INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact factor: 4.295
(Volume 4, Issue 4)
Available online at: www.ijariit.com

Local storage in web application through HTML 5

Vidhan Chand
vidhanchand@gmail.com
Babasaheb Bhimrao Ambedkar Bihar University, Muzaffarpur, Bihar

Dr. A. K. Singh
ajaykrsingh51@gmail.com
Babasaheb Bhimrao Ambedkar Bihar University, Muzaffarpur, Bihar

Dr. Pankaj Kumar
ccpplinux@gmail.com
Babasaheb Bhimrao Ambedkar Bihar University, Muzaffarpur, Bihar

ABSTRACT

In recent years a number of sophisticated mechanisms for storing and managing data on web clients have emerged. These client-side storage mechanisms bring along several benefits, including faster websites and improved user experience. In the conventional model all data stored on the server side storage. When a user requests new data, a request is made by the client to the server and the data is retrieved from the database residing on the server. In this interaction model transferring of data is done between the client and server so takes more time and thus slows down the website leading poor user experience. In Modern Interaction Model, there is a provision of client-side storage that is local storage. The first mechanism used for local storage was HTTP cookie, which enabled the web application to store data on web clients. However, cookies do not meet the requirements of the modern web application. There are several techniques evolved for local storage like HTTP Cookie, Flash Cookie, and Google Gears. Nowadays HTML5 plays a major role in local storage.

Keywords— *HTML5, Local Storage, Application Cache, HTTP Cookie, Flash Cookie, Google Gears*

1. INTRODUCTION

Local storage means storage of data on the client machine. In recent years a number of sophisticated mechanisms for storing and managing data on web clients have emerged. These client-side storage mechanisms bring along several benefits, including faster websites and improved user experience.

The web application has two options to store data.
1. on the web server.
2. on the web client.

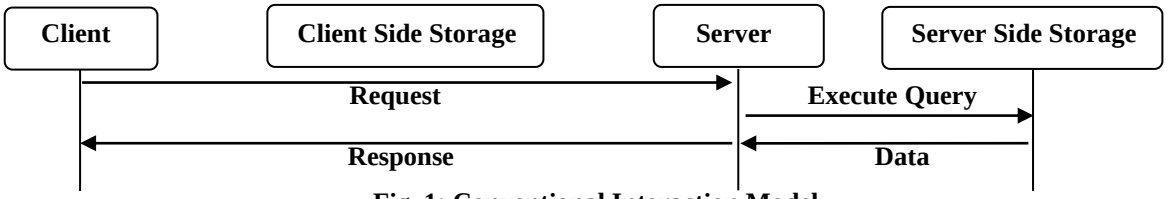

Fig. 1: Conventional Interaction Model

In the conventional model all data stored on the server side storage. When a user requests new data, a request is made by the client to the server and the data is retrieved from the database residing on the server. In this interaction model transferring of data is done between the client and server so takes more time and thus slows down the website leading poor user experience.

There are following disadvantages of Conventional Interaction Model:
1. Unnecessary network traffic.
2. Require an internet connection
3. Increase the server load.

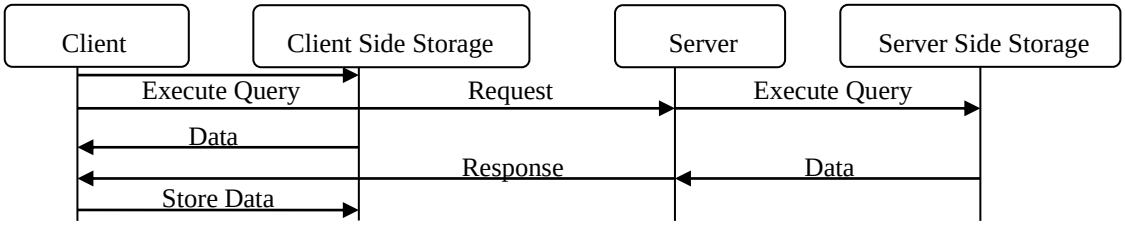

Fig. 2: Modern Interaction Model

In Modern Interaction Model, there is a provision of client-side storage that is local storage. The first mechanism used for local storage was HTTP cookie, which enabled the web application to store data on web clients. However, cookies do not meet the requirements of the modern web application.

2. EVOLUTION OF LOCAL STORAGE

There are various techniques evolved for local storage.

- HTTP Cookie
- Flash Cookie
- Google Gears

2.1 HTTP Cookie

HTTP cookie invented by Netscape in 1994 and later standardized by IETF, is a popular mechanism that lets web servers and web clients to maintain and manage a stateful session between each other over the mostly stateless HTTP protocol. Cookies allow a small piece of simple key-value pair data (4 kB per cookie, 50 cookies per origin) to be stored in a user's web client (browser). Cookie data can be set to last over a single session (session cookie) or multiple sessions (persistent cookie). Typically, web applications use them for session management, authentication purposes, storing user preferences, and user tracking. All major browsers have native support for cookies.

Although cookies provide a simple and well-supported storing mechanism, they suffer from several problems. First, each cookie is sent back and forth with every HTTP request (via HTTP headers), which adds a lot of unnecessary overhead. Second, their storage size is way too small for modern web applications. Third, the same web application using cookies cannot be run in multiple web browser tabs simultaneously.

2.2 Flash Cookie

In 2002, Adobe introduced a feature called Flash cookie (Local Shared Object) which is an Adobe Flash Player based mechanism to store and manage data on a user's computer. Flash cookies are exactly like HTTP cookies, except that they can store more complex data types than just plain strings and offer more storage space (defaults to 100 KB per origin). The easiest way to access Flash cookies from JavaScript is via the External Interface object, which was introduced in Adobe Flash 8 in 2006.

In order to use Flash cookies, Adobe Flash Player 6+ plug-in needs to be installed to a web browser. Mobile browsers, however, are gradually moving away from Flash, and therefore the use of Flash cookies is no longer an option on mobile devices.

2.3 Google Gears

Google Gears is open source software that was first developed by Google and allows online web applications to be used offline.

Google Gears is composed of the following:

- A local server component, that acts as a local web server by storing static resources such as HTML, CSS and Javascript images and multimedia files on the client machine.
- A Desktop component, that gives a web application access to the local machine resources.
- A Geolocation component, that allows a web application to become aware of the geographical location of the machine.
- Database component, that resides on the client machine and uses a relational database management system called SQLite.
- A workpool component, that allows javascript to run in the background.

3. HTML 5 AND LOCAL STORAGE

Caching data in web browsers is nothing new. The majority of web browsers use some kind of cache to store often needed data as images, style sheets or scripts. HTML5 gives developers an advantage of manual and explicit control of the certain type of browser's cache to clear the way for creating WebPages capable of running offline. HTML5 also goes much further and offers technologies as Web Storage, web databases, and sandbox file system.

Creating an offline web application also brings one related advantage: The application will almost certainly run much more smoothly as fewer data has to be exchanged between client and server and data synchronization can take place in the background of user's working process.

3.1 Web Storage

Applications (desktop or web) need to store data. Desktop applications can store data on the machine by saving it in local files (.conf, .ini, .dat, .xml or any other developers do design), registries and embedded databases, or developers can implement almost any solution that comes in their mind.

Web applications were designed to store all data on the server and just visualize the data on the client. Cookies were invented to remember the state of the website and can be used as a persistent storage but they are limited to 4 KB of data and are included in every HTTP request to the related site, which unnecessarily loads the network and can be compromised. The first possibility to store data right in the browser without cookies brought Internet Explorer with its JavaScript object userData which managed to save up to 64 KB and even 640 KB on trusted domains. LSOs (Local Shared Objects) in Flash could keep up to 100 KB per domain. Google Gears offered API to an embedded SQLite database and some more solutions were established (Adobe AIR, Silverlight) as developers desired a tool to store data. All of these solutions are either platform dependent or third-party plugins. HTML5 standard solves these difficulties and offers a unified solution.

The web storage interface is the standard which tries to solve this inconsistency. It is widely supported even in nowadays browsers (even Internet Explorer which usually adopts new technologies only when they are declared as official standard supports this feature since version 8).

The interface is really simple as it stores data as an associative array (in terms of Java it reminds Map) – key-value format. The official interface proposed by W3C looks like this

```
interface Storage {
        readonly attribute unsigned long length;
        DOMString? key(unsigned long index);
        getter DOMString getItem(DOMString key);
        setter creator void setItem(DOMString key, DOMString value);
        deleter void removeItem(DOMString key);
        void clear();
};
```

Web Storage interface is in every browser implemented by two JavaScript objects:

- **localStorage** object manages data which is supposed to be stored permanently unless users clear the browser cache.
- **sessionStorage** object manages temporary data supposed to be remembered only for a current session – one window or tab. After the end of session data vanishes.

As both objects implement the same interface the only difference is how long the data should remain. The data is stored for each website domain and other browsers or even users cannot manipulate or obtain the data. JavaScript syntax is quite simple and all objects are traversable so the next example of getters and setters are equivalent. This example checks the availability of feature and then sets and get the data from local storage:

```
if (window.localStorage) {
        //these three examples of setting data is equivalent
        localStorage["key"] = "data";
        localStorage.setItem("key","data");
        localStorage.key = "data";
        //these three examples of getting data is equivalent
        output = localStorage["key"];
        output = localStorage.getItem("key");
        output = localStorage.key;
}
```

The data stored in Web Storage is saved in a web browser and can be accessed any time even when the browser is offline and is trying to access some web page, which makes this technology a good use to store dynamic data on client's side and synchronize it when the internet connection is available again.

3.2 Databases

The Web Storage is useful for storing small amounts of unstructured data but when developers need to emulate storage capabilities of the server there is a better solution.

There were two standard proposals in the beginning: The Web SQL Database and Indexed Database API (IndexedDB).

The Web SQL Database was meant to create databases and query them with a dialect of Structured Query Language but now is not furthermore developed and standardization process stopped.

The official disclaimer of W3C: "This document was on the W3C Recommendation track but specification work has stopped. The specification reached an impasse: all interested implementors have used the same SQL backend (SQLite), but we need multiple independent implementations to proceed along a standardization path. "

IndexedDB introduces an object store mechanism by which data is stored in the database. IndexedDB is not a relational database and is very similar to NoSQL server databases or more precisely object-oriented databases. IndexedDB does not use Structured Query Language but the object store has methods to create cursors or ranged queries. IndexedDB makes possible to store whole JavaScript objects (in the not relational schema) and create indexes and sorting on them. The API is completely asynchronous that means that developers do not obtain data but they only request data and when the data is ready callback function is executed to process it. It reminds of concept of XMLHttpRequest (AJAX) objects.

```
        //standard synchronous data access
        //usually developers obtain data by assigning
        //a returned value of some method to a variable
        result = database.get ("key");
        //Indexed DB asychnronous request:
        var request = objectStore.get("key");
```



```
request.onerror = function(event) {
//when IndexedDB fails when obtaining the key,
//it triggers the onerror function
};
request.onsuccess = function(event) {
console.log ("Value of key is" + request.result.value);
};
```

The IndexedDB can currently use up to 50 MB per origin but user interface will just ask permission for storing blobs bigger than 50 MB.

3.3 File API

JavaScript was not supposed to access any files on a computer because of security concerns. HTML5 (or more precisely standard related to HTML5) brings a standardized API to read and write files in a secure way. Read and write of files can be made to a sandbox inside of computer's file system so applications can read only files which were programmatically stored. The API does not have a direct access to the standard file system but users can select the file(s) (in a dialog window or by drag and drop) and these files can be accessed by the API.

The specification declares an interface of FileReader which is capable of reading the contents of files. There are four options for reading:

- readAsBinaryString
- readAsText
- readAsDataURL
- readAsArrayBuffer

The reading is asynchronous and FileReader informs about changes by triggering events.

File API can be used for unzipping a file or for using any file imported in the application (and the file does not have to be uploaded to a server), e.g. creating thumbnails or verifying mime types. When the user is offline and wants to upload a file, API can easily save it to the sandbox and attempt the upload when the connection is working. Any files can be saved in the sandbox file system and accessed while offline.

3.4 Offline detection

Developers often need to find out, whether the browser is online or not. JavaScript defines attribute onLine in object navigator recognizing the connection of the computer. It also defines two events (offline and online) which can be listened on the object window. The update occurs on click on a link or when a script requests a page. The problem is that browsers implement this behavior differently and uselessly. Firefox and Internet Explorer sets onLine to false when the browser is in offline mode, true otherwise. Chrome and Safari (Webkit core) sets an offline flag when the browser is not able to reach Local Area Network or router, therefore when the router is working but internet provider is off it sets the attribute true and developer gets a false positive. This is the main reason why JavaScript developers created their own solutions to detect whether browsers are offline. Very simple solutions are detecting an error state of Application Cache.

```
window.applicationCache.addEventListener ("error", function (event) {
        //if the developer is sure that manifest exists
        //then the browser cannot reach the web and therefore is offline
});
```

The solution above is sufficient when the developer needs to check online state on a load of the web page. If the checking is needed even after, developers have to use other techniques. The best known are polling with XMLHttpRequest or using WebSockets. The application sends a request every e.g. 30 seconds and defines maximum timeout. If the response does not come in the timeout, the application is probably offline or the server is off. These states are almost equivalent to the site is not reachable.

3.5 Application Cache

The manifest attribute on the HTML element is responsible for ensuring that web applications remain available even when the user is not connected to the network. Even though most modern browsers have caching capabilities, they are largely unreliable and inefficient. HTML5 addresses these issues with the application cache API. The application cache API enables offline browsing, stores cached data locally, and therefore, making data load faster and reduces the workload on servers. Application caching is quite simple and easy to use.

4. CONCLUSION

After a study of different approaches for storing and managing data in a web application, we can say that simple HTTP cookie was the only considerable option used for session management due to their limited storage space. Later on, after the development of Google Gears, unlimited storage space and a rich set of features were included in the web applications. Now a day's database storage: (Web Storage, Web SQL Database, Indexed Database API (IndexedDB) of HTML5 offers a simple key-value pair data storage mechanism supported by all major browsers, whereas IndexedDB includes a number of advanced features but is still a work in progress. Studies have also shown how the use of client-side storage mechanisms can significantly improve both the performance and user experience of web applications.

5. REFERENCES

[1] Mark Pilgrim, (2010). "HTML5 Up and Running"
[2] Brian P. Hogan, (2011). "HTML5 and CSS3: Develop with Tomorrow's Standard Today"
[3] Gobe Donnini, (2012). "Is HTML5 a New Battleground for Browsers", Search Engine Journal.
[4] Rick Rogers, (2011). "Augmented Reality with HTML5", Linux Journal
[5] Ivan Bayross, (2011). "HTML5 and CSS3 Made Simple"
[6] Bruce Lawson, (2011). "Introducing HTML5".
[7] Elizabeth Castro (2011). "HTML5 and CSS3"
[8] Christopher Murphy (2012). "Beginning HTML5 and CSS3, The web evolved".
[9] Bill Sanders (2010). "Smashing HTML5"
[10] Denise Woods (2012). "HTML5 and CSS: Complete".
[11] Jennifer Kyrnin (2012. "Sams Teach Yourself HTML5, Mobile Application Development.
[12] Steve Fulton (2013). "HTML5 Canvas"
[13] Paul. S. Wang (2012), "Dynamic Web Programming and HTML5".
[14] www.w3schools.com/html5.
[15] Google. Gears Improving Your Web Browser. (2010). <http: //code.google.com/p/gears/.>
[16] Hickson I.: Web Storage (2013) from <http://www.w3.org/TR/webstorage/>
[17] Hickson I.: Web SQL Database (2013) from <http://www.w3.org/TR/webdatabase/>
[18] Ranganathan A. and Sicking J.: File API, W3C Working Draft (2012) from <http://www.w3.org/TR/FileAPI/>
[19] Kilan P.: Working off the grid with HTML5 offline, (2011) from <http://www.html5rocks.com>

