Comp 422 - Software Development for Wireless and Mobile Devices

Fall Semester 2016 - Week 8

Dr Nick Hayward

Contents

Image - Designing our app

Designing our app - fundamentals are important

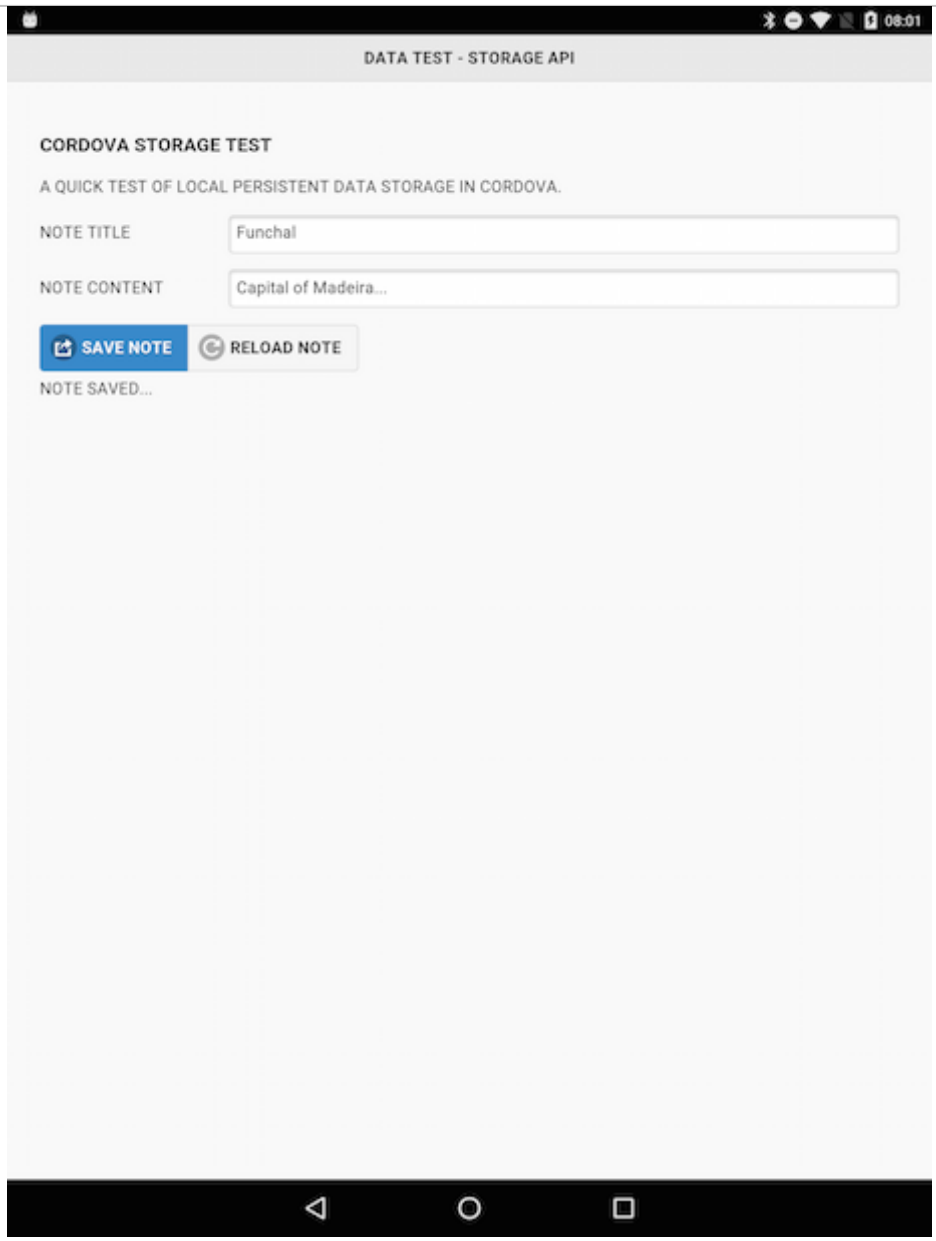
Video - Pyramid builders

Minions Pyramid Builders - Source: YouTube

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...*

Image - Data Tester



DataTest1 - save a note

Cordova app - LocalStorage - data test I

app logic - save.js

- need to handle events for our `reloadNote` button
- retrieve our notes data
  - *loaded by calling the `reloadNoteData()` function*
- uses the main app object, `storageNotes`
  - *gets the defined key for our notes*
- use this key to retrieve stored *stringified* JSON object
- then use `JSON.parse()` to convert the *stringified* object to a plain JSON object
  - *contains our note information*
- use this note information
  - *populate form fields*
  - *output our notes for rendering to the DOM*

Cordova app - LocalStorage - data test I

app logic - save.js - reload button handler

- event handler for reload button

```
// handler for reload note button
$("#reloadNote").on("tap", function(e) {
  e.preventDefault();
  reloadNoteData();
  $("#saveResult").html("note reloaded...");
});
```

- reload note data

```
function reloadNoteData() {
  var noteInfo = JSON.parse(storageNotes.get(NOTE_KEY));
  loadFormFields(noteInfo);
  noteOutput(noteInfo);
}
```

- load form fields data

```
function loadFormFields(data) {
  if (data) {
    $("#noteName").val(data.noteName);
    $("#noteContent").val(data.noteContent);
  }
}
```

Cordova app - LocalStorage - data test I

app logic - save.js

- pageinit event
  - *eg: check and validate the rendered form for our notes*
- to validate our form we specify
  - *a set of options as a parameter to* `validate()`
  - *many different options available*
  - *eg: add a* `rules` *object,* `messages` *object...*
- in the `rules` object
  - *set both input fields as required*
- then reload our note data
  - *update the application accordingly*

Cordova app - LocalStorage - data test 1

app logic - save.js - pageshow event

```
$("#noteForm").validate({
  rules: {
    noteName: "required",
    noteContent: "required"
  },
  messages: {
    noteName: "Add title for note",
    noteContent: "Add your note"
  }
});
```

Cordova app - LocalStorage - data test I

app logic - storagenotes.js

- add another new JS file, `storagenotes.js`
  - *store the logic for getting and setting of data with `localStorage`*
- start by creating a singleton object for this instance
- creating this object to ensure that we only have one instance
- create this object by calling the `getInstance()` function
  - *in effect, the guardian to the instance object for the application*
- function also highlights a pattern known as `Lazy Load`
  - *checks to see if an instance has already been created*
- if not, create one and then store for future reference
- all subsequent calls will now received this stored reference
- this pattern is particularly useful for mobile development
- helps us save CPU and memory usage within an application
  - *an object is only created when it is actually needed*
- gives us a single object with getters and setters for the local storage

Cordova app - LocalStorage - data test I

app logic - storagenotes.js

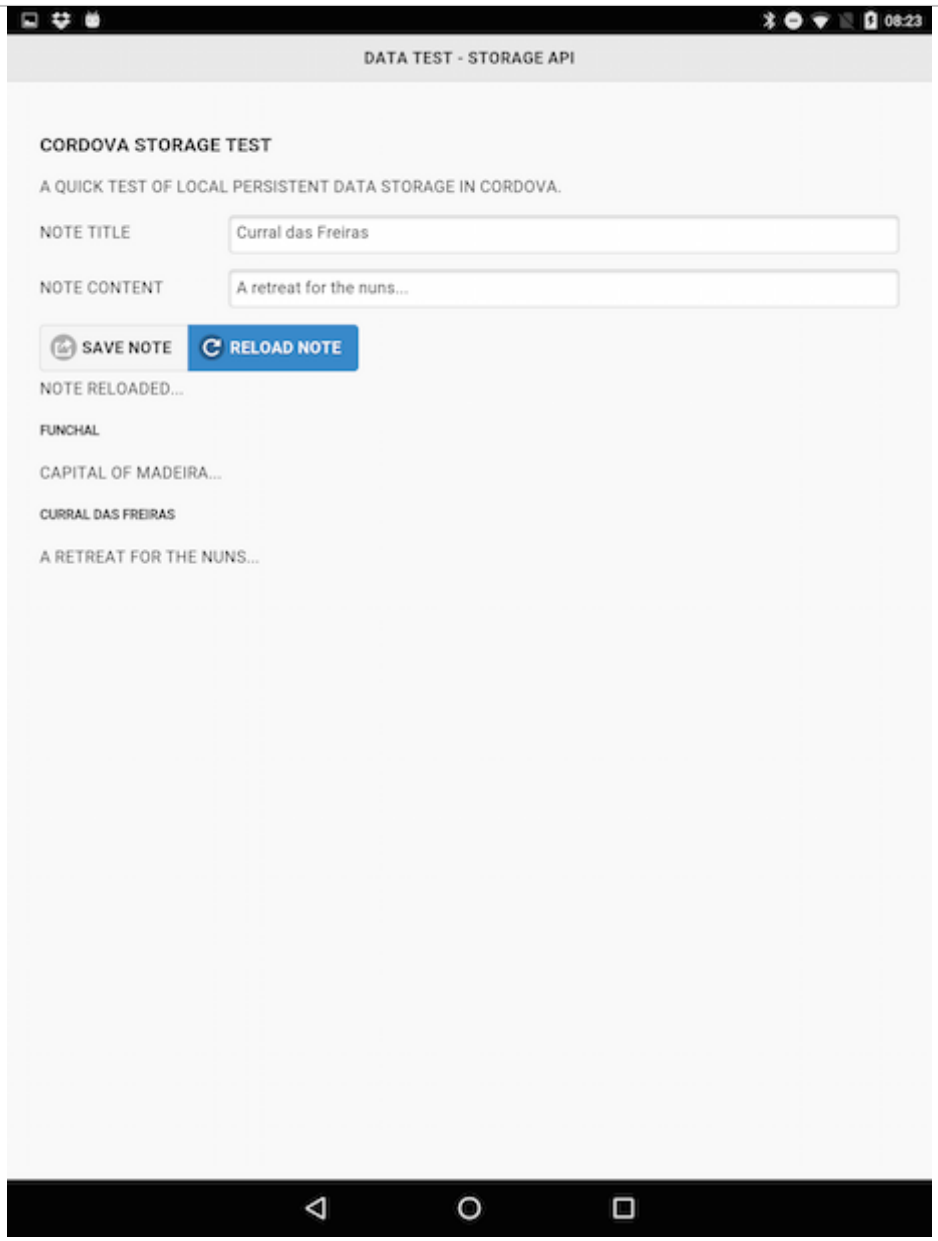
```
var NotesManager = (function () {
  var instance;

  function createNoteObject() {
      return {
        set: function (key, value) {
          window.localStorage.setItem(key, value);
        },
        get: function (key) {
          return window.localStorage.getItem(key);
        }
      };
  };

  return {
    getInstance: function () {
      if (!instance) {
        instance = createNoteObject();
      }
      return instance;
    }
  };

})();
```

Image - Data Tester



DataTest1 - update the notes

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

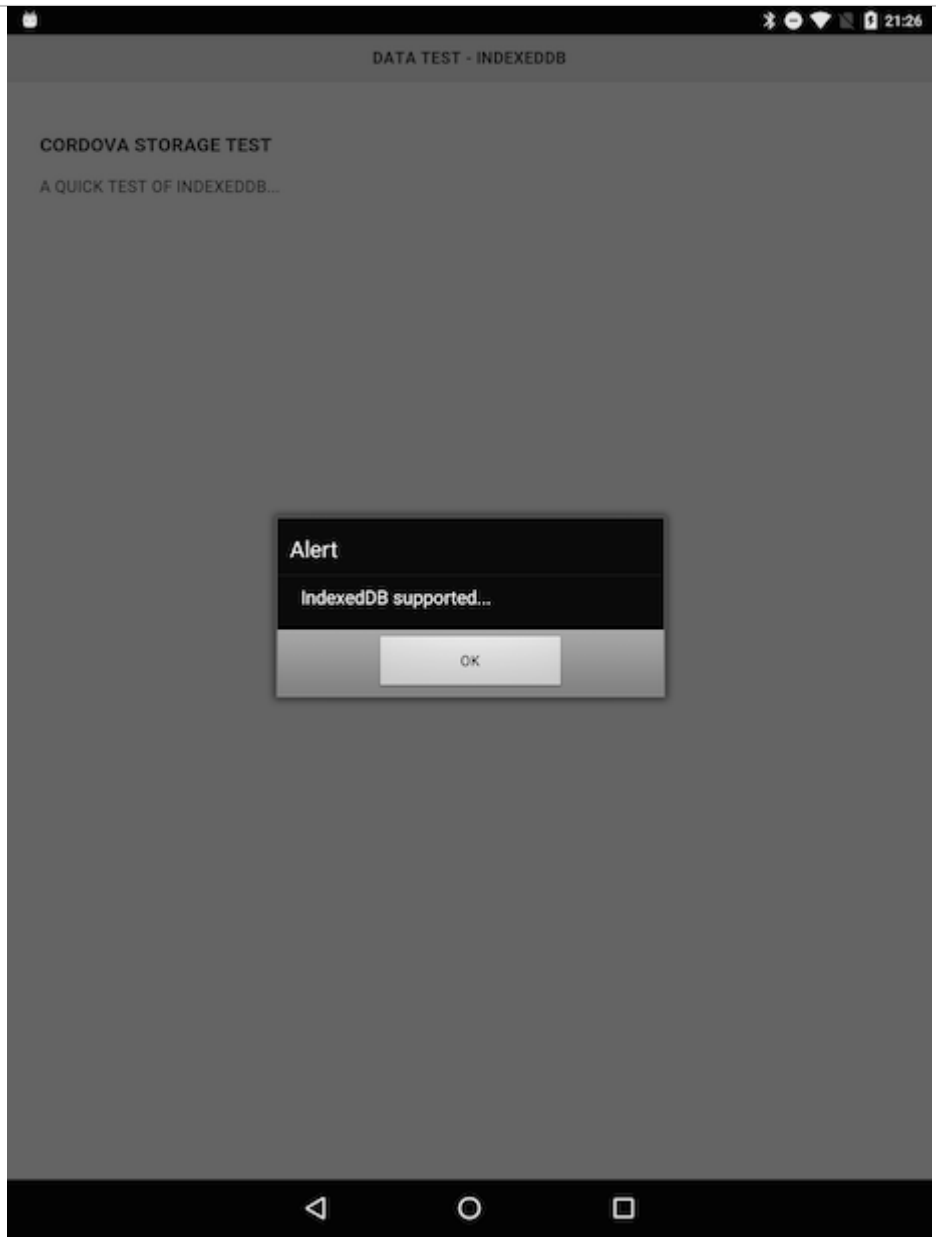
setup and test - part 1

- 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...");
}
```

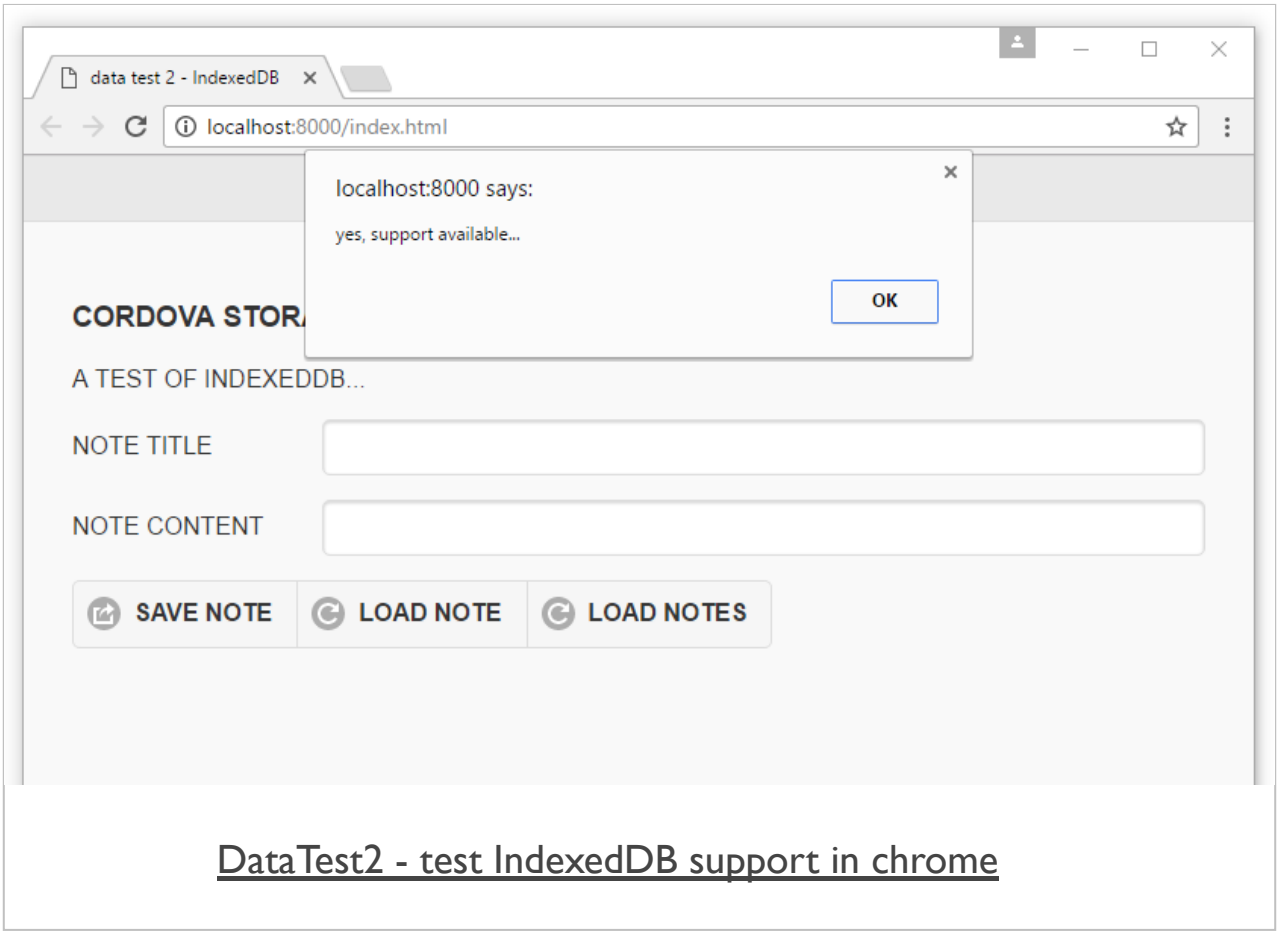
- Android support is available...

Image - IndexedDB Support



DataTest2 - test IndexedDB support in webview

Image - IndexedDB Support



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

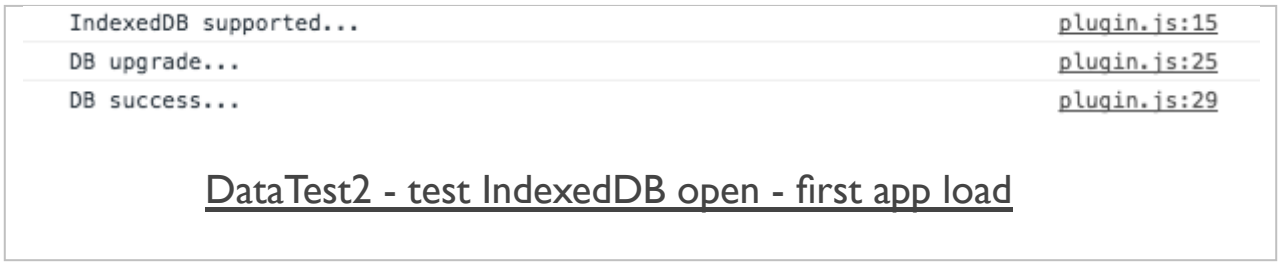
database - part 3 - create

- 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



DataTest2 - test IndexedDB open - first app load

Cordova app - IndexedDB - data test 2

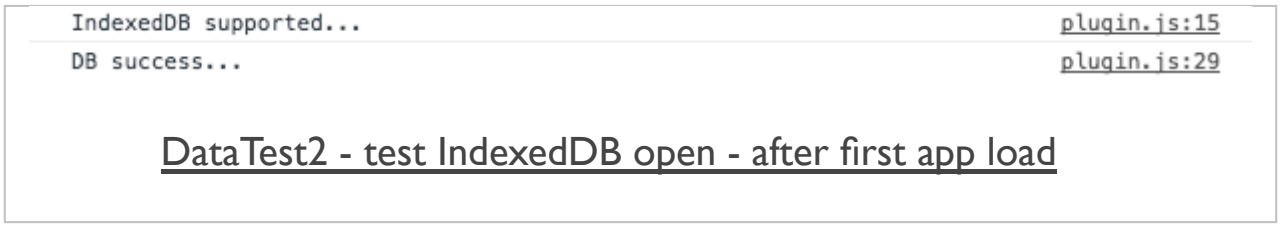
database - part 4 - success

- 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



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

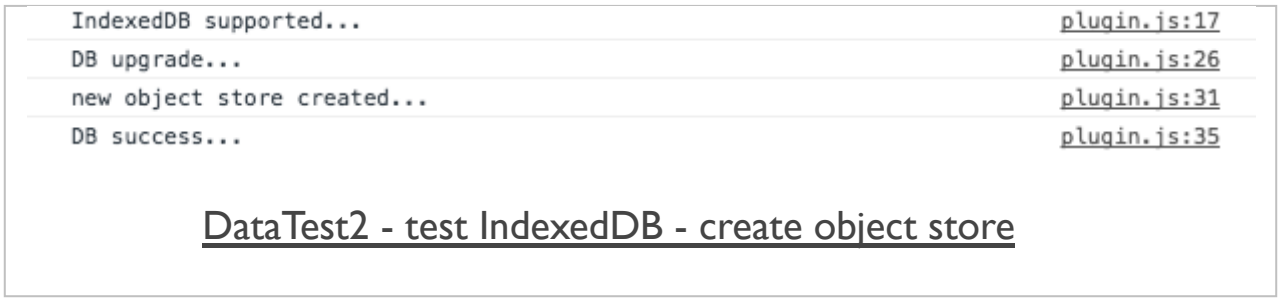
database - part 6 - data stores

- 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



DataTest2 - test IndexedDB - create object store

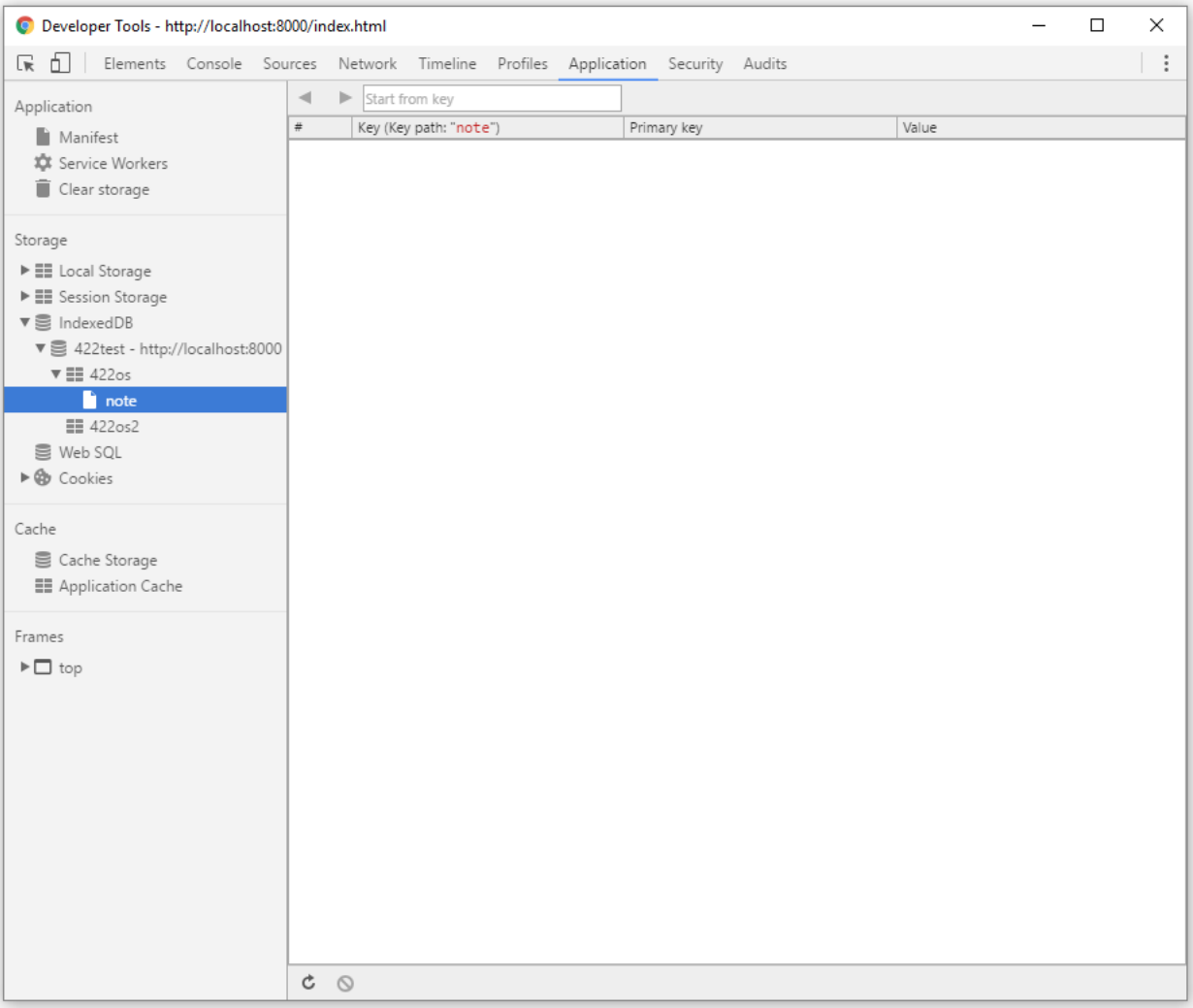
Cordova app - IndexedDB - data test 2

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="true"/>
  </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);
});
```

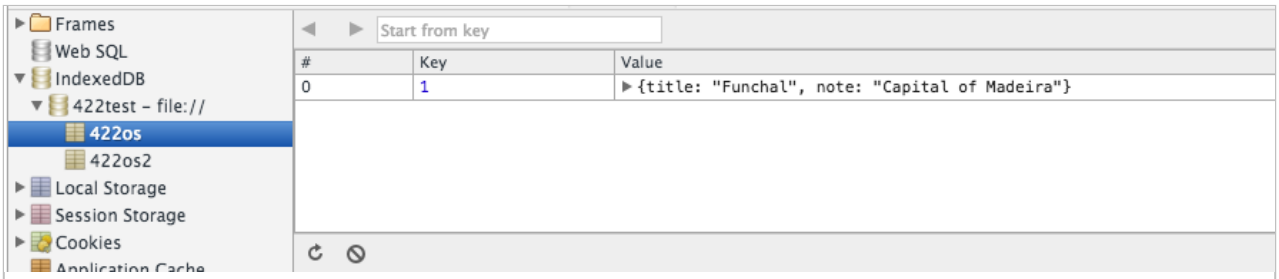
Cordova app - IndexedDB - data test 2

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...
  }
}
```

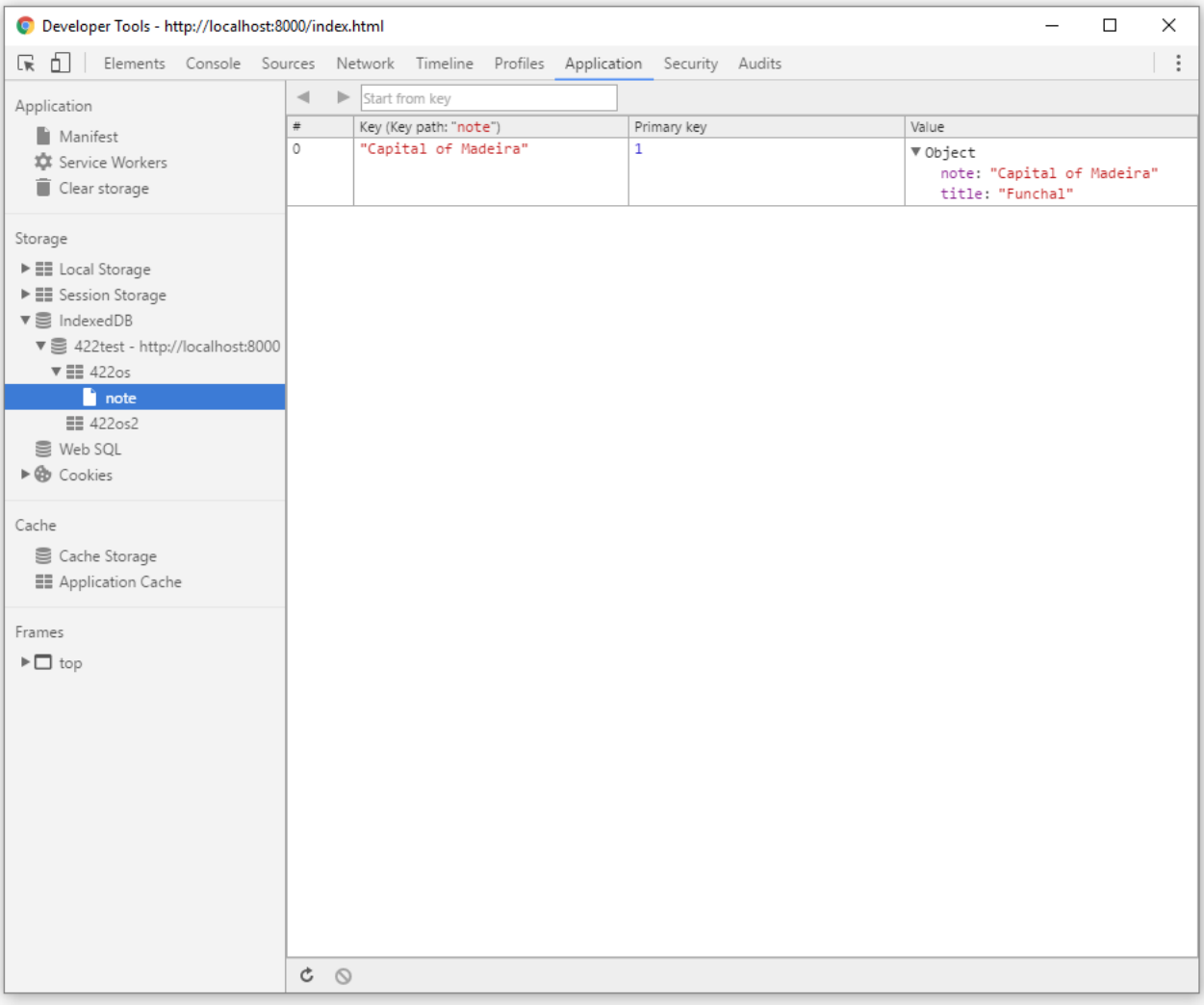
Image - IndexedDB Support

Image - IndexedDB Support



DataTest2 - test IndexedDB - save data to store 2

Image - IndexedDB Support



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***

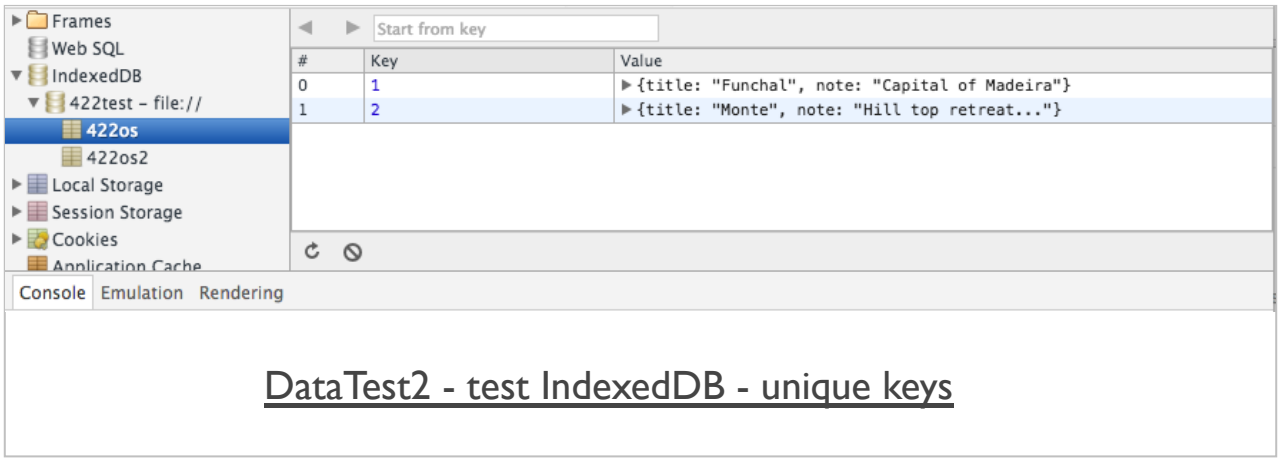
Cordova app - IndexedDB - data test 2

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`

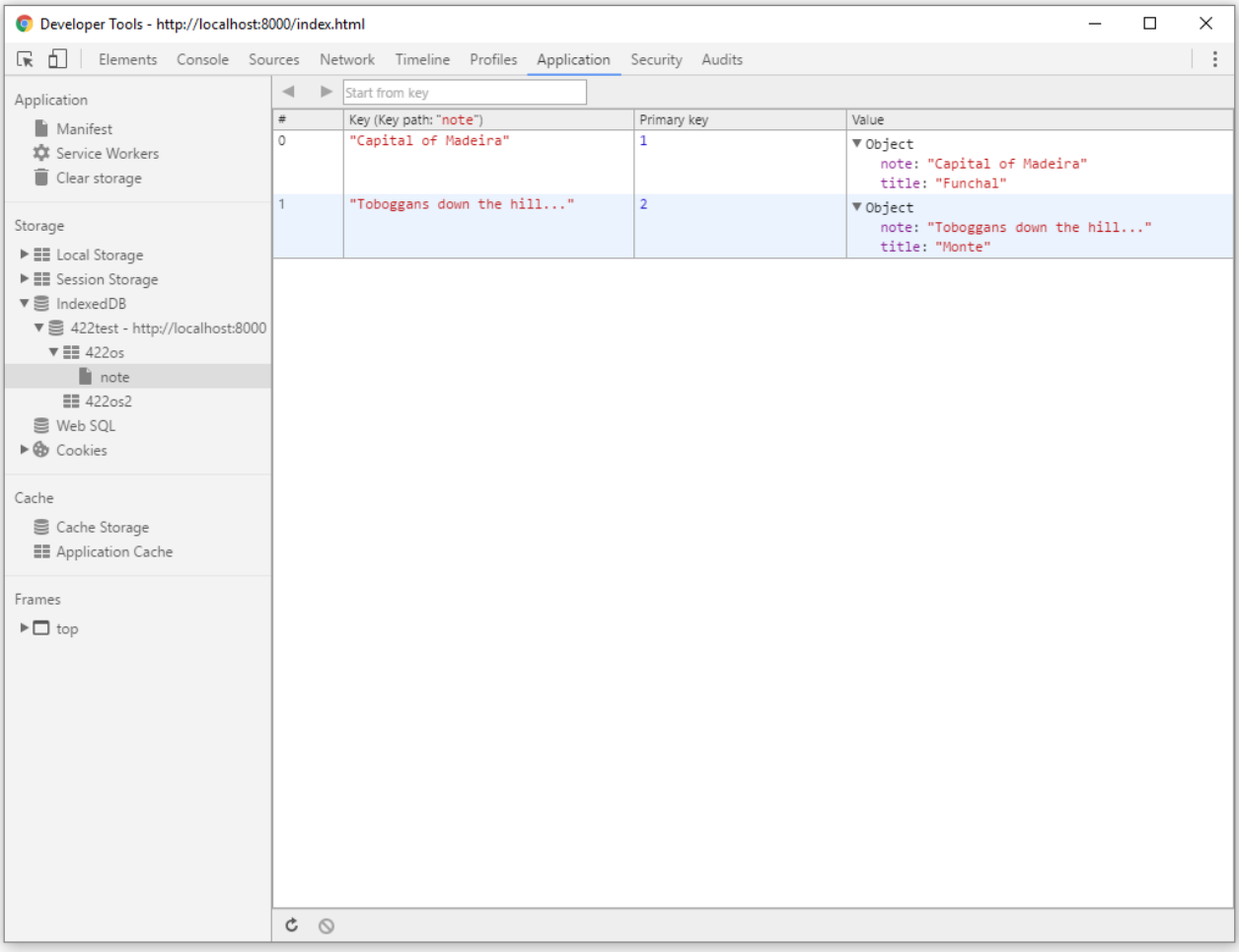
```
db.createObjectStore("422os", { autoIncrement: true });
```

Image - IndexedDB Support



DataTest2 - test IndexedDB - unique keys

Image - IndexedDB Support



DataTest2 - test IndexedDB - unique keys 2

References

- Cordova API
  - *Storage*
- GitHub
  - *cordova-plugin-indexeddb*
  - *cordova-plugin-websql*
- HTML5
  - *HTML5 File API*
- MDN
  - *IndexedDB)*
  - *Web APIs - FileError*
- W3
  - *Web storage specification*