

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

Fall Semester 2018 - Week 5

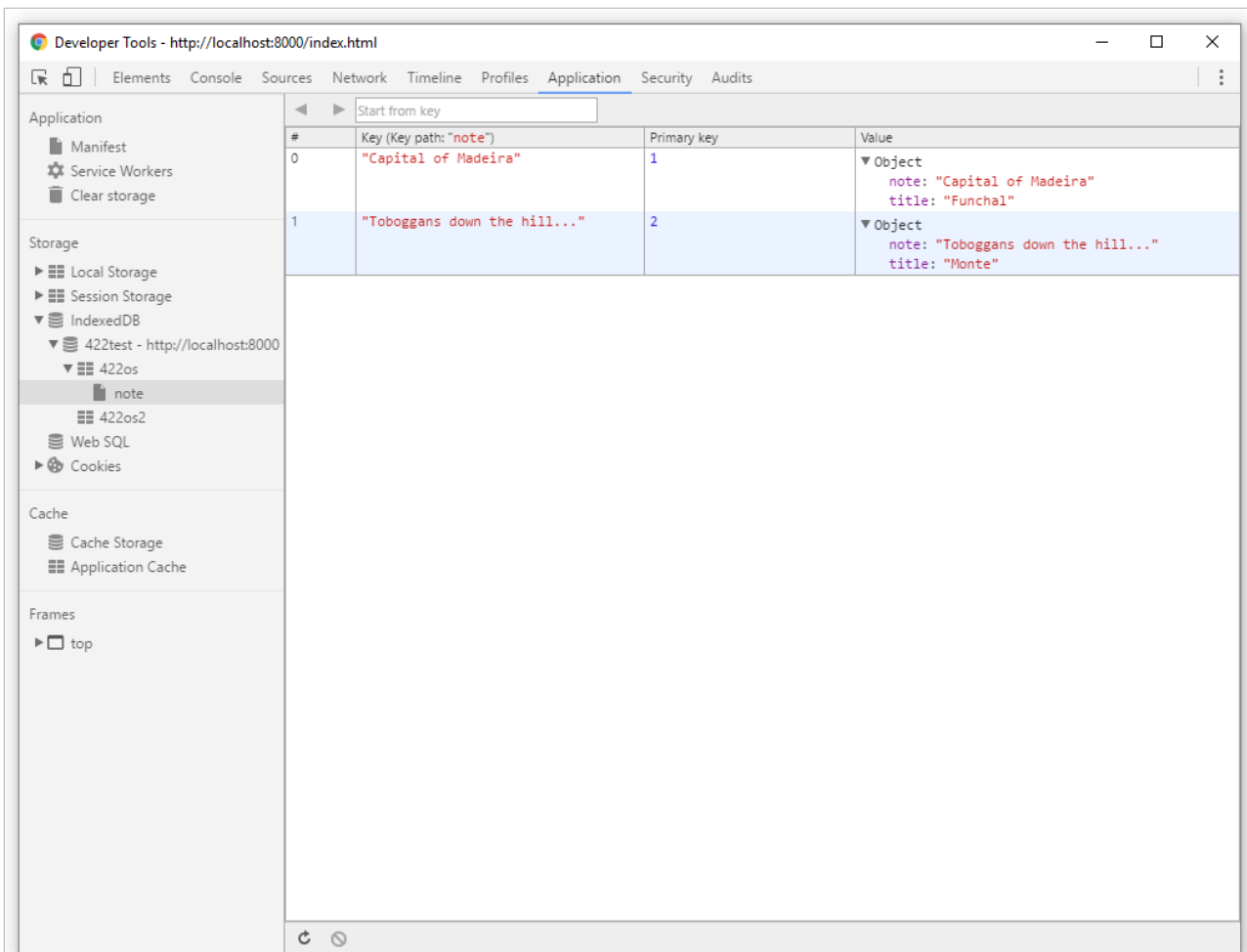
Dr Nick Hayward

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 = dbTransaction2.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"  
...
```

- 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

IndexedDB supported...	plugin.js:17
DB success...	plugin.js:39
▼ Object  note: "Capital of Madeira" title: "Funchal" ► __proto__: Object	plugin.js:81
found value...	plugin.js:82

DataTest2 - test IndexedDB - get data

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 cursor = 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*

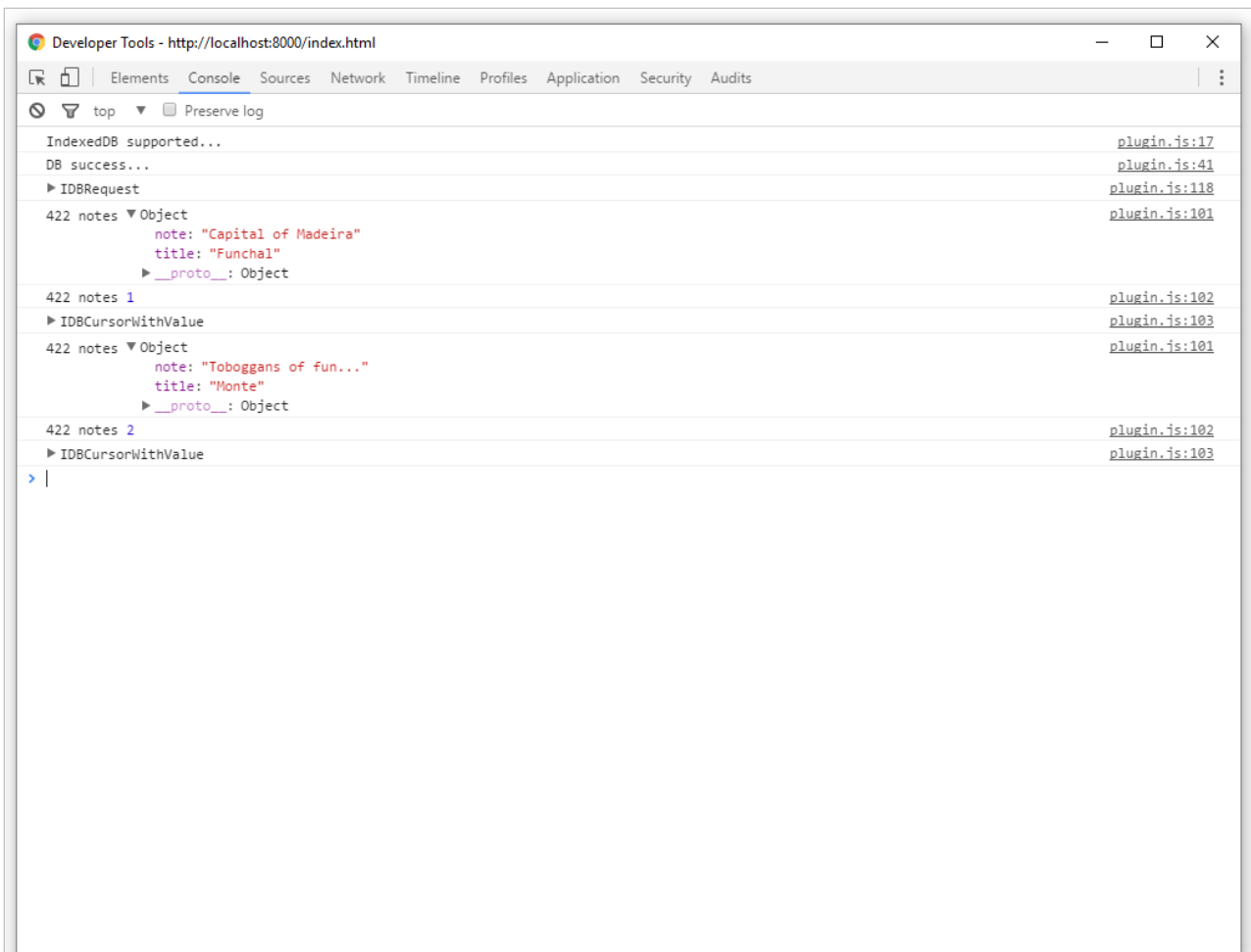
Cordova app - IndexedDB - data test 2

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

IndexedDB supported...	plugin.js:17
DB upgrade...	plugin.js:26
new object store created...	plugin.js:32
new index created	plugin.js:33
new object store 2 created...	plugin.js:37
DB success...	plugin.js:41

DataTest2 - test IndexedDB - create index

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

```
▼ IDBRequest ⓘ plugin.js:120  
  error: null  
  onerror: null  
  onsuccess: null  
  readyState: "done"  
  ▼ result: Object  
    note: "Capital of Madeira"  
    title: "Funchal"  
    ► __proto__: Object  
  ► source: IDBIndex  
  ► transaction: IDBTransaction  
  ► __proto__: IDBRequest
```

DataTest2 - test IndexedDB - query index

Image - IndexedDB Support

Frames

Web SQL

IndexedDB

422test - file://

422os

note

422os2

Start from key

#	Key (Key path: "note")	Primary key	Value
0	"Capital of Madeira"	1	{title: "Funchal", note: "Capital of Madeira"}
1	"Hill top retreat..."	2	{title: "Monte", note: "Hill top retreat..."}

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*

Server-side considerations - data storage

SQL or NoSQL

- common database usage and storage
 - *often thought solely in terms of SQL, or structured query language*
- SQL used to query data in a relational format
- relational databases, for example MySQL or PostgreSQL, store their data in tables
 - *provides a semblance of structure through rows and cells*
 - *easily cross-reference, or relate, rows across tables*
- a relational structure to map authors to books, players to teams...
 - *thereby dramatically reducing redundancy, required storage space...*
- improvement in storage capacities, access...
 - *led to shift in thinking, and database design in general*
- started to see introduction of non-relational databases
 - *often referred to simply as **NoSQL***
- with NoSQL DBs
 - *redundant data may be stored*
 - *such designs often provide increased ease of use for developers*
- some NoSQL examples for specific use cases
 - *eg: fast reading of data more efficient than writing*
 - *specialised DB designs*

Server-side considerations - data storage

Redis - intro

- Redis provides an excellent example of NoSQL based data storage
- designed for fast access to frequently requested data
- improvement in performance often due to a reduction in perceived reliability
 - *due to in-memory storage instead of writing to a disk*
- able to flush data to disk
 - *performs this task at given points during uptime*
 - *for majority of cases considered an in-memory data store*
- stores this data in a **key-value** format
 - *similar in nature to standard object properties in JavaScript*
- Redis often a natural extension of conventional data structures
- Redis is a good option for quick access to data
 - *optionally caching temporary data for frequent access*

Server-side considerations - data storage

MongoDB - intro

- MongoDB is another example of a NoSQL based data store
 - *a database that enables us to store our data on disk*
- unlike MySQL, for example, it is not in a relational format
- MongoDB is best characterised as a **document-oriented** database
- conceptually may be considered as storing objects in collections
- stores its data using the BSON format
 - *consider similar to JSON*
 - *use JavaScript for working with MongoDB*

Server-side considerations - data storage

MongoDB - document oriented

- SQL database, data is stored in tables and rows
- MongoDB, by contrast, uses **collections** and **documents**
- comparison often made between a collection and a table
- **NB:** a document is quite different from a table
- a document can contain a lot more data than a table
- a noted concern with this document approach is duplication of data
- one of the trade-offs between NoSQL (MongoDB) and SQL
- SQL - goal of data structuring is to normalise as much as possible
- thereby avoiding duplicated information
- NoSQL (MongoDB) - provision a data store, as easy as possible for the application to use

Server-side considerations - data storage

MongoDB - BSON

- BSON is the format used by MongoDB to store its data
- effectively, JSON stored as binary with a few notable differences
 - *eg: ObjectId values - data type used in MongoDB to uniquely identify documents*
 - *created automatically on each document in the database*
 - *often considered as analogous to a primary key in a SQL database*
- ObjectId is a large pseudo-random number
- for nearly all practical occurrences, assume number will be unique
- might cease to be unique if server can't keep pace with number generation...
- other interesting aspect of ObjectId
 - *they are partially based on a timestamp*
 - *helps us determine when they were created*

Server-side considerations - data storage

MongoDB - general hierarchy of data

- in general, MongoDB has a three tiered data hierarchy

1. database

- *normally one database per app*
- *possible to have multiple per server*
- *same basic role as DB in SQL*

2. collection

- *a grouping of similar pieces of data*
- *documents in a collection*
- *name is usually a noun*
- *resembles in concept a table in SQL*
- *documents do not require the same schema*

3. document

- *a single item in the database*
- *data structure of field and value pairs*
- *similar to objects in JSON*
- *eg: an individual user record*

Server-side considerations - data storage

Firestore - mobile platform - what is it?

- other data store and management options now available to us as developers
- depending upon app requirements consider
 - *Firestore*
 - *Cassandra*
- as a data store, Firestore offers a hosted NoSQL database
 - *data store is JSON-based*
 - *offering quick, easy development from webview to data store*
- syncs an app's data across multiple connected devices in milliseconds
 - *available for offline usage as well*
- provides an API for accessing these JSON data stores
 - *real-time for all connected users*
- Firestore as a hosted option more than just data stores and real-time API access
- Firestore has grown a lot over the last year
 - *many new features announced at Google I/O conference in May 2016*
 - *analytics, cloud-based messaging, app authentication*
 - *file storage, test options for Android*
 - *notifications, adverts...*

References

- GitHub
 - *cordova-plugin-indexeddb*
- MDN
 - *IndexedDB*