Client-Side Storage

Kenneth Janssen

Rasmussen College

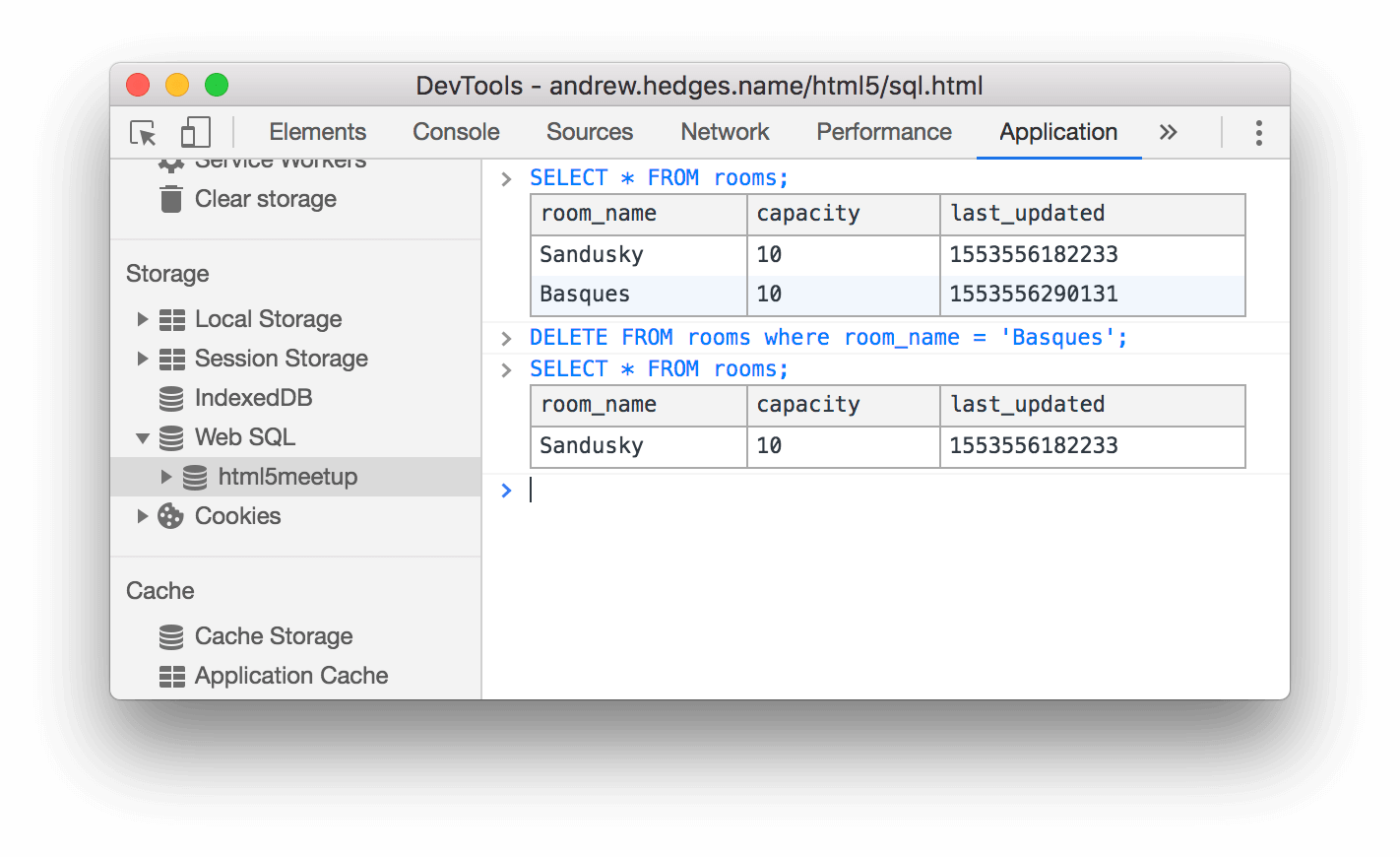
Author Note

This paper is being submitted on September 7, 2019, for Darice Corey’s CIS4655C Advanced Mobile Web Application course.

Client-Side Storage

In today's environment where everything is connected it is hard to think of a time when you need to worry about a web site working while not connected to the internet. There are many good reasons to store data on the user’s computer instead of online in a database. For example, sometimes users just do not want to sign up for a login account but they want their settings to be saved. Client-side storage is a great way to handle that. Another good example is to temporarily store data such as the current playing song for Internet music streaming sites. There are several storage options available to web developers. You might remember hearing the term ‘cookies’ but have increasingly become obsolete due to technical limitations such as only being able to store 4KB each (Kyrnin, 2019). Developers have Web Storage but there are more modern APIs now available so let us take a look at them in more detail.

**Web SQL Database**

Web SQL is a structured database using free-form key-value pairs but also allows for indexing and indexing is what helps speed up database queries. The data model is in table form. Google has a good image showing the data model using Chrome DevTools (Google, 2019).  


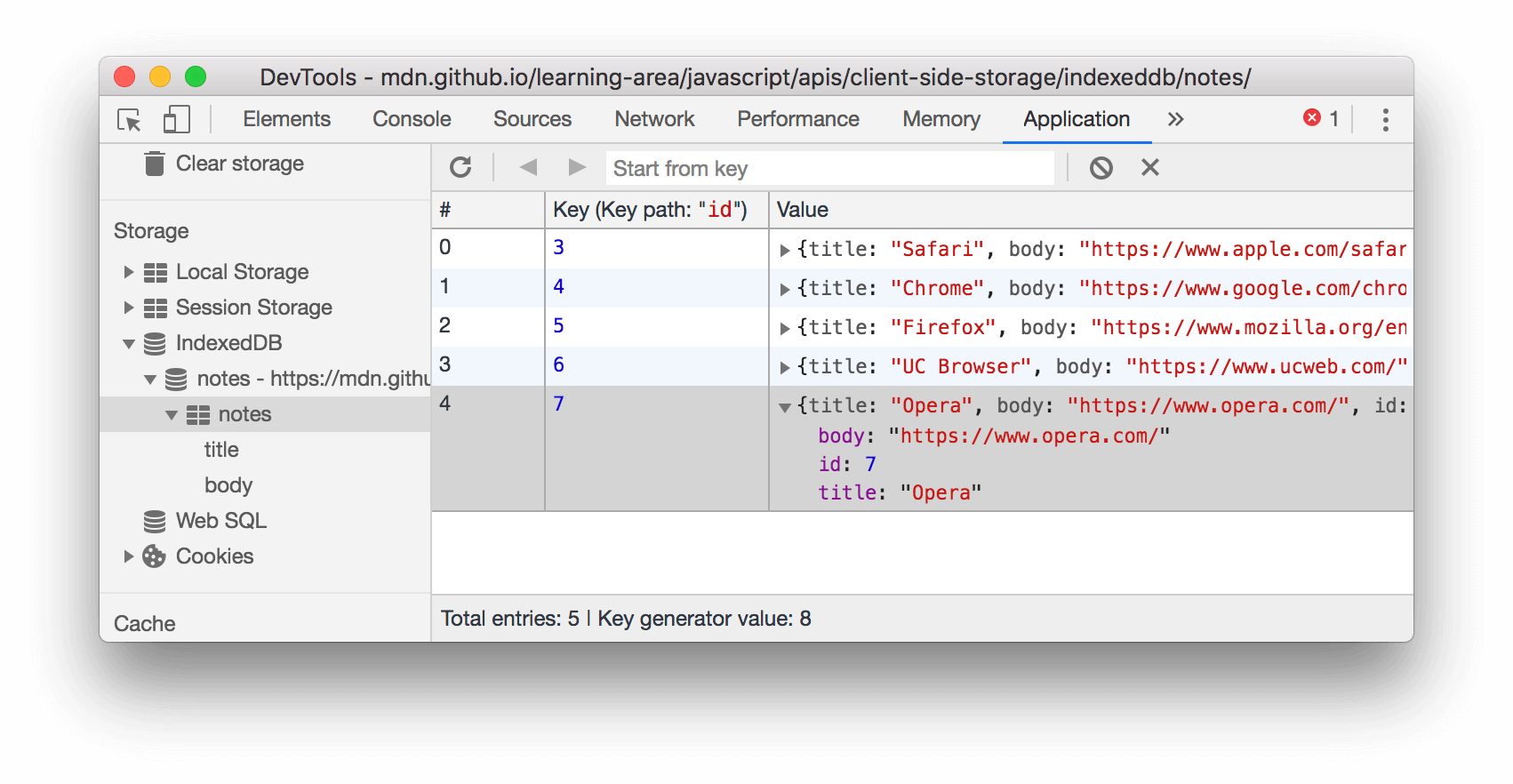
**Benefits**

Web SQL has many benefits such as being supported by all modern browsers. It has good asynchronous API functionality that prevents locking up the user interface while accessing the storage. As mentioned since we can index this storage, it is useful when your web application needs fast searches. Web SQL has SQL in its name for a reason and that it benefits from all the use cases where you would want a transactional database and as a by-product of the rigidly structured database we get the benefit of data integrity.

**Weaknesses**

After all, listed out all those strengths its unfortunate that Internet Explorer and Firefox have started to deprecated it meaning that soon in the future this type of storage is not possible for users which is a huge compatibility issue. All the boons from it being SQL relational also comes at the cost that web developers also have to fully understand SQL databases so there is an education cost behind using them. As with all relation databases, you need to develop the schema first which is not very Agile.

**Indexed Database**

Indexed databases are the result of the lessons learned from standard web storage and web SQL storage. They are still transactional databases, but unlike SQL you do not need to frontload the creation of schema or constraints (Mozillia, 2019). Again, Chrome Devtools is a great resource to check out the data model showing the key-values structure.  
  


**Benefits**

Indexed DB comes with the same performance boon from asynchronous API calls that Web SQL storage has which makes it better than standard Web Storage where you do not want users to lose user interface functionality. It comes at no surprise that Indexed DB get their name because they are indexed which means they also keep the searching performance bonus that SQL Web Storage has. The data model is easy to pick up and since at its heart Indexed DBs are transactional they have strong data integrity.

**Weaknesses**

Being a newer API, it also is very complex which can lead to nested API callbacks.

**File Access**

HTML5 supported Web browsers can also access files using File-API which essentially just allows the site to access files on the client's native file system. This API is actually three different parts: FileReader, FileWriter, and FileSystem. Since this is a paper about Storage, the FileReader API will be glossed over but it is still a valuable component with working with client storage (Marini, 2011).

**FileReader**

**Benefits**

Allows access to local files which means a web developer could query a photo to make sure it is not larger than 4mbs and is in the jpg format. It could also generate thumbnails or even perform operations on them such as creating thumbnails since the web app can read the file prior to uploading. Since FileReader-API is just accessing files on the client’s computer there really is not a data model to show.

**FileSystem**

This portion of the API has a standard now but is still quite new. While the previous storage types are useful for text or ‘structured data’, this new API helps tackle large files or binary content.

**Benefits**

The FileSystem is well suited for large content such as video files, audio, pdf, images. It also comes with asynchronous API support.

**Weaknesses**

It is not a standard API as of now meaning that there is no support for Internet Explorer or Safari. Microsoft has supported it on its new browser Edge though but since IE has not been fully decommissioned there is a large compatibility issue (Mozilla, 2019). It is not transactional so there is less data integrity than other choices. It is also not indexed so there is no searching support.

**Conclusion**

There are quite a few ways to deal with the client-side storage needs for a web application. If you go the SQL route, you will have the large task of developing Entity relation diagrams and trying to understand how to structure the data model while if you go with IndexDB, you can approach in a less Waterfall development approach. The File-API is a good one to remember because its not typically what you would think about when talking about storage because it is more about working with the user’s storage not a database or data model that the web developer has created themselves but it is none the less an important part of a web developer’s toolkit.

# References

Google. (2019, May 29). *Chrome DevTools*. Retrieved from Developers.Google.com: https://developers.google.com/web/tools/chrome-devtools/storage/websql

Kyrnin, J. (2019, June 26). *Learn the Maximum Size That a Web Cookie Can Be*. Retrieved from Lifewire: https://www.lifewire.com/cookie-size-limit-3466810

Marini, J. (2011, December 2). *HTML5: File API*. Retrieved from Lynda: https://www.lynda.com/HTML-5-tutorials/HTML5-File-API-in-Depth/90369-2.html

Mozilla. (2019, March 18). *FileSystem*. Retrieved from Developer.Mozilla.Org: https://developer.mozilla.org/en-US/docs/Web/API/FileSystem

Mozillia. (2019, September 8). *IndexedDB API*. Retrieved from Mozilla Developer Network: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB\_API