Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 9 Notes

Dr Nick Hayward

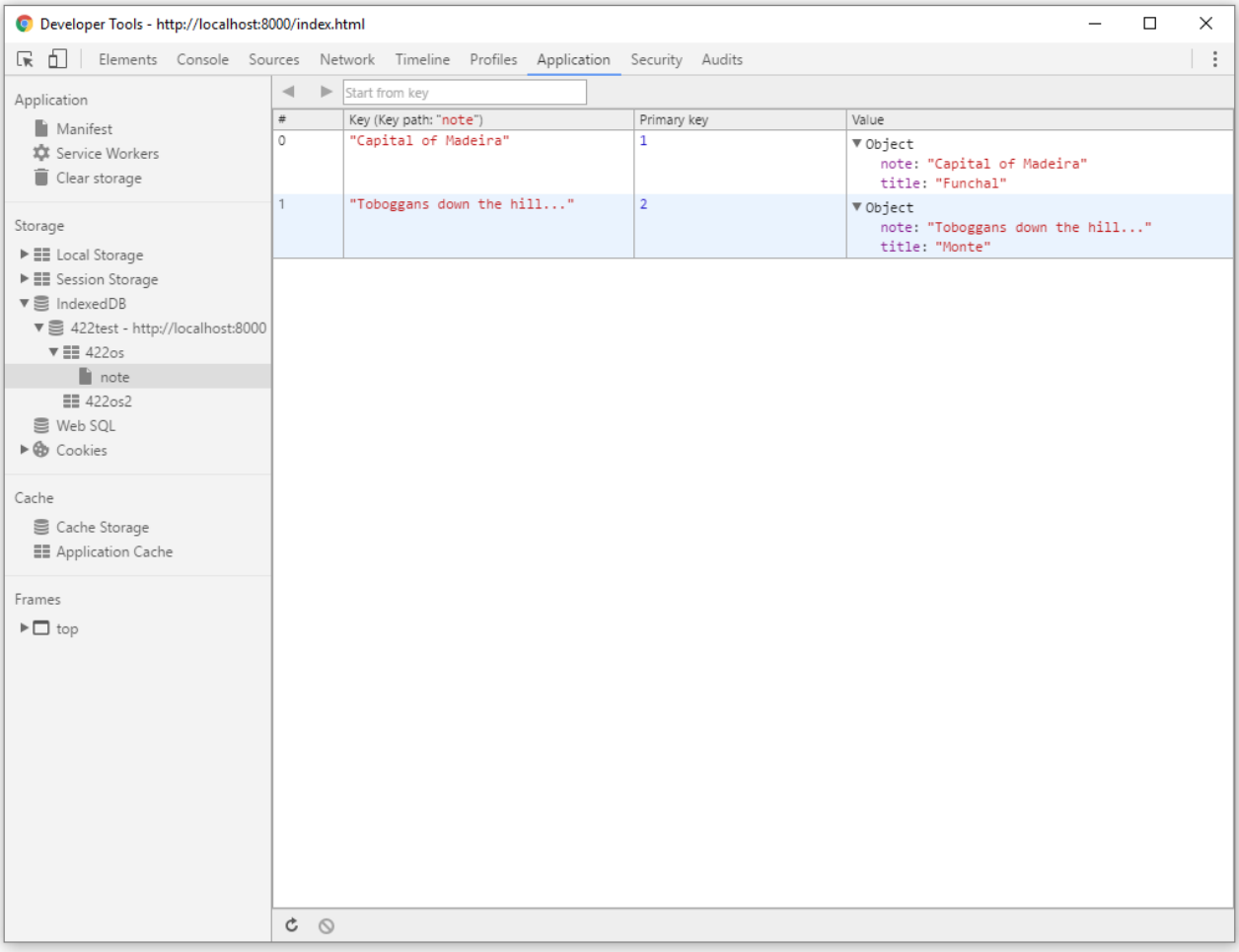
Contents

Cordova app - IndexedDB - Recap

Material covered so far:

- general intro
- checked IndexedDB availability as part of `deviceready` event
  - *created reference for later use...*
- general usage
  - *connection &c.*
- event listeners
  - `success`, `error`, `upgradeneeded`, `blocked`
- create a new DB
  - *check persistence*
  - *work with success and fail callbacks*
- object stores
- add data
- work with data handlers
- multiple object stores, notes...
- keys
- ...

Image - IndexedDB Support



DataTest2 - test IndexedDB - unique keys 2

Cordova app - IndexedDB - data test 2

database - part 16 - read data

- now able to save our notes to the IndexedDB
- need to read this data, and then load it into our application
- use the same underlying pattern for read and write
  - *use a transaction, and the request will be asynchronous*
  - *modify our transaction for* `readonly`

```
// create transaction
var dbTransaction2 = db.transaction(["422os"],"readonly");
```

- then use our new transaction get the required object store,

```
// define data object store
var dataStore2 = dbTransaction.objectStore("422os");
```

- then request our value from the database,

```
// request value - key &c.
var object1 = dataStore2.get(key);
```

- then use returned value for rendering...

Cordova app - IndexedDB - data test 2

database - part 17 - read data

- update our HTML with a button to load and test our data from IndexedDB,

```
...
<input type="button" id="loadNote" data-icon="refresh" value="Load Note" data-inline="true"/>
...
```

- add our event handler for the button
  - *allows us to call the `loadNoteData()` function for querying the IndexedDB*

```
// handler for load note button
$("#loadNote").on("tap", function(e) {
  e.preventDefault();
  // get requested data for specified key
  loadNoteData(1);
});
```

Cordova app - IndexedDB - data test 2

database - part 18 - read data

- need to add our new function to load the data from the object store

```
function loadNoteData(key) {
 var dbTransaction = db.transaction(["422os"],"readonly");
 // define data object store
 var dataStore2 = dbTransaction.objectStore("422os");
 // request value - use defined key
 var object1 = dataStore2.get(key);
 // do something with return
 object1.onsuccess = function(e) {
   var result = e.target.result;
   //output to console for testing
   console.dir(result);
   console.log("found value...");
 }
}
```

- use transaction to create connection to specified object store in IndexedDB
- able to request a defined value using a specified key
  - *in this example `key 1` for the object store `422os`*
- process return value for use in application

Image - IndexedDB Support

Cordova app - IndexedDB - data test 2

database - part 19 - read more data

- retrieving a single, specific value for a given key is obviously useful
  - *may become limited in practical application usage*
- IndexedDB provides an option to retrieve multiple data values
- uses an option called a `cursor`
  - *helps us iterate through specified data within our IndexedDB*
- use these cursors to create iterators with optional filters
  - *using range within a specified dataset*
  - *also add a required direction*
- creating and working with a cursor requires
  - *a transaction*
  - *performs an asynchronous request*

Cordova app - IndexedDB - data test 2

database - part 19 - read more data

- create our transaction,

```
var dbTransaction = db.transaction(["422os"],"readonly");
```

- retrieve our object store containing the required data

```
// define data object store
var dataStore3 = dbTransaction.objectStore("422os");
```

- now create our `cursor` for use with the required object store,

```
var 402cursor = dataStore3.openCursor();
```

- with this connection to the required object store in our specified IndexedDB
  - *now process the return values for our request*

Cordova app - IndexedDB - data test 2

database - part 20 - read more data

- use cursor to iterate through return results
  - *work with specified object store within our standard `success` handler*

```
cursor.onsuccess = function(e) {
  var result = e.target.result;
  if (result) {
    console.dir("notes", result.value);
    console.log("notes", result.key);
    result.continue();
 }
}
```

- new success handler is working with a passed object for the result from our IndexedDB
- object, `402result`, contains
  - *required keys, data, and a method to iterate through the returned data*
- `continue()` method is the iterator for this cursor
  - *allows us to iterate through our specified object store*

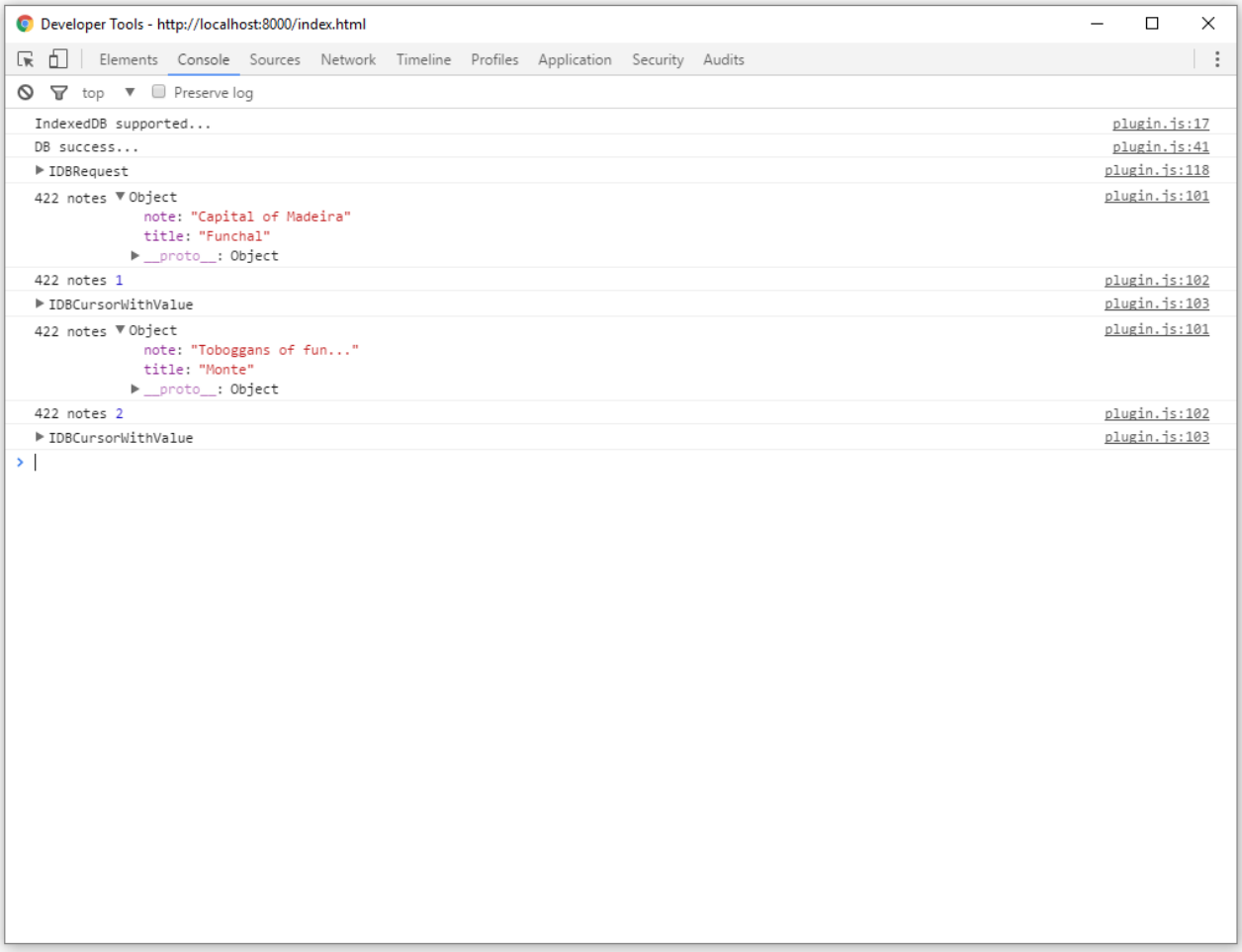
Cordova app - IndexedDB - data test 2

database - part 21 - read more data

- add an option to view all of the notes within our IndexedDB
- using the following new function, `loadNotes()`

```
function loadNotes() {
  // create transaction
  var dbTransaction = db.transaction(["422os"],"readonly");
  // define data object store
  var dataStore3 = dbTransaction.objectStore("422os");
  var cursor = dataStore3.openCursor();
  // do something with return...
  cursor.onsuccess = function(e) {
    var result = e.target.result;
    if (result) {
      console.log("422 notes", result.value);
      console.log("422 notes", result.key);
      console.dir(result);
      result.continue();
    }
  }
}
```

Image - IndexedDB Support



DataTest2 - test IndexedDB - read more data

Cordova app - IndexedDB - data test 2

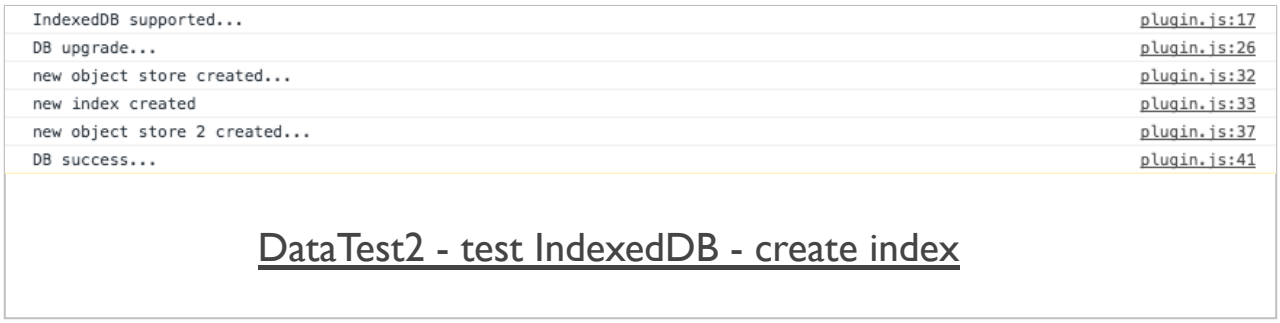
database - part 22 - index

- a primary benefit of using IndexedDB
  - *its support for indexes*
  - *retrieve data from these object stores using the data value itself*
  - *in addition to the standard key search*
- start by adding this option to our object stores
- create an index by using our pattern for an `upgrade` event
  - *creating the index at the same time as the object store*

```
var dataStore = db.createObjectStore("422os", { autoIncrement:true});
// set name of index
dataStore.createIndex("note","note", {unique:false});
```

- creating our object store, `422os`
  - *then using object store result to create and index using `createIndex()`*
  - *first argument for this method is the name for our index*
  - *second is the actual property we want indexing within the object store*
  - *add a set of options, eg: unique or not*
- IndexedDB will then create an index for this object store

Image - IndexedDB Support



Cordova app - IndexedDB - data test 2

database - part 22 - index

- new index now created
  - *start to add options for querying the database's values*
- need to specify a required index from the applicable object store
- use a transaction to retrieve a given object store
  - *then able to specify required index from that object store*

```
// create transaction
var dbTransaction = db.transaction(["422os"],"readonly");
// define data object store
var dataStore = dbTransaction.objectStore("422os");
// define index
var dataIndex = dataStore.index("note");
```

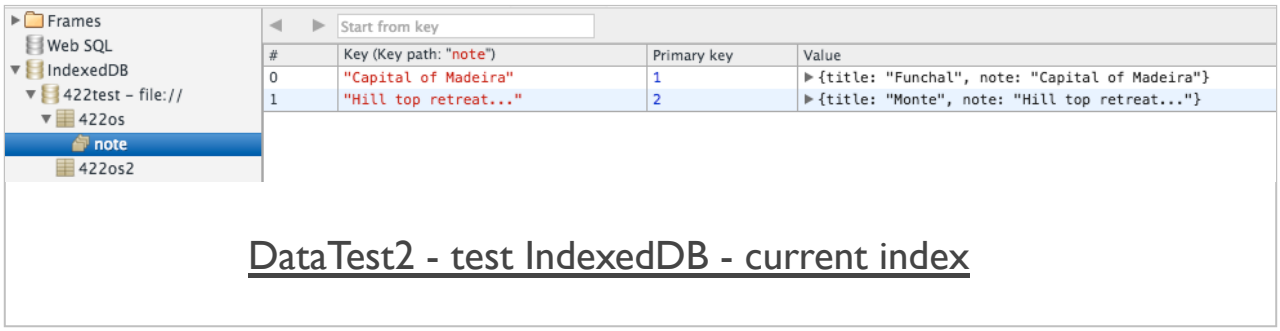
- we can then request some values using a standard `get` method with this index

```
var note = "Capital of Madeira";
var getRequest = dataIndex.get(note);
```

Image - IndexedDB Support

DataTest2 - test IndexedDB - query index

Image - IndexedDB Support



DataTest2 - test IndexedDB - current index

Cordova app - IndexedDB - data test 2

database - part 23 - index

- we will need to consider queries against an index in much broader terms
- we need to consider the use and application of ranges relative to our index
- use of ranges returns a limited set of data from our object store
- IndexedDB helps us create few different options for ranges
  - ***everything above...***, ***everything below...***, ***something between...***, ***exact***
  - *set ranges either inclusive or exclusive*
  - *request ascending and descending ranges for our results*
- an example range might be limiting a query to a specific word, title, or other key value...

```
// Only match "Madeira"
var singleRange = IDBKeyRange.only("Madeira");
```

- by default, IndexedDB supports the following types of queries
  - `IDBKeyRange.only()` - *Exact match*
  - `IDBKeyRange.upperBound()` – *objects = property below certain value*
  - `IDBKeyRange.lowerBound()` – *objects = property above certain value*
  - `IDBKeyRange.bound()` – *objects = property between certain values*

Data considerations in mobile apps

- worked our way through Cordova's File plugin
  - *tested local read and write for files*
- test JS requests with JSON
  - *local and remote files*
  - *remote services and APIs*
- work natively with JS objects
  - *webview*
  - *controller*
  - *local or remote data store or service*

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 1

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 1

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 1

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 1

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 1

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);
  });
}
```

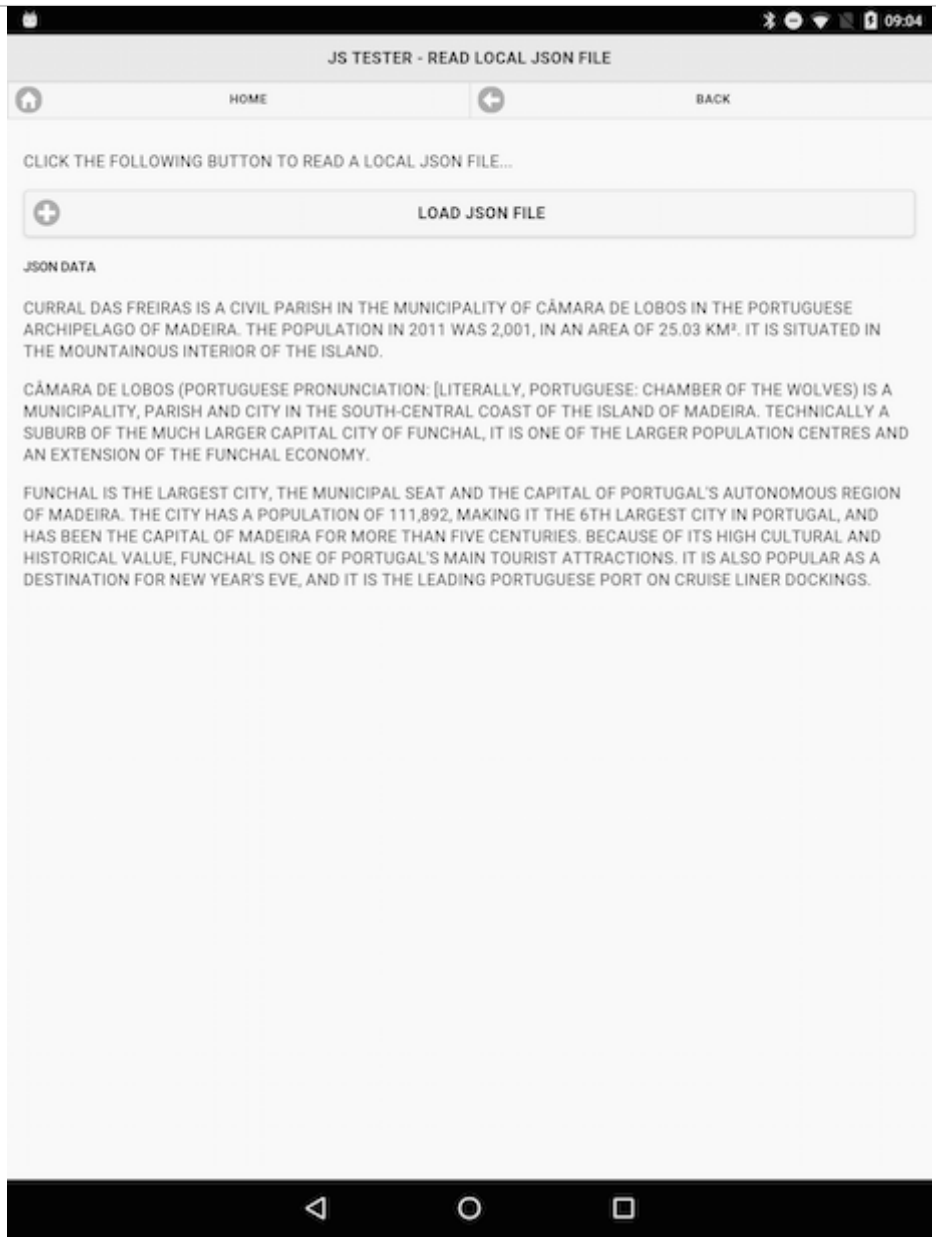
Cordova app - JS data options - JS data test 1

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

References

- Cordova API
  - *Storage*
  - *Whitelist plugin*
- GitHub
  - *cordova-plugin-indexeddb*
- MDN
  - *IndexedDB*
- W3
  - *Web storage specification*