Comp 388/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2015 - Week 10

Dr Nick Hayward

Contents

Data considerations in mobile apps

- no one size fits all model for mobile
- can't just default to the server-side for reading and writing data
- our app may become useless if we rely heavily on remote data
  - *lose our network connection*
  - *run out of monthly data allowance*
  - *or end up with throttled or restricted data on a poor 2G network*
- Facebook's recent introduction of **2G Tuesdays**
  - *remind employees, developers of 2G limitations and issues around the world*
- also need to consider
  - *data security, read and write privileges for certain data stores, authentication for remote sources...*
- careful consideration of the options for reading and writing data
  - *a crucial aspect of our app's planning and subsequent development*

Cordova app - API plugin examples - plugin test 4

setup

- create our initial plugin test shell application

```
cordova create plugintest4 com.example.plugintest4 plugintest4
```

- add any required plaforms, eg: Android, iOS, Windows Phone...
  - *we'll add iOS as well*

```
cordova platform add android
```

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

Cordova app - API plugin examples - plugin test 4

setup

- also add support for iOS development

```
cordova platform add ios
```

- running a test application on iOS is not as simple as Android
- need to add support to Cordova for a local iOS simulator
  - *add package for iOS simulator using **npm***
  - ***NB**: may require admin or sudo permissions to install correctly*

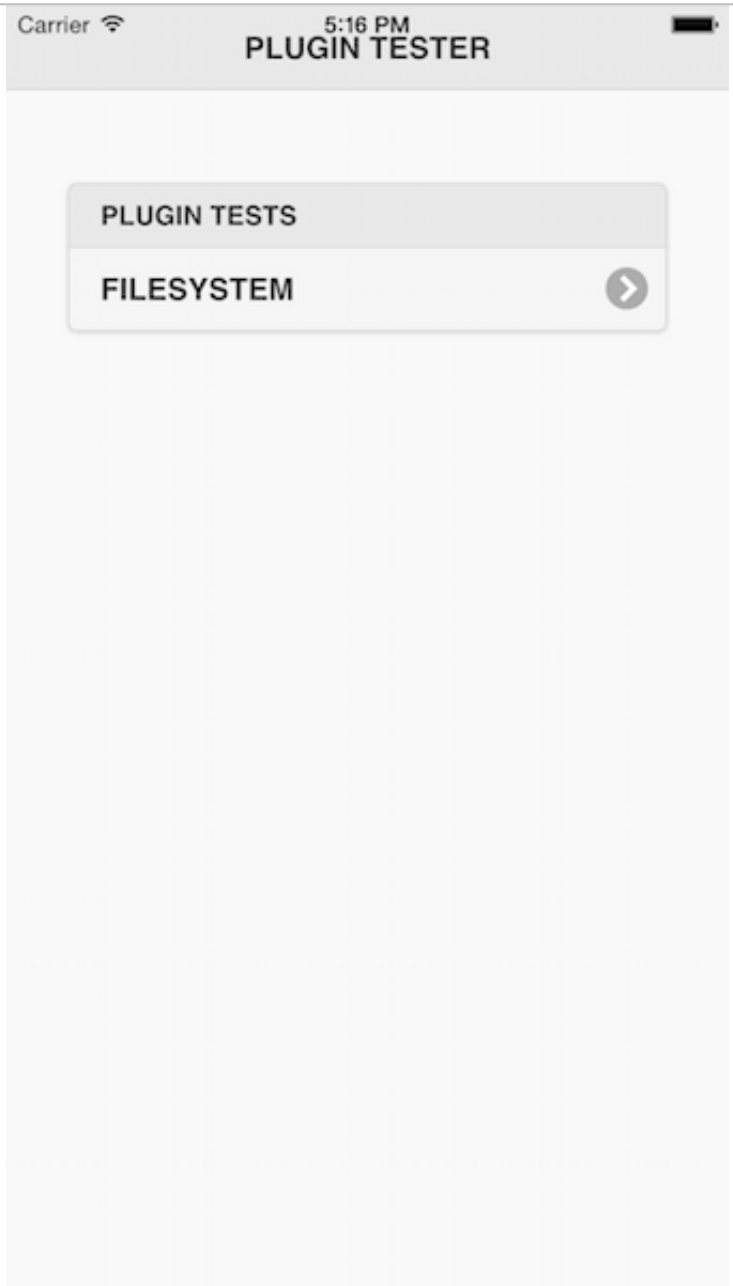
```
npm install -g ios-sim
```

- then run our Cordova app from the working directory

```
cordova run ios
```

- Cordova will try to load the application using this local simulator
  - *without defaulting to XCode (and Apple's $99 annual fee)*
- quickly test our iOS application with this simulator

Image - iOS Local Simulator



iOS Simulator - running locally on OS X

Cordova app - API plugin examples - plugin test 4

iOS simulator - options

- iOS simulator gives us many useful options
  - *helpful ways to test our local Cordova based iOS applications*
- emulate many different devices
  - *from the iPhone 6 Plus to the iPad Air*
- mimic many of these device's hardware features
  - *such as rotate, shake, different keyboards...*
  - *also output to a simulated Apple Watch device, 38mm & 42mm*
- various debugging options available within this simulator
  - *including ability to mimic locations for GPS enabled applications*
- quickly take a screenshot of the current application screen within the simulator

Cordova app - API plugin examples - plugin test 4

application structure

- now updated our initial Cordova template
  - *better structure for plugin test application*
  - *structure is now 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 - API plugin examples - plugin test 4

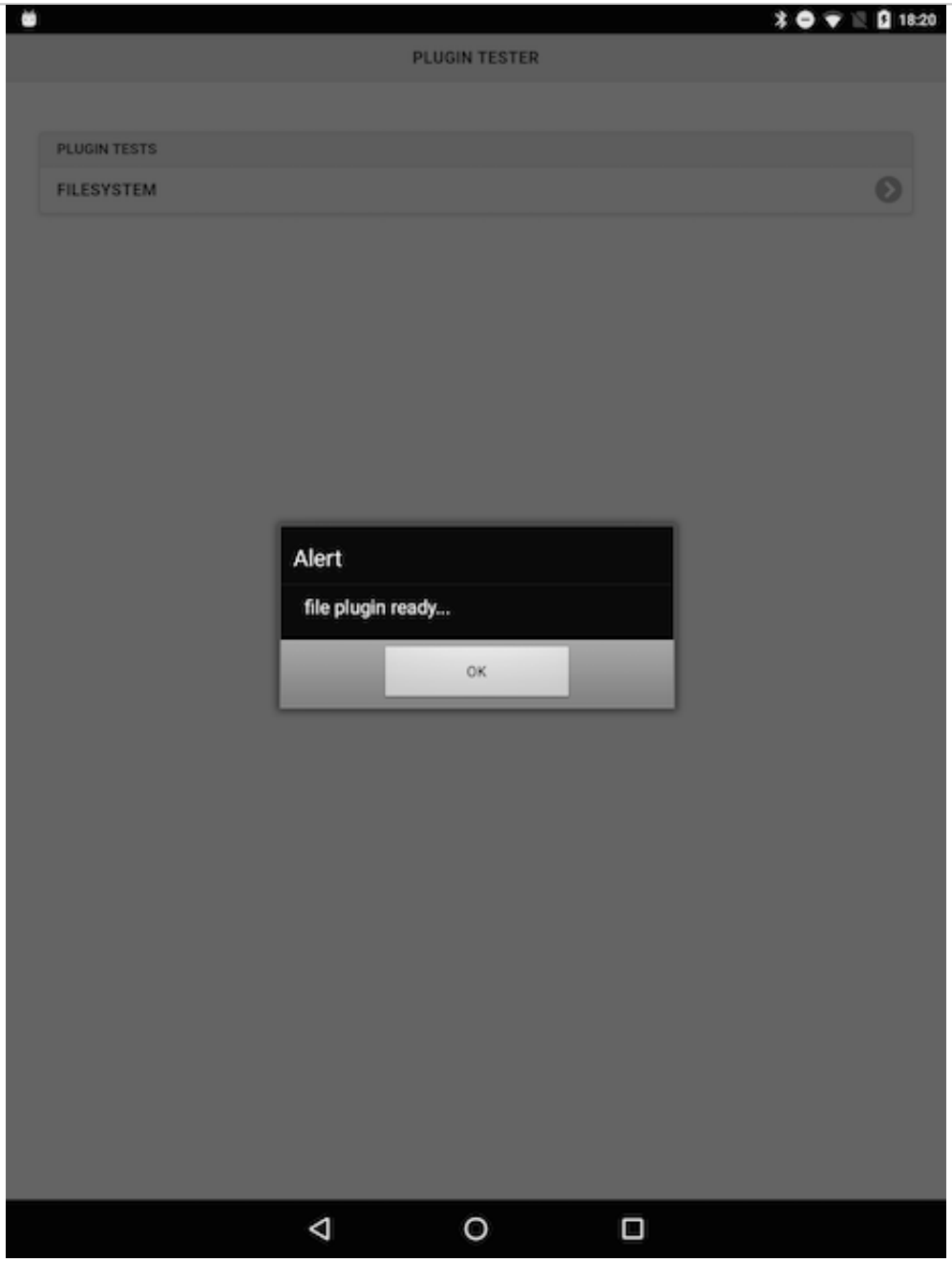
plugins - add filesystem

- add and use the **file** plugin
- plugin has been designed to permit read and write access to files
  - *files are stored on the local device for Cordova applications*
- **file** plugin is initially based on open specifications
  - *includes the **HTML5 File API**, W3's **FileWriter** specification...*
- add the file plugin to our test application using the standard CLI command

```
cordova plugin add cordova-plugin-file
```

- command will install plugin for all current platforms
  - *includes Android and iOS for our test application*

Image - API Plugin Tester - file



API Plugin Tester - file plugin ready

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem

- using this plugin we can read local files from within the filesystem
- we could read a file from within our Cordova application
  - *eg: located in the following directory*

```
...
|- www
   |- docs
      |- txt
         |- madeira.txt
```

- we can use the available global `cordova.file` object
- to be able to use the URL for our text document in the file-system directory
  - *convert it to a `DirectoryEntry` using*

```
window.resolveLocalFileSystemURL()
```

- in our standard `onDeviceReady()` function
  - *use this global object to resolve the URL of our file*
  - *then pass to specified callbacks for success and fail*

```
window.resolveLocalFileSystemURL(cordova.file.applicationDirectory +
"www/docs/txt/madeira.txt", onSuccess, onFail);
```

Image - API Plugin Tester - file

API Plugin Tester - read an app txt file

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem `onSuccess`

- render this text after retrieving from the requested file
  - *update our `onSuccess()` function to output the file's content*

```
function onSuccess(data) {
  data.file(function(file) {
    var readFile = new FileReader();
    readFile.onloadend = function(e) {
      //use jQuery selector to add returned file data
      $("#file-output").html(this.result);
    }
    readFile.readAsText(file);
  });
}
```

- call the `file()` method on our returned file data
  - *effectively gives us a hook/handle into the file*
  - *we can now work with the returned file data*
- then call the `FileReader()` method from the **FileAPI**
  - *and process the returned text*
- output to our specified HTML element
  - *using a standard jQuery selector with the `html()` method*

Cordova app - API plugin examples - plugin test 4

plugins - test filesystem `onFail()`

- complement to the `onSuccess()` function
- now add our function `onFail()` for the fail callback
- test it with the returned error code

```
function onFail(error) {
  console.log("FileSystem Error"+error.code);
  $("#file-output").html("file plugin error - "+error.code);
}
```

- uses the passed error object
  - *returns a code for rendering in the specified jQuery selector*
- obviously does not make a lot of sense to our user

Cordova app - API plugin examples - plugin test 4

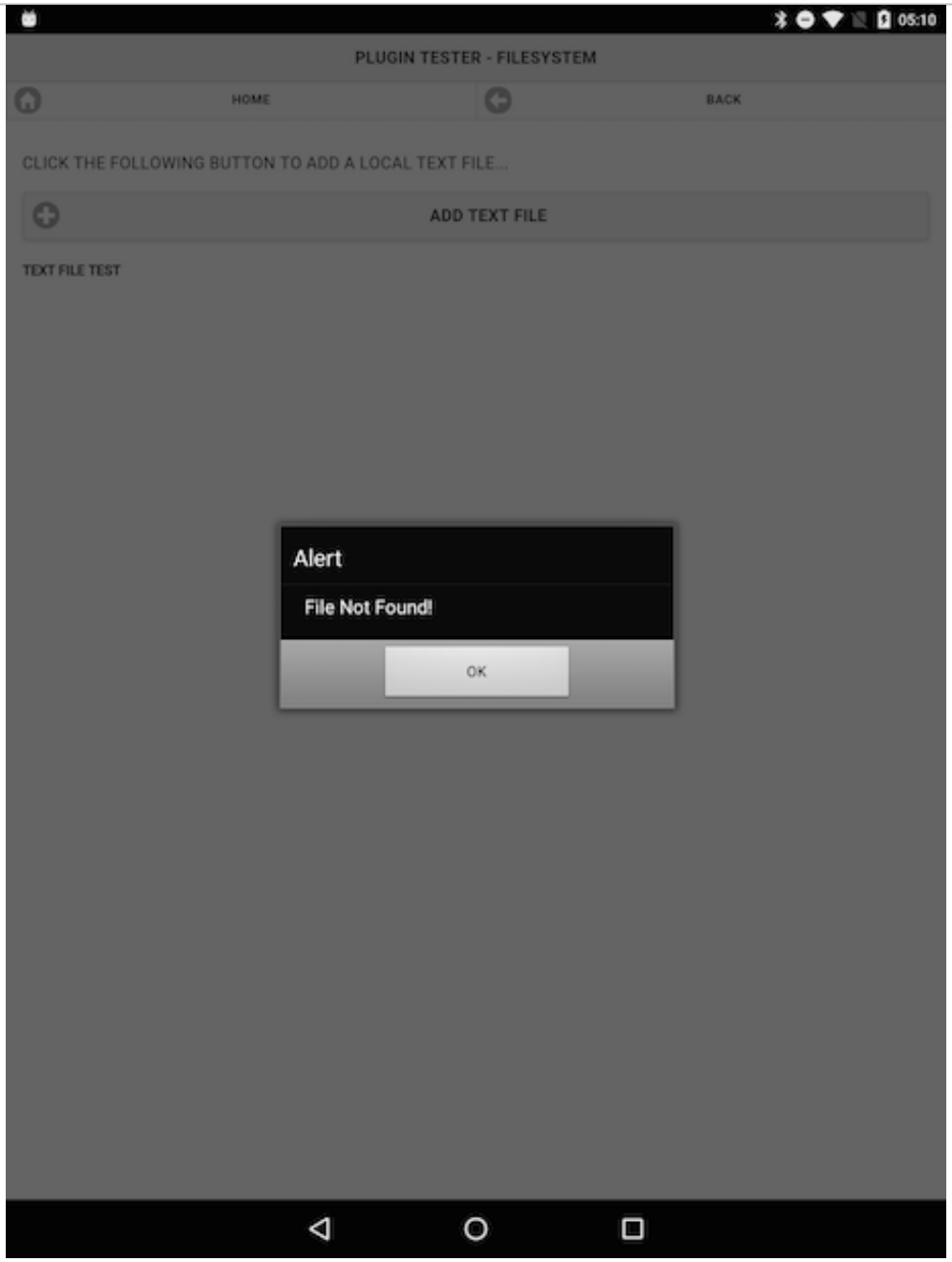
plugins - test filesystem `onFail()`

- we can use a conditional statement to check for certain returned error codes
  - *then output a meaningful error message to the user in the application*

```
function onFail(error) {
  switch(error.code) {
    case 1:
    alert('File Not Found!');
    break;
    //add other options to cover additional error codes...
    default:
    alert('An error occurred reading this file.');
  };
}
```

- now output more graceful error messages and feedback to the user
- Web APIs - FileError

Image - API Plugin Tester - file



API Plugin Tester - output error message

Cordova app - API plugin examples - plugin test 4

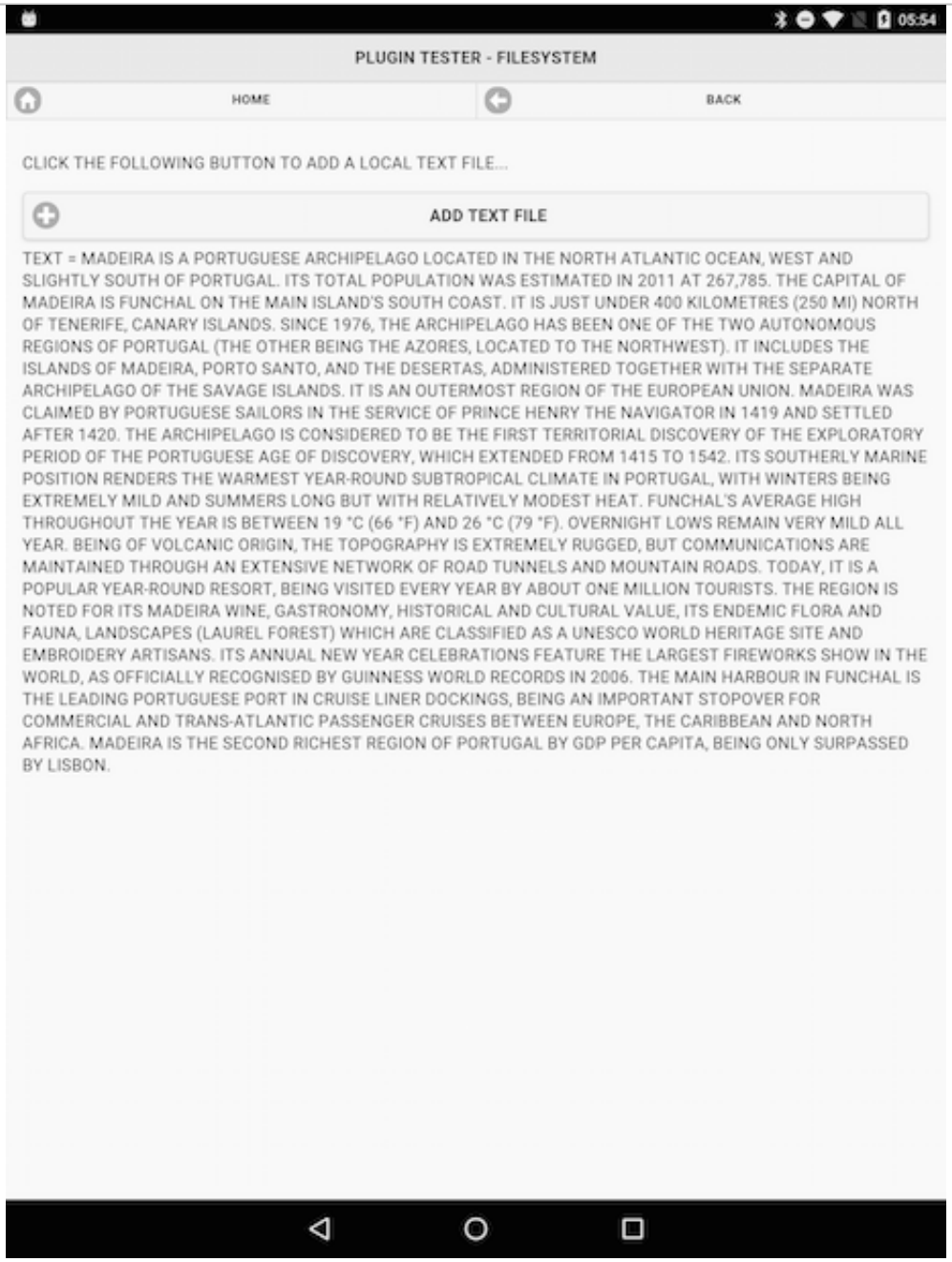
plugins - test filesystem with event

- easily link file loading to a given event, such as a user tap event
- instead of loading the file by default with the `onDeviceReady()` function
  - *get the contents of our file when needed by the user*
- link this to a button event, a separate page init event...

```
//handle button press for file load
$("#getFile").on("tap", function(e) {
  e.preventDefault();
  getTxtFile();
});
```

- then call our local file as before within its own function, `getTxtFile()`

Image - API Plugin Tester - file



API Plugin Tester - event file loader

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- now read files from the local device's native storage thanks to Cordova's File plugin
- file plugin also offers an option to write to files in the same local filesystem
- quickly create a test app for writing to files
  - *eg: plugintest5*
- create your project

```
cordova create plugintest5 com.example.plugintest5 plugintest5
```

- `cd` to app's working directory
- add required platforms

```
cordova platform add android
```

- add our required Cordova API plugin for working with the file system

```
cordova plugin add cordova-plugin-file
```

- run usual initial tests for app loading, `deviceready` event...

Cordova app - API plugin examples - plugin test 5

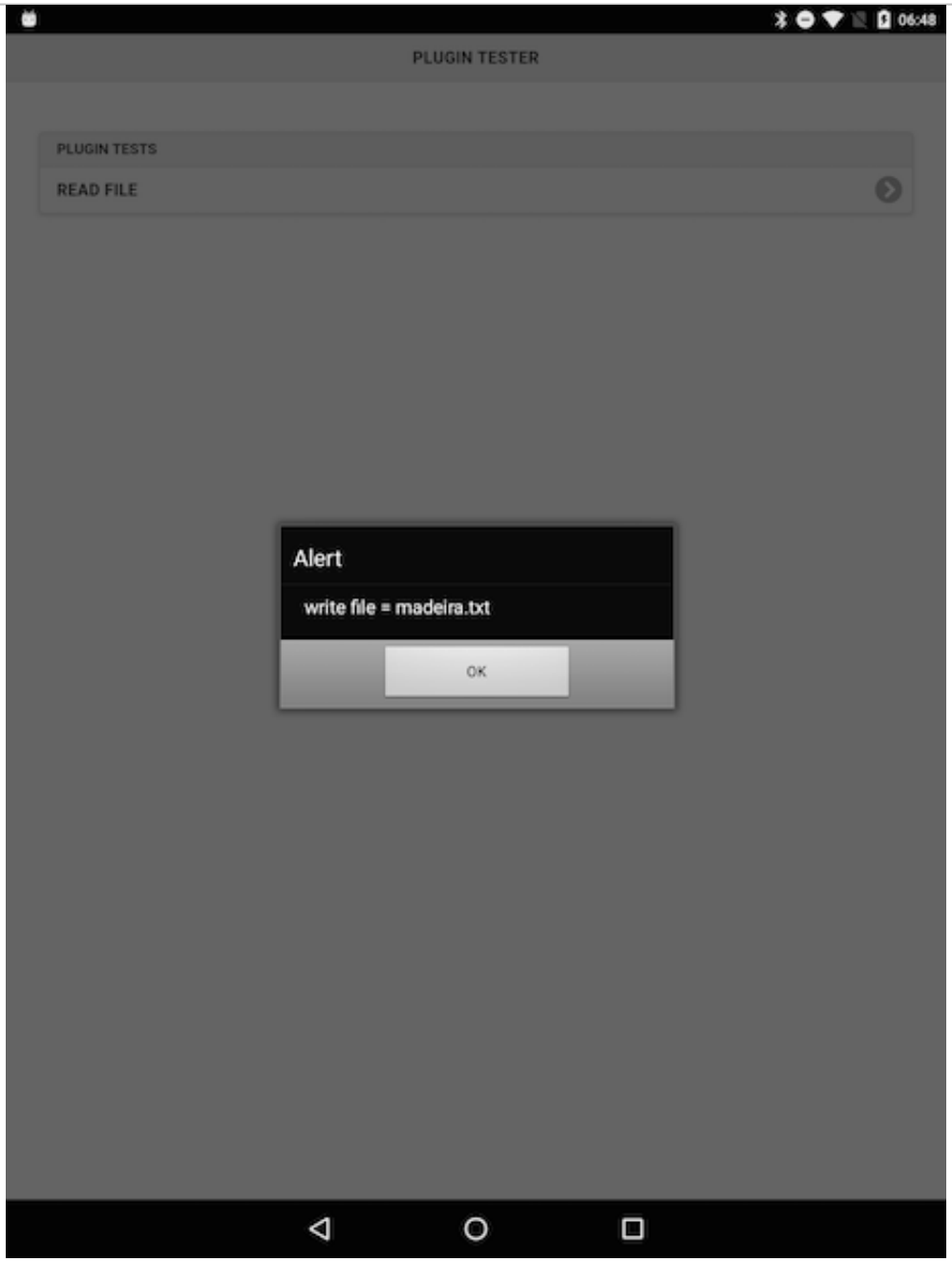
plugins - test filesystem with file write

- now start to add writing to a file to our test app
- start, as we did with file reading, by getting a hook/handle to a file
- we can then write to a file within the assigned app's data directory
  - *specific app directory has read and write access*
  - *allows us to create files as needed for our app*
  - *then read and write within the confines of the native app*
- use `window.resolveLocalFileSystemURL` to allow us to work with this data directory

```
var fileDir = cordova.file.dataDirectory;
window.resolveLocalFileSystemURL(fileDir, function(dir) {
// do something useful...
});
```

- in application specific directory get our required file for writing

Image - API Plugin Tester - file



API Plugin Tester - get file for writing

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- create a new file if it doesn't exist on app loading
- use directory object with `getFile()` method etc...
  - *set flag to create a new file*

```
window.resolveLocalFileSystemURL(fileDir, function(dir) {
dir.getFile("madeira.txt", {create:true}, function(file) {
//do something useful
});
});
```

- pass file object to other functions for processing...
- create our write function to check and write to specified file within app's data directory

Cordova app - API plugin examples - plugin test 5

plugins - test filesystem with file write

- now write some simple text to our file

```
function writeTxtFile(data) {
  //check passed data for writing
  if (data !== "") {
    //new text to write to file
    var text = data;
    //use write file object
    writeObj.createWriter(function(writeFile) {
      //call seek() to ensure we append to end of file
      writeFile.seek(writeFile.length);
      //create raw Blob for writing
      var textBlob = new Blob([text], {type:'text/plain'});
      //write to the end of the file
      writeFile.write(textBlob);
    });
  }
}
```

Cordova app - API plugin examples - plugin test 5

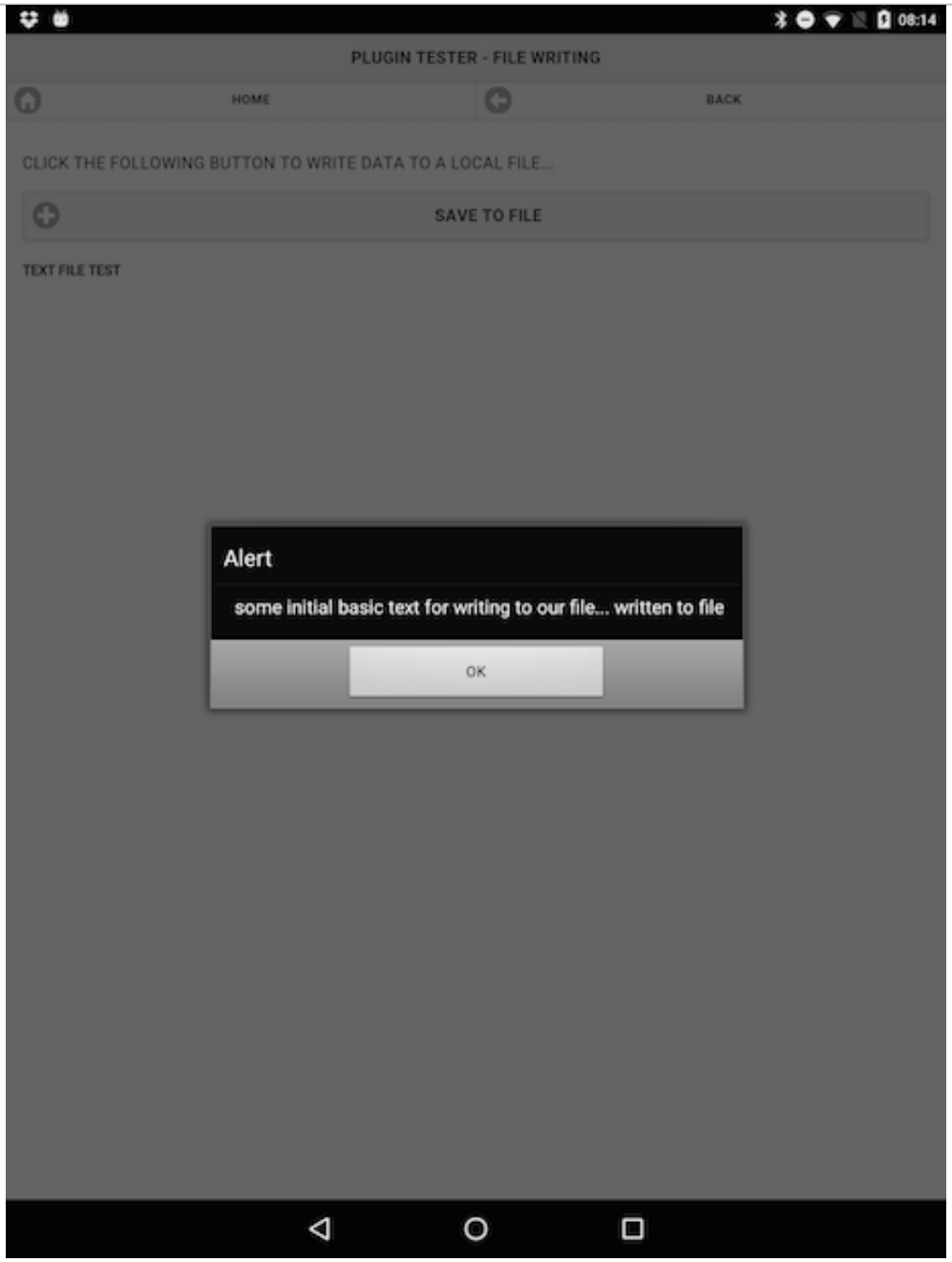
plugins - test filesystem with file write

- then call this `writeTxt()` as needed within our application
  - *eg: calling it from event handler for a button tap*

```
//handle button press for file write
$("#saveFile").on("tap", function(e) {
  e.preventDefault();
  writeTxtFile("some initial basic text for writing to our file...");
});
```

- could easily get text to write from an input field, from metadata...
- then pass it to our `writeTxtFile()` function for writing

Image - API Plugin Tester - file



API Plugin Tester - text written to file

Cordova app - JS data options - JS data test 1

read local JSON file - .getJSON()

- return to working with the native local filesystem for TripNote app
  - *eg: we'll look at reading and writing to an SD card on Android*
- we'll now consider other options available to us
  - *due to our use of JavaScript in the Cordova stack*
  - *use many jQuery functions to access local and remote data*
- test loading some local JSON files
  - *we can use jQuery's `.getJSON()` method*
- create our new project, **jstest1**, add required platforms, new JS and JSON files...

```
|- www
   |- assets
      |- scripts
         |- app.js
         |- json_loader.js
```

```
...
|- www
   |- docs
      |- json
         |- madeira.json
```

Cordova app - JS data options - JS data test I

read local JSON file - .getJSON()

- use the following initial test JSON

```
{
  "travelNotes": [{
    "created": "2015-10-12T00:00:00Z",
    "note": "Curral das Freiras is a civil parish..."
  }, {
    "created": "2015-10-13T00:00:00Z",
    "note": "Câmara de Lobos (Portuguese pronunciation: [literally, Portuguese: chamber of
  }, {
    "created": "2015-10-14T00:00:00Z",
    "note": "Funchal is the largest city, the municipal seat and the capital of Madeira...
  }]
}
```

Cordova app - JS data options - JS data test I

read local JSON file - .getJSON()

- add an inital loader for our JSON file
- initially use in response to the `deviceready` event
- then we can move it to our separate `json_loader.js` file
  - *use with various handlers and other functions...*

```
//returns a deferred object - limited scope promise
var $deferredNotesRequest = $.getJSON (
  "docs/json/madeira.json",
  {format: "json"}
);
```

- returns a deferred object with a limited scope **promise**
- then use `$deferredNotesRequest` to work with returned JSON

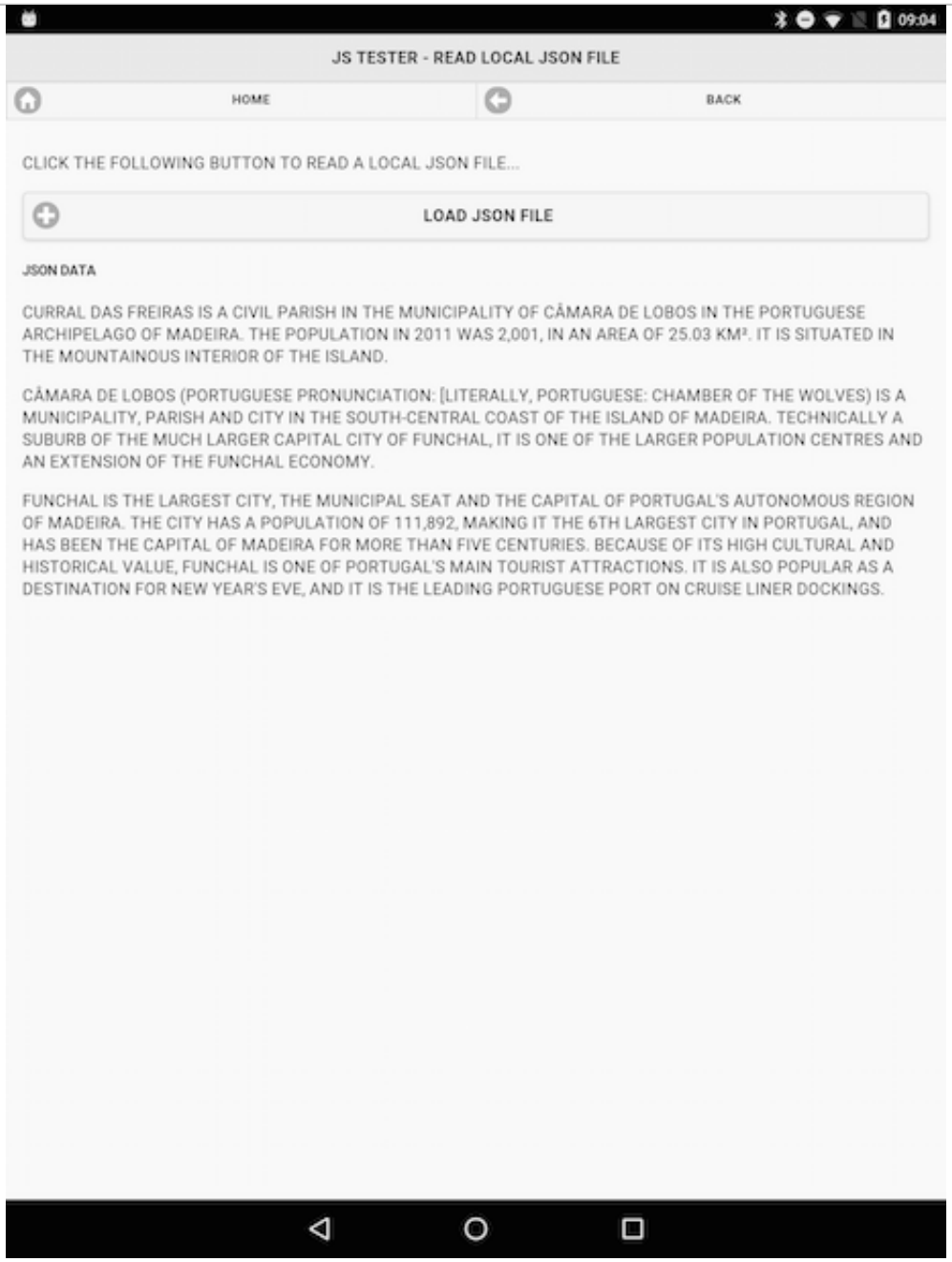
Cordova app - JS data options - JS data test 1

read local JSON file - .getJSON()

- we can use the **deferred** object to get our notes from the JSON
- build each note as a paragraph, and append to DOM

```
$deferredNotesRequest.done(function(response) {
    //get travelNotes
    var $travelNotes = response.travelNotes;
    //process travelNotes array
    $.each($travelNotes, function(i, item) {
      if (item !== null) {
        var note = item.note;
        //create each note's <p>
        var p = $("<p>");
        //add note text
        p.html(note);
        //append to DOM
        $("#note-output").append(p);
      }
    });
});
```

Image - API Plugin Tester - file



JS Tester - load and render local JSON

Cordova app - JS data options - JS data test 1

read local JSON file - jQuery deferred pattern

- jQuery provides a useful solution to the escalation of code for asynchronous development
- known as the `$.Deferred` object
  - *effectively acts as a central despatch and scheduler for our events*
- with the **deferred** object created
  - *parts of the code indicate they need to know when an event completes*
  - *whilst other parts of the code signal an event's status*
- **deferred** coordinates different activities
  - *enables us to separate how we trigger and manage events*
  - *from having to deal with their consequences*

Cordova app - JS data options - JS data test I

read local JSON file - using deferred objects

- now update our AJAX request with **deferred** objects
- separate the asynchronous request
  - *into the initiation of the event, the AJAX request*
  - *from having to deal with its consequences, essentially processing the response*
- separation in logic
  - *no longer need a success function acting as a callback parameter to the request itself*
- now rely on `.getJSON()` call returning a **deferred** object
- function returns a restricted form of this **deferred** object
  - *known as a **promise***

```
deferredRequest = $.getJSON (
  "file.json",
  {format: "json"}
);
```

Cordova app - JS data options - JS data test 1

read local JSON file - using deferred objects

- indicate our interest in knowing when the AJAX request is complete and ready for use

```
deferredRequest.done(function(response) {
  //do something useful...
});
```

- key part of this logic is the `done()` function
- specifying a new function to execute
  - *each and every time the event is successful and returns complete*
  - *our AJAX request in this example*
- **deferred** object is able to handle the abstraction within the logic
- if the event is already complete by the time we register the callback via the `done()` function
  - *our **deferred** object will execute that callback immediately*
- if the event is not complete
  - *it will simply wait until the request is complete*

Cordova app - JS data options - JS data test I

read local JSON file - error handling deferred objects

- also signify interest in knowing if the AJAX request fails
- instead of simply calling `done()`, we can use the `fail()` function
- still works with JSONP
  - *the request itself could fail and be the reason for the error or failure*

```
deferredRequest.fail(function() {
  //report and handle the error...
});
```

Cordova app - JS data options - JS data test 1

read local JSON file - working with deferred objects

`resolve()`

- use this method with the deferred object to change its state, effectively to complete
- as we resolve a deferred object
  - *any **doneCallbacks** added with `then()` or `done()` methods will be called*
  - *these callbacks will then be executed in the order added to the object*
  - *arguments supplied to `resolve()` method will be passed to these callbacks*

`promise()`

- useful for limiting or restricting what can be done to the deferred object

```
function returnPromise() {
  return $.Deferred().promise();
}
```

- method returns an object with a similar interface to a standard `deferred` object
  - *only has methods to allow us to attach callbacks*
  - *does not have the methods required to resolve or reject deferred object*
- restricting the usage and manipulation of the deferred object
  - *eg: offer an API or other request the option to subscribe to the deferred object*
  - ***NB:** they won't be able to resolve or reject it as standard*

Cordova app - JS data options - JS data test 1

read local JSON file - working with deferred objects

- still use the `done()` and `fail()` methods as normal
- use additional methods with these callbacks including the `then()` method
- use this method to return a new promise
  - *use to update the status and values of the deferred object*
  - *use this method to modify or update a deferred object as it is resolved, rejected, or still in use*
- can also combine promises with the `when()` method
  - *method allows us to accept many promises, then return a sort of master deferred*
- updated `deferred` object will now be resolved when all of the promises are resolved
  - *it will likewise be rejected if any of these promises fail*
- use standard `done()` method to work with results from all of the promises
  - *eg: could use this pattern to combine results from multiple JSON files*
  - *multiple layers within an API*
  - *staggered calls to paged results in a API...*

Cordova app - JS data options - JS data test I

read local JSON file - update test app

- now start to update our test AJAX and JSON application
  - *begin by simply abstracting our code a little*

```
//get the notes JSON
function getNotes() {
    //return limited deferred promise object
    var $deferredNotesRequest = $.getJSON (
      "docs/json/madeira.json",
      {format: "json"}
    );
    return $deferredNotesRequest;
}

function buildNote(data) {
    //create each note's <p>
    var p = $("<p>");
    //add note text
    p.html(data);
    //append to DOM
    $("#note-output").append(p);
}
```

Cordova app - JS data options - JS data test 1

read local JSON file - working with a `promise`

- requesting our JSON file using `.getJSON()`
  - *we get a returned **promise** for the data*
- with a **promise** we can only use the following
  - *deferred object's method required to attach any additional handlers*
  - *or determine its state*
- our **promise** can work with
  - *`then`, `done`, `fail`, `always`...*
- our **promise** can't work with
  - *`resolve`, `reject`, `notify`...*
- one of the benefits of using **promises** is the ability to load one JSON file
  - *then wait for the results*
  - *then issue a follow-on request to another file*
  - *...*

Cordova app - JS data options - JS data test I

read local JSON file - update test app

- add our `.when()` function to app
  - *`.when()` function accepts a deferred object*
  - *in our case a limited promise*
- then allows us to chain additional deferred functions
  - *including required `.done()` function*
- for returned data, use standard response object to get `travelNotes`
  - *then iterate over the array for each property*
  - *for each iteration, we can simply call our `buildNote` function*
  - *builds and renders required notes to the app's DOM*

```
$.when(getNotes()).done(function(response) {
  //get travelNotes
  var $travelNotes = response.travelNotes
  //process travelNotes array
  $.each($travelNotes, function(i, item) {
    if (item !== null) {
      var note = item.note;
      console.log(note);
      buildNote(note)
    }
  });
});
```

Cordova app - JS data options - JS data test I

read local JSON file - update test app

- use this `.when()` function in a new function, called `.processNotes()`
- call our deferred promise object from an event handler...

```
function processNotes(){
  $.when(getNotes()).done(function(response) {
    //get travelNotes
    var $travelNotes = response.travelNotes
    //process travelNotes array
    $.each($travelNotes, function(i, item) {
      if (item !== null) {
        var note = item.note;
        console.log(note);
        buildNote(note)
      }
    });
    console.log("done..."+response.travelNotes[0].note);
  });
}
```

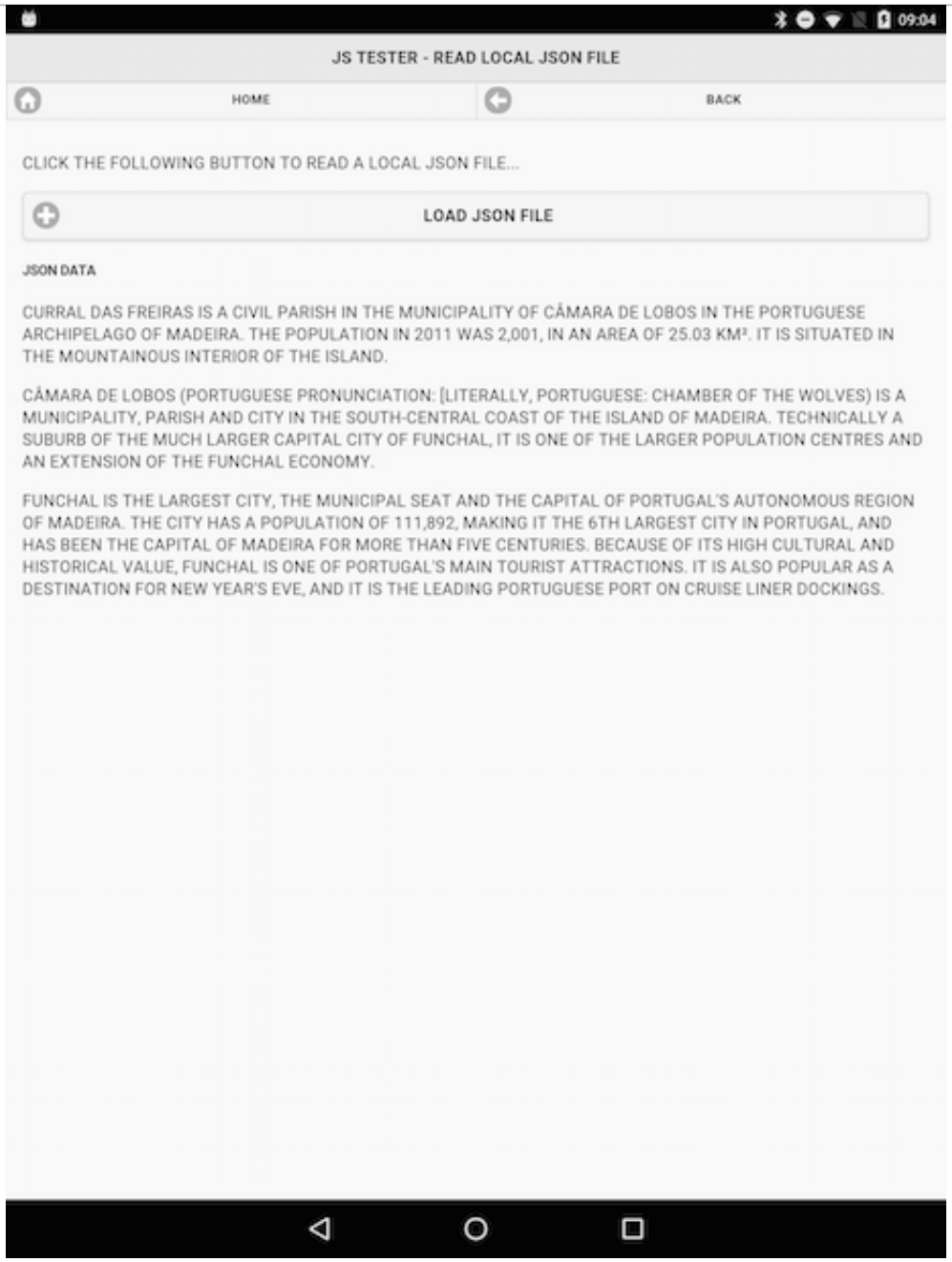
Cordova app - JS data options - JS data test 1

read local JSON file - update test app

- as we navigate to our JSON page in the test app
  - *call this function from an event handler...*

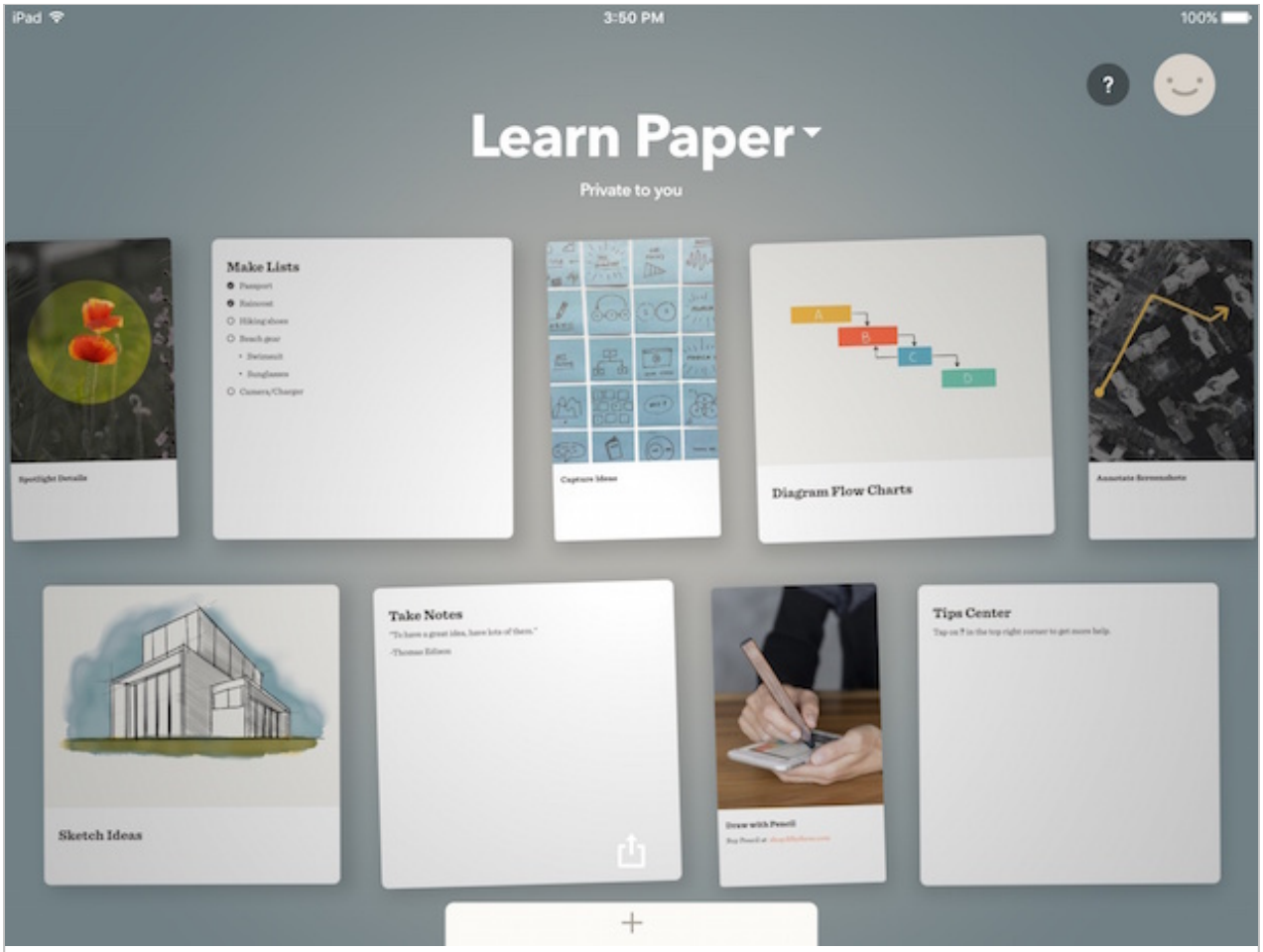
```
//handle button press for file write
$("#loadJSON").on("tap", function(e) {
  e.preventDefault();
  processNotes();
});
```

Image - API Plugin Tester - file



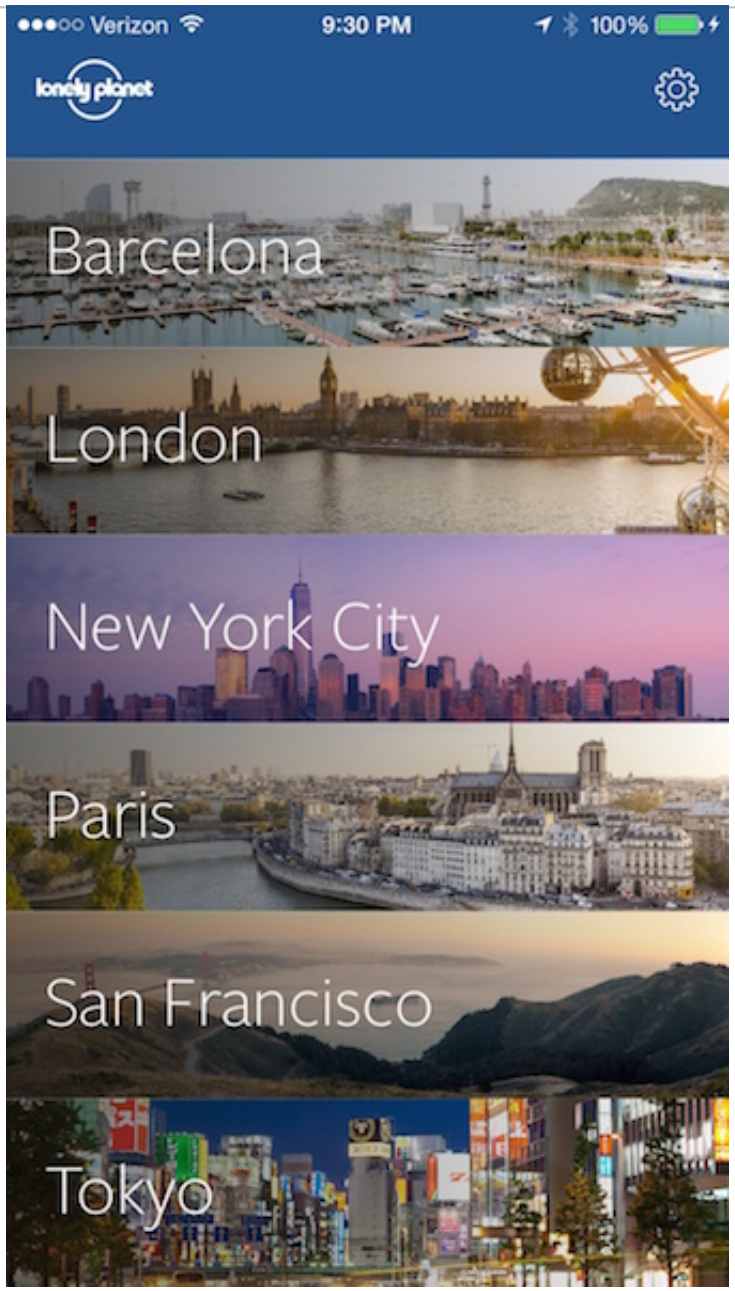
JS Tester - JSON deferred pattern

Considering mobile design patterns - screen 1



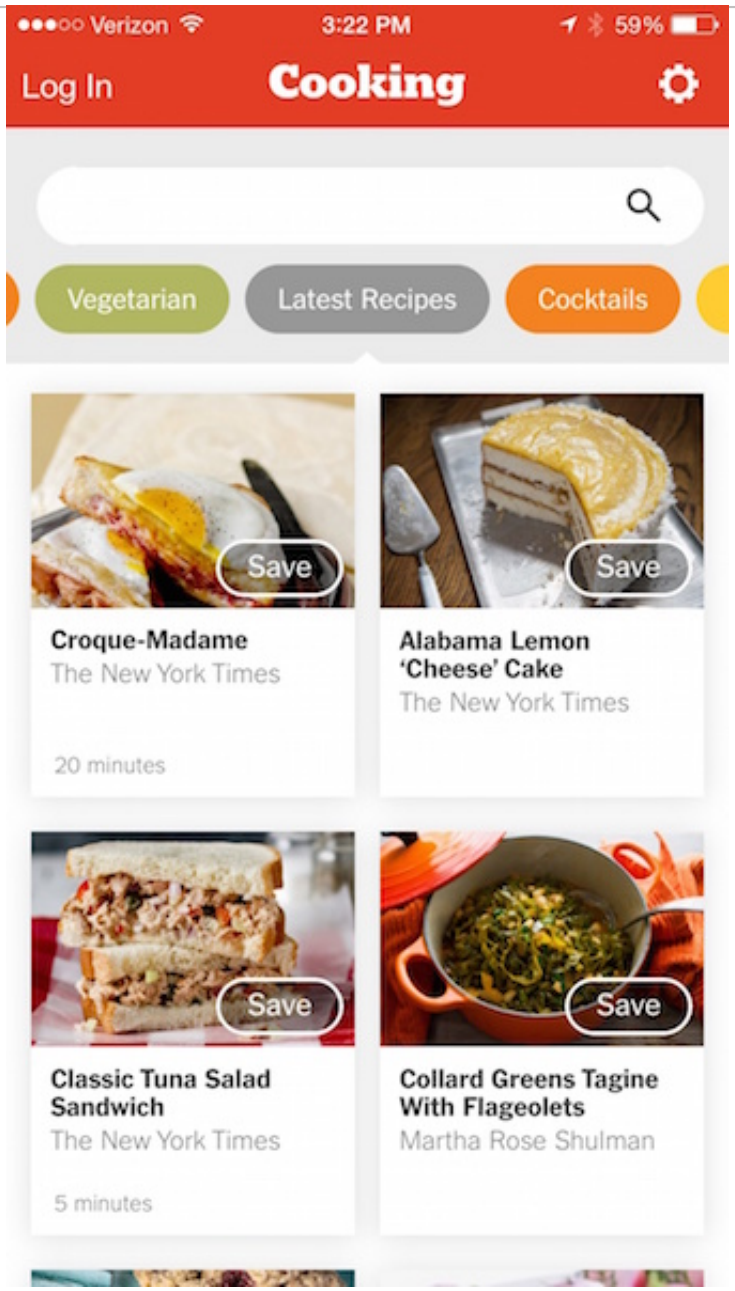
Paper on iOS

Considering mobile design patterns - screen 2



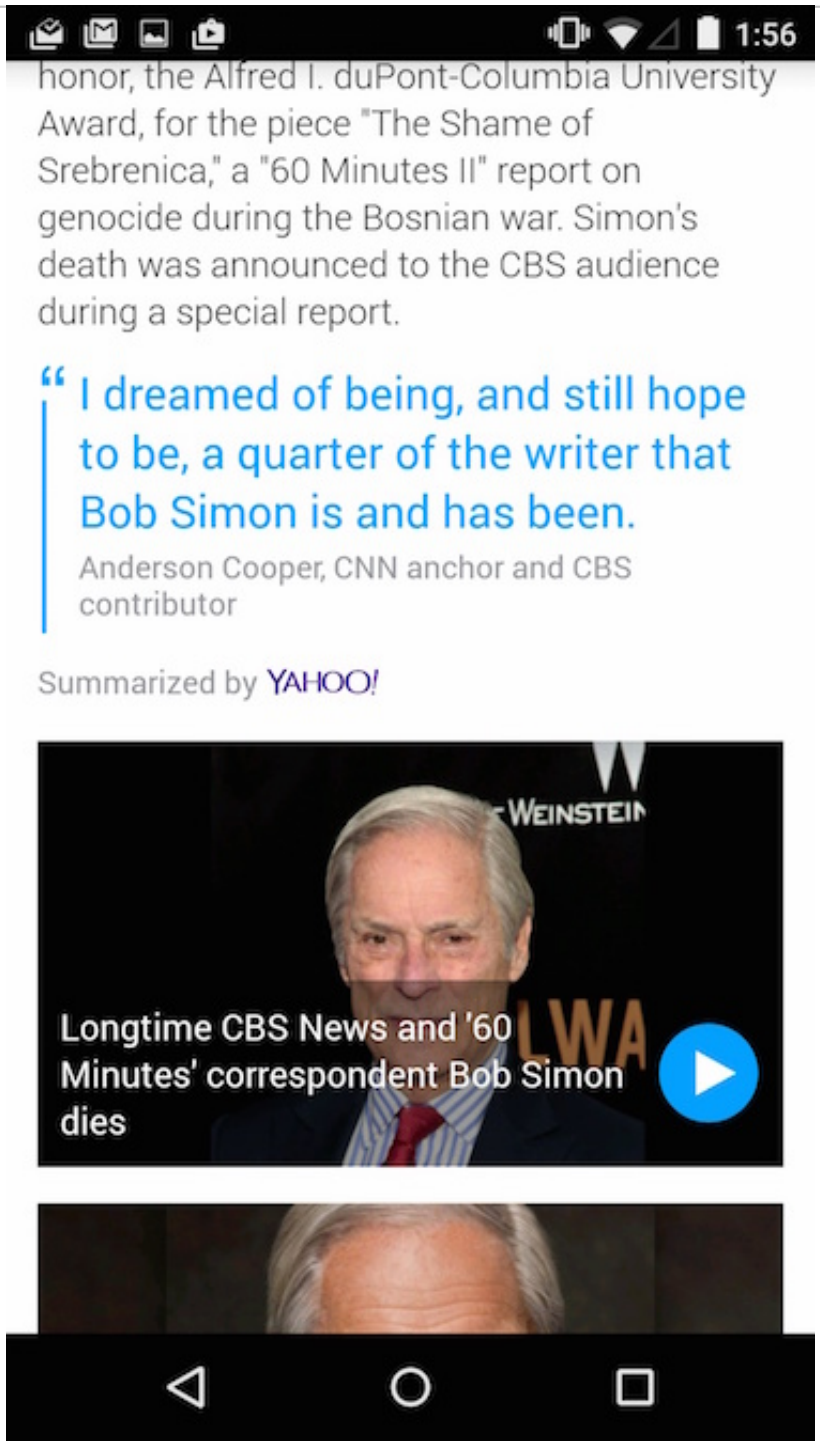
Lonely Planet on iOS

Considering mobile design patterns - screen 3



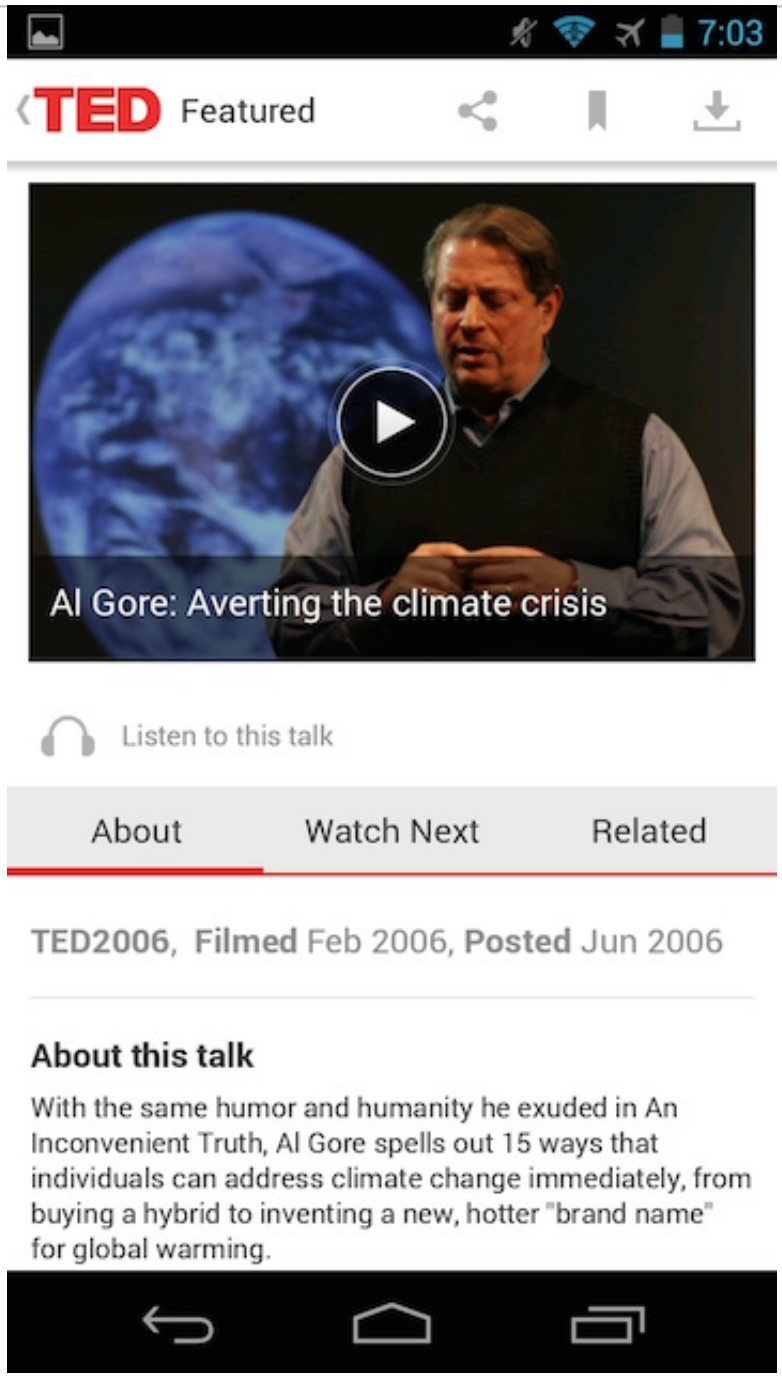
NY Times Cooking on iOS

Considering mobile design patterns - screen 4



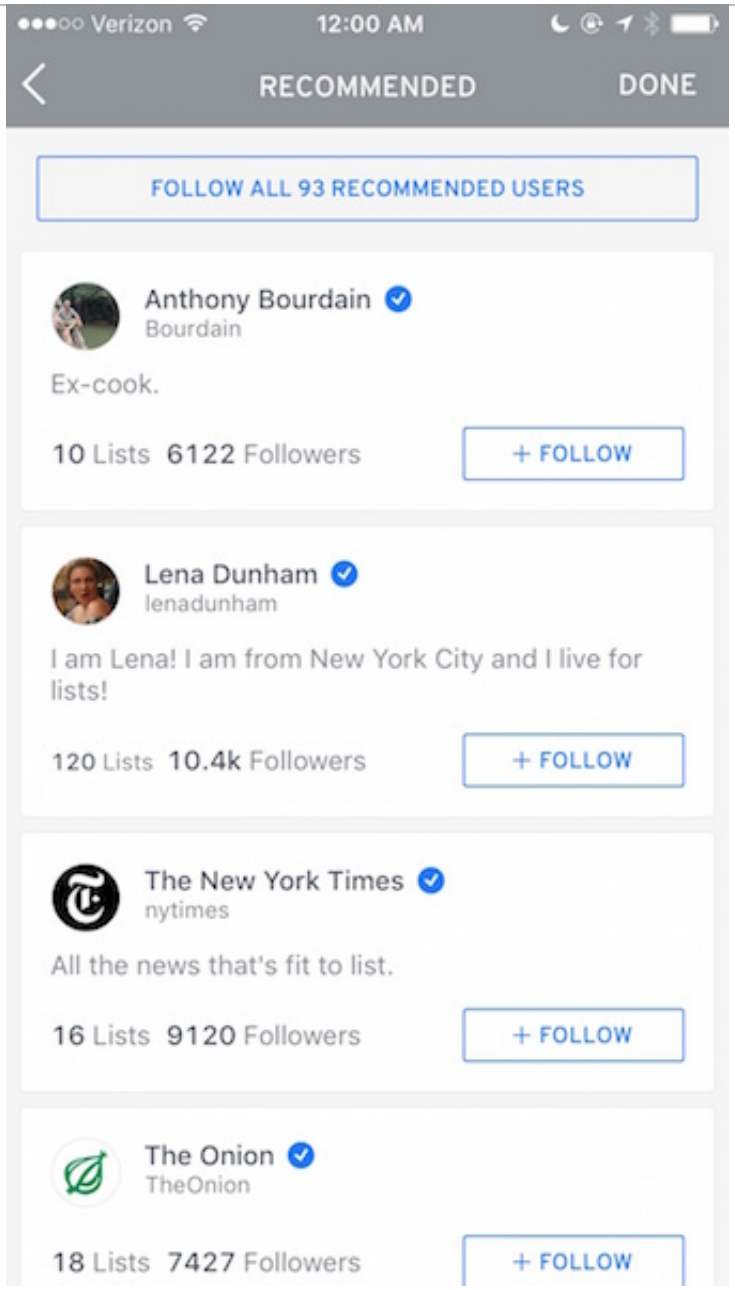
Yahoo News Digest on Android

Considering mobile design patterns - screen 5



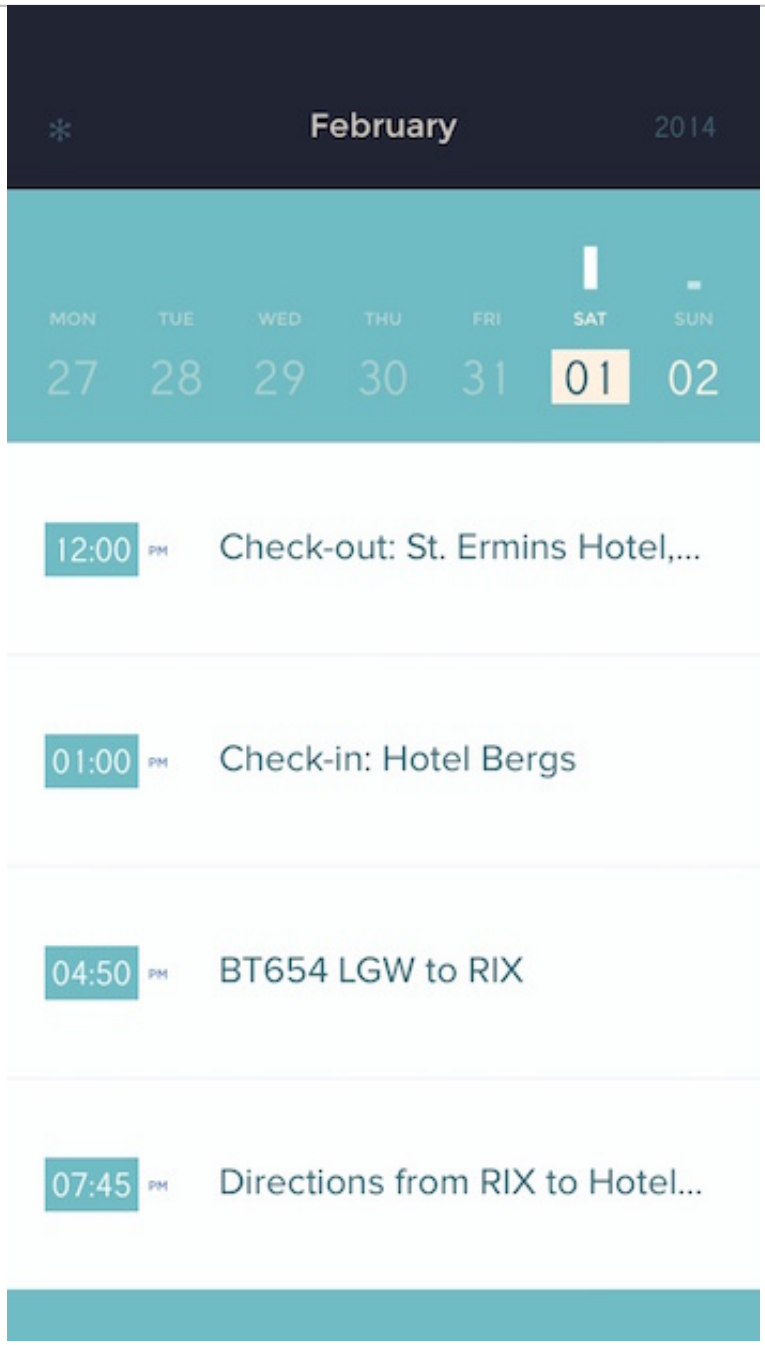
TED on Android

Considering mobile design patterns - screen 6



The List App on iOS

Considering mobile design patterns - screen 7



Peek on iOS

Demos

- Cordova test apps
  - *plugintest4*
  - *plugintest5*
  - *jstest1*

References

- Cordova
  - *Cordova API - filesystem plugin*
  - *Cordova API - file transfer plugin*
  - *Cordova Storage*
- GitHub
  - *cordova-plugin-file*
- HTML5
  - *HTML5 File API*
- MDN
  - *Web APIs - FileError*