UNIVERSITY OF GOTHENBURG

# Comparison of IndexedDB and SQLite Based on Developers' Concerns

Bachelor of Science Thesis in Software Engineering and Management

*SORUSH AREFIPOUR*

*MASSIH MOZAHHEBI*

**Comparison of IndexedDB and SQLite Based on Developers' Concerns**

SORUSH AREFIPOUR
MASSIH MOZAHHEBI



Examiner: MORGAN ERICSSON

University of Gothenburg
Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

# Comparison of IndexedDB and SQLite Based on Developers' Concerns

*Sorush Arefipour*
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
sorush@student.gu.se

*Massih Mozahhebi*
Department of Computer Science and Engineering
University of Gothenburg
Gothenburg, Sweden
gusmozse@student.gu.se

***Abstract*— The recent development of web applications has caused companies and developers to have more alternatives while choosing a new solution. The aim of this study is to compare IndexedDB, the local storage of the latest web applications' standard "HTML5", with Android SQLite. This comparison is designed to clarify the position of web application against native application in the field of local data storage according to developers' concern that are obtained through exploratory interviews and Volvo Group Target Architecture (VGTA). The investigation is conducted using mixed method approach and for data collection literature review and prototype testing were used; also we utilized thematic analysis along with descriptive analysis for data analyzing.**

***Keywords: Android, HTML5, IndexedDB, SQLite, Security, Performance***

## I. INTRODUCTION

There has always been a competition between native and web applications due to the cross-platform nature of web applications and access of native application to operating system's features. However, being cross-platform would be a great advantage in web application but developers do not neglect web applications' weaknesses because of this advantage. One of the weaknesses of web applications that developers are dealing with is local data storage, which would lead them to use native applications (Pilgrim, 2010). Recently, HTML5 has introduced a new local data storage feature, which is known as IndexedDB, to address developers' needs for local data storage in web applications (Mozilla, 2013).

The purpose of this thesis is to compare HTML5's IndexedDB and Android SQLite databases based on software developers' concerns and Volvo Group Target Architecture (VGTA). These concerns were obtained through exploratory interviews, where according to our interviewees performance and security were the most important concerns. A mixed approach of qualitative and quantitative research was chosen to cover these concerns (Creswell, 2009). Between the two popular mobile operating system, Google Android and Apple iOS, we chose Android since Apple iOS does not support IndexedDB at the time of this research. Since, Android ships with SQLite (Android Developers, 2013), therefore, we chose SQLite to compare with IndexedDB. Our contribution to the community is to determine how well web applications' local storage compete with native application's local storage and its shortcomings and strengths. The research questions of this study are:

1. What concerns do developers have for choosing a local storage solution?
   1.1. How is the performance of IndexedDB compared to SQLite?
   1.2. How is the security of IndexedDB compared to SQLite?

This thesis is structured as follows; Section 2 represents theoretical background to familiarize readers of this paper with technologies that was investigated in this thesis. Section 3 represents research method, where the chosen approaches and their processes are explained explicitly. In Section 4, results of both qualitative and quantitative approaches are demonstrated. Following to that, in Section 5 results are being discussed and finally, Section 6 will represent conclusion and outlook of the thesis.

## II. THEORERTICAL BACKGROUND

This section outlines theoretical background about the two databases, which will be compared in this research. In the following, characteristics, features, and shortcomings of IndexedDB, in Subsection 2.1, and SQLite, in Subsection 2.2, will be discussed. In addition, Subsection 2.3 will describe VGTA document briefly.

### *A. IndexedDB*

IndexedDB (formerly WebSimpleDB) is an API for client-side data storage, which is able to store structured data. In 2009, Oracle proposed IndexedDB as new web browser's standard interface for local databases (Oracle Corp, 2013). In this type of database, data is stored as a key-value pair but keys can be referred as properties of objects that are stored in the values.

In general there is no limitation on capacity of IndexedDB but every browser has different policy on capacity. As an example, FireFox does not force any limitation, however it asks for permission for storing files bigger than 50MB (Mozilla, 2013). Also, Google Chrome

lets web application to use 20% of shared storage pool[1], though it is not possible to query for more space (Google Developers, 2013).

IndexedDB is an object-oriented NoSQL database, which sets it apart from most traditional relational databases. NoSQL databases are schemaless, where saving and retrieving is not following traditional structured query language (SQL). Moreover, IndexedDB is not a relational database (RDBMS), where tables contain rows and columns (table-oriented). In fact, it is an object database (ODBMS[2]) that represents and treats data in the form of objects, which is called "objectstore"(Mozilla, 2013).

As shown in Figure 1, ODBMS eliminates process of copying and translating data between programming languages and databases.

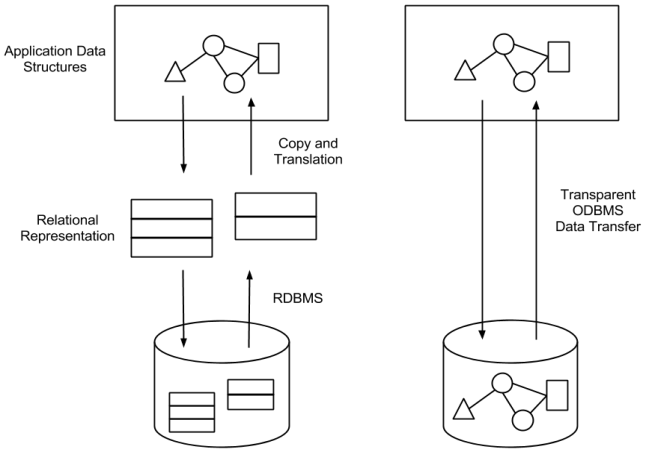


Figure 1. RDBMS vs. ODBMS (Cattell and Barry, 1997)

IndexedDB API can be used via JavaScript in order to enter, retrieve, remove, and update data. This API is using DOM[3] events to provide status of database operations. All primitive JavaScript data types (String, Date, etc.) and hierarchical objects such as JSON[4] are supported as an entry in this API (Laine, n.d.). In addition to primitive data types, IndexedDB is using BLOB to save and retrieve files including images, videos, etc. A BLOB represents a file-like object of immutable, raw data. BLOB represents data that isn't necessarily in a JavaScript-native format (Mozilla, 2013).

IndexedDB is still considered as a draft version (W3C, 2013), however, it is expected that easy to use libraries will be built on top of the API. According to Mozilla developer network (2013), IndexedDB is missing a few features such as:

- **Internationalized sorting:** Not all languages sort strings in the same way; so internationalized sorting is not supported.
- **Synchronizing:** The API is not designed to take care of synchronizing with a server-side database.
- **Full text searching:** The API does not have an equivalent of the LIKE operator in SQL.

IndexedDB is built on a transactional database model (Kimak, Ellman and Laing, 2012). Transactional database model prevents overwrite of data or interference of two or more executing processes on the same resource. This prevention happens in situations like when a user opens two instances of a web application in different tabs simultaneously, and applies distinct modification on the same resource (Mozilla, 2013).

### *B. SQLite*

SQLite is an open source embedded relational database. It is referred as an embedded database, since it symbiotically coexists inside the application it serves rather than a standalone process (Owens, 2006). SQLite is licensed as "Public Domain", where there is no ownership or copyright rule in this type of license and it is free to be used in proprietary and free softwares (GNU, 2013). The size limitation of a SQLite database is 140 Terabytes (SQLite, 2013). Kreibich (2010) describes some of the significant features of SQLite as:

- **Serverless:** SQLite does not need a separate server process or system to operate. The SQLite library accesses its storage files locally or on a server.
- **Zero Configuration:** Since no server is needed, thus no configuration needs to be done between server and client. Creating an SQLite database instance is as easy as opening a file.
- **Cross-Platform:** The entire database instance resides in a single cross-platform file, requiring no administration.
- **Self-Contained:** A single library contains the entire database system, which integrates directly into a host application.
- **Small Runtime Footprint:** The default build is less than a megabyte of code and requires only a few megabytes of memory. With some adjustments, both the library size and memory use can be significantly reduced.
- **Transactional:** SQLite transactions are fully ACID [5] -compliant, allowing safe access from multiple processes or threads.

1 Shared storage pool can be up to half of free memory of a device

2 Object-oriented database management system

3 Document Object model is a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents (W3C, 2013).

4 JavaScript Object Notation

5 ACID (Atomicity, Consistency, Isolation, Durability) assures database's transactions to be processed reliably (USING SQLite).

SQLite inherits most of SQL features that are listed in SQL-92 standard (IBM, 1992), however it lacks features such as: RIGHT and FULL OUTER JOIN, Complete ALTER TABLE support, Complete trigger support, Writing to VIEWs, GRANT and REVOKE (SQLite, 2013). These concepts are not in the scope of this thesis, and complementary information can be found on SQLite official website.

### C. Volvo Group Target Architecture

Volvo Group has documented ten architectural principles and obliges developers and designers to follow these principles in order to increase the quality of their products. These principles are as follows:

*1) Conformity to standards:* Drive usage of open and industry standards at Volvo.

*2) Autonomous and loose coupling, between components and applications:* Flexible subsystem and granular component setup, avoiding monoliths.

*3) Simplicity in solutions and work methods:* Clean solutions from technical, application and user perspective.

*4) Strive for usage of existing Volvo services:* Whenever possible avoid application specific infrastructure and instead use already existing services at Volvo.

*5) Robust solutions:* Strive for robust solutions securing uptime.

*6) Performance focus from the start:* Strive for good performance in solutions from the start.

*7) Secure solutions:* Strive for secure solutions from the start.

*8) Good integration solutions:* Follow Volvo Group integration policies and guidelines.

*9) Usage of Agile work methods and design principles:* Use Agile system development and implementation principles.

*10) Maintainable solutions:* Deliver maintainable solutions to Maintenance.

## III. Research Method

This research was an empirical investigation solicited by Volvo IT. In spring 2013, the company had a plan to perform a wide research around the capability of web applications using HTML5. This project was divided into smaller subprojects to cover multiple aspects of using HTML5 to develop web applications. One of the subprojects was analyzing IndexedDB to determine if it can fulfill the company's requirements according to VGTA document. For this mean, we decided to compare IndexedDB with SQLite.

In order to determine our comparison's criteria, we decided to find out mobile application developers' concerns for choosing a database. These concerns were obtained by conducting exploratory qualitative interviews (Creswell, 2009), where interview questions were designed based on VGTA. Following to that, we evaluated these two databases based on these concerns.

According to the conducted interviews, the important quality attributes for developers were performance and security. In fact, the common point of view among all interviewees implied that performance plays an important role while choosing any solution especially in enterprise systems. As a matter of fact, our interviewees considered performance as system's response time. In addition, one of Volvo IT's software architect mentioned his own experience about security issues of some solutions that they were dealing with, which forced them to stop using those solutions to protect their credential data. Therefore, we based our comparison on performance and security of the two databases. For this mean, we chose quantitative research method (Creswell, 2009) to measure performance of SQLite and IndexedDB statistically. On the other hand, we chose qualitative approach to compare security of these two databases, in order to obtain deep understanding of underlying causes of the differences between the two databases.

### A. Data Collection

#### 1) Exploratory Interviews

Interviews were semi-structured, which means that some questions were prepared in advance. However, some improvisational questions were asked when there were vague points during interviews. We chose interviewees based on their level of experience to cover different aspects of our research. Therefore, we chose two junior software developers with less than one year of experience, a senior software developer with 5 years of experience, and one software architect with more than 25 years of experience in the field of software developments. Interview questions are available in Appendix.

#### 2) Measuring Performance

To collect data for indexedDB and SQLite's performance comparison, we decided to perform different benchmarks on both DBs. These benchmarks were aimed to measure the amount of time, which is required to perform specified number of queries on both databases. For this mean, we developed two prototypes, a native Android application using SQLite database and a web application developed in HTML5 using IndexedDB. Both prototypes are available at

website: http://web.student.chalmers.se/~seyedma/indexeddbspeedtest.

Our benchmarks were based on CRUD (create, retrieve, update, delete) operations of databases. In addition, we considered indexing, which can have impact on benchmark's results, to achieve more reliable results. All test queries were applied on simple tables and objectstores of the both databases, where no joining or sub-queries were involved. The data that we used to perform the benchmarks was JSON and XML text files, which their size would not exceed 1 megabyte. Although, all benchmarks were performed on small amount of data and cannot measure how well these databases can handle large scale data but they can provide guidelines and set basic operational expectations (Oracle, 2006). In order to generate data for our benchmarks, we used Databasetestdata.com website.

The two host platforms for our benchmarks were:

1. LG Nexus 5 phone with a Quad-core 2.3 GHz Krait 400 CPU, 2GB RAM, 16GB Internal memory, running Android 4.4.2 (Kitkat).

2. ASUS Nexus 7(first generation) tablet with a Quad-core 1.2 GHz Cortex-A9 CPU, 1Gb RAM, 32Gb Internal memory, running Android 4.4.2 (Kitkat).

Both devices were using default configuration without any optimizations. All background processes and applications were terminated to prevent I/O speed manipulation, and to get more precise results during all tests. IndexedDB benchmarks were executed on the latest version of Google Chrome browser (version 31.0.1650.59), which was the latest version by the time of benchmarking. Also, SQLite version 3.4.0 was used for SQLite benchmarks.

*3) Literature Review on Security*

Data collection for security was conducted using literature review technique to identify the previous works, which had been done in this field. According to Creswell (2009), literature review becomes a basis for comparing and contrasting findings of a qualitative study. To collect our sources for literature review, we searched through digital libraries to find relevant articles, books, and journals. Also, since IndexedDB is introduced recently and there are not adequate sources in this area, we also used blogs, and bug reports. Table I presents the searched libraries and number of found articles from each.

| Sources | Related | Selected |
|---|---|---|
| **IEEEXplore** | 16 | 9 |
| **ACM** | 11 | 5 |
| **Other** | 18 | 12 |

TABLE I. QUANTITY OF PAPERS AND BOOKS FOUND AND SELECTED.

The following are the search terms that we used for collecting related resources:

"HTML5", "Android", "IndexedDB", "SQLite", "Security", "Relational database", "Object-oriented database", "IndexedDB" AND "Security", "SQLite" AND "Security", "IndexedDB" AND "Vulnerabilities", "SQLite" AND "Vulnerabilities".

## B. Data Analysis

*1) Quantitative Data Analysis*

We used descriptive statistics to analyse our captured data through benchmarking. Babbie (2009) states, "bivariate analysis is not only simple descriptive statistical analysis, but also it describes the relationship between two different variables". Since there were two variables included in our tests, which are "Time" and "Number of queries", bivariate statistical analysis was suited for our purpose to summarize and represent our collected data on graphs.

*2) Qualitative Data Analysis*

The collected data, in qualitative part, was analyzed using thematic analysis. Braun and Clarke (2006) describe thematic analysis as "a method for identifying, analyzing and reporting patterns (themes) within data". After collecting raw data through literature review, all data was reviewed several times in order to extract patterns, which are known as codes. Following to that, common concepts of the extracted codes with their linked data were identified and reviewed to determine suitable themes for them. If at anytime during data analysis a new code were merged, we did not start over our analysis and treated the merged code separately or if it were possible, we would include it in one of the existed themes.

# IV. RESULT

## A. Performance Measurement

In this subsection, we present result of benchmarks on both SQLite and IndexedDB databases. For this mean, we divided our results according to each basic operation of a database, which are "insert", "select", "update", and "delete". Benchmark of each operation is represented on two distinct graphs, where one is dedicated to an unindexed table/objectstore and the other one is dedicated to an indexed table/objectstore. In fact, both prototype used two tables/objectstores, which have the same structure and consist of three string attributes, but in one of them two attributes were used as "index" in the table/objectstore.

Each graph is a relation of number of queries and the amount of time to execute them, where time is measured in millisecond. Execution time in all graphs is average of performing each benchmark ten times. We performed each benchmark with three different number of queries that are 1000, 3000, and 10,000. All benchmarks are performed on two hosts for both databases, therefore, every graphs consists

of twelve bars. However, processing 10,000 entries in local databases of portable devices is rare, but we considered this amount to examine both databases on high load of data.

There are different performance tests to measure performance of a database but to conduct a performance comparison on IndexedDB and SQLite, which have different structures, we selected those that are applicable on both databases. In addition, the same database's schema was used in both developed prototypes to have an equal testing environment.

**Performance test 1: Insert**

In this performance test, the time of inserting set of tuples into a table or an objectstore is being measured. Less execution time indicates higher performance.

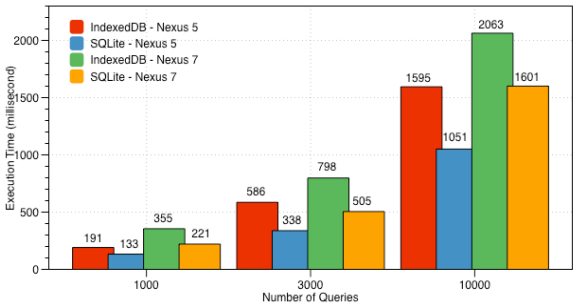


Figure 2. Unindexed insert benchmark

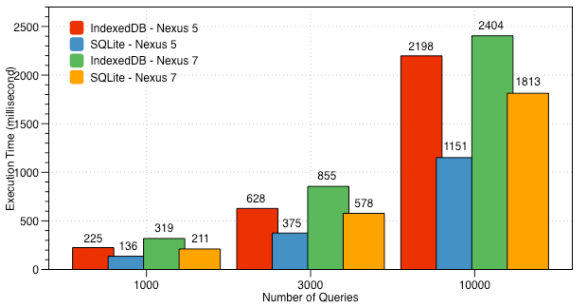


Figure 3. Indexed insert benchmark

**Performance test 2: Select**

In this performance test, selection criteria segregates 100 tuples on each iteration. For instance, in a 1000 entry table/objectstore, this test iterates 10 times and on each iteration it selects 100 tuples to have a full scan of the table or the objectstore. Less execution time indicates higher performance.

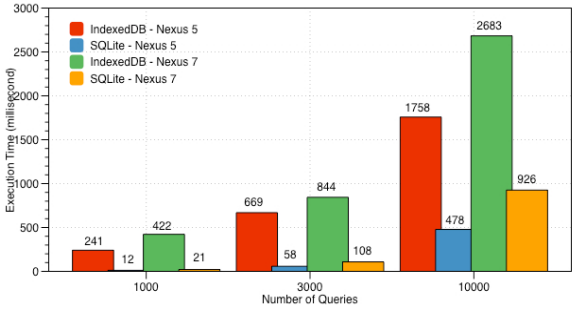


Figure 4. Unindexed select benchmark

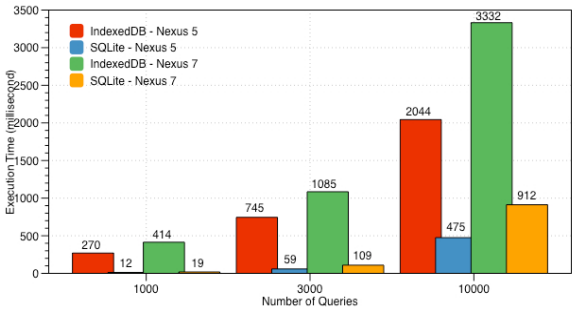


Figure 5. Indexed select benchmark

**Performance test 3: Update**

This performance test updates one non-index attribute of all 100 tuples on each iteration. For instance, in a 1000 entry table/objectstore, this test iterates 10 times and updates one attribute of all 100 tuples, to update all tuples of a table or an objectstore. Less execution time indicates higher performance.

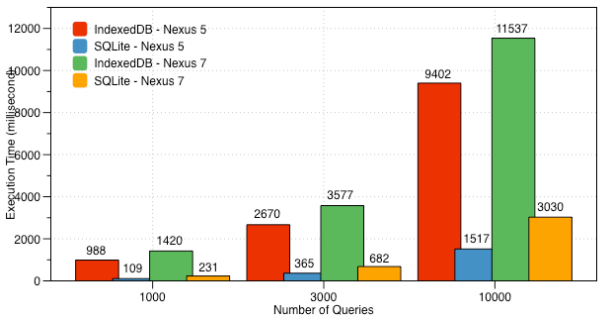


Figure 6. Unindexed update benchmark

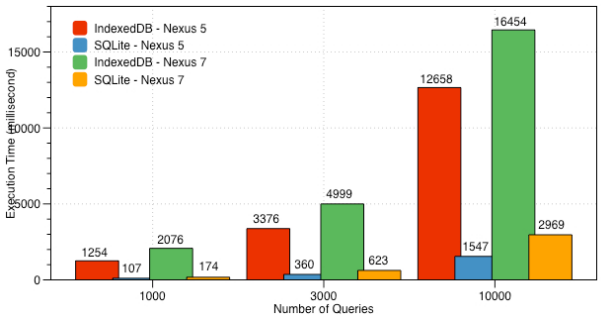


Figure 7. Indexed select benchmark

**Performance test 4: Delete**

This performance test deletes 100 tuples from a table/objectstore on each iteration. For instance, in a 1000 entry table/objectstore, this test iterates 10 times and deletes 100 tuples on each iteration, to delete all entries of a table or an objectstore. Less execution time indicates higher performance.

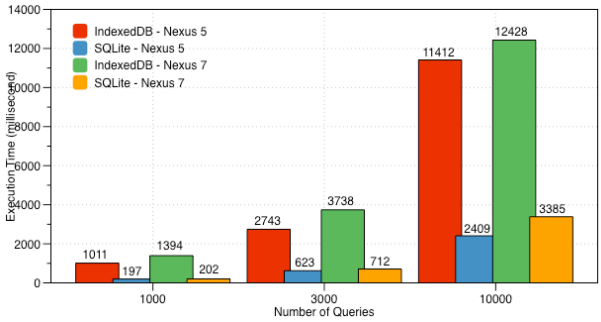


Figure 8. Unindexed delete benchmark

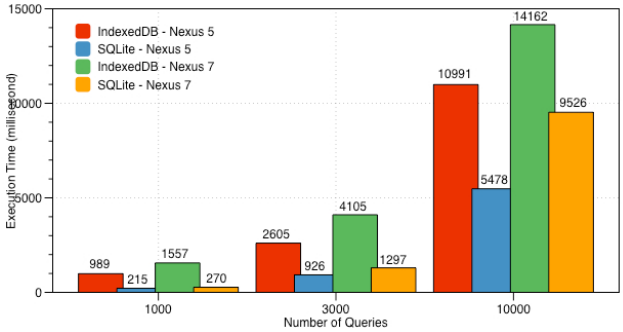


Figure 9. Indexed delete benchmark

## B. *Security Evaluation*

There are number of security features available such as: access control, authentication, data encryption, audit, and input validation in order to protect data of a database from security vulnerabilities (Kimak, Ellman and Laing, 2012; Okman, et al., 2011; Liu and Gong, 2013). In the following, comparison of IndexedDB and SQLite for each feature is brought.

### 1) *Access Control*

In database context, access control specifies the privileges, which each user of a database has, to access to specific set of data or actions (Bertino and Sandhu, 2005). There are different models of access control such as: Discretionary access control (DAC), where the model is governing the accesses of a subject to data based on the subject's identity and authorization rules, Mandatory access control (MAC), which regulates accesses to data by subjects on the basis of predefined classifications of subjects and objects in the system, and Role-based access control (RBAC), where in this model subjects are having different accesses to data according to their role in a system (Bertino and Sandhu, 2005).

Our investigation indicates that there is no form of access control in SQLite and the whole database is stored in a single file, which can be accessed and modified by anyone who has READ/WRITE permission in the system (Liu and Gong, 2013). On the other hand, although, IndexedDB does not have any access control mechanism but it is utilizing Same-origin-policy (SOP), which is a feature of web browsers to prevent access to data from other domains than its original one (Kimak, Ellman and Laing, 2012). However, Saiedian and Broyle (2011) state that the SOP is not the correct security mechanism and requires redesign to meet the access control requirements of Web-based assets but it is the only security mechanism in web browsers against potential security threats (Kimak, Ellman and Laing, 2012).

### 2) *Authentication*

Authentication is any process by which someone is allowed to go where they want to go, or to have information that they want to have (Needham and Schroeder, 1978). Authentication usually requires user's username and password to give the requested privileges to the user.

Neither SQLite nor IndexedDB have implemented authentication mechanism, therefore, in order to increase the security of stored data, it is suggested to implement authentication on the application layer to prevent access of unauthorized parties to a database.

### 3) *Encryption*

Encryption is the process of encoding data in order to prevent unauthorized parties from reading the data (Barrett and Silverman, 2001). Encryption process needs a KEY to perform encryption and decryption on data, where the key should be kept secure. In an application, data can be encrypted on database layer and be encoded and decoded respectively during storing and retrieval or on application layer.

According to our research, SQLite does not have any encryption implementation within its source code but there is an interface reserved for encryption (Liu and Gong, 2013). Furthermore, SQLite website introduces SQLite Encryption Extension (SEE) as a library for encrypting data on database level. In addition to SEE that is official encryption implementation of SQLite, there are number of encryption libraries available for SQLite such as: SQLCipher, SQLiteCrypt, wxSQLite, etc. On the other hand, IndexedDB also lacks data encryption within its source code. One of the suitable and easy to use JavaScript encryption libraries that we suggest is "Stanford Javascript Crypto Library".

### 4) *Audit*

Database auditing is the process of recording and monitoring users' actions to prevent or stop any security threads (Stephens, 2009). The same as authentication, audit is not implemented in the both databases; however, it can be implemented on application layer. Also, it is possible to use logging system of OS, if such a feature is available.

*5) Input Validation*

Input validation or data validation is the process of validating all the input data before an application using it (Kimak, Ellman and Laing, 2012). No or inefficient input validation can lead to "Code Injection", where attackers insert a piece of code into an application to change the behaviour of the application (Kimak, Ellman and Laing, 2012). SQL injection is one of the common types of code injections.

SQLite introduces "check constraint" as an approach to validate data before storing it in a database (Kreibich, 2010). Check constraint is attached to column definition or specified as a table constraint. On the other hand, IndexedDB is not utilizing any form of input validation and it is threatened by code injection. In order to mitigate code injection vulnerability, we suggest implementing input validation on application layer.

Table II summarizes our findings about IndexedDB and SQLite databases' security and provides some recommendations.

| Security Technique | IndexedDB | SQLite |
|---|---|---|
| Access Control | Not implemented - Utilize SOP | Not implemented |
| Authentication | Not implemented - Should be implemented on application layer | Not implemented - Should be implemented on application layer |
| Encryption | Not implemented - External libraries available | Not implemented - Official and external libraries available |
| Audit | Not implemented | Not implemented |
| Input Validation | Not implemented - Should be implemented on application layer | Check Constraint |

TABLE II. SECURITY FEATURES OF INDEXEDDB AND SQLITE

## V. DISCUSSION

In this section we discuss the result of our literature findings and benchmarks and map them to developers and Volvo Group's concerns that are brought in VGTA document.

According to our benchmarks, IndexedDB performed slower in all of the defined performance tests. Both SQLite and IndexedDB executed "insert" queries almost in the same amount of time and the differences between them were neglectable, however, the difference slightly increased in 10,000 entries. SQLite performed "select" queries significantly faster compared to IndexedDB. Interestingly, the execution time of selecting in 1000 entries is approximately 20 times faster in SQLite but the difference is decreasing in 3000 and 10,000 entries. Also, SQLite has shown a better performance for "update" and "delete" queries and the difference in all three data sets is tangible. From security perspective, both databases are closely at the same level, with IndexedDB taking advantage of SOP mechanism and SQLite having built-in "input validation". However, it should be considered that IndexedDB API is still under development and it is expected to improve in the future.

Our overall results indicate that for a large corporation like Volvo Group, which usually works with enterprise systems, IndexedDB is not recommended because of its low performance. In addition, security of both databases cannot address Volvo Group's security requirements, especially while dealing with credential data. However, IndexedDB is an applicable solution to those applications, where security and performance of them are in low priority and structure of the database is needed to be loos.

### *A. Threats to Validity*

The result of our research can be affected or biased by factors such as: researchers' background, collected data, etc. Here we discuss four threats we identified and how we managed to mitigate them.

*1) Raw data selection*

Raw data selection should be ensured to be unbiased. We aimed to collect all type of related data without assigning any restriction to them such as: date of publication, number of pages, and etc. For all the papers, abstracts and conclusions were reviewed to ensure if they are relevant to our topic. Moreover, all selected books were partially read to select suitable ones for our research.

*2) Missing data*

However, we put our full effort to collect all available data, but there are still missing resources in our literature review data pool. To minimize this threat, data collection was done iteratively, even while analyzing captured data.

*3) Prototypes optimization*

Unoptimized prototypes would effect on the result of our benchmarks. In order to lower the risk of this threat, pair-programming technique was used in development phase and all codes were reviewed to find possible flaws.

*4) Unreliable measurements*

The results of benchmarks in both prototypes are dependent on background processes of their host environments such as: Android memory controller, Java

garbage collector, Google Chrome JavaScript engine, and etc. To cope with this threat, we performed each performance test ten times and calculated the average.

## VI. Conclusion and Outlook

HTML5 is a recent technology, where few researches have been conducted in this area. In this investigation, the aim was to compare HTML5's local data storage mechanism "IndexedDB" with "SQLite" on Android devices. This comparison was performed based on Volvo Group and developers' concerns while choosing a solution for local data storage, where these concerns were captured through exploratory interviews. This study has found that IndexedDB is noticeably slower in performance and at the same security level compared to SQLite. However, different solutions are being chosen according to projects' requirements, therefore, IndexedDB would fit to those projects, where performance is not the main concern.

The outlook of this research is to improve the quality of the research by creating more optimized prototypes, and defining more consistent performance tests along with collecting more data about security of both databases, since additional research is needed to understand security issues in depth. Also, the existing data would be updated by release of any new version of the databases.

## Contribution

The main contribution is the comparison of IndexedDB and SQLite, in order to determine if web applications can compete with native applications in the field of local data storage, which shows that currently, SQLite is the preferable local storage.

## Acknowledgement

We would like to appreciate the effort of our academic supervisor Eric Knauss and also our industrial supervisors Micael Andersson and Mohamed Seifeddine who helped us to make this research possible and also thanks the interviewees who gave us their valuable time.

## Refereces

Anon, 2013. android.database.sqlite | Android Developers. [online] Available at: <http://developer.android.com/reference/android/database/sqlite/package-summary.html> [Accessed 8 Dec. 2013].

Anon, 2013. Authentication and Authorization - Apache HTTP Server. [online] Available at: <http://httpd.apache.org/docs/2.2/howto/auth.html> [Accessed 2 Dec. 2013].

Anon, 2013. Basic concepts - IndexedDB | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/IndexedDB/Basic_Concepts_Behind_IndexedDB> [Accessed 13 Nov 2013].

Anon, 2013. Blob - Web API reference | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/Web/API/Blob> [Accessed 2 Nov 2013].

Anon, 2013. Categories of Free and Nonfree Software - GNU Project - Free Software Foundation. [online] Available at: <https://www.gnu.org/philosophy/categories.en.html#PublicDomainSoftware> [Accessed 15 Dec. 2013].

Anon, 2013. Implementation Limits For SQLite. [online] Available at: <http://www.sqlite.org/limits.html> [Accessed 8 Nov. 2013].

Anon, 2013. IndexedDB | MDN. [online] Available at: <https://developer.mozilla.org/en-US/docs/IndexedDB> [Accessed 5 Nov. 2013].

Anon, 2013. Indexed Database API. [online] Available at: <http://www.w3.org/TR/IndexedDB/> [Accessed 23 Nov 2013].

Anon, 2013. Managing HTML5 Offline Storage - Google Chrome — Google Developers. [online] Available at: <https://developers.google.com/chrome/whitepapers/storage#temporary> [Accessed 2 Dec 2013].

Anon, 2006. Oracle Berkeley DB: Performance Metrics and Benchmarks. [pdf] Redwood Shores: Oracle. Available at: <http://www.oracle.com/technetwork/products/berkeleydb/berkeley-db-perf-128909.pdf>[Accessed 14 Nov. 2013].

Anon, 2013. SQL Features That SQLite Does Not Implement. [online] Available at: <http://www.sqlite.org/omitted.html> [Accessed 15 Dec. 2013].

Babbie, E. R., 2009. The Practice of Social Research. 12th ed. Unknown: Wadsworth Publishing.

Barrett, D. J. and Silverman, R. E., 2001. SSH, the Secure Shell: The Definitive Guide. Sebastopol: O'Reilly.

Bertino, E., and Sandhu, R., 2005. Database Security, Concepts, Approaches, and Challenges, IEEE Trans. Dependable and Secure Computing, 2(1), pp.2-19.

Braun, V. and Clarke, V., 2006. Using thematic analysis in psychology. Qualitative Research in Psychology. 3(2), pp.77-101

Cattell, R.G.G., Barry, D.K. eds., 1997. The object database

standard: ODMG 2.0. San Francisco: Morgan Kaufmann Publishers.

Creswell, J. W., 2009. Research design: Qualitative, Quantitative, and Mixed Methods Approaches. 3rd ed, Sage Publications: London.
Kimak, S., Ellman, J., & Laing, C., 2012. An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention. Liverpool: PGNET.

Kreibich, J.A., 2010. Using SQLite. Sebastopol: O'Reilly.

Laine, M., n.d. Client-Side Storage in Web Applications. [pdf] Aalto: Department of Media Technology, Aalto University. Available at: <http://media.tkk.fi/webservices/personnel/markku_laine/client-side_storage_in_web_applications.pdf> [Accessed 16 Nov. 2013].

Liu, H., and Gong Y., 2013. Analysis and Design on Security of SQLite. International Conference on Computer, Networks and Communication Engineering: Beijing.

Needham, R.M., Schroeder, M.D., 1978. Using encryption for authentication in large networks of computers. Communications of the ACM, 21(12), pp.993-999.

Okman, L., Gal-Oz, N., Gonen, Y., Gudes, E., Abramov, J., 2011. Security Issues in NoSQL Databases. Trust, Security and Privacy in Computing and Communications (TrustCom), IEEE 10th International Conference. pp.541-547.

Oracle Corp, 2013. WebSimpleDB API. [online] Available at: <http://www.w3.org/TR/2009/WD-WebSimpleDB-20090929/> [Accessed 2 Nov. 2013].

Owens, M., 2006. The Definitive Guide to SQLite. Unknown: Apress.

Pilgrim, M., 2010. Dive Into HTML5. Unknown: Apress.

Stephens, R., 2009. Beginning Database Design Solutions. Unknown: Wiley Publishing Inc.

Saiedian, H., and Broyle, D., 2011. Security Vulnerabilities in the Same- Origin Policy: Implications and Alternatives. Computer Journal. 44(9), pp.29-36.

## APENDIX

Interview questions:

1. Have you done any web application project personally or within your company?

1.1. If yes:

1.1.1. Why did you decide to do web application instead of native one?

1.1.2. What approach did you use for storing data? If Server-side: why?

If Client-side:

1.1.3. Have you used HTML5 LocalStorage within the project? (Why?)

1.1.4. How was learning curve of using HTML5 LocalStorage? (Simplicity)

1.1.5. Have you faced any security issues when using HTML5 LocalStorage? (Explain) (Security)

1.1.6. How do you evaluate performance of HTML5, specifically LocalStorage? (Performance)

1.1.7. How was the situation while modifying some part of the application? (Modifiability)

1.1.8. How was coping with runtime error handling? (Availability)

1.2. If no: (These questions are asking to check if what interviewees mention, is implemented in HTML5)

1.2.1. What were the reason you decide to go for native application over web application?

1.2.2. Have you faced any security issues when using native application's Storage? (Explain) (Security)

1.2.3. How was the situation while modifying some part or maintaining the application? (Modifiability)

1.2.4. How was coping with runtime error handling? (Availability)

These questions are asking according to the situation:

- Do you have any experience with NoSQL database? (if yes, how do you evaluate it?)

- Do you have any experience with Object-oriented database? (if yes, how do you evaluate it?)

- Do you have any experience with Offline API of HTML5? (if yes, how do you evaluate it?)