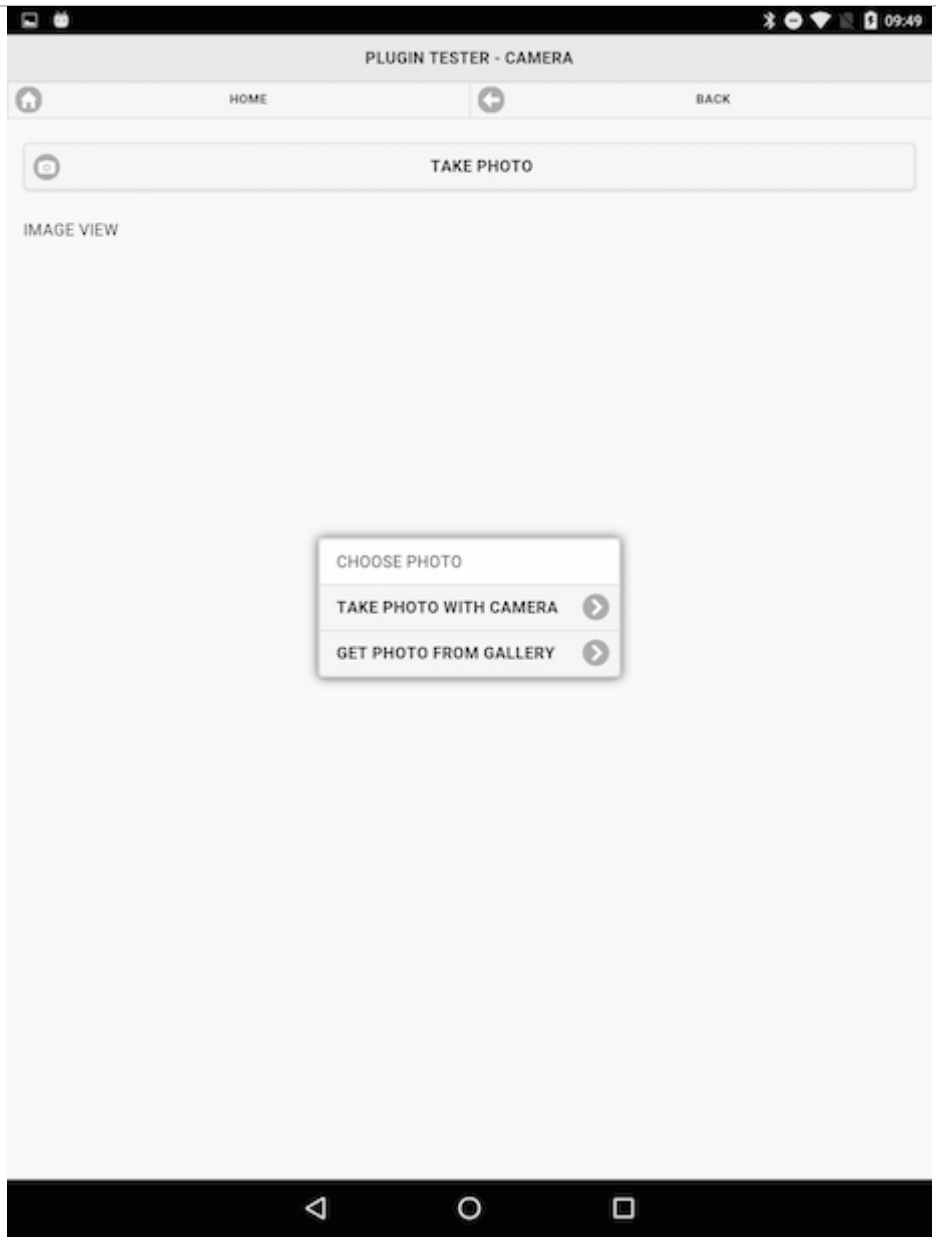
- add our handlers for the tap events
- initial handlers for `takePhoto`, `cameraPhoto`, and `galleryPhoto`

```
$("#takePhoto").on("tap", function(e) {
  e.preventDefault();
  //show popup options for camera
  $("#photoSelector").popup("open");
})

$("#cameraPhoto").on("tap", function(e) {
  e.preventDefault();
  //hide popup options for camera
  $("#photoSelector").popup("close");
})

$("#galleryPhoto").on("tap", function(e) {
  e.preventDefault();
  //hide popup options for camera
  $("#photoSelector").popup("close");
})
```

Image - API Plugin Tester - Camera



API Plugin Tester - camera photo selector

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

- capture an image using this plugin with the native device's camera hardware
- use the provided `navigator` object for the camera
  - *then call the `getPicture` function*
- also specify required callback functions for the camera
  - *and add some required options for quality...*

```
//Use from Camera
navigator.camera.getPicture(onSuccess, onFail, {
  quality: 50,
  sourceType: Camera.PictureSourceType.CAMERA,
  destinationType: Camera.DestinationType.FILE_URI
});
```

- quality option has been reduced to `50` for testing
  - *choose a value between 0 and 100 for our final application*
  - *100 being original image file from the camera*
- option for `destinationType` now defaults to `FILE_URI` could be changed to `DATA_URL`
  - ***NB:*** *`DATA_URL` option can crash an app due to low memory, system resources...*
  - *returns a base-64 encoded image*
  - *then render in a chosen format such as a JPEG*

Cordova app - API plugin examples - plugin test 2

plugins - add camera logic

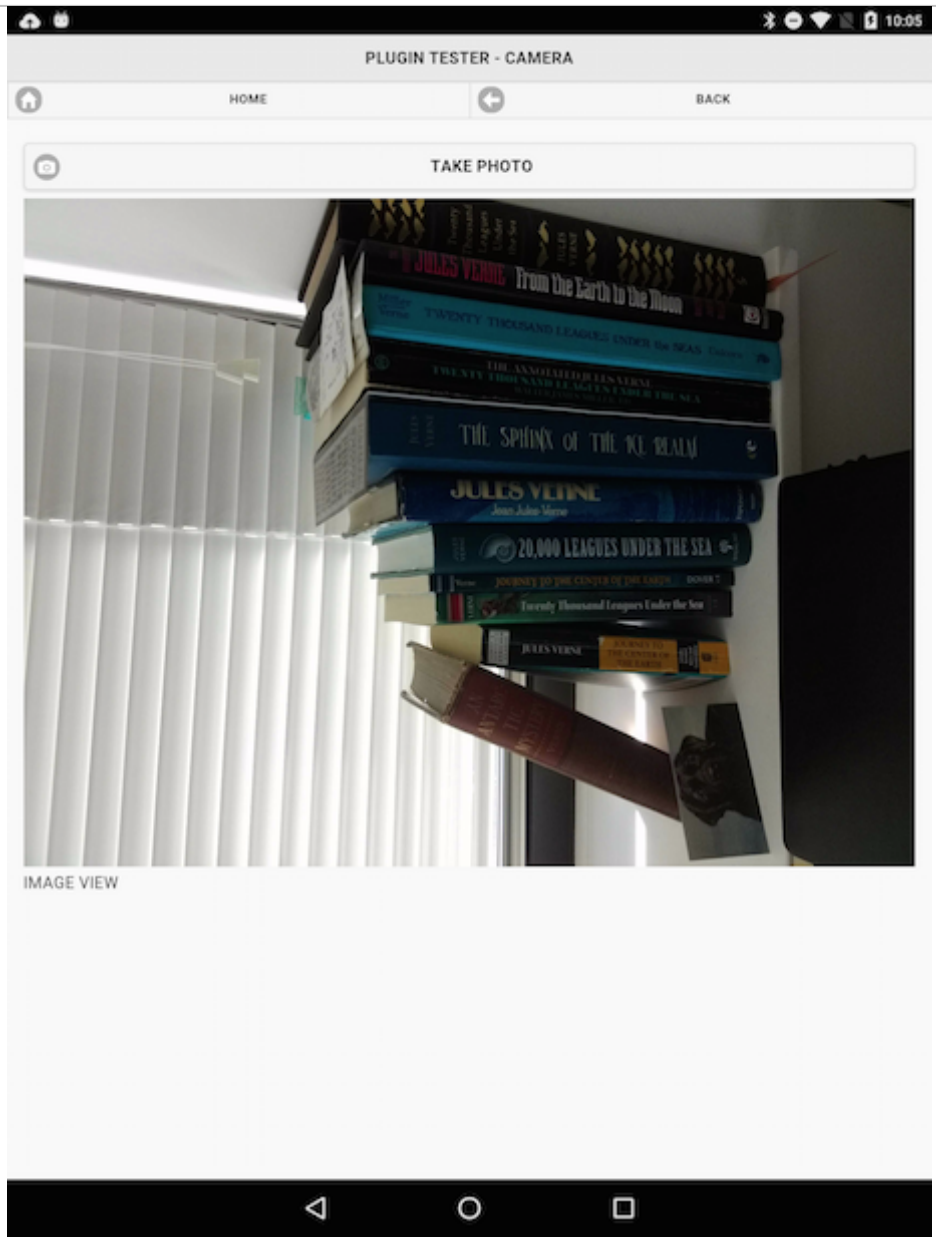
- two callback functions are `onSuccess` and `onFail`
  - *set logic for returned camera image and any error message*

```
function onSuccess(imageData) {
  //JS selector faster than jQuery...
  var image = document.getElementById('imageView');
  image.src = imageData;
}

function onFail(message) {
  alert('Failed because: ' + message);
}
```

- `onSuccess` function accepts a parameter for the returned image data
- using returned image data to output and render our image in the test `imageView`
- `onFail` function simply outputting a returned error message
- we can use these two callback functions to perform many different tasks
  - *we can pass the returned image data to a save function, or edit option...*
  - *they act like a bridge between our own logic and the native device's camera*

Image - API Plugin Tester - Camera



API Plugin Tester - image rotated

Cordova app - API plugin examples - plugin test 2

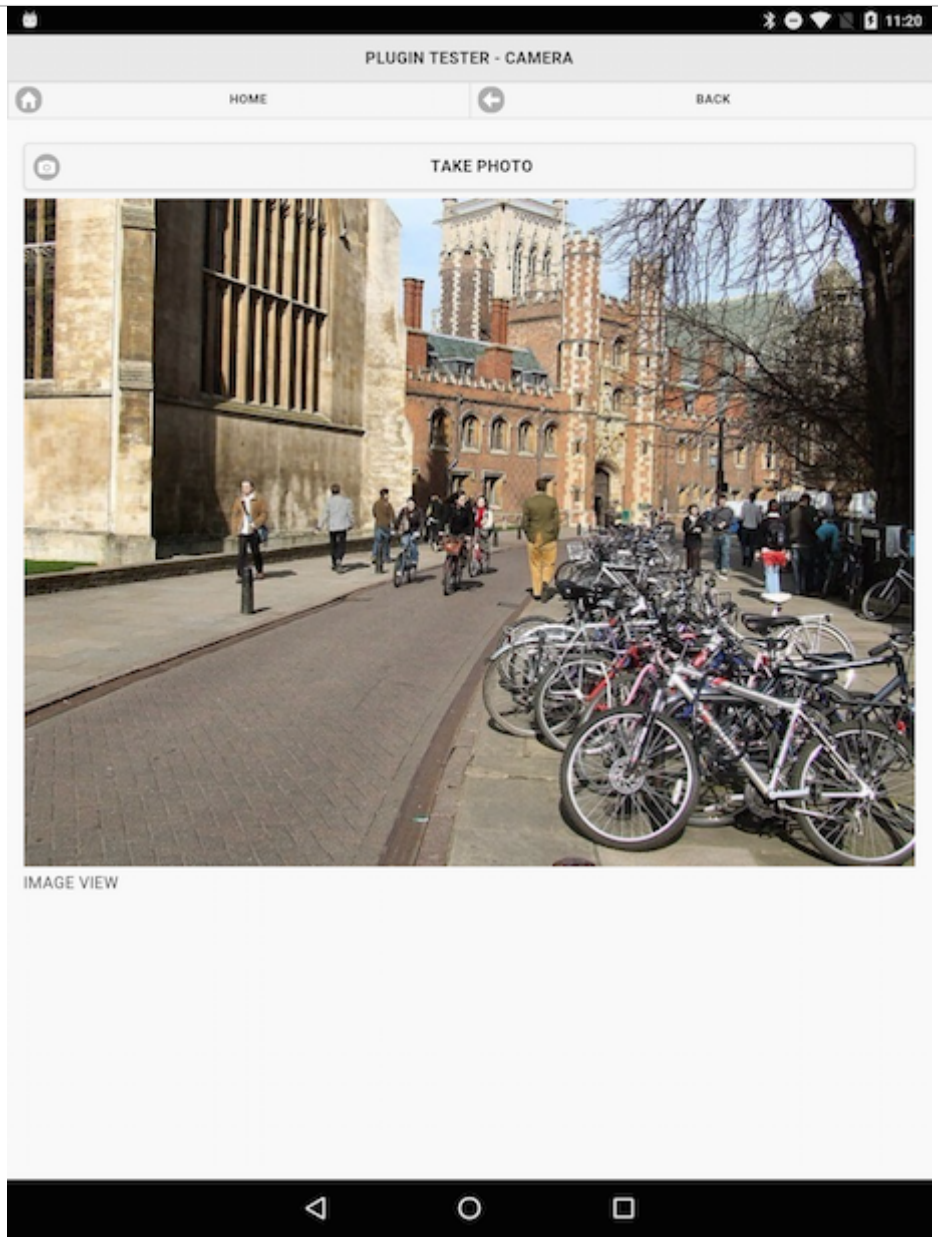
plugins - update camera logic

- returned an image from the camera
- update our application to select an image from gallery application
- add a conditional check to our `getPhoto()` function
  - *allows us to differentiate between a camera or gallery request*

```
navigator.camera.getPicture(onSuccess, onFail, {
  sourceType: Camera.PictureSourceType.PHOTOLIBRARY,
  destinationType: Camera.DestinationType.FILE_URI
});
```

- update in the `sourceType` from `CAMERA` to `PHOTOLIBRARY`
- returned image respects original orientation of gallery image

Image - API Plugin Tester - Camera



API Plugin Tester - image from gallery

Cordova app - API plugin examples - plugin test 2

plugins - fix camera logic

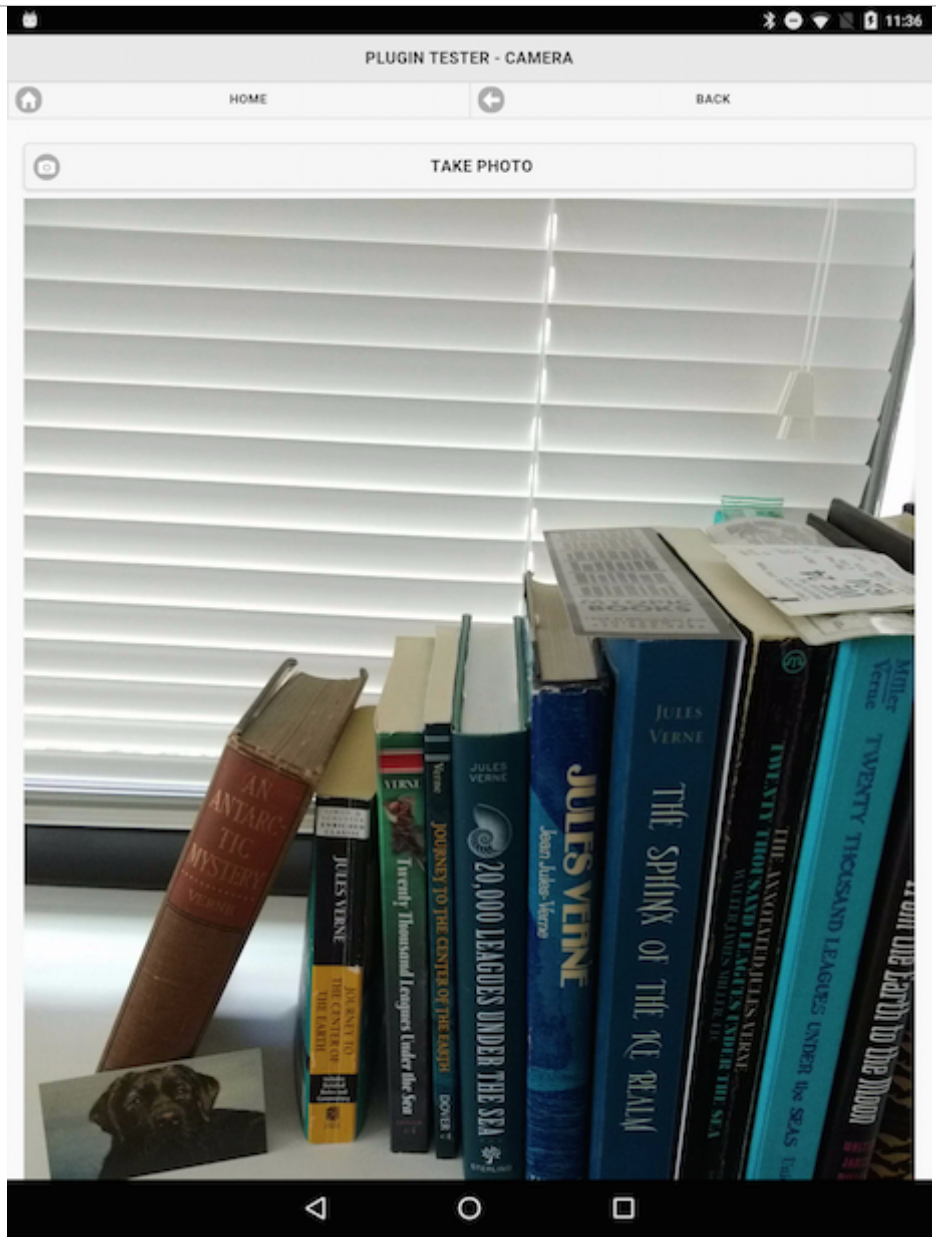
- need to fix the orientation issue with the returned image from the camera
- options for this plugin make it simple to update our logic for this requirement
  - *add a new option for the camera*

```
correctOrientation: true
```

- ensures that the original orientation of the camera is enforced
- updated logic is as follows

```
//Use from Camera
navigator.camera.getPicture(onSuccess, onFail, {
  quality: 50,
  correctOrientation: true,
  sourceType: Camera.PictureSourceType.CAMERA,
  destinationType: Camera.DestinationType.FILE_URI
});
```

Image - API Plugin Tester - Camera



API Plugin Tester - correct image orientation

Cordova app - API plugin examples - plugin test 2

plugins - camera updates

- continue to add many other useful options
  - *specifying front or back cameras on a device*
  - *type of media to allow*
  - *scaling of returned images*
  - *edit options...*
- in the app logic, also need to abstract the code further
  - *too much repetition in calls to the `navigator` object for the camera*
- then add more options and features
  - *save, delete, edit options*
  - *organise our images into albums*
  - *add some metadata for titles etc*
  - *add location tags for coordinates...*

Extra notes - mobile design considerations

Extra design notes will start to be added to the course website, GitHub...e.g.

- design mockups
- design and interface
- design and data
- ...

& extra notes on JS &c.

References

- Cordova API
  - *plugin - camera*
  - *plugin - Splashscreen*
- Cordova Guide
  - *app templates*