Client-side Storage. Offline Availability Of Data

Submitted: 6th July 2014; accepted 28th July 2014

Arkadiusz Janik, Szymon Kiebzak

DOI: 10.14313/JAMRIS_4-2014/30

Abstract:
Since paradigm shift from thick clients to thin ones and through growing requisition of remote services we have been facing continuous evolution of web applications. Standards and approaches of building HTML clients have been changing and now one of the biggest challenges for web application developers is issue related to client-side data storage. This paper applies to new technologies specified recently by World Wide Web Consortium. These specifications refer to new methods of storing data in web clients, which can be accessed via JavaScript in modern web browsers. In addition, the paper introduces a design and a reference implementation of a framework for Java language and Java Server Pages technology that simplifies implementation of web based applications with client-side storage. The framework is designed for integrating Java language with above mentioned new W3C specifications to create more powerful, modern, thin client web applications easily, without a need of knowledge of implementation details of JavaScript API for new technologies but using only Java language. The framework can be used to build thin clients, including mobile web-based clients that may replace not –portable, native mobile applications.

Keywords: *HTML5, Offline-storage, WebSQL, Web Storage, IndexedDB, File API, client-side storage*

1. Introduction

W3C organization has recently introduced HTML5 specification. Since then lot of effort has been made on implementing a new client-side storage technologies to give developers opportunity to create powerful web applications that utilize HTML5 and CSS3. Previous technologies of client-side storage, such as HTTP Cookies or URL parameters, were not enough. The lack of sophisticated client-storage technologies was particularly important in a world of mobile applications. This led to expansion of thick, native clients that are not portable and have to be implemented for each mobile platform (Android, iOS, BlackBerry, Windows Phone etc.). Additionally, there are more and more users of Internet and, due to limited servers' efficiency, storing data on a client side is an obvious way of increasing applications performance (especially now when clients machine nowadays have relatively big CPU and RAM capabilities).

The new client-side storage technologies could be used in improvement of distributed data acquisition systems in the field of automation and measurements where problems with an access to the network may occur.

Web Storage, Indexed Database and File API are modern technologies and there has not been much research done in relevance to them (last two have not been marked with Recommendation status – W3C still works on the specification). They still require thorough examination. In the paper, we summarize the results of our investigation and viability study on the subject of above mentioned technologies. In the first part of the paper we present previous ways of storing data in a web browser on a client-side. Then, we summarize Web Storage, Indexed Database, File API. We also present a design of a framework and reference implementation of a library for Java language that can be used to integrate web applications written in Java (web frameworks based on JSP technology) with the mentioned client-side technologies.

2. An Overview of Previous Client-side Storage Methods

2.1. HTTP Cookies

HTTP Cookies were introduced in 1994. The main motivation was the ability to store state of customers' basket in web stores. It was a very important matter, especially that e-commerce businesses were growing ones.

A concept behind cookies is very simple. A cookie is a small piece of information that consists of unique key and its value. The cookies are stored as files in a client's file system. Obviously web browser can access these files. When HTTP server wants to set a cookie in a particular client computer, it sends a key and a value of the cookie in the HTTP Response.

After that when a client browser generates the new HTTP Request, it attaches all the previously stored cookies to a header in the HTTP Request thus all the cookies for a relevant web domain are sent to the web server. A drawback of using the HTTP Request for transporting cookies is a limitation of the cookies size (4kB for a web domain).

Another weakness of the cookies is that the technology is susceptible to web attacks. Even though some mechanisms to prevent such threats have been introduced still not cope with all attacks [6].

Nowadays, the HTTP cookies are still widely used by web applications, primarily to store session ID but

also to provide website personalization. The last but not least – the cookies can be used to track user activities on websites.

2.2. URL Parameters

In some cases, cookies can be substituted by URL parameters. It is quite important since users can disable cookies in web browsers. Therefore, URL parameters are often used to keep users session. The main difference between this mechanism and cookies is a fact that information transferred in URL parameters is lost once a user closes a web browser window. Cookies last on users' file system.

3. An overview of W3C client-side storage specifications

3.1. HTML5 Manifest

New HTML5 specification introduces noteworthy feature, which is a manifest file. In the file, which location is referred in HTML documents of web applications, developers may specify pages that should be cached in web browsers. This feature can be used to provide an offline web application. A manifest file is a great idea as it allows a developer to actively affect behavior of partially offline applications [3].

3.2. Web Storage

Web Storage is a set of key-value pairs stored in a web browser for each origin [9]. It is divided into Local Storage (which is persistent) and Session Storage (where data is stored for the lifetime of a web browser window). A web application can only store strings. It means that in order to save JavaScript object, they have to be serialized (converted to JSON format) before. Stringifying objects to JSON can be time consuming so developers should be aware of that. It is particularly important because calls to WebStorage API are synchronous.

3.3. IndexedDB

IndexedDB is a database, which is built in a web browser [10]. It can store more types of data than the Web Storage because it can handle every object that can be serialized by "structured clone" algorithm. One origin may have many databases (identified by a name and a version) and each one may have many data stores, so that one store has its own definition of a key and is designed to store similar objects.

Similarly to the WebStorage, data in IndexedDB's store is represented by a pair of a key and a value. The key can be automatically generated or set programmatically (in fact, the key is also one of the stored objects attributes). Obviously, the keys must be unique across a store.

IndexedDB has two APIs – asynchronous and synchronous one. It is recommended to use the asynchronous API from JavaScript code of the web application as it can work on quite big data sets. Executing JavaScript code can hang a web browser card or even the whole window (depending on how a given web browser is implemented).

As a great feature of IndexedDB, a developer can define indexes for chosen attributes for a selected store. As in regular databases, this improves performance when searching for entities. What is more, indexes can be used as constraints thus guarding attribute uniqueness.

IndexedDB is transactional. Every operation on its data is performed in a scope of a transaction. A developer can choose between two types of a transaction – read-only and read-write – depending on a type of data manipulation required. There is also another type of transaction – "versionchange" but it is only used during a database creation or changing its version.

It is worth mentioning that there have been some ideas of synchronizing data from a server (with an SQL database deployed) with a web client running different engines (IndexedDB, WebSQL Database, Google Gears). The authors decided to build a custom SQL interpreter [4].

An interesting summary about usage of above mentioned technologies in JavaScript can be found in [7].

3.4. File API

W3C introduced File API to manipulate files in a web browser before they are sent to a web server [11]. This approach allows some operations to be performed on files using client CPU power (thus saving server resources). Moreover, files can be read (as a binary string, text or DataURL) and their content can be saved in Web Storage or IndexedDB.

Another great function of File API is a sandboxed filesystem [12], which web application can access to operate [13] on files (create, write, read and delete) and folder. There are two types of filesystems – persistent and temporary. While using temporary filesystem, a web browser takes care about files lifetime – if a web browser needs some disk space, it can delete files without notice. But when a web application wants to use persistent filesystem it requires user permission and a user can approve or decline this request.

3.5. Browser's Support

It should be mentioned that not all of these specifications have been already announced as final. Only Web Storage has W3C Recommendation status. Indexed Database has W3C Candidate Recommendation status, whereas File API has Last Call Working Draft status (but its related specifications: File API: Directories and System and File API: Writer has only Working draft status). For this reason, not all mechanisms are supported by different web browsers [1]. Web Storage is supported by each popular modern web browsers except for Opera Mini. IndexedDB is not supported by Safari, iOS Safari, Android Browser (but will be from 4.4 version) and Opera Mini (and it is supported by Internet Explorer version 10.0 or newer). Some features of File API are supported by all browsers except for Opera Mini. However, key features like a sandboxed filesystem and CRUD operations on files in it are currently supported solely by web browsers with Webkit engine, which are Google Chrome, Opera, Blackberry Browser, Opera Mobile and Chrome for Android (and there are no signs of implementing full File API support in other web browsers in near future). The Fig. 1 summarizes availability of HTML5 features in modern web-browsers.

	WebStorage	IndexedDB	File API	FileSystems, FileWriter
Internet Explorer	8.0	10.0	10,0	-
Mozilla Firefox	3.5	10.0	3.6	-
Google Chrome	4.0	23.0	13.0	13.0
Safari	4.0	-	6.0	-
Opera	10.5	15.0	11.1	15.0
iOS Safari	3.2	-	6.0	-
Opera Mini	-	-	-	-
Android Browser	2.1	-	3.0*	-
Blackberry Browser	7.0	10.0*	10.0	10.0
Opera Mobile	11.0	14.0	11.1	14.0
Google Chrome for Android	25.0	25.0	25.0	25.0
Firefox for Android	19.0	19.0	19.0	-

Fig. 1. Compliance of HTML5 features for web-browsers available on the market

4. Web Applications on Mobile Devices

Due to specific limitations of mobile devices and development frameworks available for them majority of web applications intended for them are native applications. It is caused by unreliable access to Internet from such devices. One cannot simply assume that there will be an access to Internet all the time. On top of that there are also other situations when the devices cannot be connected to Internet, e.g. during flight. Therefore, mobile applications need to have ability to work both online and offline. However, working offline they generate a demand for storing application state on mobile devices' storage. Unfortunately, it was really hard to meet this requirement when using web browsers on mobile devices.

There are two main types of mobile applications. The first one is often called "Wireless Internet" applications. They consist of client – WWW browser and a web server connected to a data source. Application state is stored on the server. To use such an application, a user needs to establish connection to Internet on their machine and launch a web browser. There is no need to install any additional software. Developers implement only one version of a web application, which is suitable for all modern browsers be it a mobile web browsers of a desktop one.

Another type of a mobile application is "Smart Client". This is a native application, which must be downloaded and intentionally installed by a user on his device. As an advantage, such application can access device's storage and can work offline (e.g. some operations are performed offline and then are synchronized when connection is established). A native application is allowed to access a device's OS features such as geo-location, detection of available connection channel (WiFi, 3G, etc.) thus can offer more sophisticated features. A big drawback of native applications is variety of mobile platforms to be considered by application developers (i.e. iOS, Android, Windows Phone, Blackberry). An application provider has to implement many applications to cover all mobile device users.

For above reasons, there were some attempts made to speed up development process for web applications in thin client architecture for mobile devices. PhoneGap is one of them [2] [5]. This tool allows a developer to create an application using HTML, Javascript and CSS. From a developer's point of view, the process is similar to developing a standard web-based application. However, once the application source code is ready, it is compiled into a native application dedicated for each mobile platform (using based on web views controls specific for each platform). Therefore, the compiled application has to install on a device by a user and undergoes the process of validation before being placed in platform market (i.e. Play Store for Android or App Store on iOS).

Another approach is RWD (Responsive Web Design). Because it is sometimes hard to prepare a web page that looks properly on both big Full-HD monitor and relatively small mobile device screen, developers tend to choose creating native applications adapted to screen size (sometimes different software for phones and tablets). The HTML5 introduce media queries – elements of CSS that can be used by developers to adjust appearance of web page depending on web browser's window size.

It seems that, due to availability of HTML5 and above mentioned technologies, developers have great opportunities to boost their web applications in thin client architecture. Storing data on client side can be easily achieved using Web Storage, IndexedDB and File API even with relatively big data sets and binary files. Certainly, there are some types of mobile application that still have to be native, e.g. applications running as daemon but still majority can be provided as HTML5, web-based clients with client-side offline storage. These applications can run in both: online and offline modes.

5. Possible Adoption of New W3C Technologies

As W3C organization claims (and it is hard to disagree with it), there are many useful cases related to mentioned technologies to be adopted in web applications. First of all, a module that sends big sets of data from a client web browser to a web server. It splits a set of data into smaller pieces, stores them in a web browser's space and sends sequentially. If a web browser window is closed, it remembers the last sent part. After next access to the web page, remaining parts are sent as well. Without ability to store data in a web browser, in case of any transfer interruption (regardless of a reason) a data set would have to upload once again and sent from the very beginning.

Another scenario is also about sending files. Imagine that an application running in a web browser needs to download many files from a server. It is faster to transport one ZIP package than many files separately. In this case, after receiving client's HTTP request, several files can be zipped on the server and sent back as a single package. Then, it can be unpacked in the web browser. The package's content can be stored in a web browser storage space to be accessed from it (also offline).

The next case is offline graphic editor that runs only in a web browser without need to send files to a server. In addition, files may be stored in a web browser's storage space for the further manipulation. In case of mobile devices, this is an example of useful application that could successfully replace native software.

Another example is a web portal providing access to movies. Now, the portal can be streamlined with an ability to store movies in browsers, so that they can be watched when no Internet connection is available due to network problems or flight.

When using new HTML5 technologies, it is possible to implement a web-based e-mail client allowing to access e-mail content offline (including attachments that have been previously downloaded and stored in a web browser's storage space). It allows a user to use the application in the offline mode but it also decreases network load when working online. Another helpful feature useful offline is saving working copies, as well as attachments, as drafts that will be synchronized with the server once the user is back online (when Internet is reconnected items will be taken from web browser space and sent as e-mail message).

6. An Tdea of the Framework

Web applications are created in popular programming languages, such as PHP, C#, Java, Python, Ruby or many others. In fact, developers always use proven web frameworks for those languages. A large part of the market is possessed by Java and there are many popular libraries for it to help developing web applications. Most of them (Spring MVC, Struts, JSF etc.) are based on Java Server Pages specification, in which an application has its view layer built on JSP files. A JSP file can contain HTML tags and also Java code that integrates view with other layers. Of course, web frameworks have mechanisms that facilitate this type of integration and introduce their own mechanisms (for instance actions and forms definition in Struts or view controllers in Spring MVC). When using JSP files, developers can also take an advantage of previously created libraries of JSP tags (or define new ones). JSP tags can for example execute methods on a server side or place previously prepared piece of HTML code or JavaScript code.

New HTML5 technologies (Web Storage, IndexedDB and File API) can be accessed from JavaScript scripts inside HTML documents. Therefore, in web applications created with Java frameworks based on JSP technology, the best place for JavaScript code is a JSP file. As it was presented above, the new technologies have a lot of possible applications, however developers need to learn new APIs or sometimes even JavaScript language (because Java web frameworks allow a user to successfully create working applications without need to know JavaScript). It is worth noticing that both IndexedDB and File API have asynchronous API, which requires quite a different approach from web developer. It is an obstacle for the use of the technologies.

Considering the above problems, our objective was to create a Java framework that would help developers using new HTML5 client-side storage technologies. The framework should be intuitive and easy to learn and designed in a way that keeps an application code properly structured and also easy to test.

There have been several studies on usefulness of new client-side storage technologies [1], [8]. Some articles also claims that these mechanisms will be important when developing modern web applications (mobile ones as well). These papers describe ways of adopting HTML5 technologies to web applications. However, there is nothing about designing a way of propagate these technologies.

In the next section we present a design of a framework allowing Java developers to write web-based applications streamlining HTML5's offline storage feature. We will present the framework's four main elements that can be leveraged by developers.

7. Project of the Framework

Some ideas significantly guided the final process of designing our framework. First of all, the key point was to identify and group separable elements of the framework. As the result it was found that one of these elements will be JavaScript library. The library contains a set of functions, which call Web Storage, IndexedDB and File API methods in a proper, browser-specific way. What is important, the library is the only place in the framework where client-side storage technologies APIs are used. This is very essential since, JavaScript language as well as JavaScirpt library API specifications tend to change frequently. Especially when one considers that above mentioned technologies still have not been officially specified by W3C organizations and their specification is still under development. Therefore, if any element of these API changes, it is required to adapt changes only in the JavaScript library. What is more, although our framework is designed to be a Java library, JavaScript library can be also distributed as independent project and can be used with alternative web frameworks written in different programming languages. High level architecture of the solution proposed is presented in Fig. 2).

The most important element is JavaScript library html5storage.js – a set of objects and methods that maps browser specific code onto browser-independent code to be used by client applications. Another element of a framework is a Java library with a set JSP tags definitions and Java annotations that can be used by developers to annotate classes to be cached on the client side. The whole framework is distributed as a WAR package that can be used in the customer-specific Java Web applications.

Another important feature of a project is that it is easy to use by Java web developers. To make this happen, we have focused on two things. First of all, we introduced JSP tag library that contains many JSP tags to use on JSP pages. The set of tags covers all possible calls to functions from framework's JavaScript library described above. JSP tags should be well-known to Java web developers as oppose JavaScript language for HTML5. Secondly, in our framework we use a proven

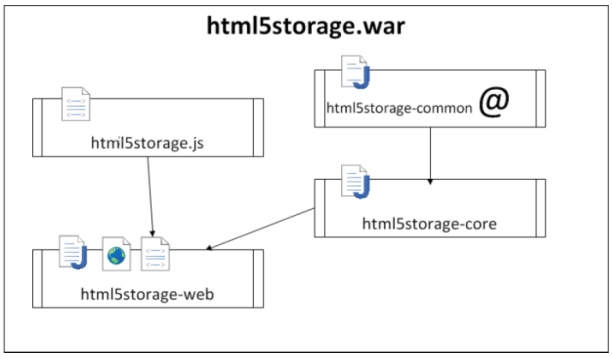

Fig. 2. High level architecture of the solution

way of defining a set of objects that can be stored in Indexed Database. We choose similar way that is used in Java Persistence API – there is configuration file with a list of fully-qualified Java classes, which instances can be stored in the database. Additionally, we introduced a set of Java annotations for both: classes and fields. They have to be used to specify classes which instances should be made storable. There are also annotations to mark fields to be stored in the database as well as an annotation to indicate a primary key and specify for which field indexes should be generated.

The most important thing about our framework is wide range of operations that can be performed in web applications that adapts it. We introduce four main elements that take advantage of client-side storage HTML5 technologies and they are briefly described in below sections.

7.1. HTTP Request Caching

In a web application clicking on a link or an input field, which has type "submit" (simply a button) causes generating HTTP Request for a particular resource identified by URL. It always works when Internet connection is available. Otherwise, an appropriate error message will be displayed in a web browser window. However, if a web application uses our framework, a user can go to offline mode. A framework will take care about caching actions that generate HTTP Request (simplifying – a resource URL, HTTP method type and/or additional parameters). These requests will be stored persistently in a web-browser (they will last even when the web-browser is closed). Once the Internet connection is reestablished, the previously stored requests can be sent to the web server.

For example, a developer can mark a HTML form as conditionally cacheable. It means that if a user works online, the form will be submitted in a standard way. Otherwise, a URL with parameters generated by the form submission will be cached and sent to the web server when the user starts working online again.

7.2. HTML Forms Caching

Another feature of this framework implementation is caching state of HTML forms. In a usual scenario, when a user has to close a web browser when filling in an HTML form the form's state will simply vanish. Using our framework it is possible to save the form's state in web browser storage. It can be done automatically by marking particular form or it can be done on-demand by the proper user's action. Form state could be saved either for web browser's tab's life or persistently. Then, the form's state can be restored (either automatically or when needed).

The framework provides following operations to developers:

- to mark a form to be cached conditionally,
- to mark a link for conditional send (if the application is offline the system will store HTTP request for the link instead of trying reloading the page),
- to add an element that will call an action that will cache a HTTP request in a web browser's cache,
- to add an element that will call an action that will call a stored HTTP request,
- add an element that will remove a cached HTTP request.

7.3. Cache Data Between Pages

The next framework's element is ability to cache data when navigating between pages. This mechanism is very similar to session or request scope data storing on server side available in popular web frameworks. A data can be stored in a web browser. Of course, it is still possible to use traditional mechanisms: HTTP cookie mechanism or URL parameters. However, in this approach, all data will be sent in each HTTP Request. And like in "HTML forms caching", a developer can specify scope of stored data and framework will obey it. But in this case, one of available scopes is a session scope, which keeps data stored until a web browser's tab is closed. Another one is a request scope, which makes data accessible only for the next loaded page.

The framework provides following operations to developers:

- to define, which data should be cached and be available after loading another page
- to load previously cached data after another page is loaded,
- to remove previously cached data,
- to define scope of data availability (a session scope or a request scope).

7.4. Access to Browser Data Space

One of the most important parts of the framework is set of tools to manipulate data stored. The framework provides many JSP tags which deal with that. Each type of operation can be done in two ways - automatically when page is loaded or after specific action of a user. From server side perspective (i.e. Java code), by different sort of data we assume simple, primitive data types, Java objects (properly annotated) and binary data (e.g. resources – music, graphics, etc. on web server).

Depending on kind of data it is stored in the most suitable way. Simple data types are placed in Web Storage under a given key. Single Java also objects can be stored in Web Storage but if their classes are listed in frameworks configuration file, framework will cre-

ate in client web browser database several stores for each class. Then one or more Java object of same class can be put in Indexed Database from Java code (using JSP tags). Developer can put in JSP iterators to access their attributes (for example to present collection data in a HTML table). In addition, iterated data can be filtered by attributes (lower and/or upper bound; strict or open; equality). This filtering can only be applied to attributes that were annotated to additionally be store indexes. All storing operations can be done in two ways. The first one is to explicitly define objects to be stored in a JSP file. The second one is to load them from URL (programmers should then provide for example dedicated Java Servlet, Spring controller, REST web service etc.). Files can be cached in browser's sandbox filesystem from given URL or from HTML5 input (type=file). What is more, there are ready tags that allow web application users to download files from sandboxed filesystem to hard disk. On top of that there are tags to display stored graphics in a web page or place audio/video control to play stored music and video files.

The framework provides following operations to developers:
- to cache data,
- to cache data from a given URL,
- to cache data from a given URL and add it to HTML document,
- to cache data from a given URL and process it,
- to read data,
- to read data and add it to HTML document,
- to remove data,
- to transfer data to server using AJAX,
- to transfer data to server using GET/POST method,
- to modify attributes of JavaScript objects previously cached.

The framework provides following operations on object collections to developers:
- to add elements to a data storage,
- to add elements to a data storage from a given URL,
- to read (or search using defined criteria) elements from a data storage,
- to read elements from a data storage and add them to HTML documents,
- to remove elements from a data storage,
- to send elements to a server using AJAX,
- to send elements to a server using GET/POST methods,
- to modify attributes of elements from a data storage.

The framework provides following operations on binary data to developers:
- to add a binary data to a virtual file system,
- to add a binary data to a virtual file system and process it,
- to add a binary data from a given URL to a virtual file system,
- to add a binary data from a given URL to a virtual file system and process it,
- to read a binary data from a virtual file system either as a text or as a data URL or as a binary string,
- to add a link to a binary data from a virtual file system to HTML document (so that a user can download it),
- to add an image/video/audio content from a virtual file system to HTML document,
- to add a file from a user's file system to a virtual file system,
- to convert image format of stored binary data,
- to apply filters, resize and modify image data stored in a virtual file system using Pixastic library [14],
- to rename, move and remove files from a virtual file system,
- to add, remove folder in a virtual file system,
- to send binary data to a server using AJAX.

7.5. Using Framework

Let's assume that we want to create Java web application using a framework that is based on JSP technology. A project will be built using Maven. As a result a WAR file will be created.

First, the dependencies needs to be defined in a pom.xml file:

```
<dependencies>
  <dependency>
    <groupId>pl.edu.agh.html5storage</
groupId>
    <artifactId>html5storage-core</
artifactId>
    <version>1.0.0</version>
  </dependency>
  </dependencies>
```

In order to use caching of HTTP requests or HTTP forms does not require any special configuration. However, in order to use caching data between pages and accessing browser data space a developer needs to annotate Java classes, which instances can be cached, and create a configuration file.

Java classes, which instances can be cached do not have to implement any specific interface nor to inherit any specific class. Neither getters/setters have to be defined. The only required modification is a Java annotation defined in pl.edu.agh.html5storage.annotations.ClientStore.

A Java class needs to have a field to be used as ID. The ID will be used as object's key for Javascript objects stored in IndexedDB data storage. A pl.edu.agh.html5storage.annotations.KeyPath annotation has to be used. A value of a key will be generated by Java code (using a code provided by a developer) or can be generated automatically. In the latter case the field has to be int or long (java.lang.Integer or java.lang.Long respectively). Additionally, ClientStore annotation needs to have autoGenerateKey attribute set to true.

All fields that should be available when caching an objects in a data storage need to be annotated by pl.edu.agh.html5storage.annotations.Cached. All non-annotated fields (except for the ID) will be ignored.

There is another annotation available pl.edu.agh.html5storage.annotations.Index that can be used to mark fields for which a system should create an index.

This is useful to speed up filtering of data o searching data storage.

Another step that needs to be taken by a developer is to prepare a configuration file (html5storage-config.xml) and place it in src/main/resources/META-INF location. The simplest configuration file is presented below:

```
<?xml version="1.0" encoding="UTF-8"?>
<html5storage>
  <db>
    <name></name>
    <version></version>
  </db>
  <classes>
    <class></class>
  </classes>
 </html5storage>,
```

db.name – a name of a database to be used on a web browser side,

db.version – a version of a database to be used on a web browser side; whenever there are any changes in Java classes whose objects will be stored, a developer needs to increate a database version,

classes.class – zero or more fully-qualified names for classes whose instances will be stored.

8. Test Cases

Browser's support for framework depends on their support for client-side storage HTML5 technologies. For this reason, some framework elements are available on every modern web client, but some are not. Hopefully it seems that process of standardization of Indexed Database and File API is progressing in a good pace. After they reach Recommendation status, all modern browser vendors will not have choice but to introduce implementation in their products.

Our framework is distributed as a Java Archive library. To use it, it must be simply added to classpath for JVM where one wants to run his/her application. The simplest way is to start new project with Maven and add framework dependency. Then developer should create in the project html5storage-config.xml file. It contains information about database name and version, which will be created in client's web browser and list of classes, for which corresponding stores in browser database will be also created. The next step is to mark desirable classes and class fields with framework's annotations. From this point a developer can freely use JSP tags in their projects.

After finishing the first reference implementation of our framework we decided to write two sample web applications that utilizes client-side storage technologies. This was an offline graphics editor and a web-based e-mail client. The next two sections briefly describe the sample applications.

8.1. Offline Graphics Editor

The first application we built to validate the framework was web-based GUI editor. The application is capable of working offline. A user is allowed to upload an image from web or from local file system. Then, an image can be processed, different image-filters may be applied to it. Then, the modified image can be stored on local file system. Once a file is uploaded to a client it is stored on the client's side (File API and WebStorage) and can be processed offline. Even though there is no Java code executed on the server side (which means that the application executes in web browser only) it was written using the JSP tags and other elements delivered by the framework.

8.2. Web Based E-mail Client

The second application is an e-mail client that runs in a web browser. We focused on storage-side of the implementation thus, the IMAP/POP3/SMTP part of it is just a mock up because handling e-mail protocols was negligible in that case. We only provided server methods required to imitate e-mail sending and receiving. The e-mail client presents data stored in a web browser (previously received e-mail messages with attachments), therefore it can work offline. It also implements mechanisms to synchronize its state with the server (e.g. receive new messages). This application can work both offline and online depending on client's needs and access to Internet. All operations executed by a user when offline are automatically synchronized with the server when online again.

Let's consider pl.emailclient.Message class that represents a single e-mail message. We want ID to be a message identifier. An identifier will be assigned by email server rather than the framework. We also predict the need for searching by email subject, sender and recipient. Considering the above a Java class can be defined in a following way:

```
package pl.emailclient;
@ClientStore
public class Message {
  @KeyPath
  private long id;

  @Index
  @Cached
  private String title;

  @Cached
  private String content;

  @Index
  @Cached
  private String sender;

  @Index
  @Cached
  private String receiver;

  @Cached
  private List<String> attachements;
 }.
```

The simplest configuration file is following:

```
<?xml version="1.0" encoding="UTF-8"?>
<html5storage>
  <db>
    <name>test_project</name>
    <version>1</version>
  </db>
  <classes>
    <class>pl.emailclient.Message</class>
  </classes>
</html5storage>.
```

9. Conclusion and Future Works

We researched powers of new HTML5 client-side storage technologies and found them really helpful in boosting capabilities of web applications. We also tried to show alternatives to making several native mobile applications representing the same product for different mobile platforms – it yet can be done in a thin client architecture streamlining HTML5 storage capabilities. We designed framework that integrates Java web framework based on JSP technology (which nowadays takes significant part of web market) with Web Storage, Indexed Database and File API. We also tried to proof usefulness of our work by implementing two sample applications that operates on web browser client-side storage thanks to our framework.

There is much space to improve our solution. It is obvious that this is a tool made for developers, so it is important that it meets as many theirs requirements as possible as well as needs to be ease use. It was really difficult to predict all scenarios. Furthermore, when we started to implement sample applications we discovered that we also need to add additional functionality to framework to fulfill our needs. We also already have agreed that there are some places where framework can be changed to be easier to use. For example, our plans is to abandon configuration file and base all setup on Java annotations and Java reflection.

Finally, there are some cases that we really need to work on accroding to specific environment that web application is. We live in times that one person often has one or more device (PC, notebook, tablet, smartphone etc.). However, a situation that two different people uses same machine as well as same browser is also very common. Web Storage, Indexed Database or File API do not provide any mechanism that can divide browser space for different user (yet). We must think on suitable solution for that as it is really important in terms of security and privacy.

AUTHORS

Arkadiusz Janik* – AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. Mickiewicza 30, 30-059 Kraków, Poland, E-mail: arkadiusz.janik@agh.edu.pl

Szymon Kiebzak – AGH University of Science and Technology, Faculty of Computer Science, Electronics and Telecommunications, Department of Computer Science, al. Mickiewicza 30, 30-059 Kraków, Poland, E-mail: s.kiebzak@gmail.com

*Corresponding author

REFERENCES

[1] Onyedikachi Jeffrey Anyansi, Lonergan, Ian Patrick, *Oracle HTML5 Rich Web Application*, Worcester Polytechnic Institute, 2012.
[2] Cha Si-Ho, Yeomun Yun, *Smartphone Application Development using HTML5-based Cross-Platform Framework*, Chungwoon University, 2013.
[3] Ram Sandilya K., Sudheer R., Sreenivas V., Kiran K.V.D., "Effective Use of HTML5 for Developing Offline Web Applications", IIJAR, 2013, 80–87.
[4] Leblon R., "Building advanced, offline web applications with HTML5", UPC-Barcelona Tech, Ghent University, 2010.
[5] Hasan Y., Mustafa Zaidi M., Haider N., Hasan W.U., and Amin I., "Smart Phones Application development using HTML5 and related technologies: A tradeoff between cost and quality", *International Journal of Computer Science Issues* (IJCSI), vol. 9, issue 3, May 2012, 455.
[6] West W., Monisha Pulimood S., "Analysis of privacy and security in HTML5 web storage", *Journal of Computing Sciences in Colleges Archive*, vol. 27, issue 3, January 2012, 80–87.
[7] Laine M., "Client-Side Storage in Web Applications", Aalto University, Technical Report, 2012.
[8] Ijtihadie R.M., Chisaki Y., Usagawa T., Bekti Cahyo H., and Affandi A., "Offline web application and quiz synchronization for e-learning activity for mobile browser", TENCON 2010, November 2010, 2402–2405. DOI: 10.1109/TENCON.2010.5685899.
[9] http://www.w3.org/TR/webstorage/.
[10] http://www.w3.org/TR/IndexedDB/.
[11] http://www.w3.org/TR/FileAPI/.
[12] http://www.w3.org/TR/file-system-api/.
[13] http://www.w3.org/TR/file-writer-api/.
[14] http://www.pixastic.com/.