Conclusion

Hmmm... The two courses, HTML5 Part 1 and HTML5 Part 2, have covered a lot of material, and you may have trouble identifying which of the different techniques you have learnt about best suits your needs.

We have borrowed two tables from HTML5Rocks.com that summarize the pros and cons of different approaches.

TO SUM UP:

- **If you need to work with transactions** (in the database sense: protect data against concurrent access, etc.), or do some searches on a large amount of data, if you need indexes, etc.,**then use IndexedDB**
- **If you need a way to simply store strings, or JSON objects, then use localStorage/sessionStorage**. *Example*: store HTML form content as you type, store a game hi-scores, preferences of an application, etc.
- **If you need to cache files for faster access or for making your application run offline, then use the cache API (as of today).** In the future, look for the Service Workers API. They will be presented in future versions of this course, when browser support will be wider and the API more stabilized.
- **If you need to manipulate files (read or download/upload), then use the File API, and use XHR2.**
- If you need to manipulate a file system, there is a FileSystem and a FileWriter API which are very poorly supported and will certainly be replaced with HTML 5.1. We decided not to discuss these in the course because they don't have existing agreements from the community or the browser vendors.
- If you need an SQL database client-side: No! Forget this idea, please! There was once a WebSQL API, but it disappeared rapidly.

Note that all the above points can be used all together: you may have a Web app that uses localStorage and IndexedDB that caches its resources so that it can run in offline mode.

WEB STORAGE

Strengths of Web Storage	Weakness of Web Storage
1. Supported on all modern browsers, as well as on iOS and Android, for several years (IE since version 8). 2. Simple API signature. 3. Simple call flow, being a synchronous API. 4. Semantic events available to keep other tabs/windows in sync.	1. Poor performance for large/complex data, when using the synchronous API (which is the most widely supported mode). 2. Poor performance when searching large/complex data, due to lack of indexing. (Search operations have to manually iterate through all items.) 3. Poor performance when storing and retrieving large/complex data structures, because it's necessary to manually serialize and de-serialize to/from string values. The major browser implementations only support string values (even though the spec says otherwise). 4. Need to ensure data consistency and integrity, since data is effectively unstructured.

INDEXEDDB

Strengths of IndexedDB	Weakness of IndexedDB
1. Good performance generally, being an asynchronous API. Database interaction won't lock up the user interface. (Synchronous API is also available for WebWorkers.) 2. Good search performance, since data can be indexed according to search keys. 3. Supports agile development, with no need to flexible data structures. 4. Robust, since it supports a transactional database model. 5. Fairly easy learning curve, due to a simple data model. 6. Decent browser support: Chrome, Firefox, mobile FF, IE10.	1. Somewhat complex API. 2. Need to ensure data consistency and integrity, since data is effectively unstructured. (This is the standard flipside weakness to the typical strengths of NOSQL structures.)