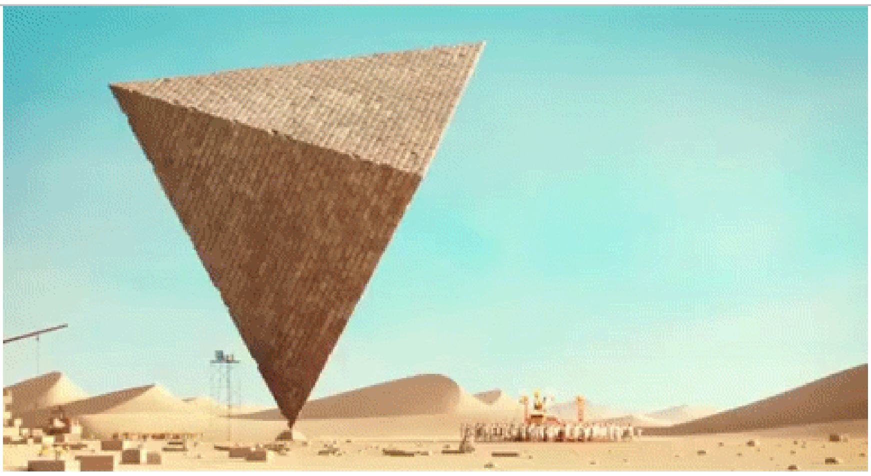
Comp 322/422 - Software Development for Wireless and Mobile Devices

Fall Semester 2019 - Week 4

Dr Nick Hayward

Image - Designing our app



Designing our app - fundamentals are important

Video - Pyramid builders

Minions Pyramid Builders - Source: YouTube

Extra notes - mobile 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.

Mobile Design & Development - Data Usage and Persistency

Fun Exercise

Four apps, one per group

- Books - http://linode4.cs.luc.edu/teaching/cs/demos/422/videos/week4/books/
- Cinema - http://linode4.cs.luc.edu/teaching/cs/demos/422/videos/week4/cinema/
- Plants - http://linode4.cs.luc.edu/teaching/cs/demos/422/videos/week4/plants/
- Travel - http://linode4.cs.luc.edu/teaching/cs/demos/422/videos/week4/travel/

For your assigned app, consider the following

- UI and UX in the app that requires data loading
  - *local or remote*
  - *how to update this data?*
- required data persistency in the app
  - *local or remote*
  - *temporary or long-term*
  - *account or session*

~ 10 minutes

Cordova app - API plugin examples - plugin test 2

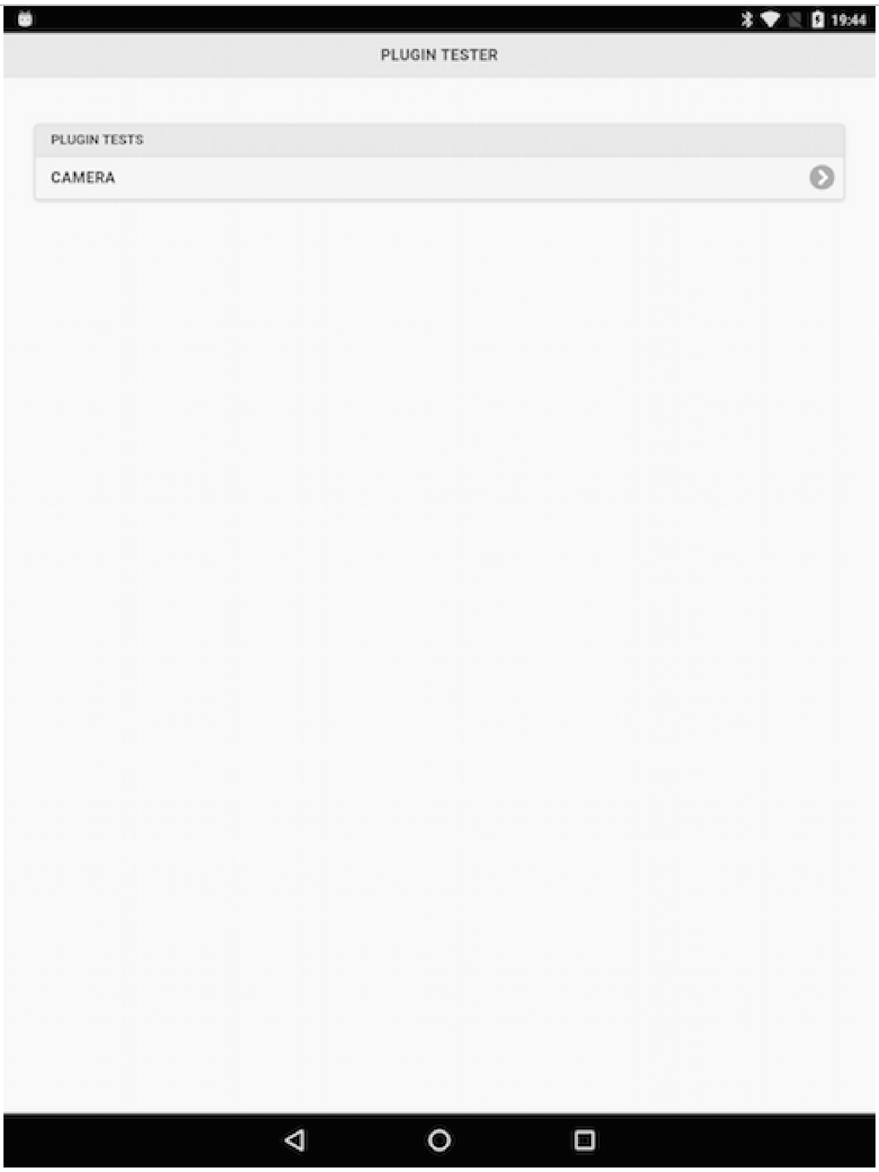
plugins - add camera plugin

- 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
```

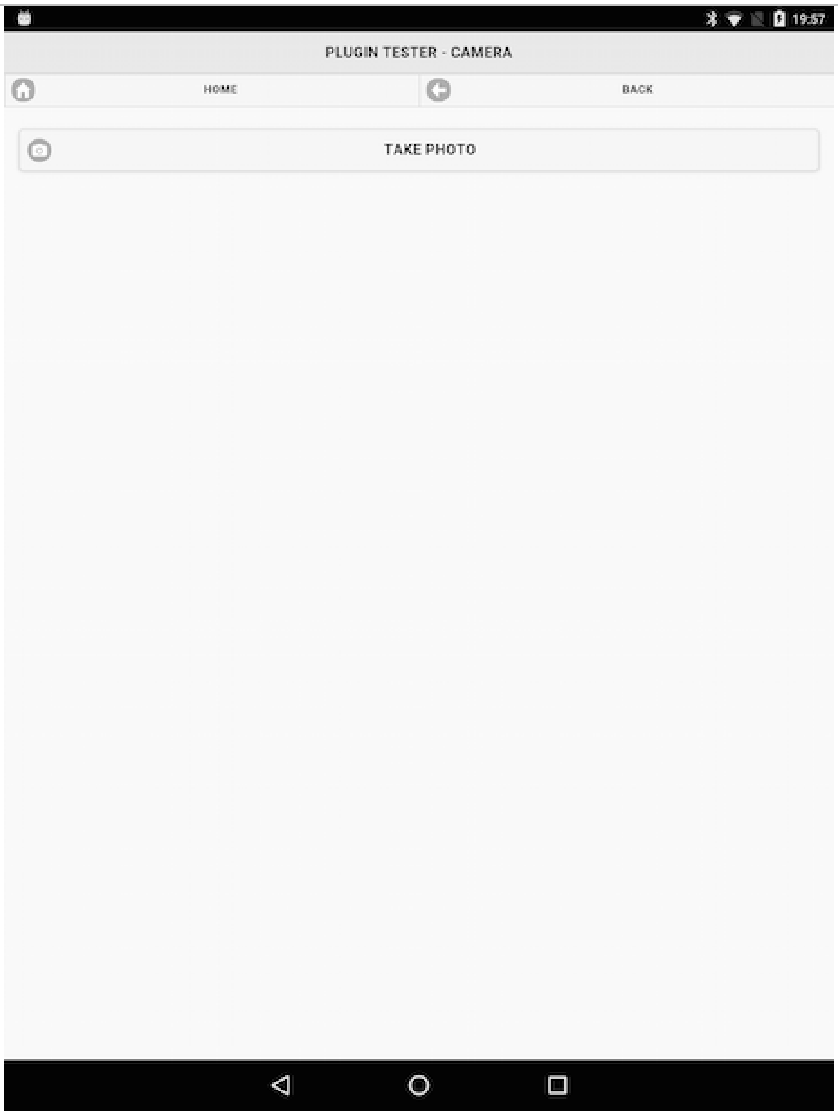
- provides standard `navigator` object
  - *enables taking pictures, and choose images from local image library*

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

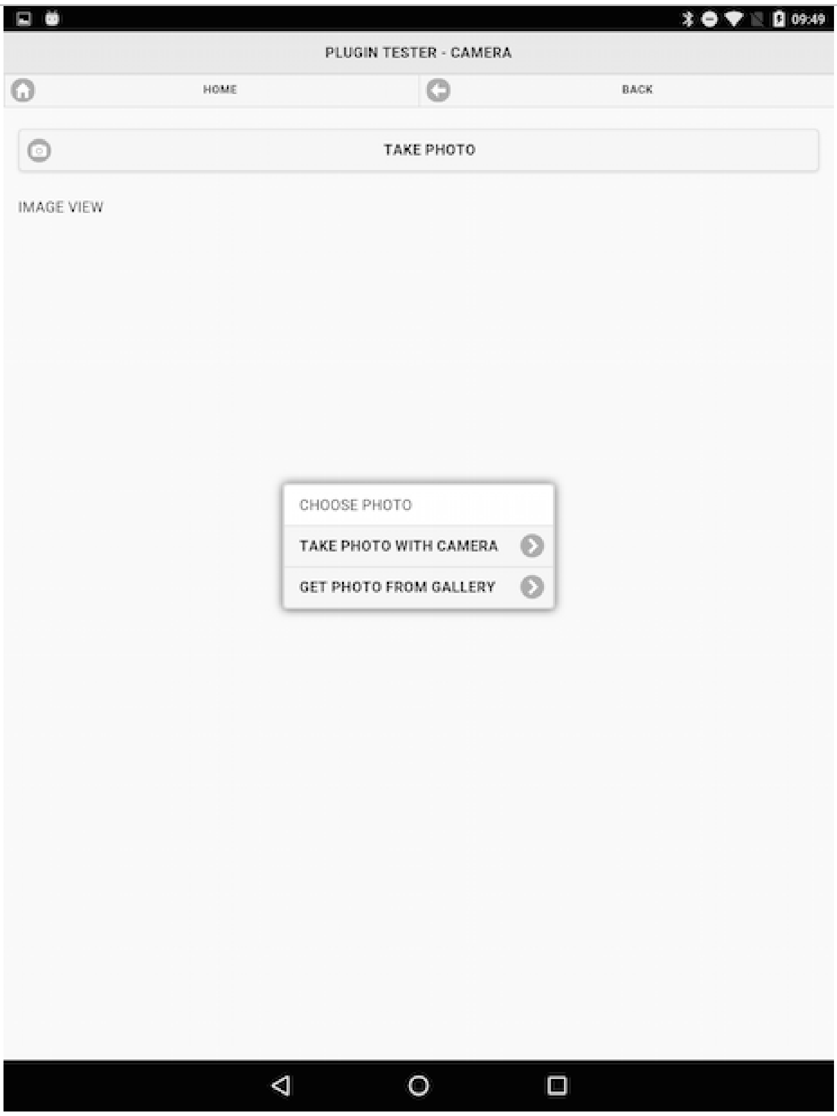
- add our handlers for the tap events
- initial handlers for `takePhoto`, `cameraPhoto`, and `galleryPhoto`

e.g.

```
let shutter = document.getElementById('takePhoto');
playButton.addEventListener('touchstart', takePhoto, false);

function takePhoto() {
  // show modal for camera options...
  // different call relative to chosen UI option...
}
```

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

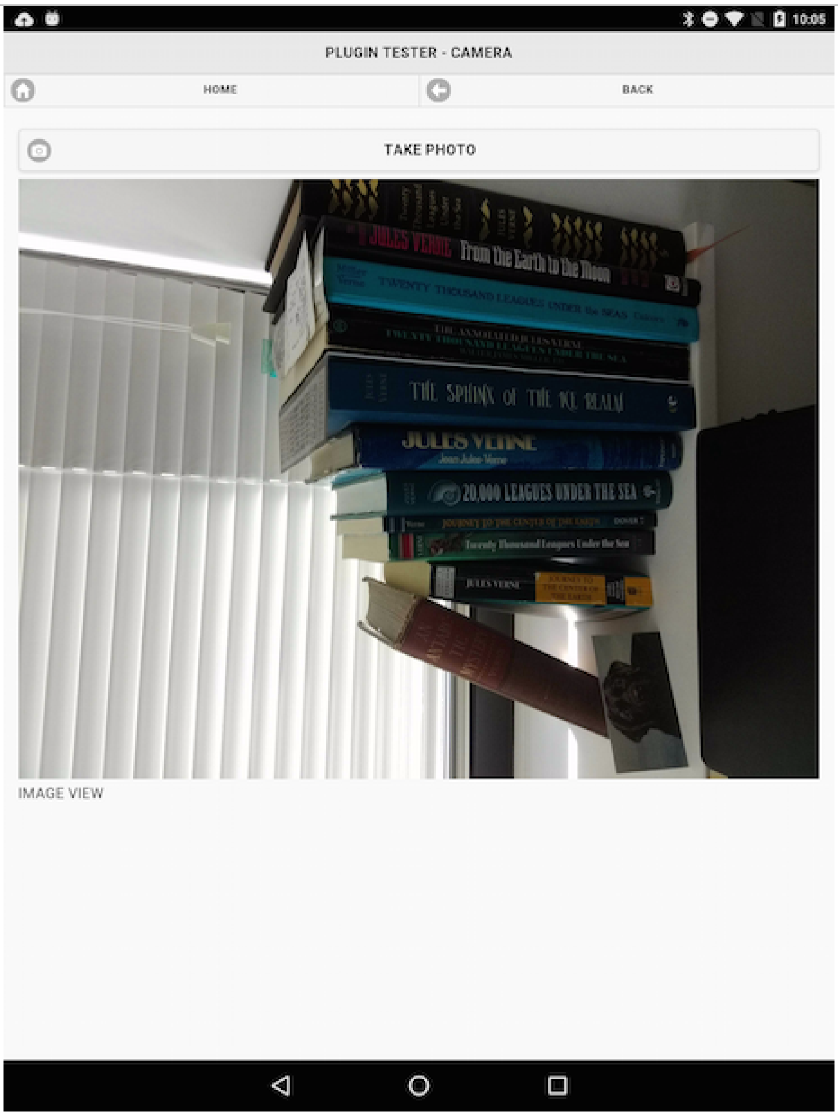
- two callback functions are `onSuccess` and `onFail`
  - *set logic for returned camera image and any error message*

```
function onSuccess(imageData) {
  //JS selector...
  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

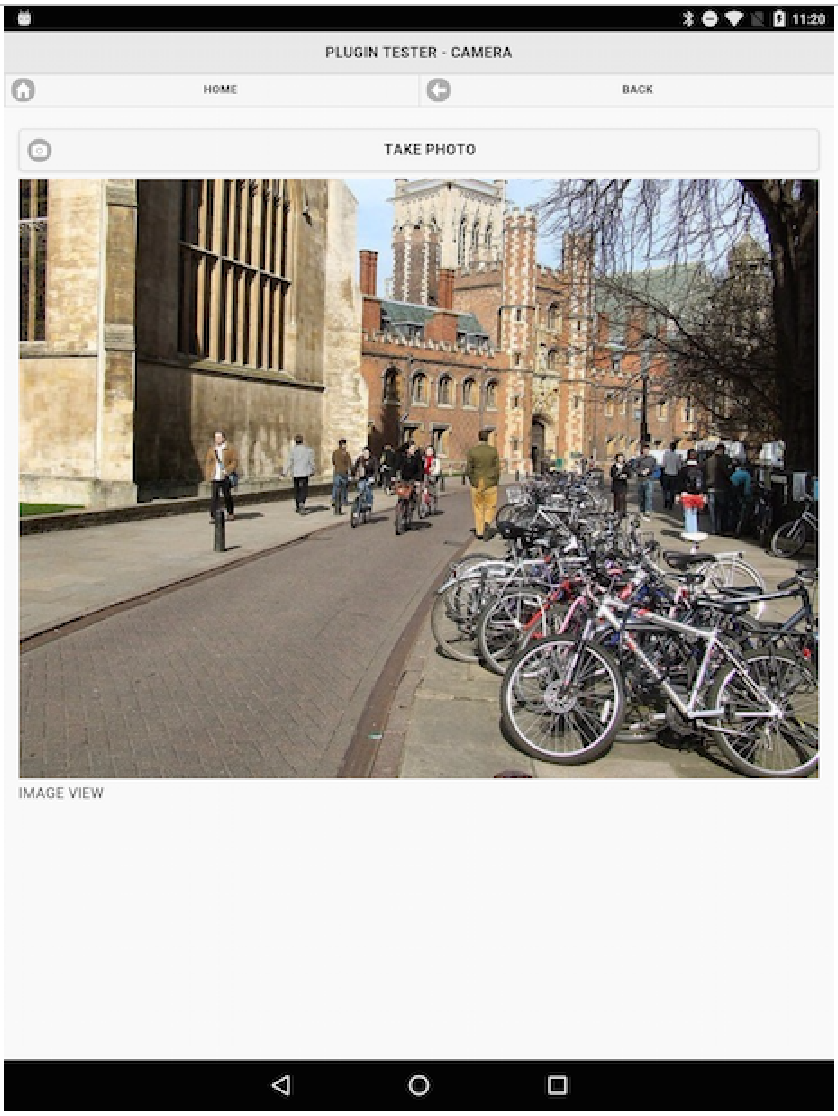
plugins - update camera logic

- 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

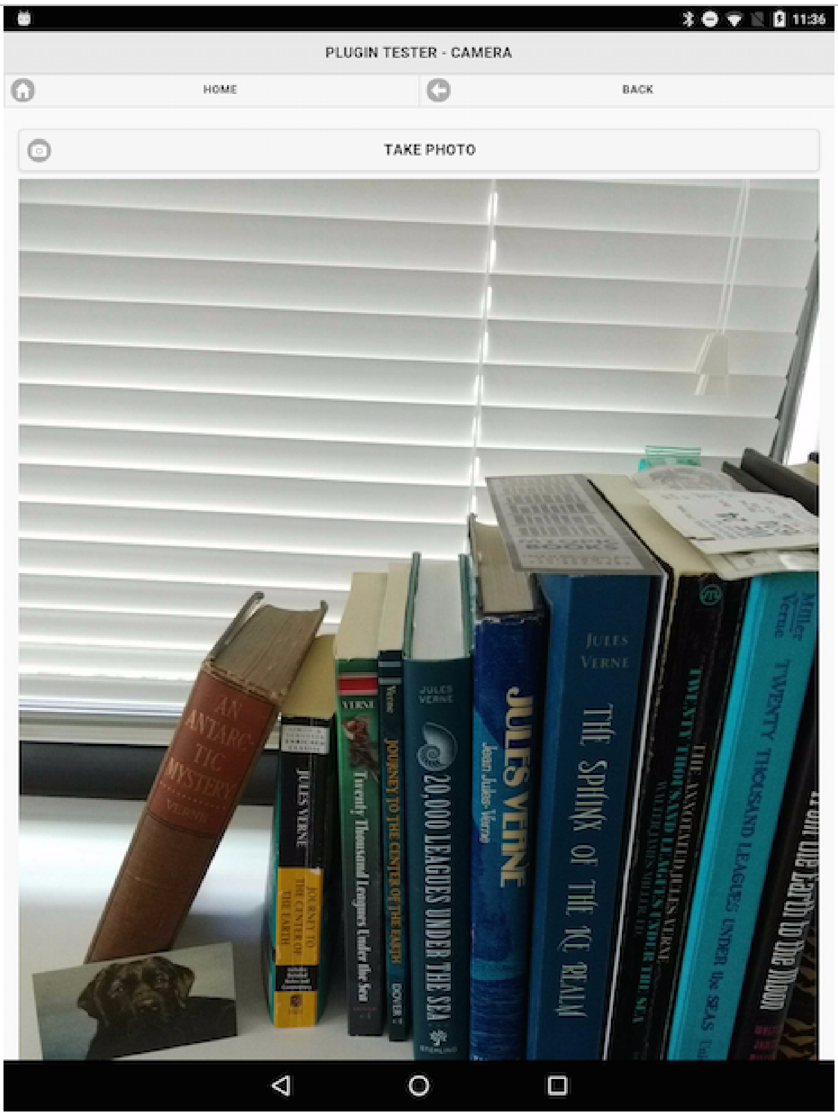
- need to fix the orientation issue with the returned image from the camera
- options for this plugin make it simple to update our logic for this requirement
  - *add a new option for the camera*

```
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...*

Data considerations in mobile apps

- worked our way through Cordova's File plugin
- tested local and remote requests with JSON
- initial considerations for working with LocalStorage
- many other options for data storage in mobile applications
  - *IndexedDB*
  - *hosted NoSQL options, such as Redis and MongoDB*
  - *Firebase*
  - *query hosted remote SQL databases*
  - *and so on...*

Cordova app - IndexedDB

intro

- browser storage wars of recent years
  - *IndexedDB was crowned the winner over WebSQL*
- what do we gain with IndexedDB?
  - *useful option for developers to store relatively large amounts of client-side data*
  - *effectively stores data within the user's webview/browser*
  - *useful storage option for network apps*
  - *a powerful, and particularly useful, indexed based search API*
- IndexedDB differs from other local browser-based storage options
- localStorage is generally well supported
  - *limited in terms of the total amount of storage*
  - *no native search API*
- different solutions for different problems
  - *no universal best fit for storage...*
- browser support for mobile and desktop
  - *Can I use___?*
- Cordova plugin to help with IndexedDB support
  - *MSOpenTech - cordova-plugin-indexeddb*

Cordova app - IndexedDB - data test 2

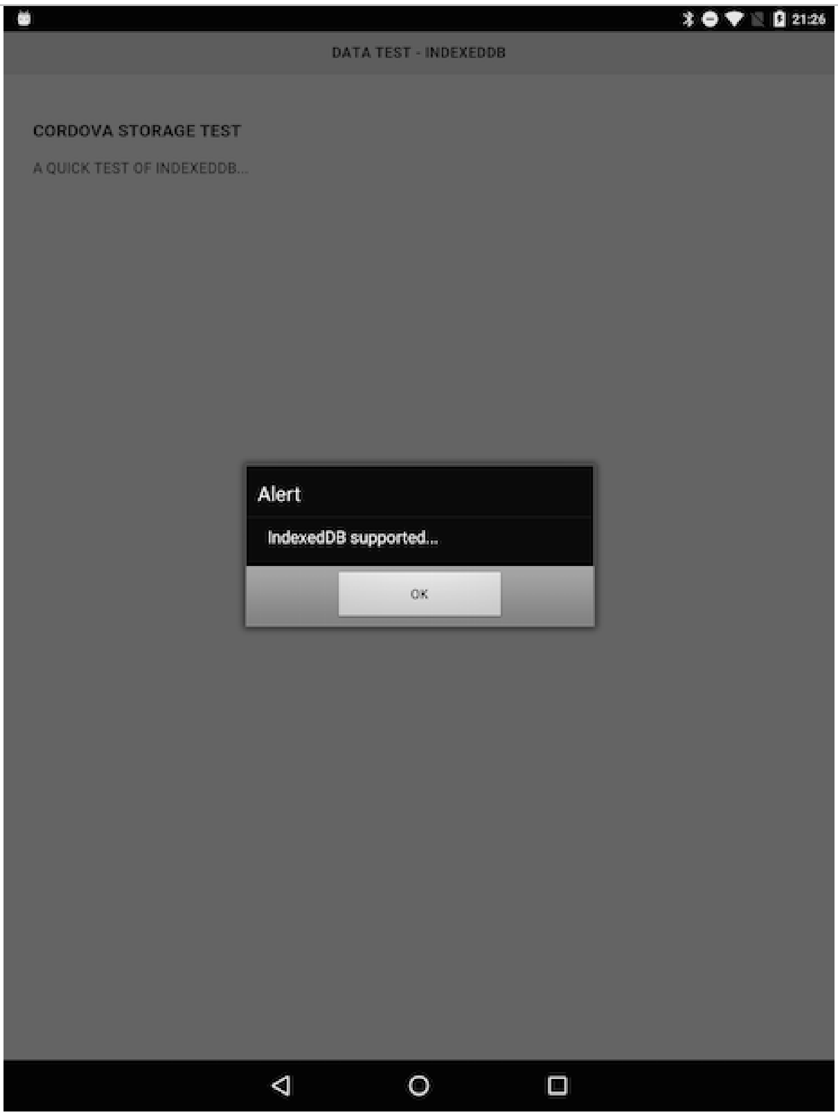
setup and test - part 1

- testing our IndexedDB example with Cordova and Android
- perform our standard test for the `deviceready` event
  - *going to add a check for IndexedDB support and usage*
- in `onDeviceReady()` function
  - *add a quick check for IndexedDB support in the application's webview*

```
if("indexedDB" in window) {
  console.log("IndexedDB supported...");
} else {
  console.log("No support...");
}
```

- Android support is available...

Image - IndexedDB Support



CORDOVA STORAGE TEST

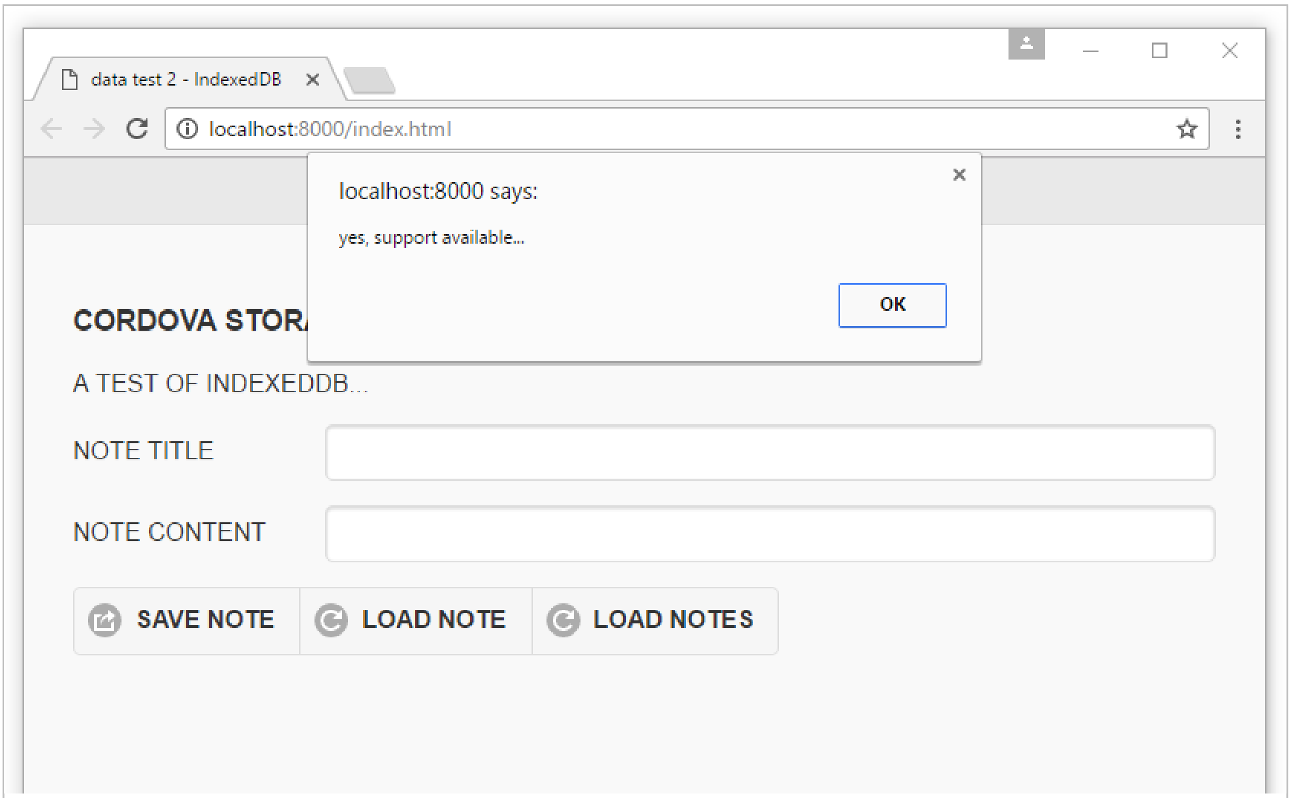
A QUICK TEST OF INDEXEDDB...

Alert

IndexedDB supported...

DataTest2 - test IndexedDB support in webview

Image - IndexedDB Support



data test 2 - IndexedDB

localhost:8000/index.html

localhost:8000 says:

yes, support available...

OK

CORDOVA STOR

A TEST OF INDEXEDDB...

NOTE TITLE

NOTE CONTENT

SAVE NOTE | LOAD NOTE | LOAD NOTES

DataTest2 - test IndexedDB support in chrome

Cordova app - IndexedDB - data test 2

setup and test - part 2

- update this check to ensure we have a quick reference later

```
//set variable for IndexedDB support
var indexedDBSupport = false;
//check IndexedDB support
if("indexedDB" in window) {
  indexedDBSupport = true;
  console.log("IndexedDB supported...");
} else {
  console.log("No support...");
}
```

- create initial variable to store the boolean result
- check variable after `deviceready` event has fired and returned successfully

Cordova app - IndexedDB - data test 2

database - part 1 - getting started

- start to build our IndexedDB database
- database is local to the browser,
  - *only available to users of the local, native app*
- IndexedDB databases follow familiar pattern of read and write privileges
  - *eg: browser-based storage options, including localstorage*
- create databases with the same name, and then deploy them to different apps
  - *remain domain specific as well*
- first thing we need to do is create an opening to our database

```
var openDB = indexedDB.open("422test", 1);
```

- creating a variable for our database connection
  - *specifying the name of the DB and a version*
- open request to the DB is an asynchronous operation

Cordova app - IndexedDB - data test 2

database - part 2 - getting started

- open request to the DB is an asynchronous operation
  - *add some useful event listeners to help with our application*
  - *`success`, `error`, `upgradeneeded`, `blocked*
- `upgradeneeded`
  - *event will fire when the DB is first opened within our application*
  - *also if and when we update the version number for the DB*
- `blocked`
  - *fires when a previous or defunct connection to the DB has not been closed*

Cordova app - IndexedDB - data test 2

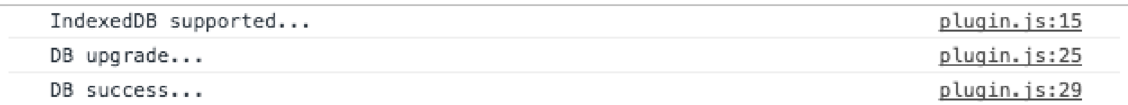
database - part 3 - create

- test creating a new DB
  - *then checking persistence during application loading and usage*

```
if(indexedDBSupport) {
  var openDB = indexedDB.open("422test",1);
  openDB.onupgradeneeded = function(e) {
      console.log("DB upgrade...");
  }
  openDB.onsuccess = function(e) {
      console.log("DB success...");
      db = e.target.result;
  }
  openDB.onerror = function(e) {
      console.log("DB error...");
      console.dir(e);
  }
}
```

- `console.log()` - outputs a string representation
- `console.dir()` - prints a navigable tree

Image - IndexedDB Support



```
IndexedDB supported...                                          plugin.js:15
DB upgrade...                                                   plugin.js:25
DB success...                                                   plugin.js:29
```

DataTest2 - test IndexedDB open - first app load

Cordova app - IndexedDB - data test 2

database - part 4 - success

- performed a check to ensure that IndexedDB is supported
  - *if yes, open a connection to the DB*
  - *also added checks for three events, including `upgrade`, `onsuccess`, and `errors`*

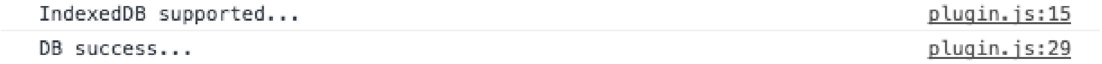
- now ready to test the `success` event
  - *event is passed a handler via `target.result`*

```
...
openDB.onsuccess = function(e) {
    console.log("DB success...");
    db = e.target.result;
}
...
```

- handler is being stored in our global variable db

- run this test and check log output
  - *outputs initial connection and upgrade status*
  - *then the success output for subsequent loading of the application*

Image - IndexedDB Support



```
IndexedDB supported...                                          plugin.js:15
DB success...                                                   plugin.js:29
```

DataTest2 - test IndexedDB open - after first app load

Cordova app - IndexedDB - data test 2

database - part 5 - data stores

- now start building our data stores in IndexedDB
- IndexedDB has a general concept for storing data
  - *known as **Object Stores***
  - *conceptually at least, known as (very) loose database tables
- within our object stores
  - *add some data, plus a **keypath**, and an optional set of indices (indexes)*
- a **keypath** is a unique identifier for the data
- Indices help us index and retrieve the data
- object stores created during `upgradeneeded` event for the current version
  - *created when the app first loads*
  - *create object stores as part of this `upgradeneeded` event*
- if we want to upgrade our object stores
  - *update version*
  - *upgrade the object store using the `upgradeneeded` event*

Cordova app - IndexedDB - data test 2

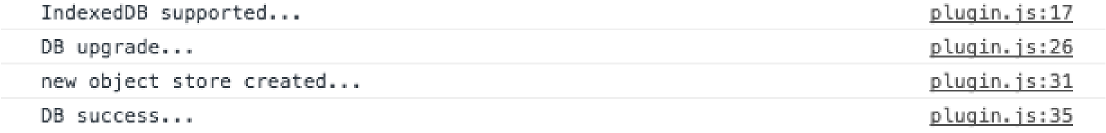
database - part 6 - data stores

- update our upgrade event to include the creation of our required object stores

```
...
openDB.onupgradeneeded = function(e) {
    console.log("DB upgrade...");
    //local var for db upgrade
    var upgradeDB = e.target.result;
    if (!upgradeDB.objectStoreNames.contains("422os")) {
      upgradeDB.createObjectStore("422os");
      console.log("new object store created...");
    }
}
...
```

- check a list of existing object stores
  - *list of existing object stores available in the property `objectStoreNames`*
- check this property for our required object store using the `contains` method
- if required object store unavailable we can create our new object store
  - *listen for result from this synchronous method*
- as a user opens our app for the first time
  - *the `upgradeneeded` event is run*
  - *code checks for an existing object store*
  - *if unavailable, create a new one*
  - *then run the `success` handler*

Image - IndexedDB Support



```
IndexedDB supported...                                      plugin.js:17
DB upgrade...                                               plugin.js:26
new object store created...                                 plugin.js:31
DB success...                                               plugin.js:35
```

DataTest2 - test IndexedDB - create object store

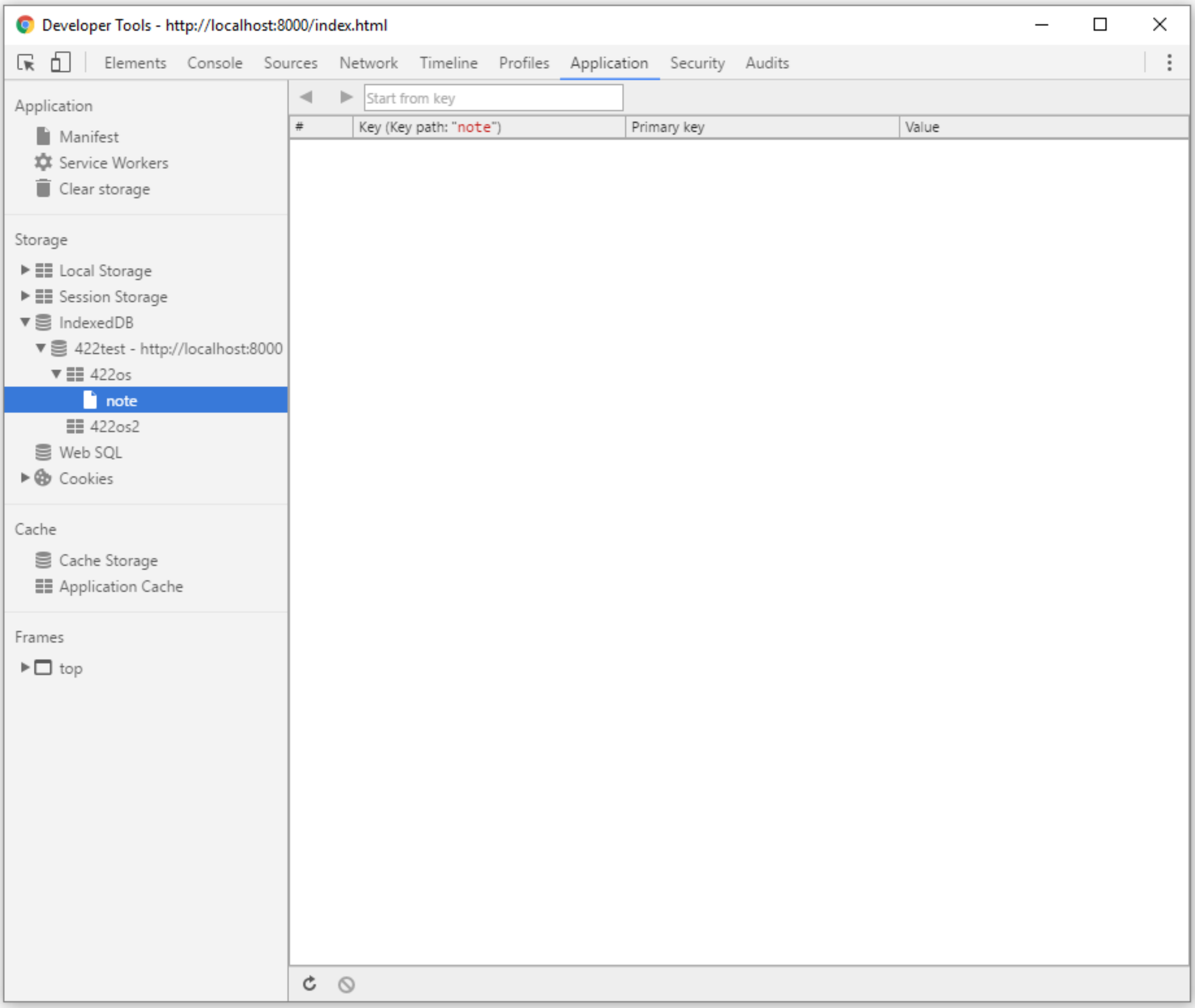
Cordova app - IndexedDB - data test 2

database - part 7 - extra data stores

- start to add further object stores
- can't simply create a new object store due to the `upgradeneeded` event
- increment the version number for the current database
  - *thereby invoking the `upgradeneeded` event*
- reate our new object store using the same pattern

```
var openDB = indexedDB.open("422test",2);
openDB.onupgradeneeded = function(e) {
  console.log("DB upgrade...");
  //local var for db upgrade
  var upgradeDB = e.target.result;
  if (!upgradeDB.objectStoreNames.contains("422os")) {
    upgradeDB.createObjectStore("422os");
    console.log("new object store created...");
  }
  if (!upgradeDB.objectStoreNames.contains("422os2")) {
    upgradeDB.createObjectStore("422os2");
    console.log("new object store 2 created...");
  }
}
```

Image - IndexedDB Support



DataTest2 - test IndexedDB - initial object stores

Cordova app - IndexedDB - data test 2

database - part 8 - add data

- our database currently has two object stores
  - *now start adding some data for our application*
- IndexedDB allows us to simply store our objects in their default structure
  - *simply store JavaScript objects directly in our IndexedDB database*
- use transactions when working with data and IndexedDB
- transactions help us create a bridge between our app and the current database
  - *allowing us to add our data to the specified object store*
- a transaction includes two arguments
  - *first for the object store*
  - *second is the type of transaction*
  - *choose either `readonly` or `readwrite`*

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

Cordova app - IndexedDB - data test 2

database - part 9 - add data

- use transaction to retrieve object store for our data
  - *requesting the `422os` in this example*

```
var dataStore = dbTransaction.objectStore("422os");
```

- add some data using the new `datastore`

```
// note
var note = {
  title:title,
  note:note
}
// add note
var addRequest = dataStore.add(note,key);
```

- for each object we can define the underlying naming schema
  - *best fit our applications*
- then add our object, with an associated key, to our `dataStore`

Cordova app - IndexedDB - data test 2

database - part 10 - add data

- now added an object to our object store
- request is asynchronous
  - *attach additional handlers for returned result*
  - *add a success and error handler*

```
// success handler
addRequest.onsuccess = function(e) {
  console.log("data stored...");
  // do something...
}
// error handler
addRequest.onerror = function(e) {
console.log(e.target.error.name);
// handle error...
}
```

Cordova app - IndexedDB - data test 2

database - part 11 - add data

- add a form for the note content and title
- set a save button to add the note date to the IndexedDB

```
<form id="noteForm">
  <div class="ui-field-contain">
    <label for="noteName">Note Title</label>
    <input type="text" id="noteName" name="noteName"></input>
  </div>
  <div class="ui-field-contain">
    <label for="noteContent">Note Content</label>
    <input type="text" id="noteContent" name="noteContent"></input>
  </div>
  <div data-role="controlgroup" data-type="horizontal">
    <input type="button" id="saveNote" data-icon="action" value="Save Note" data-inline="t
  </div>
</form>
```

- bind event handler to save button for click
  - *submit add request to IndexedDB*
  - *store object data*

Cordova app - IndexedDB - data test 2

database - part 12 - add data handlers

- now add our event handler for the save button
- handler gets note input from note form
- passes the data to the `saveNote()` function

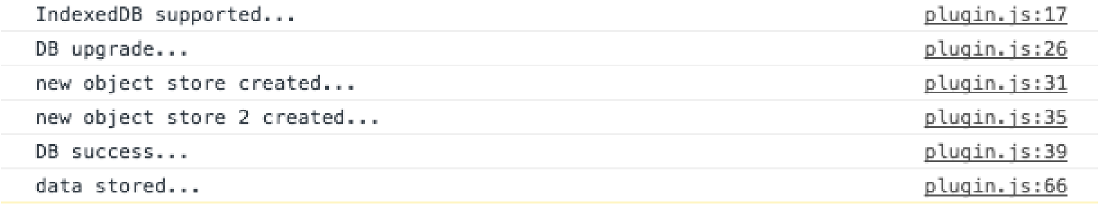
```
// handler for save button
$("#saveNote").on("tap", function(e) {
  e.preventDefault();
  var noteTitle = $("#noteName").val();
  var noteContent = $("#noteContent").val();
  saveNote(noteTitle, noteContent);
});
```

Cordova app - IndexedDB - data test 2

database - part 13 - add data handlers

```
//save note data to indexeddb
function saveNote(title, content){
  //define a note
  var note = {
    title:title,
    note:content
  }
  // create transaction
  var dbTransaction = db.transaction(["422os"],"readwrite");
  // define data object store
  var dataStore = dbTransaction.objectStore("422os");
  // add data to store
  var addRequest = dataStore.add(note,1);
  // success handler
  addRequest.onsuccess = function(e) {
    console.log("data stored...");
    // do something...
  }
  // error handler
  addRequest.onerror = function(e) {
  console.log(e.target.error.name);
  // handle error...
  }
}
```

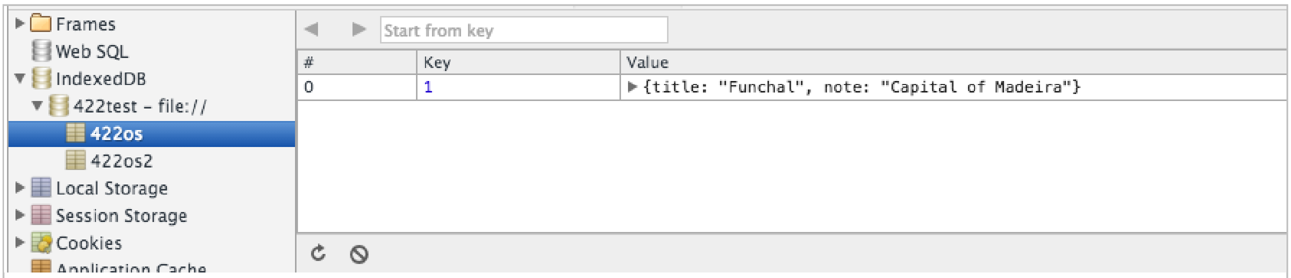
Image - IndexedDB Support



```
IndexedDB supported...                                                    plugin.js:17
DB upgrade...                                                             plugin.js:26
new object store created...                                               plugin.js:31
new object store 2 created...                                             plugin.js:35
DB success...                                                             plugin.js:39
data stored...                                                            plugin.js:66
```

DataTest2 - test IndexedDB - save data to store

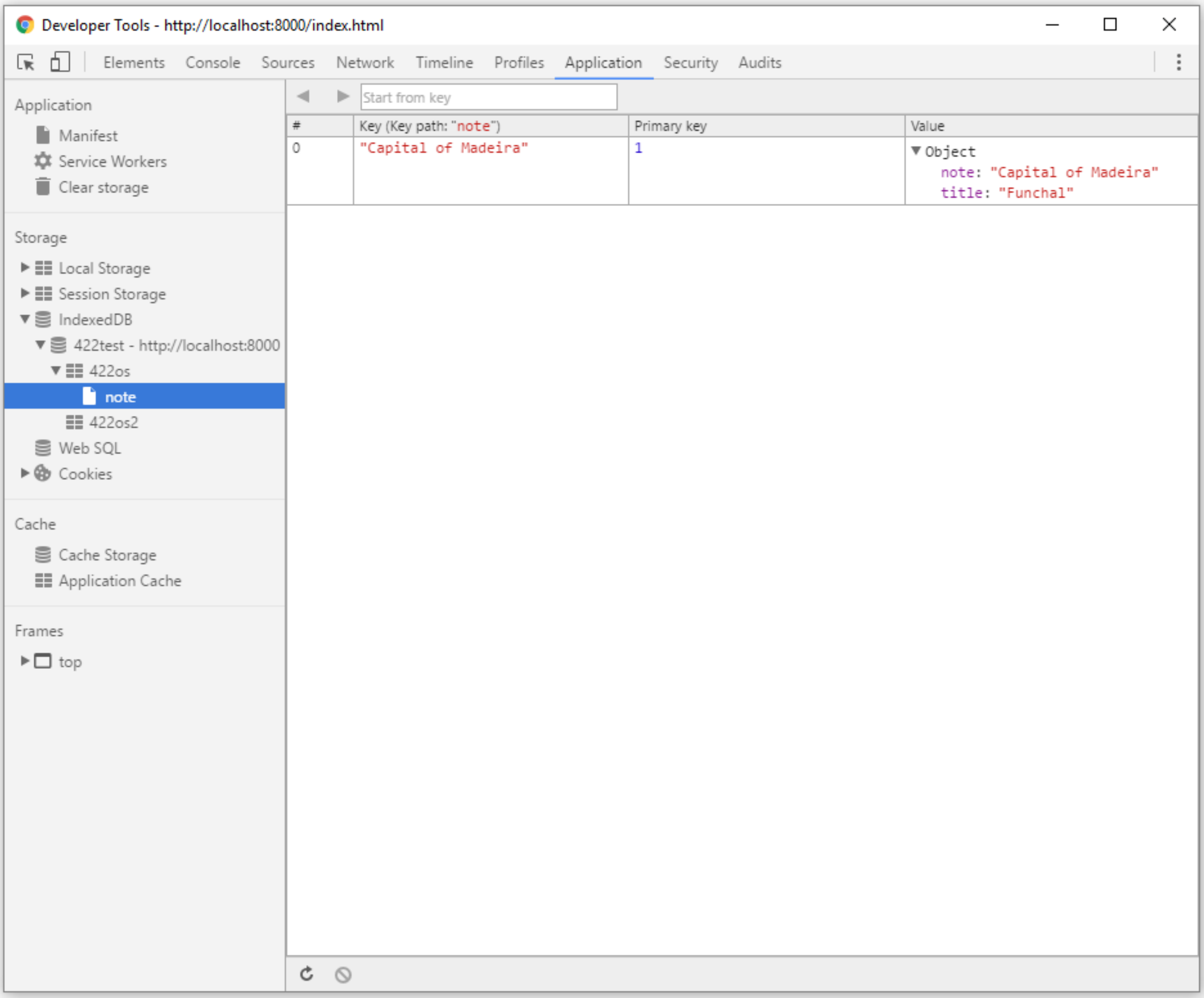
Image - IndexedDB Support



#	Key	Value
0	1	▶ {title: "Funchal", note: "Capital of Madeira"}

DataTest2 - test IndexedDB - save data to store 2

Image - IndexedDB Support



#	Key (Key path: "note")	Primary key	Value
0	"Capital of Madeira"	1	▼Object note: "Capital of Madeira" title: "Funchal"

DataTest2 - test IndexedDB - save data to store 3

Cordova app - IndexedDB - data test 2

database - part 14 - multiple notes

- now created our IndexedDB
- created the object store
- setup the app's HTML and form
- and saved some data to the database...
- update our application to allow a user to add multiple notes to the database
- currently setting our key for a note in the `saveNote()` function
  - *add another note, we get a constraint error output to the console*
  - *we're trying to add a note to an existing key in the database*
- need to update our logic for the app
  - *to allow us to work more effectively with **keys***

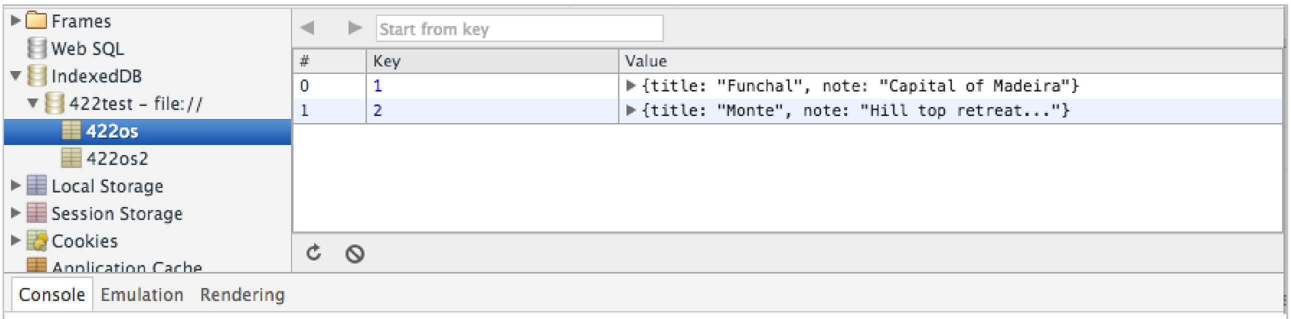
Cordova app - IndexedDB - data test 2

database - part 15 - keys

- keys in IndexedDB often considered similar to primary keys in SQL...
    - *a unique reference for our data objects*
- traditional databases can include tables without such keys
    - ***NB:*** *every object store in IndexedDB needs to have a* ***key***
    - *able to use different types of keys for such stores*
- first option for a key is simply to create and add a key ourselves
    - *could programatically create and update these keys*
    - *helps maintain unique ID for keys*
- could also provide a **keypath** for such keys
    - *often based on a given property of the passed data...*
    - *still need to ensure our key is unique*
- other option is to use a key generator within our code
    - *similar concept to SQL `auto-increment`*

```
db.createObjectStore("422os", { autoIncrement: true });
```

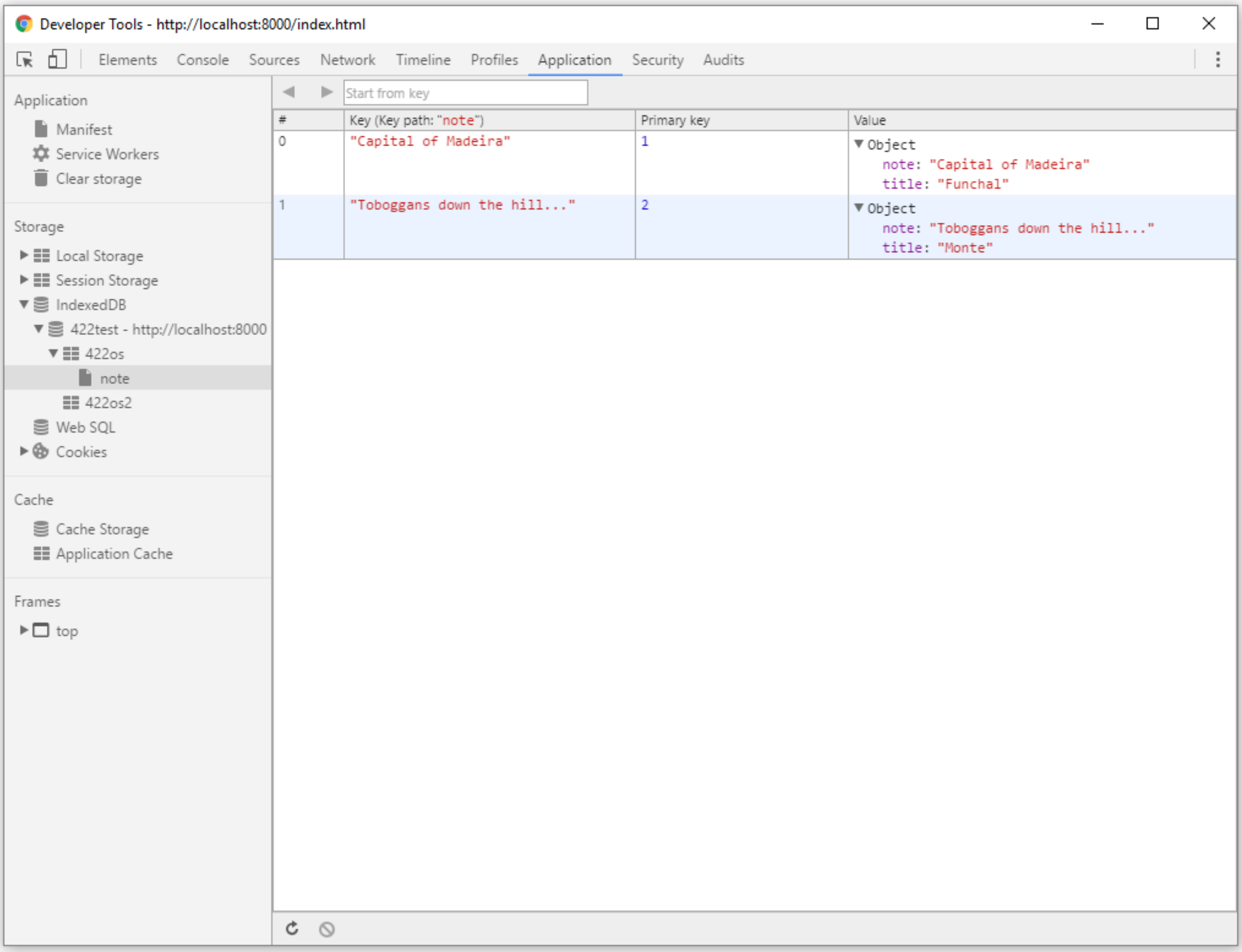
Image - IndexedDB Support



#	Key	Value
0	1	▶ {title: "Funchal", note: "Capital of Madeira"}
1	2	▶ {title: "Monte", note: "Hill top retreat..."}

DataTest2 - test IndexedDB - unique keys

Image - IndexedDB Support



#	Key (Key path: "note")	Primary key	Value
0	"Capital of Madeira"	1	▼ Object note: "Capital of Madeira" title: "Funchal"
1	"Toboggans down the hill..."	2	▼ Object note: "Toboggans down the hill..." title: "Monte"

DataTest2 - test IndexedDB - unique keys 2

References

- Aaron, Marcus. *Graphic Design for Electronic Documents and User Interfaces.* ACM Press. 1992.
- Cordova API
  - *plugin - camera*
- GitHub
  - *cordova-plugin-indexeddb*
- MDN
  - *IndexedDB*