TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Science
Chair of Network Software

The Analysis of Offline-capable Web Application Solution and its Implementation Based on Whiteboard Application

Master's Thesis

Student's name: Risto Novik

Student's code: 132307IAPMM

Supervisor: Ago Luberg

Tallinn
2015

Declaration Of Authorship

I declare I have written the master's thesis independently.

All works and major viewpoints of other authors, data from other sources of literature and elsewhere used for writing this paper have been referenced.

---------- ----------

(date) *(signature)*

Annotatsioon

Töö eesmärgiks on arendada kollaboratiivne veebirakendus, mis on võimeline töötama võrguühenduseta või piiratud võrguühendusega. Igapäevaselt suureneb mobiilsete seadmete kasutajate hulk, kelle põhitegevus hõlmab interneti kasutamist. Küll aga pole mobiilse interneti ühenduse tagamine kõikjal maailmas samasugune ning sageli esineb teisi tehnilisi võrguühendusega seotud probleeme. Lisaks on muutunud veebirakendused üha rohkem kollaboratiivsemaks ja keerulisemaks, seega on ka ootused ja nõudmised nende kasutajatelt kasvanud. Eeldades, et veebirakendused suudavad töötada ilma interneti ühenduseta ning ühenduse taastudes andmed automaatselt serveriga sünkroniseerida.

Antud töö käigus uuriti põhjalikult mobiilse interneti ühendusega ja erinevate seadmetega seotud probleeme. Tutvuti olemasolevate ja sarnaste lahenduste käsitlustega, millest igaüht põhjalikult analüüsiti. Analüüsi tulemusena valiti välja lahendus, mida kasutati prototüübi arendamiseks. Enne arenduse alustamist, uuriti detailselt eelnevalt valitud lahendust ning kirjeldati täpsemalt metoodikat komponentide tasemel.

Töö tulemusena valmis kollaboratiivne joonistamise veebirakendus, mis töötab piiratud interneti ühenduse korral. Lisaks ka analüüs hetkel olemasolevate võrguühenduseta sünkroniseerimise lahenduste kohta.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 66 leheküljel, 6 peatükki, 23 joonist, 4 tabelit.

Abstract

The purpose of thesis is to develop collaborative web application which is able to work without connection or with in limited connectivity. Every day the number of mobile device users increases whose activity involves internet connection. Although the network connection coverage in world is not similar and often occurs other network related problems. Additional the web applications have changed more collaborative and complex, because of that the expectations from the users have increased. Assuming that the web applications could work without connection and on connection restore synchronize data with server.

In thesis the mobile network connection and different device related problems were researched in depth. In the search of existing and related work each were analysed. As a result of analysis the solution were chosen for prototype development. Before the development the chosen solution details were studied and described the methodology behind.

The result of thesis were collaborative web applications which works with limited connectivity. In addition to application the analysis of different offline data synchronization solutions.

The thesis is in English and contains 66 pages of text, 6 chapters, 23 figures, 4 tables.

List of abbreviations and terms

API	Application Programming Interface
CRDT	Conflict-free Replicated Data Type
CLI	Command Language Interpreter
2G	second-generation wireless telephone technology
3G	third-generation wireless telephone technology
4G	fourth-generation wireless telephone technology
LPWA	Low Power Wide Area
REST	Representational State Transfer
HTTP	Hypertext Transfer Protocol
PDA	Personal Digital Assistant
SSL	Secure Sockets Layer
TLS	Transport Layer Security
WSS	Secure WebSocket connection
SDK	Software Development Kit
DTIM	Delivery Traffic Indication Message

List of Figures

List of Tables

Table of Contents

1 Introduction

With the increasing number of mobile devices including tablets in our everyday lives, the problem of connectivity plays an important role. Each device requires its own internet connection and the signal quality varies a lot between the providers and locations. There are many places where the connectivity is limited or not available at all. More and more applications are developed and designed with an understanding that there are no internet connection interruptions or disconnections. This approach is incorrect, instead the applications should also be operational within low connectivity or even without the connection at all. In developing countries people do not have the access to high quality internet infrastructure or the prices of the Internet Service Providers' services are too high. Even in the developed countries the mobile connectivity has bandwidth limitations. That is the reason why the web application development for mobile devices should be overseen.

More companies choose to develop mobile device friendly web sites and applications instead of the native applications. In order to support all the major mobile platforms that are Android, iOS and Windows Phone, the native application development requires a huge effort. On the other hand, the native application provides more possibilities and control over the device, but in most cases it is not needed. Developing a web application for a mobile platform has an advantage by supporting multiple platforms. The downside is that there is a wide variety of devices with different hardware specifications, also the browser technology is still limited compared to the native solution.

With the improvements in browser technology and in mobile hardware in recent years, it is possible to create a near native like experience using the browser technology. Currently there is no built-in solution for the offline synchronization of a mobile device. This component is critical when one is developing internet based applications, especially collaborative tools such as drawing, chat, document editing applications etc. One good example is the Google Docs [27] product which allows working without internet connection and later, when the connection restores, synchronizes the data.

1.1 Problem statement

A mobile device without internet connection is mostly quite useless. It is impossible to guarantee the persistent connection for the web applications. There is a need for a solution which allows the mobile applications to work without the internet connectivity. The following are the main issues addressed in the thesis and the options how to solve them:

- Search for an existing solution for mobile *offline data synchronization* in *collaborative* web application in case not found develop a custom solution.
- Analyse data synchronization *conflicts* and *merging* methods.
- *Security* problems by exposing user data on the client side.
- *Development* and *testing* tools for network condition simulation.
- *Web technologies* that enables the application usage in offline mode.

1.2 Methodology

Based on the problem these are the primary steps to reach the purpose.

- Research more about the causes and the existence of connectivity problem.
- Analyse existing tools, frameworks, libraries, methods which have implemented the offline synchronization technology.
- Based on an analysis, choose one solution which would be used to develop a practical collaborative prototype. This is needed to understand the technology behind the scenes and how the components work together.
- Analyse the positive and negative sides that occurred while using the chosen solution, also provide possible solutions for the problems.

2 Problem analysis

The need for offline data synchronization in collaborative applications is mostly related to network reliability and stack related problems. In the following chapters is the overview of connectivity in chapter 2.1 and network stack problems in chapter 2.2 that affect mobile devices the most.

2.1 Connectivity and network reliability

Device Type	**2014**	**2015**	**2016**	**2017**
PC Market				
Traditional PCs (Desk-Based and Notebook)	277	253	244	236
Ultramobile (Premium) *	37	53	74	91
Total PC Market	**314**	**306**	**318**	**327**
Mobile Devices				
Ultramobiles (Tablets and Clamshells) **	227	237	258	276
Mobile Phones	1879	1944	2018	2056
Total Mobile Devices	**2106**	**2181**	**2276**	**2332**
Total Devices Market	**2420**	**2487**	**2594**	**2659**

Table 1: Worldwide devices shipments by device type, 2014-2017 ***

** - The Ultramobile (Premium) category includes devices such as Microsoft's Windows 8 Intel x86 products and Apple's MacBook Air.*

*** - The Ultramobile (Tablets and Clamshells) category includes devices such as, iPad, iPad Mini, Samsung Galaxy Tab S 10.5, Nexus 7 and Acer Iconia Tab 8.*

**** - thousands of units*

Internet plays a more relevant role with the increase of wide variety mobile devices and gadgets (Table 1 [26]). With the increase of devices there has also been also a huge growth in mobile broadband subscriptions. The fixed cable broadband provided a stable connection and usually a problem free network, excluding only the dial-up technology. Moving more and more to mobile broadband means that there is a need for solutions that are capable of handling instability. That is discussed more in the following chapter.

Most of the devices connect to the Internet through the WiFi or mobile networks such as 3G, 4G etc. This enables us to use all the devices wirelessly which is a great benefit and is more comfortable. The mobile network is a physically shared resource and that causes unexpected behaviour with larger user base on a single cellular station. Connection speed could be lower or the connectivity is limited. If you are travelling a lot or do remote work from different places you may have noticed that the quality and level of the connection is

quite different and unpredictable. Another important topic is the indoor mobile network coverage which is much lower or missing at all.

Usually the web pages and applications are built with the need for a good internet connection. Some popular pages like Facebook [23], Instagram [29], Google Docs, which make a lot of background queries for fetching new data. When the network connection is dropped, some data might not be synchronized or even be missing. That is the reason why the web pages and applications should also have an option to work offline mode or in a limited connectivity network.

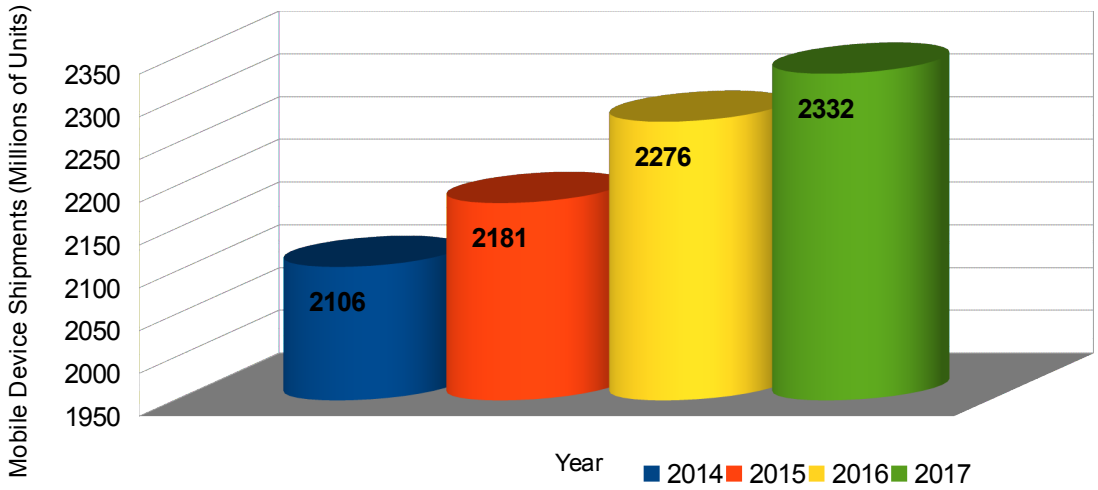

*Figure 1: Worldwide devices shipments by mobile device type, 2014-2017**

** - millions of units*

The trend of mobile device is increasing as could be seen and more people visit the web pages from mobiles (Figure 1 [26]). That is one of the reasons it is important to focus on this problem. Improving the user experience on the mobile web application should be similar or better than the native application.

2.1.1 **Estonian mobile network coverage**

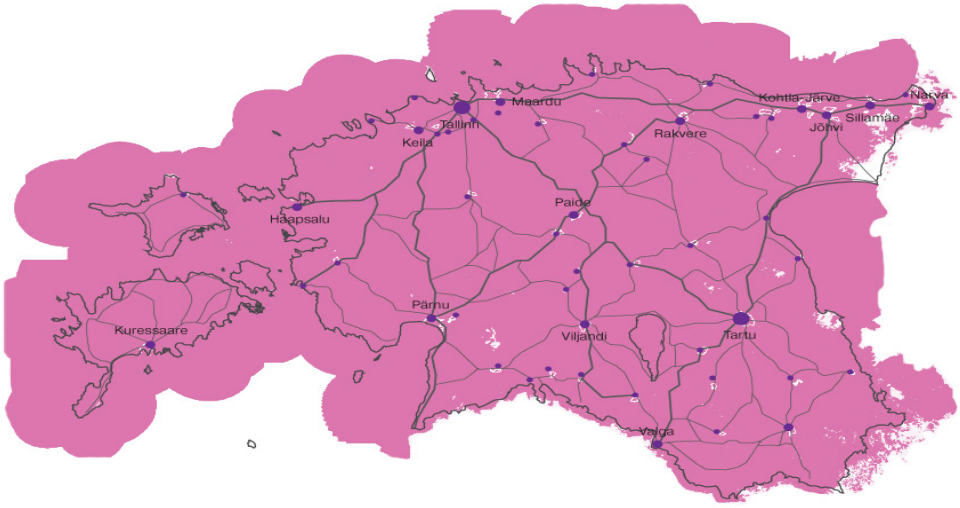

Figure 2: 4G network coverage in Estonia January 2015

It has been predicted that by the end of year 2018 the mobile internet usage in Estonia will be about 61% [16]. With the 4G mobile networks the overall network coverage is quite good, according to the results of EMT in January 2015 it was 99.9% and reached 70 kilometres over sea (Figure 2 [22]). Using the latest mobile generation 4G enables to have much lower latency compared with the older technologies 3G and 2G. The synchronization process is faster for the end user and changes will be visible immediately.

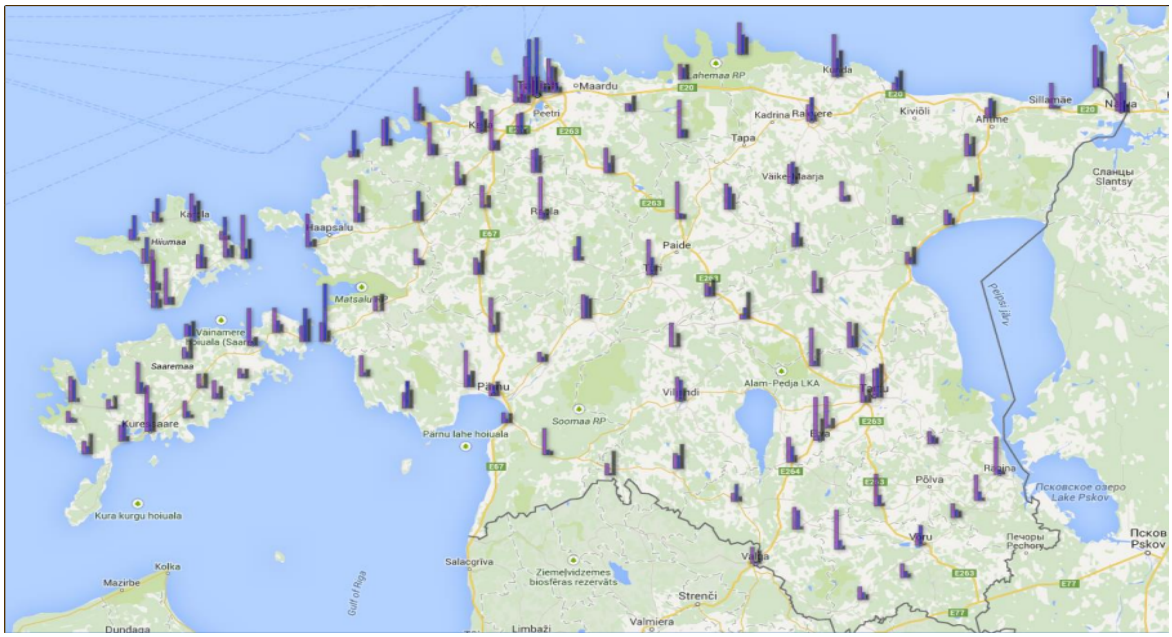

*Figure 3: Internet service providers' download speeds in January 2015**

** - Shown in columns from left to right EMT, Elisa, Tele2*

The signal quality is quite different in various places and the quality improvements are made in big towns and villages. In many places in the countryside has quite bad connectivity or is missing at all. These are the 4G test measurements by The Technical Regulatory Authority (Figure 3 [35]), which shows the quality difference between the internet service providers. In smaller towns the signal coverage is not as good as in larger towns like Tallinn, Tartu, Pärnu etc.

2.1.2 World mobile network coverage

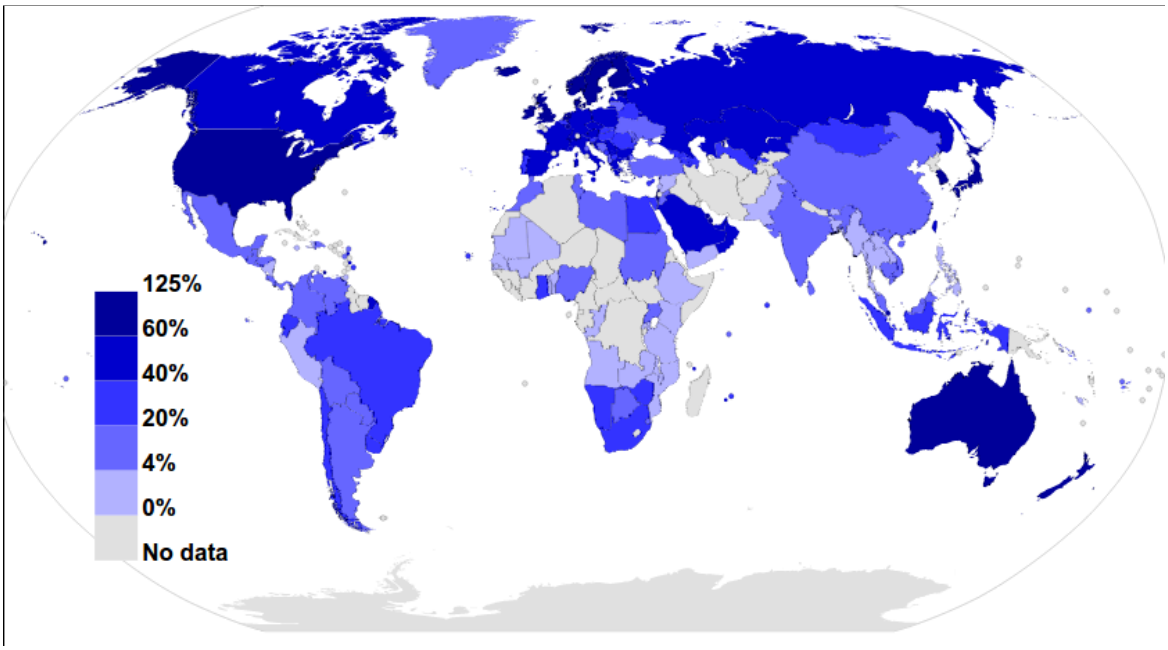

*Figure 4: Mobile broadband Internet subscriptions in 2012**

** - a percentage of a country's population*

The network coverage in Estonia is good compared to world overall statistics (Figure 4 [31]). Roaming is more important on the world scale because of many people love to travel and while flying with a plane the user wants to use application without any interruptions even when there is no connection.

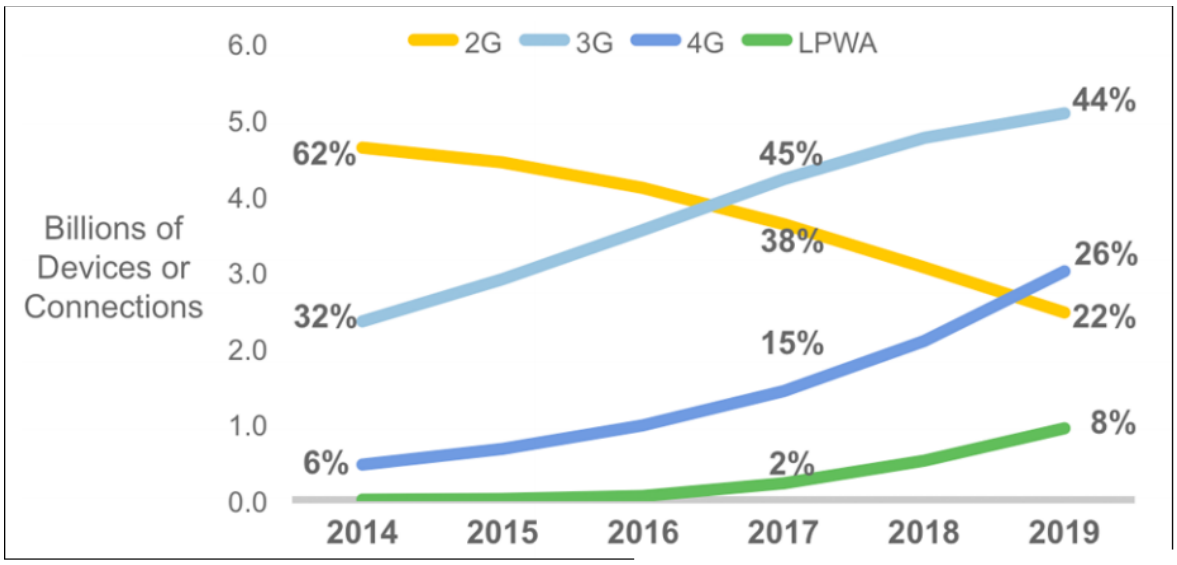

Figure 5: Global mobile devices and connections by 2G, 3G, LPWA and 4G

The newer generation network connectivity is slowly getting more popular by the year 2017 3G and 3.5G will surpass the 2G and in 2019 4G will surpass 2G as showed in Figure 5 [20]. This means that developers must also consider high latency and slower mobile networks which has the majority of the total world connection usage.

2.2 Device and network stack problems

Besides the connectivity problems there are also network stack and technical problems of mobile devices which might occur during data synchronization. Usually these are the problems of architecture or the network stack which can not be so easily changed and adapted.

Table 2: Data rates and latency for mobile connection

■ - *high latency*
■ - *low latency*

Generation	Data rate	Latency
2G	100 – 400 Kbit/s	300 - 1000 ms
3G	0.5 – 5 Mbit/s	100 – 500 ms
4G	1 – 50 Mbit/s	< 100 ms

- **Packet loss** – is higher when dealing with cellular network compared to wired connections. If the transferred data is text based then the packet loss would not be as critical compared to voice and video streams where the problem is more visible

to the end user.

- **Latency based problems** – the latency differences between the mobile connection generations are quite large (Table 2 [6] page 104). Even if the ISP has good downlink and uplink for the connection, the latency causes problems with real time synchronization. In this thesis the real time is defined as an event which occurs under one second.

- **Battery lifetime** – another important topic is how the mobile device connectivity affects the battery lifetime. Most of the time the mobile device is in the idle state. For example using WiFi technology which allows changing transmit power that could be mainly between the 30 – 200 mW range. WiFi DTIM (Delivery Traffic Indication Message) is a multiplier of the beacon interval, with shorter beacon intervals the wireless traffic is increased and this drains battery faster. Mobile connections 3G and 4G consume only 15 mW in an idle state, but on data transfer bursts this could change to 1000 – 3500 mW. It is important to know when to send data and when not to. For example the periodic connection wake up state drains the device battery level significantly more. Android SDK has support for job scheduling API which is responsible for buffering the responses and requests and then send these as a single batch. Similar technology could be used in browser side by adding an extra abstraction layer for that. Applying such techniques makes the energy consumption considerably lower and the battery lasts longer.

- **ISO OSI model** – the stack consists of multiple layers, starting from top to down: application, presentation, session, transport, network, data link, physical. The model was designed for the wired network not for wireless. The model also introduces overhead for the wireless network which could be avoided using cross-layer design [11].

- **Device features and capabilities** – when looking the Android market there are quite many phone manufacturers, each of these devices has different radio capabilities and characteristics. Even when the ISP has the support for the latest 4G networks the device could be limited only to 3G.

2.3 End user stories

There are lot of categories and use cases where the offline data storage and collaborative applications could be used and applied. In this chapter we are going to discuss the main categories and also the extended use cases.

1. **Collaborative whiteboard application –** with the increase of tablet devices it is natural to draw on a device not in desktop PC. The problem is that the content should not be static and saved into the file system where you have to share the file with other users. That is why the collaborative whiteboard applications automatically synchronize the content and also enable working even without the connectivity.

2. **Email client** – the received emails with the content should be accessible even in offline mode. Without internet connection it is impossible to transfer new composed messages, but it should still be possible to write messages in offline mode.

3. **Chatting application** – access older chats even while in offline mode.

4. **Multi form editing at the same time with presence** – apparently Information System (IS) includes usually form based editing and those are allowed to be edited by multiple users. The concurrent editing ends with the saving of the data and one of the user's data being overwritten. Instead of the previous design, the forms should be editable by multiple users without locking and it should be possible to see and save the form in real time.

5. **Collaborative document writing** – allows edit the same document by multiple people, Google Docs [27] is a good example.

2.3.1 Extended use case: Wiki/Wikipedia

"A wiki is an application, typically a web application, which allows collaborative modification, extension, or deletion of its content and structure" [44].

The content is community driven and without the user contributors wikis could not exist. That means that wikis are collaboration tools, the negative side of this is that when the users are in edit mode, it locks the content for other users. Popular writings and articles get many simultaneous edits which usually end in a conflict. Trying to edit the whole document content locks the posting permission like it is implemented in DokuWiki [21]. Even if using locking for subsections it is not convenient for the end user. Another approach commonly used in MediaWiki [34] based engine is the post conflict handling. This allows the user to save the content and then solve the conflict if there was one. Instead of using the old locking or conflict solving model, it would be better if one could see the content editing in real time without locking specific parts. That would also allow writing the document offline and then later synchronizing the article.

2.3.2 Extended use case: technician tool

More and more companies find that the use of mobile devices makes the communication inside the company much easier and more transparent. Instead of using paper based orderings as communication between the customer and inside the company, it is a much better solution to use *PDA-s.* This makes the technician's job in the user's first time customer visit much easier. That is due to the fact that the technician gets the latest information about the amounts of equipment in storage and can also offer a more specialized service for the needs of customer. This kind of tool should also have a requirement to work within offline mode because there are many places where the mobile network coverage has no signal at all. That is the reason why the developed tools need an option to later synchronize the data changes. Besides the syncing there might be some conflict problems which need confirmation from the technician.

3 Related work

Before starting one's own custom solution development it is good to have a common understanding of related work which is solving a similar problem. Analyse each solutions cons and pros. Find the best matching solution and build based on top of that the prototype application.

3.1 Existing solutions

Instead of the usual data model design, the offline applications need a different approach. The offline ready applications should be designed having in mind that in client's side only a small subset of data would be available, not the full data set. The user authentication without the server side validation is impossible, instead the application should be ready without requiring signing in. Later, when the user has restored the internet connection, the application should provide merging data for the specific user.

These are the main problems to look for in the solution analysis (Alice and Bob are example users):

1. How the mode switching has been handled, from "online" to "offline" and from "offline" to "online"? Does the user get any notifications or messages about the mode switch?

2. How quickly is the change of state detected? The WebSocket based solutions could detect the connection drop instantly, which is the opposite to HTTP polling, where the detection of change could take seconds.

3. How have the lossy, high latency and low speed connection been handled? Does the application have a mechanism for detecting it?

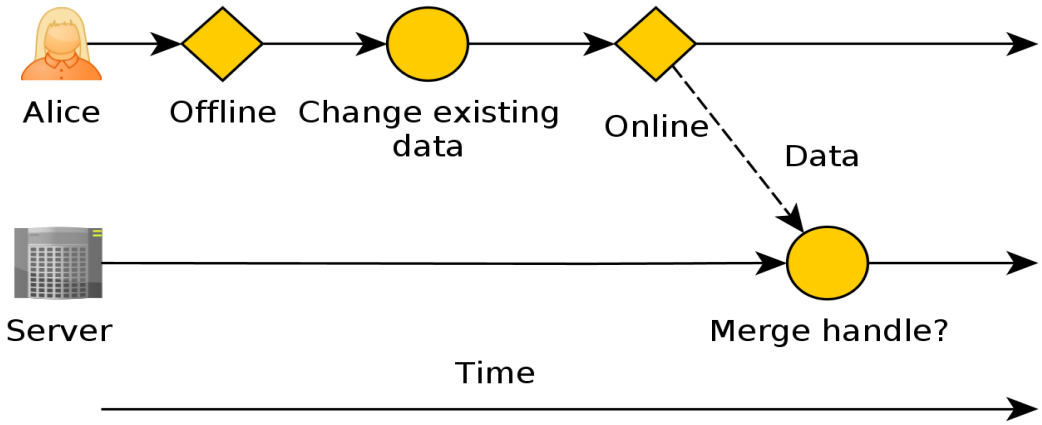

Figure 6: Case when data edited in offline and after in online state how data is merged

4. How are the data conflicts handled? For example, Alice is in offline state and edits a text item, after that she moves into online state. How is the data synchronized in server, does it require the intervention from the user (Figure 6)?

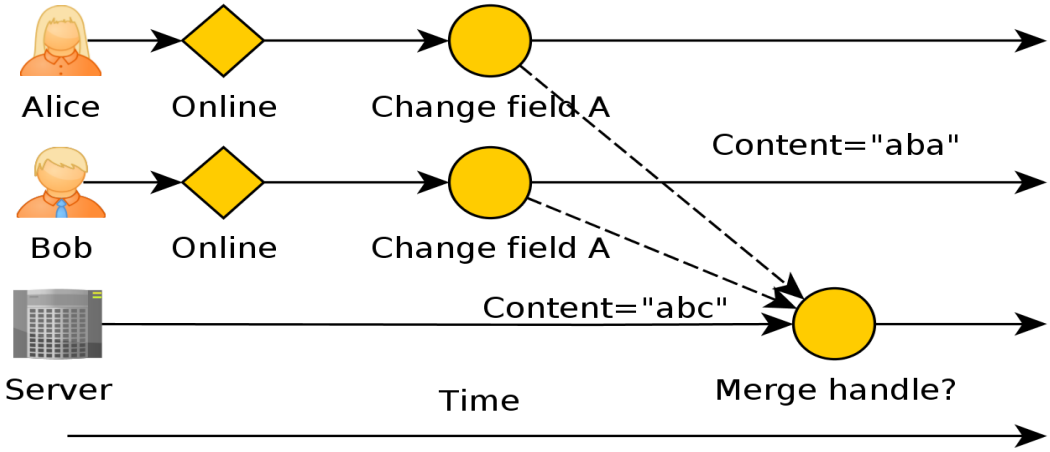

Figure 7: Case when users edits the same field

5. Alice and Bob are simultaneously editing the same data field, how is the conflict handled (Figure 7)? In following case both users edit the same text field received from server. Merging the data with server causes conflict as there is already existing field and also merging occurs on the same time for both users.

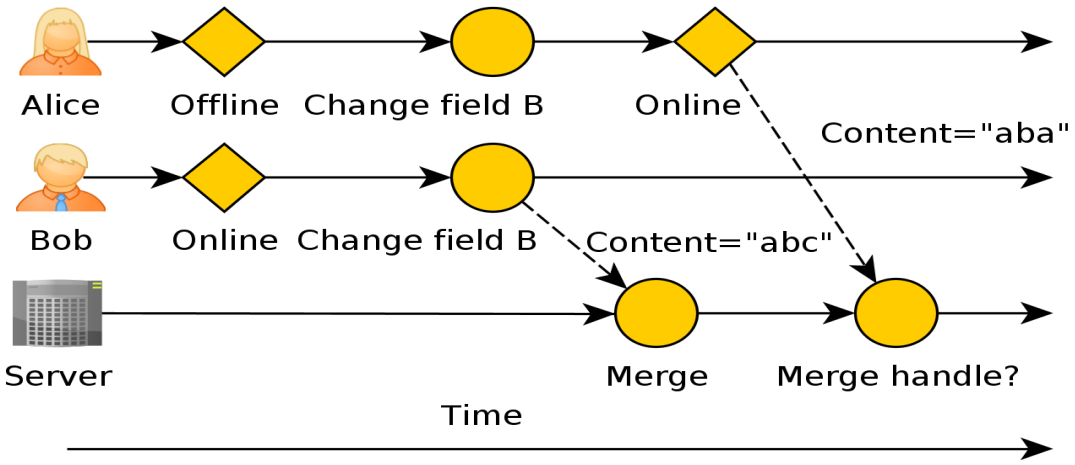

Figure 8: Case when one of the users edits same field in offline

6. Alice is in offline state and edits the same field as Bob in online state, when Alice comes online and synchronizes the data, how is the conflict handled (Figure 8)? In this case is important how the offline edited content will be merged back or is the online synchronized data only primary source.

7. Does the solution allow real time streaming data (for example WebSocket support) or are all the changes synced with HTTP protocol which is enough for text based syncing?

8. Consistency problems when there are web clients whose data might be couple months old without synchronization. In such case, should the system show the warning on data merge and how are the conflicts handled?

3.1.1 **StrongLoop LoopBack**

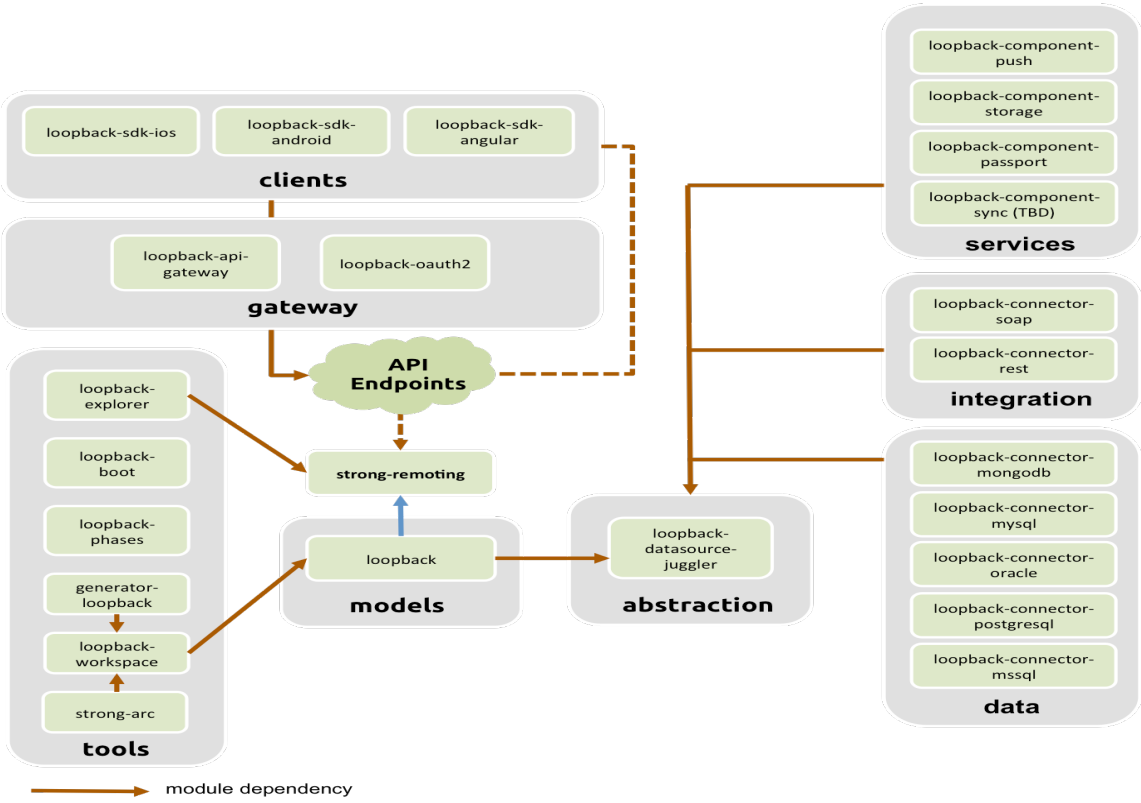

Figure 9: LoopBack modules relation and dependencies

StrongLoop LoopBack(v2.x) is a framework for building scalable REST API-s, it is built on top of the Node.js Express module. LoopBack has a variety of different features and modules (Figure 9 [41]):

1. CLI with code generation support, base structure.
2. API generation based on Data Models.
3. Automatic API documentation generation with Swagger.
4. Social logins(OAuth) support, own ACL model.
5. Easy to switch between different databases, supports multiple connectors: Oracle, SQL Server, MySQL, MongoDB etc.
6. Mobile specific implementations push notifications, offline synchronization.
7. Enterprise supported solution.

Offline capabilities:

The synchronization module [33] in LoopBack framework is still in an experimental state so there might be bugs and issues that are not yet solved. The idea of LoopBack syncing module is similar to the version control systems. The client(browser) replicates the changes to the server and the server replicates the changes to client. In the replication operation conflicts might occur, which have to be handled manually. Or if the manual conflict handling is not preferred it is possible to use the custom merging logic. The models used in the server and on the client side are common and allow sharing the same models. The client side replication of data is held in web storage(local storage) which is not a good option for larger data sets.

Positive:

1. Extensive list of documentation and examples.
2. Enterprise based support, stable releases and tested solutions.
3. Possible own hosting deployment.
4. Data integration with many database providers: Oracle, SQL Server, MongoDB, MySQL etc.
5. Quick setup with command line tools with code generation.

Negative:

1. Not the best suited for existing projects as this means the rewriting of project and designing a new data model.
2. Not practical for smaller projects which usually do not need such an extensive list of features and unnecessary abstractions.
3. The framework components are open-sourced, but the DevOps tools, monitoring, support are only for paid users.
4. Deep learning curve.

3.1.2 Apache CouchDB and PouchDB

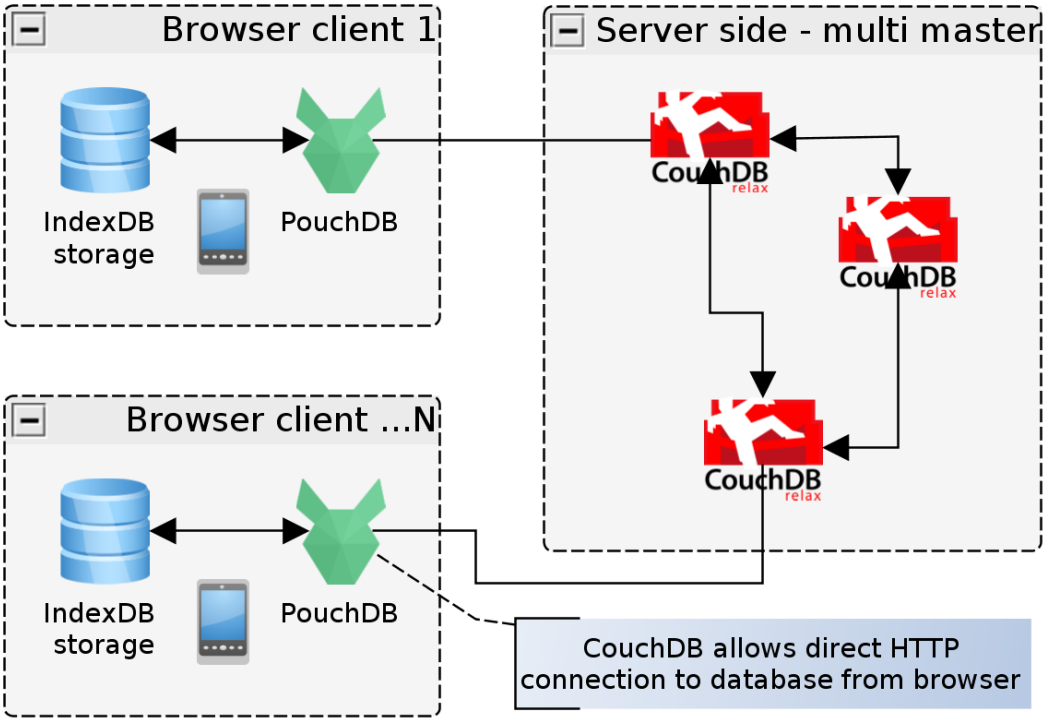

Figure 10: CouchDB and PouchDB architecture

The combination of Apache CouchDB(v1.6.x) and PouchDB(v3.4.x) is built for the covering of the mobile synchronization in mind as shown in Figure 10. On the server side CouchDB is a NoSQL database that uses JSON documents, for querying the map-reduce indexes and HTTP for API. The client side library PouchDB has similar API like the CouchDB which allows easy switching for developer and data model design. PouchDB is responsible for the client side replication and the data synchronization between the server and local databases.

Offline capabilities:

PouchDB applies all the operations and document changes locally in the user browser. Each browser client's could be thought as a separate replica that is synchronized with the master CouchDB instance. There could be multiple master nodes in CouchDB which allow direct connection to the database. The downside of this is the HTTP based synchronization ping-pong of getting changes and applying the data differences. This should be done using the WebSockets instead of HTTP requests flood, which cause high network traffic and

what are more problematic on mobile devices. The conflict resolution can be controlled manually and allows custom solutions.

Positive:

1. Full solution with server (CouchDB) and client side (PouchDB) support.
2. Implementing the CouchDB document sync protocol would allow using other database engines. Only then there would be a need for separate server service for the data exchange.
3. Multiple client side storage supports: LocalStorage, IndexDB, WebSQL, in memory storage.
4. CouchDB has support for REST API which allows using a database without the need for a separate server node layer.
5. Replication history is stored, this would allow implementing undo model.
6. Replication data filter option for not dumping all the database but only a small subset.

Negative:

1. For most of the existing projects which use a relational database this requires using a NoSQL database called CouchDB.
2. Maintenance for CouchDB, replication, backup, hosting etc.
3. Learning curve compared to other methods seems much steeper and longer. There are lots of settings needed to configure before use.
4. Data transport is limited to HTTP due to the CouchDB implementation.

3.1.3 **Firebase**

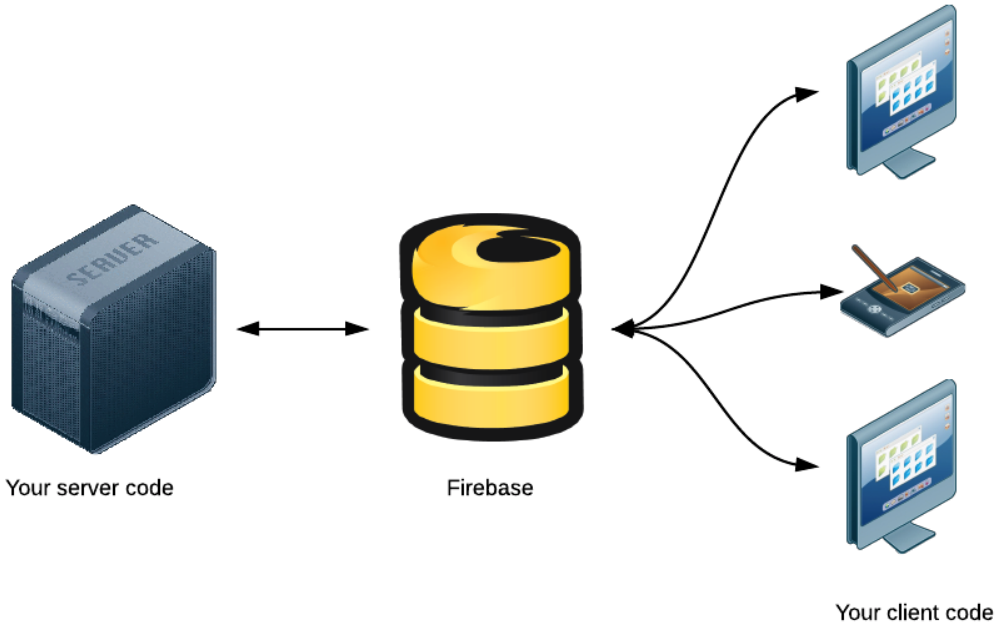

Figure 11: Firebase architecture

Firebase is a service for real-time mobile and web application platforms which solves the storing and data synchronizing problem. Recently, the Firebase team joined the Google Cloud Platform [3] team to extend the service. The simplified architecture of Firebase is illustrated on Figure 11 [1]. Besides the direct client connection to Firebase database, it is possible to access and change data from the server side using REST API.

Offline capabilities:

Firebase mentions the offline capabilities briefly in their documentation [24]. The topics they have covered are:

- Offline/online status detection works effectively and quickly.
- Reconnection problems have been well solved.
- Main problem of data sync itself is partially solved in the latest version of Firebase.

All the data which the user has created in offline state is stored only in web client's memory. That means that after refreshing the page one would not be able to access or sync the previous data. Even saving the data locally will not solve the data merging. The client

and server synchronization works on a “best-effort” basis, the data conflicts are solved with the Last Write Wins method.

Positive:

1. Quick to start development with no extra server setup needed, perfect for prototyping.
2. Server hosting, replication, backup, analytic problems are already solved.
3. Documented API with lots of examples and different anti patterns.
4. Library support for multiple mobile platforms: Android, iOS and OSX.
5. Providing the server side libraries for Java, NodeJS etc.
6. Account owner has access to the online data editing and viewing administrator UI, which also allows following the data that is currently added and edited in real time.
7. Open data [25] sets for everyone to use: earthquakes, currencies, weather in the United States.
8. Multiple authentication providers supported by OAuth, token based, email password pair.
9. Presence API.
10. The best match for developing a prototype.

Negative:

1. Only service based hosting, you can not setup your own instance on Firebase, which would give more control about the scaling of the infrastructure.
2. Firebase uses its own way to store the data (key-value storage), due to which there is need to redesign the existing database solution.
3. From the Firebase database schema design perspective, the data must be structured flat, as the nested data depth is limited. (Maximum 32 levels deep nested data)

4. The reading, writing permission and data validation is done in a single JSON file where the logic is written in strings. With the growth of the application it is difficult to maintain, understand and test such rules.

Table 3: Firebase pricing model

	The Hacker plan	**Candle**	**Bonfire**	**Blaze**	**Inferno**
Connections	50	200	750	2500	10 000
Data transfer	5 GB	20 GB	75 GB	250 GB	1 TB
Data storage	100 MB	3 GB	10 GB	30 GB	100 GB
Other features	-	Custom domains	Custom domains, private backups	Custom domains, private backups	Custom domains, private backups
Price	Free	$ 49 / month	$ 149 / month	$ 449 / month	$ 1 499 / month

The Firebase is a service based solution, it is good to take the overview of the pricing model in Table 3 according to the date 17.04.2015. For the development and the test example is the free “The Hacker plan” enough.

3.1.4 Swarm.js (CRDT)

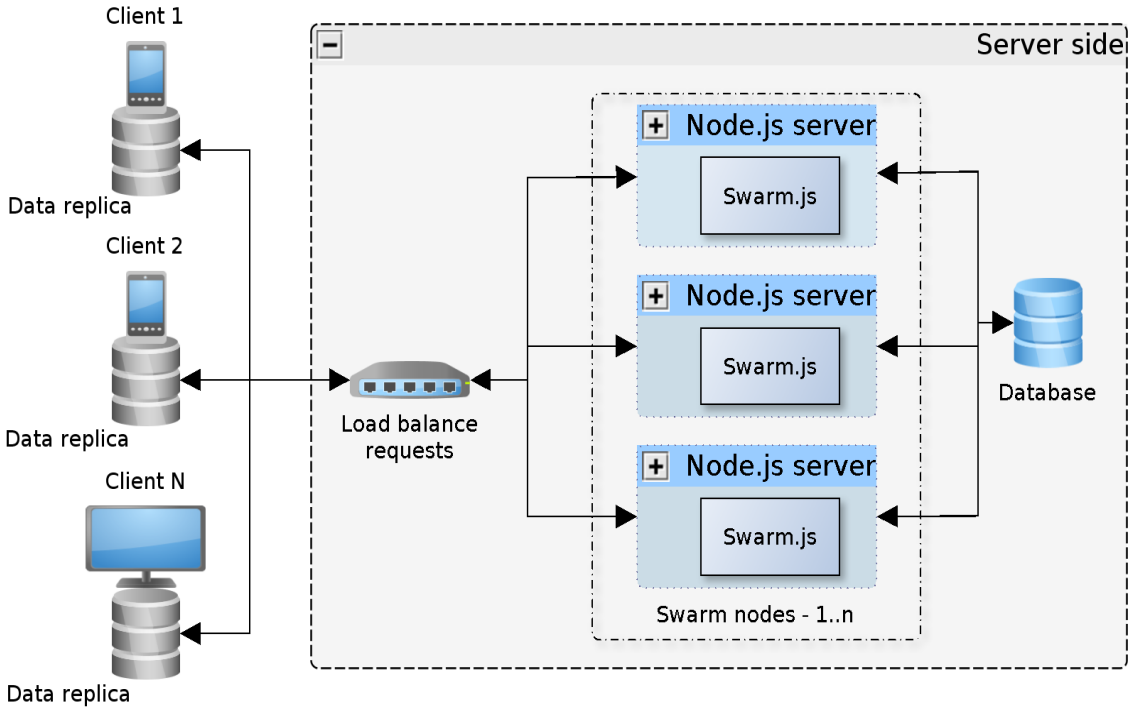

Figure 12: Swarm.js example architecture

Swarm (v0.3.x) is an isomorphic JavaScript library which synchronizes objects in real time and allows working in offline (Figure 12). The library is concentrated only on data synchronization not any other problems. Swarm has support for the complex data structures which relies on operation-based CRDT.

Offline capabilities:

Swarm uses quite a similar strategy to other solutions, all the data is replicated on the client's side. All the actions applied on the client's side are first saved to local database and then synchronized. This is perfect for the offline use cases where you can not relay on the server HTTP responses and network connection. On the restoring of the connection all the operations are synchronized between the local and replica server. The conflict resolution is handled by the conflict-free replicated data types which guarantee that there will be no conflicts and that the data is automatically merged.

Positive:

1. Small and lightweight, instead of using the full featured and heavy abstracted framework, all of this can be solved in the modular way.
2. Quick responses to Github issues from the main contributors.
3. The whole project is open source which allows easy contributions.
4. Both the server and the client side share the same data model.
5. License MIT.

Negative:

1. Project is in an early stage, so there will be changes that are not compatible with previous versions. No stable version for a production use which is well tested.
2. Missing up to date documentation, there are fragments of documentation from previous versions.
3. No clear milestones of when the next version will be usable and what are the features and fixes.
4. Not too many examples and real use stories (the project is in an early stage).
5. Learning curve is steeper because before starting with Swarm you have to learn the concept of CRDT.
6. No supported storages for the relational databases.

3.1.5 Overview

In previous subsections different existing solutions and tools were described. This section gives an overview of those tools along with some analysis. The comparison is shown in Table 4, the columns represent the analysed solutions whereas the rows represent the properties which are written out in the following paragraph.

Solution type – the selected solutions are in different types, frameworks, libraries and services. Libraries are meant to solve only one certain type of problem. This means it is easier to integrate it with an existing project. Frameworks, on the other hand are trying to solve multiple problems and tend to have a larger learning curve, also usually the existing solution would need to be rewritten. Service based solutions mainly provide libraries and SDK-s for the integration.

Deployment option – the various solutions are only used as services and can not be setup as a standalone solution. This is important when the solution is needed to be scaled horizontally, or the customer needs more control over the infrastructure, for example encrypting stored data.

Offline sync support – has the feature to synchronize the changes made in offline when the state changes to online.

Conflict handling method – in case of conflict does it need an input from the user to solve the conflict or are the changes merged automatically.

Sync channel – the protocol which is used to transfer data between server and client(browser). With older mobile version, depending on the requirements, the HTTP requests could be the only way for data transfer. WebSocket should be preferred for desktop and mobile device because it allows transferring data quicker (without the overhead of opening and handling new connections in HTTP) and with lower latency. Depending on the browser implementation the concurrent HTTP requests to the same domain are limited. In Google Chrome v34 to 6 parallel requests and in Microsoft Internet Explorer v6 to 2 parallel requests.

Open source – does the solution have full access to the code and possibly accepts the project contributions.

Contributors – show the project start date, based on Github.com, also commits of different contributors of the project. A larger contributor base indicates that the project is possibly active and growing.

License type – depending on the license type there could be limitations of forking projects and further monetizing.

Table 4: Data synchronization solutions

■ - *negative features*
■ - *positive features*

	StrongLoop Loopback	**CouchDB and PouchDB**	**Firebase**	**Swarm.js**
Solution type	Framework + offline module	PouchDB is client side library	Service	Library
Deployment option	+	+	-	+
Offline sync support	+	+	-	+
Conflict handling method	User has to handle conflicts or based on custom merge handler	Custom conflict handling	Solved as "best-effort"	Based on the CRDT data-structure, mostly auto merge
Sync channel	HTTP	Default enabled to HTTP calls	Data streamed through Websocket	Multiple channel types supported, Websocket, HTTP long polling
Open source	+	+	-	+
Contributors	45 (Apr 7, 2013 – Apr 6, 2015)	137 (Jun 6, 2010 – Apr 6, 2015)	-	4 (Feb 3, 2013 – Apr 6, 2015)
License type	Dual license: MIT + StrongLoop License	Apache License	Closed, not available	MIT

3.1.6 **Conclusion**

Based on the analysis of different data synchronization solutions it is the goal to find the best solution matching the requirements mentioned in chapter 3.1.

Looking at the solution type it is important that the solution could be hosted one's your own server without the need for buying any external service. That is one of the reasons why *Firebase* is not suitable for our needs. Additionally, the client side library does not have persistence in storage support.

The *Loopack* framework offline module requires use to the full framework and can not be used without an extracting small part of it. The offline module is more suitable for form-based applications, where conflict handling requires user action.

One of the maturest production ready solutions is the *CouchDB* + *PouchDB* combination. The downside is that you have to use the *CouchDB* NoSQL database which is not compatible with the existing architecture if there is no previous use of *CouchDB*. To use other database systems one has to write an adapter or a layer similar to the HTTP CouchDB synchronization. The data transferring could be done more effectively using long polling techniques or WebSocket.

Swarm.js has a different approach compared with other solutions, using the idea of CRDTs as data structures. CRDT's approach is more suitable for multiple data replication systems. The data merging is easier to understand compared to other methods because of the CRDT properties. *The Swarm.js* library is modular and gives the opportunity to extend it with own custom modules. Data transferring between the client and server can also be chosen based on the application and user needs. With a real time application you may use WebSocket connection instead of HTTP for smaller latency.

According to the analysis, the *Swarm.js* would be a good match to continue to work with. Before going into details it would be good to understand how the CRDT concept works, how the data is stored and how to design the data model.

3.2 Convergent or commutative replicated data type

This chapter gives a short introduction to the basics of convergent or commutative replicated data types (CRDT), how they work and what the main guidelines for designing such a system. The mobile devices could be looked at as a distributed systems, which in limited connectivity state are isolated and should be able to work without a central network server. Data replication is the key to look for, each mobile device could be looked at as a small partial replica. All the operations can be applied locally without any external synchronization and when in connected state the operations apply asynchronously to other replicas. The operations can be in a different order as to which they were first created, that means partially ordered.

Application areas of CRDT include computation in delay-tolerant networks, latency tolerance in wide-area networks and partition-tolerant cloud computing ([9] page 3). These are the properties matching with the needs of mobile devices. The current state of the mobile device's peer to peer network is still in early steps and is not as usable as the Internet. The topic of direct peer to peer communication (WebRTC [43]) has been left out because of the experimental state of browsers and small support by mobiles. This is also one of the topics which could be further researched, use cases, usability etc. In our case there are multiple clients (e.g. mobile devices) and a centralized server which is responsible for the synchronization of data. CRDT has its limitations and problems which are discussed more in chapter 3.2.4.

All the CRDTs must meet the semilattice (S) algebra identities, where $S=(S,\Lambda)$ satisfying, for all $x,y,z\in S$: ([10] chapter 5 CRDTs: Convergent replicated data types)

- associativity: $x\wedge(y\wedge z)=(x\wedge y)\wedge z$
- commutativity: $x\wedge y=y\wedge x$
- idempotency: $x\wedge x=x$

The CRDTs could be divided into two different types based on the replication: state and operation based replication. These two types differ mostly by the assumptions and

performance which are discussed in chapter 3.2.1 and 3.2.2. The state and operation based replications could also emulate each-other.

3.2.1 State based replication

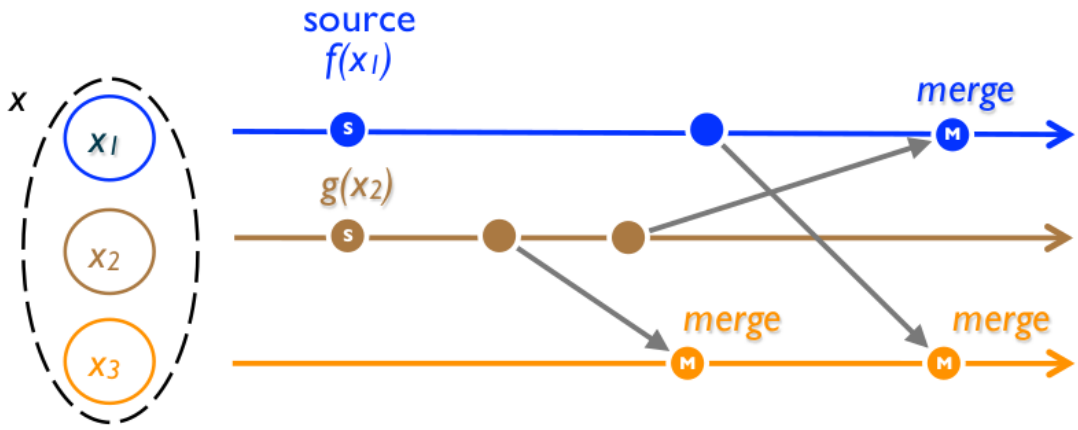

Figure 13: State based replication

State based CRDTs are called Convergent Replicated Data Types (CvRDTs). CvRDT transmits the full local state between the replicas. This allows using more unreliable channels for the transmission of the state. Compared to the operation based on the full state replication, broadcasting has overhead and increases with larger state objects. In CvRDT the object update occurs entirely at the source, then propagates by transmitting the modified payload between the replicas x_1, x_2, x_3 (Figure 13 [9] page 6). The dot with "s" represents the source at start point, the empty dot shows the state change and line with an arrow shows data transmit.

3.2.2 Operation based replication

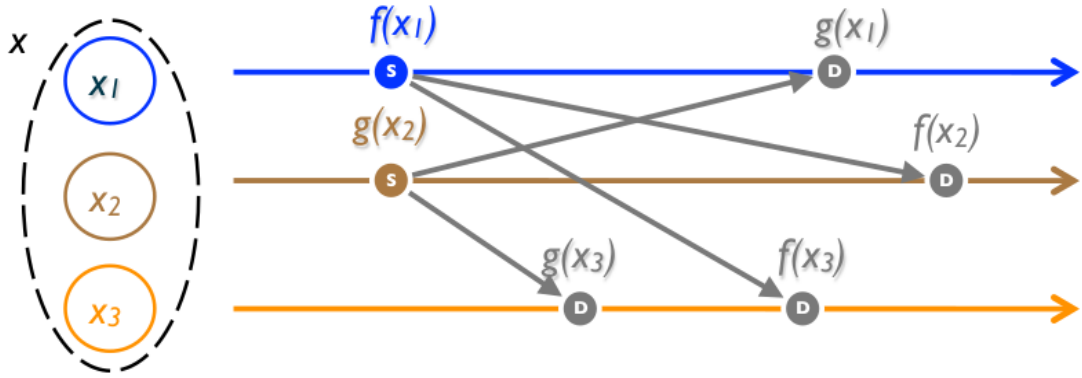

Figure 14: Operation based replication

Operation based CRDTs are called Commutative Replicated Data Types (CmRDTs). The requirement is to have a reliable broadcast channel to deliver all the updates to every replica. Instead of broadcasting the full local state, CmRDT transmits only operations to be applied on every replica $x_{1,}x_{2,}x_3$ (Figure 14 [9] page 8). The dot "s" represents the source at start point, in this time the operations are applied which is represented with dot "d".

3.2.3 Data structures

Based on the problem that one is trying to solve one has to choose a suitable data structure. In CRDT there are quite many data structures for different problems. All the CRDT data structures have its own limitations and restrictions. For example, when creating a chatting application where the order of each message matters then a Set data structure might not be the best match for it as it is not ordered and stores only unique messages. Before choosing the data structure one has to make sure to understand the conditions and requirements for the data.

The main supported data structures([10] chapter Replication):

- **Counters**
    - Grow-only – on merging uses the maximum function over values, payload is a single integer.

- Positive-negative counter (PNC) – uses two counters, one for increments and the other for decrements.

- **Registers**
  - Last Write Wins – timestamps, or version numbers, on merge get the maximum(timestamp), payload = value.
  - Multi-valued – vector clocks, on merge take multiple values.

- **Sets**
  - Grow-only set – on merge get the union of set items, payload is set itself and no removal is allowed.
  - Two-phase set – uses two sets, in one set stores the added and in other set the removed items, the downside is that the elements can be added and removed only once.
  - Unique set – an optimized version of the two-phase set.
  - Last write wins settings – on merge uses a maximum timestamp and the payload is set
  - Positive-negative set – consists of one PNC per set item
  - Observed-remove set

- **Graphs, sequences**
  - Treedoc – is a collaborative editing system which uses a binary tree to represent the document [12].
  - Logoot – similar data structure to Treedoc but instead of the binary tree the n-ary tree is used, besides that Logoot has support for undo mechanism [15].

3.2.4 **Real-world uses**

Even tough there is not many examples of client side CRDT usage, there are more examples in server side. These are the two popular projects which are also open source and a good resource for learning the concept of CRDT.

SoundCloud Roshi – is basically a high-performance index for timestamped data, the main use case is the SoundCloud activity stream [40]. Roshi implements time-series event storage with the last write wins element set. It is a stateless, distributed layer on top of the Redis [37] using the CRDT.

Database Riak – is an open source, key-value storage distributed database which offers high availability, low-latency, fault-tolerance, operational simplicity, scalability etc [38]. The core of the database is built on the CRDT data structures. The database runs only on the server side and does not have any browser based support.

Riak supports following data structures [39]:

- Flags – which allow set value to *enable* and *disable*, also could be used within maps. On convergence *enable* wins over *disable.*
- Registers – binary value which could be used within a map. On convergence the last write wins the value.
- Counters – support increment and decrement. On convergence the maximum value is used as the last, for example if one instance had the counter value of 10 and the other 20, then max(10, 20) uses the maximum value of 20.
- Sets – collection of unique binary values or strings. On concurrent convergence the *add* operation wins over the *remove* operation. Means if you have added a string "hi" and deleted it at the same time, the entry "hi" remains.
- Maps – allow storing any other type within fields. On concurrent convergence of the *add*, update or remove operations, when the elements are not existing, the *add* operation wins and if an element is already existing the update operation wins.

3.2.5 Limitations and problems

Using CRDTs gives a lot of freedom on the conflict handling but on the other hand there are also limits and problems of which the developer needs to be aware.

Garbage collection – the operation's log size could over time get pretty large and needs to be compacted. This is an important topic for the replication of mobile devices where the storage size is rather limited. There is a possible solution to compact the log with strong consistency based synchronization.

Undo-Redo models – how to implement such a feature so that when the user has undone some changes and after the replication, these would not be available again or not applied again, when the client has been offline for quite some time.

Binary formats – all the following solutions will not be able to handle the binary formats like image, video audio etc. In multimedia heavy application there is a need for a different solution to support binary synchronization.

Data model changes – the problem with replication based solutions is the backward compatibility. Changing existing data structure fields or type could lead to problem, when the client side sent data is not compatible with server side's. This has to be handled separately on the server side.

4 Solution

This chapter is more about the based on the chosen solution from the previous chapter that is going to be used to develop the prototype. The prototype should show the possible bottlenecks, limits, problems etc. Besides that there is a detailed introduction to the main components, security and license of Swarm.js. Also, a brief overview of the development and testing tools that can emulate similar conditions to a mobile network is given.

4.1 Practical collaborative example

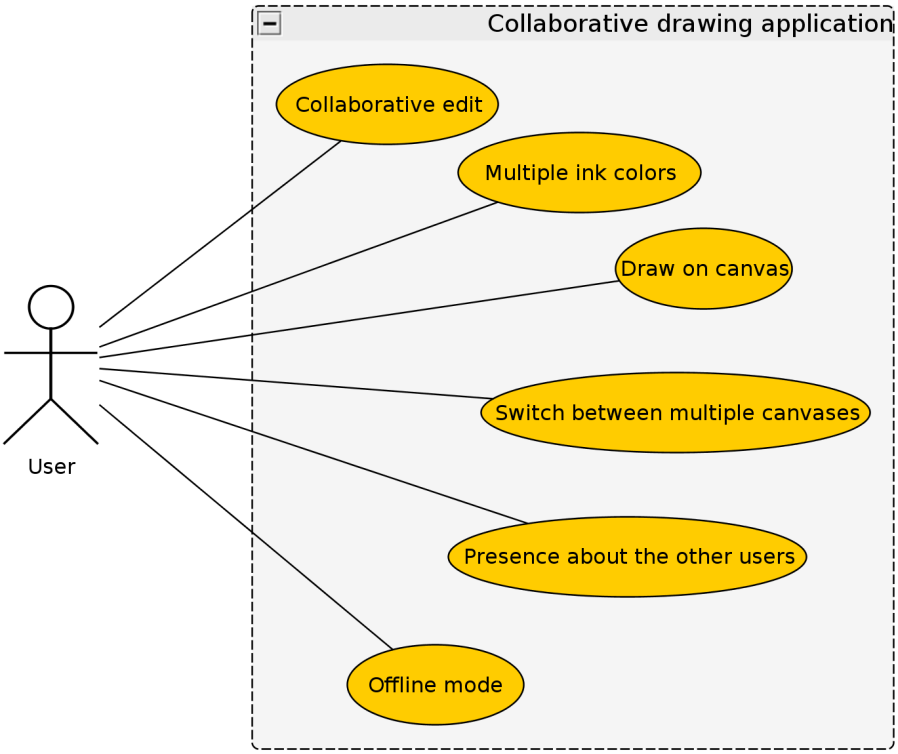

Figure 15: Use case diagram

To understand the solution and concept behind the CRDT and web application technologies, it is good to apply it in a real collaborative application. There are many possible use cases that could be implemented, but the drawing application is the clearest example of collaboration and offline data synchronization. This application could be used for sketches and brainstorming, the use case diagram is showed in Figure 15.

The requirements for the application are the following:

1. User should be able to draw on canvas.

2. User should be able to switch between multiple canvases.

3. User should be able to draw even if there is no internet connection (offline mode). For example “User A” had a meeting where he started drawing some sketches. Later, he was flying on a plane where travelling where the internet connectivity was limited and wanted to continue his work in offline mode.

4. All the users should see the same drawings on canvas when in connected state. (collaborative and syncing feature)

5. When user has drawn in offline mode then after restoring the connection the content should be synchronized. After the plane of “User A” has landed and he has restored the internet connection, all the sketches should be synchronized without conflicts.

6. Application should be accessible without the Internet after the first full load. The problem with the web browser based application is that it is difficult to cache the base application content, scripts, images, fonts. The following resources do not often change and could be cached for a quicker web page opening on mobile.

4.2 Implementation in Swarm.js

In this chapter there is going to be a more specific introduction to the technology behind the Swarm.js, the Lambort vector clocks, specifiers, streams, the data modelling for Swam.js.

The meaning of the isomorphic library is that there is a common part of the code that could be run on the server and client side. That is one of the key ideas of Swarm.js and there are quite many components that could be run on both browser and server. This is only possible because of the use of code pre-processors like Browserify [19] or Webpack [42] which bundle all the module dependencies and allow using the Node.js module system in browser. This method allows using most of the Swarm.js modules also on browser side, if these are not specific to web or WebSocket server implementations.

The main components to analyse are the specifiers, clocks, hosts, streams and data structures.

4.2.1 Specifier

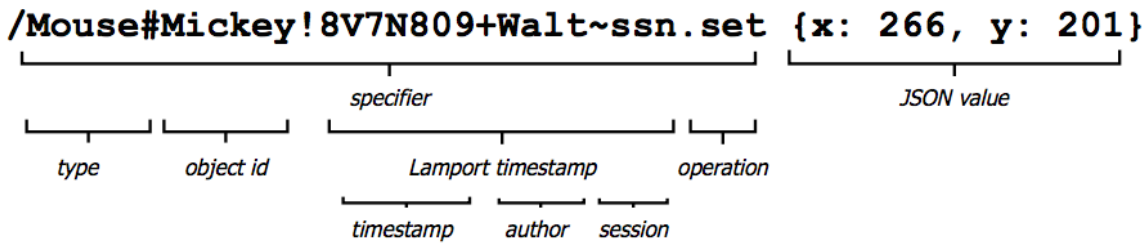

Figure 16: Specifier format in Swarm.js

Swarm.js uses the operational based data exchange described in chapter 3.2.2. This requires the creation of a special identifier for each operation. In Swarm.js the identifier consists of multiple values: class, object id, Lamport timestamps, operation and value. Example of a possible specifier can be found in Figure 16 [17].

4.2.2 Clock

```
// Second based
var Second = require('swarm').SecondPreciseClock;
var second = new Second('ristopid', 0);

var intReference = setInterval(function () {
   console.log('Second', second.seconds(), ' - ', second.issueTimestamp());
}, 100);

// Output
// Within same second generating more events increases the sequence in the end
// Second 40740380  -  2RQOS+ristopid
// Second 40740380  -  2RQOS01+ristopid
// Second 40740380  -  2RQOS02+ristopid
```

Snippet 1: Swarm SecondPreciseClock usage

With distributed systems it is almost impossible to synchronize all the clocks of the system to be exactly the same. Instead, it is only needed to know the partial order of the events, which are the bases of Lamport timestamps [13]. There is a possibility in Swarm.js to use different types of clocks such as the MinutePreciseClock, SecondPreciseClock and LamportClock. The specific clock should be chosen by the frequency of created events, for example if there are less than 64 events it is better to use a minute based clock. The second

precise clock allows creating more than 4000 events in a second, this should be enough in most cases. Following is the example of creating a second precise clock and how the time format looks.

The above Snippet 1 shows the clock format combined of time, sequence number and the process identification. The code snippet illustrates the case when there are more than one event generated in a second.

4.2.3 **Host**

Host is a singleton object which coordinates the streams, clocks and storages. Host is like a container object used on both the server and the client side. There could be multiple storages and uplink streams to send data. When adding the peer to peer support for Swarm.js, it should be controlled by the host object. Host instance is responsible for the application mode, connected, disconnected etc.

4.2.4 **Stream**

Streams are the wrappers for data connections between the host objects. The stream interface is Node.js stream API compatible which allows supporting many implementations.

Client side:

- PostMessageStream
- SockJSStream
- WebSocketStream

Server side:

- SockJSServerStream
- EinarosWSStream

4.2.5 **Storage**

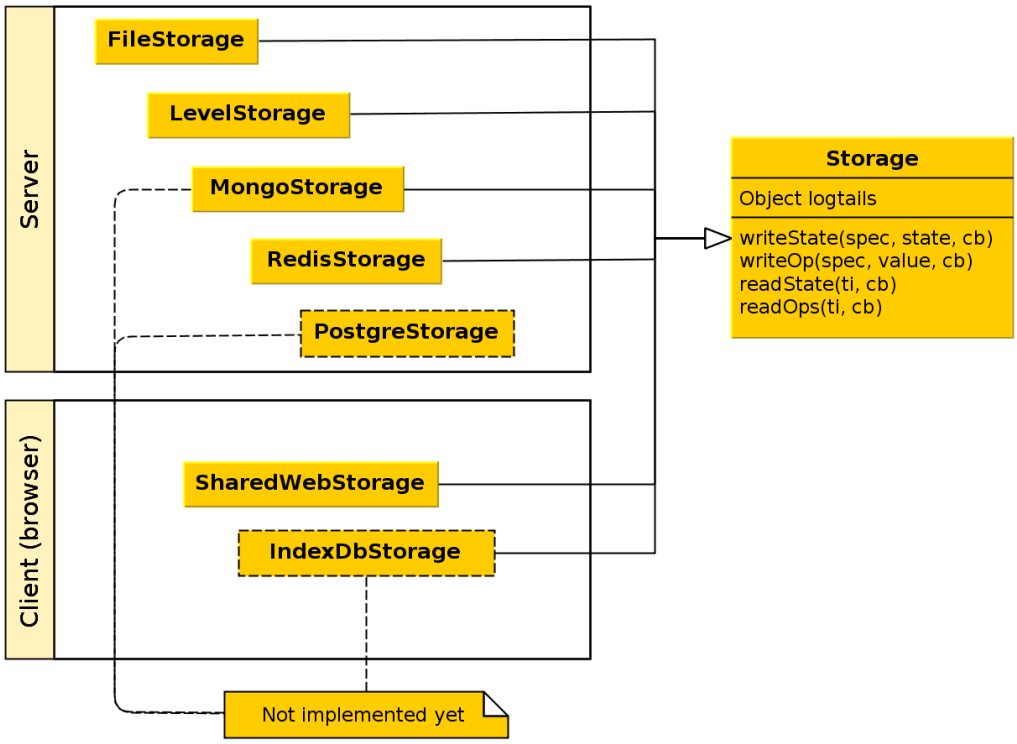

Figure 17: Server and client side storages

Data storages (Figure 17) are responsible for writing and reading the data on both on the server side and on the client side. The supported storages on the server side are FileStorage, LevelStorage, MongoStorage, RedisStorage. The client side has support from SharedWebStorage which uses internally local storage or the session storage API.

Currently, there is no storage support for a relational database, for example the PostgreSQL which needs to be implemented as a separate module. This would be needed if there was more data than fits into the memory or the file storage querying was too slow. Adding new storage support requires implementing the following four methods: state snapshots writeState, readState and appending the operation to log writeOp and readOps.

4.2.6 Data structures

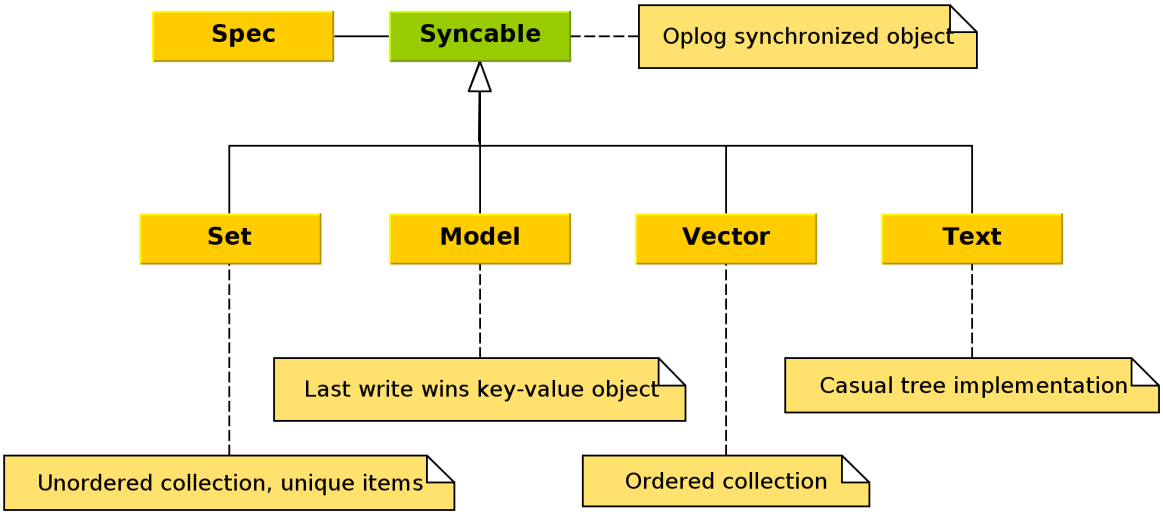

Figure 18: Data structures

These are the implemented CRDT data structures in Swarm, shown in Figure 18. All the structures extend the base object called *Syncable* which defines the base functionality of the substructures. Beside the main data structures there are some more CRDT-s that could be implemented in Swarm.js's counters, graphs etc.

4.3 License

Swarm.js's license is served as a MIT License, so all the changes and contributions to the project remain an also open source and share the same license. The developed whiteboard application and other related test projects' follow specific license rules, mostly MIT. The projects source code is available at *github.com/offline-ready-web*.

4.4 Technology stack

The example is implemented on top of these technologies, it has been mostly driven by the choice of Swarm.js. The programming language used both on server and client side is JavaScript following the ECMAScript 5 standards. The technology stack overview is shown in Figure 19.

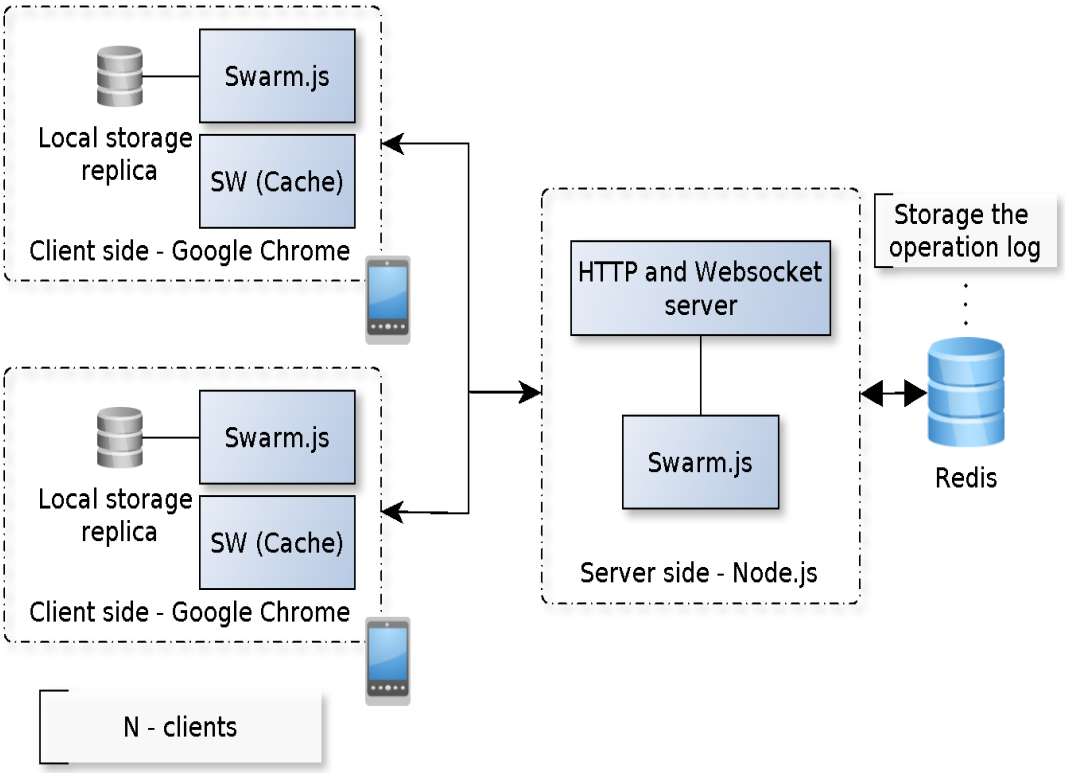

Figure 19: Technology stack

Client side:

- Google Chrome – Web browser v42 also the Chrome and Firefox Android version.

Server side:

- Node.js – Platform built on top of the Google Chrome's JavaScript engine V8, which allows developing fast, scalable and asynchronous solutions. The Node.js version used in our example is v0.12x. Node serves as the layer for the Swarm.js CRDT logics and storage. The server side could be easily scaled with multiple Node.js instances and a load balancer.

- Redis – is an in memory key value storage and cache. The Redis version used is v3.x In our example Redis is used for storing the operation's keys and values.

Node.js's modules list:

- Swarm.js – Responsible for the server and client side syncing.

- RxJS – The Reactive extensions for the JavaScript library, version used in our example is v2.5x. The library helps to handle all the data flows with common stream interface, drawing, user input, data exchange etc.
- browserify – Tool for bundling up all the client side dependencies, which allows using modules same way as in Node.js.
- hammer.js – Client side library for common event handling for touch events between different platforms Android, iOS, Windows Surface and browsers.
- simplify-geometry – Storing all the coordinates from the user input is mostly useless because the coordinates are repetitive and duplicated. To simplify the array of points the *Ramer–Douglas–Peucker* algorithm could be used for reducing the points that are needed to be base on the distance dimension.

4.5 **Data model**

One of the primary topics is what data structures to use for the implementation and how to store the data. Choosing correct data structures at first could be hard without having the domain specific knowledge. Also in Swarm.js the modelling process is not similar to relational databases where you can think of tables, columns and rows.

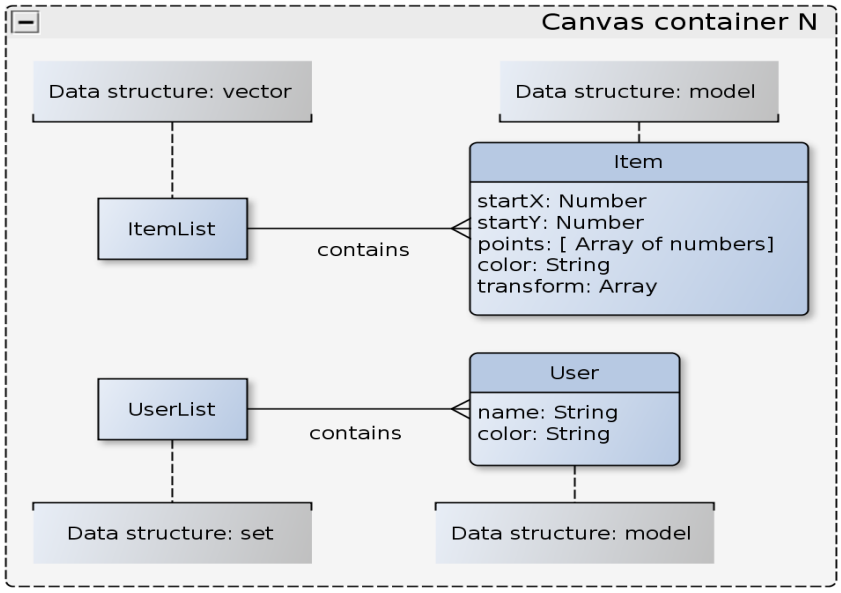

Figure 20: Data model

In our drawing application the main container is the canvas, each user could have multiple canvases and canvases multiple users (Figure 20). The users in canvas "N" are unique which means the *set* data structure guarantees that in *UserList*. *User* specific information is stored in *model* data structure, that allows the last write wins operations on model properties. Next key component for canvas is the *ItemList* containing all the drawn items. Since the order of drawn items it is important, the *vector* data structure is used. The *ItemList* itself is only container of the items, *Item* uses *model* data structure, that stores the points drawn on canvas. The *Item* points are pre-processed to lower the data storage overhead by removing unnecessary points and only store deltas from the start point.

The created data model is shared between the server and client side. By designing the offline application it is important do not to expose personal user or any other not needed information. In most use cases there is only a small fraction of data needed for the functionality.

4.6 **Security**

In order to make the communication between the server and client more secure the basic web security rules should be applied. Instead of using the HTTP the HTTPS connection should be used. For data streaming through the WebSocket it is a must have a "secure connection" transport WSS. A majority of exploits could be avoided by validating data sent from the client side in the server side.

Replicating the data to browser might cause another part of the issues:

1. Should the replicated data storage be encrypted [7]? If the encryption is one way to do it, then how and where should the private key be stored, so it would not be exposed to all the browser users?

2. The personal user data should, if possible never be replicated.

3. XSS and other script based attacks get the storage information from browser and forward it to attackers.

4. There is no default support for the access control which needs to be implemented by the case.

5. The browser side replication persists the user created data, if there are multiple users on single computer which gives access to the data.

6. Swarm.js's and the Node.js's weak spots are transferring larger chunks of data. The problematic spot of the data exchange is the message size. Swarm.js uses a text based string to transfer over the network which means the data needs to be serialized and deserialized using the V8 methods *JSON.parse* and *JSON.stringify*. When the object size gets large enough, for example 1MB the Node.js event loop starts blocking because these methods cannot be executed asynchronously and V8 has only a single thread. Meanwhile, the application can not accept any other interaction until the event loop is freed. That is the reason why on the Swarm.js's or application's side there must be limitations on the size of objects.

4.7 Tools for developing and testing

Solving the connectivity problems and testing the offline syncing is usually something that requires manual testing. Even then there is still a need for a testing plan and tools for simulating similar environment to production. Therefore, this chapter lists the tools for helping the network simulation in different levels like kernel and application. The tools should mainly follow the properties that are free/open-source and at least working in the Linux operating system. The following tools were used for testing the prototype application.

4.7.1 Google Chrome Developer tools

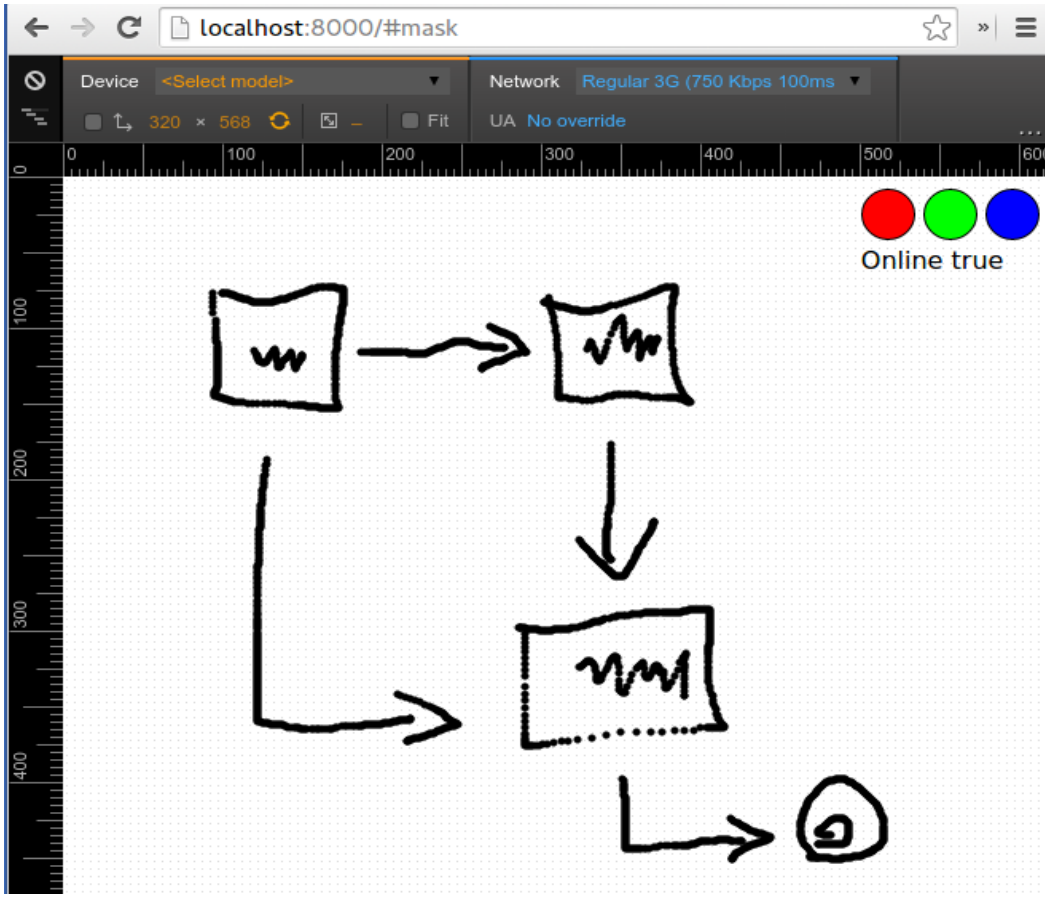

*Figure 21: Mobile device network settings **

** - Press F12, and click the mobile icon to enable device mode*

The developer tools are usually best suited for the web application developers. The developer tools provide an extensive list of features: memory time-line, CPU profile, network debugging tools. Small set also includes the tools for mobile devices (Figure 21),

like device based screen resolution emulation and the basic network quality emulation tool. The network tool allows changing the network throughput based on many well known mobile network service profiles like GPRS (50 Kbs 500 ms RTT), Good 3G (1 Mbs 40 ms RTT), offline mode etc. In the following test the 3G connection based synchronization were tested between the other user.

4.7.2 Network emulation tool - NetEm

NetEm [30] is a Linux command line network emulation tool that allows testing our application behaviour with various network properties. The tool is meant to emulate the connection similar to mobile connection. The emulation is done not only on the application level but also on the kernel level. With the Linux kernel version 2.6 the NetEm tool is by default enabled. NetEm has more control and variety options and settings to control the network compared to Google Chrome Developer tools. When there is a need for more control over the network properties, the NetEm should be preferred.

These are the main properties that we are going to emulate, the network interface name is the following *wlp3s0*:

Packet latency – with this command the latency for the packet is 100ms which is similar to a 4G network. The latency time can be set within the range of 20ms to 100ms.

```
tc qdisc add dev wlp3s0 root netem delay 100ms
```

Packet loss – simulate the random packet loss probability, in this example the loss 20%.

```
tc qdisc add dev wlp3s0 root netem loss 20%
```

Bandwidth limit – allow the limiting of the network speed to 2mbits with the latency of 100ms.

```
tc qdisc add dev wlp3s0 root tbf rate 2mbit burst 1024kbit latency 100ms
```

To remove all the applied rules.

```
tc qdisc del dev wlp3s0 root
```

4.7.3 End-to-end user test

A lot of functionality can be tested in the unit tests level which are usually faster to execute and implement, but there is still a need for integration test. The offline synchronization scenarios are these cases which need end-to-end user testing and validation by user. Also, the network emulation tool does not have an API that could make the testing quicker for the developers. For the prototype application the end-to-end tests are based on the main cases in chapter 3.1. Automated tests should be preferred over the end-to-end user tests. The next step is to validate the main test cases for the developed prototype.

Data synchronization test – for the test there is a need for at least two browsers. In this test case the Google Chrome and Mozilla Firefox are used. The test case has following steps:

1. Draw in Google Chrome text “Test”.

2. At the same time in the second browser the drawing should appear.

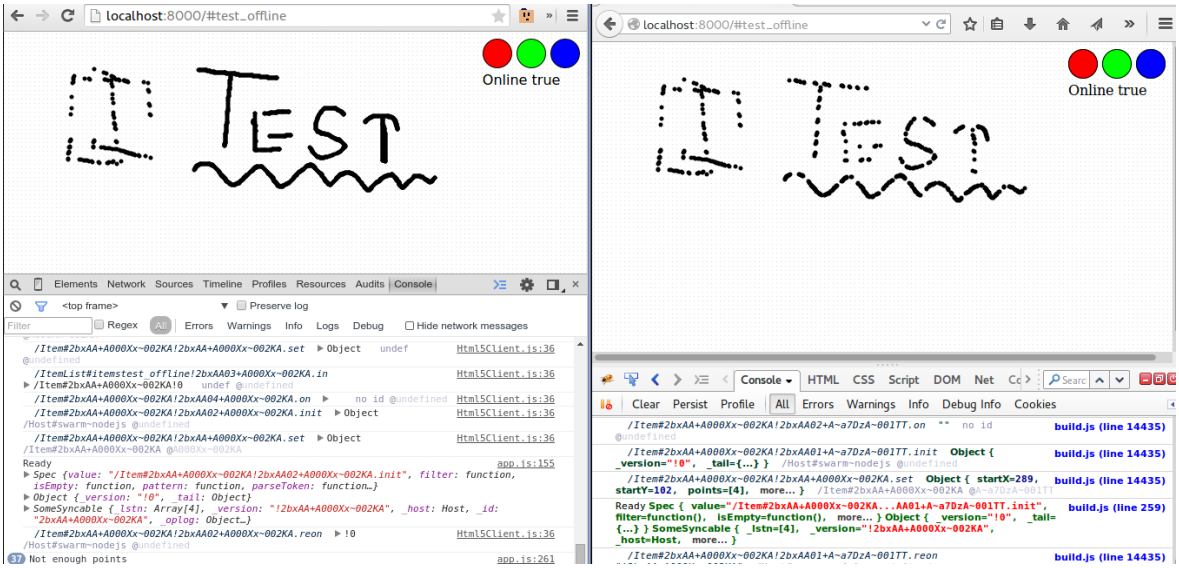

Figure 22: Data synchronization test between two browsers

Test result: Test showed that the synchronization worked between the two browsers, all the items and user’s data were replicated (Figure 22). The text drawn on the second board is not identical because of the user input is pre-processed by removing unnecessary points. This increases the performance and lowers the need for data storage size. Changing the drawing method to draw lines between the points would end up with a same result as in left

browser.

Offline data synchronization test – requires two separate browsers. The test case has following steps:

1. In one browser use the Google Developer Tools and change the connection to "offline" or disable the network connection in computer.

2. Start sketching in offline, the drawing should also be available after browser refresh.

3. On the connection restore the data synchronization to server starts and the drawing should appear also in other browser.

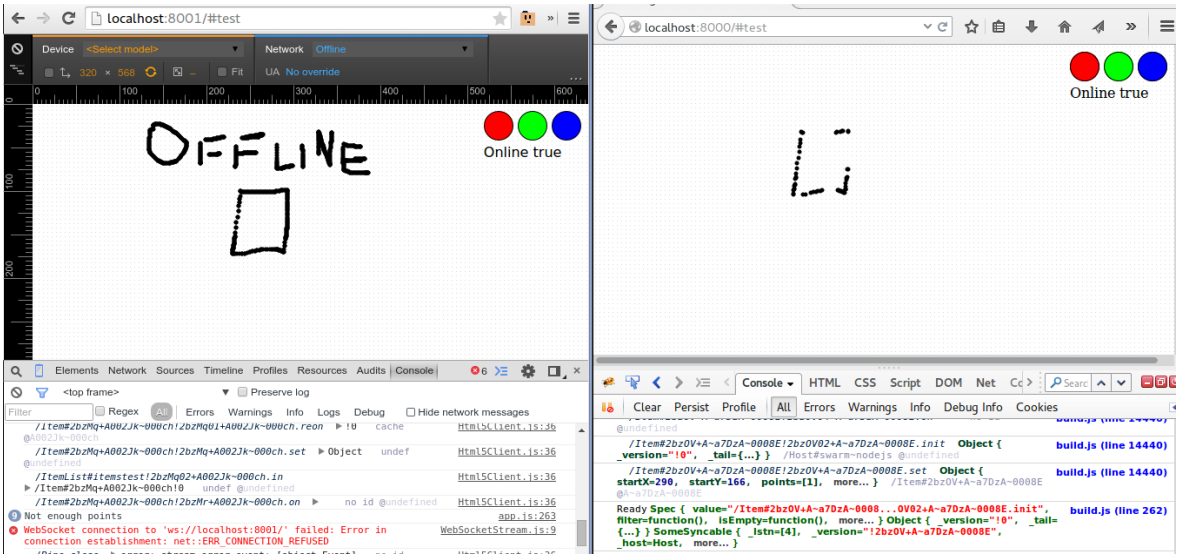

Figure 23: Offline data synchronization test

Test result: Test showed that the square were synchronized when both clients were online (Figure 23). But the text written in offline state did not appear in the second browser even after the switch to back online. After search for a bug in code, showed that the data were synchronized but the event handling on the client side were not handled, that caused the text to not redraw. This is a good example why the end-to-end tests are mandatory.

4.8 Browser technology

Over the recent years the browser technology has improved a lot by enabling more than before. Offline synchronization depends on many browser's API-s like Application Cache, ServiceWorker API, local storages etc. The concept of ServiceWorker takes an advantage of background processing that enables more complex web flows, battery optimized synchronization etc.

W3C specification of ServiceWorker API which has already been implemented in the latest Google Chrome v40+ web browsers and also in the Android WebView v42+ component. Allowing a similar native application life-cycle in web application, instead of just single page application which is loaded by the user. New caching API-s give the opportunity to create real offline applications. The current implementation Application Cache which is included in many browsers, but has problems with single page applications, also there is not much control over the cached items.

4.8.1 Browser storages for mobile devices

Table 5: Browser data storages

	Web storage	IndexDB	Web SQL
Storage size*	~5 MB	~50 MB*	~5 MB
API style	Syncrhonous	Asynchronous	Asynchronous
Mobile device support	All	iOS 8.1, from Android 4.3	iOS, Android
Query style	Key based	Key based	SQL queries

** - Every browser implementation has different condition and storage size implementations, for example IE11 has Web storage size ~10MB.*
*** - Mozilla Firefox will ask permission for storing blobs bigger than 50 MB.*

Local data replication in a mobile device depends a lot on the device capability of device using a local database. That is the reason the device should have fast data writing and reading operations. Currently, the common standards for browser implemented data storages are (Table 5):

1. **Web Storage** [5] (Local storage/session storage) – simple key value storage for storing string based content. The difference between the local storage and session

storage is timing. With session storage the data is stored until the last browser window is closed. On the other hand, the local storage persists data even if the browser windows are closed. The main API has the following methods: *getItem(key), setItem(key, value), removeItem(key), clear(), key(n-th index)*. All the following methods are synchronous which causes problems with larger data operations and blocks the UI. The specification suggested storage size is 5MB. The positive side of Web Storage is the good support from a variety of browser vendors.

2. **IndexDB** [28] – key value storage with more extensive support for different features: transactions, database version mechanism, cursors, asynchronous API etc. An improved database for more advanced usage, also the API is more complicated compared with LocalStorage. Besides, improved API the storage size is also much larger and could allow 50MB of data to be stored, with the user permission the storage size could be even bigger. The downside, compared with Web Storage, is the browser support which is not so good or could lead to unexpected behaviour in IE11, Safari 8 and iOS Safari 8.1.

3. **Web SQL [4]** – the idea is to allow data storage and query handling similar to SQLite. The W3C specification was abandoned in 2010, the only browser that supports the Web SQL API is Google Chrome but this is going to be dropped in the future versions.

The storage size limits are quite different between the browsers and platforms [2]. Choosing the right storage for a mobile device may not be so obvious, although the Web storage has the best support for different browsers, it is slower and the storage size is quite limited. There are many wrappers built to fix the compatibility issues and to use common API, one of which is *localForage* built by Mozilla [32]. Swarm.js has the implementation for the Web storage which is used also in the prototype application.

4.8.2 ServiceWorker API

The first steps towards offline based web application access were available with the browser technology called Application Cache [36]. This required all the resources and pages described in a manifest file that are needed in offline mode. On request, based on the

manifest file, the decision about which files are the cached resources and which are not is made. The application cache based solution is acceptable in use cases when the content is static based and does not change.

To get advantage of the cache control flow the ServiceWorker (SW) API must be used. SW is a special type of web worker that allows background processing even if the user is not on the web page. The main idea of SW is to give programmed control over the caching process and also the possibilities of using different storages for caching. Besides the caching there are many use cases where the mobile experience can be improved. Using the SW API also enables features for the browser Push API and Notification API. To make the synchronization process more battery efficient the one-off and periodic synchronization specification has been created [18].

From the security point of view it is required to use HTTPS, otherwise you will receive a security error and the SW will not be registered.

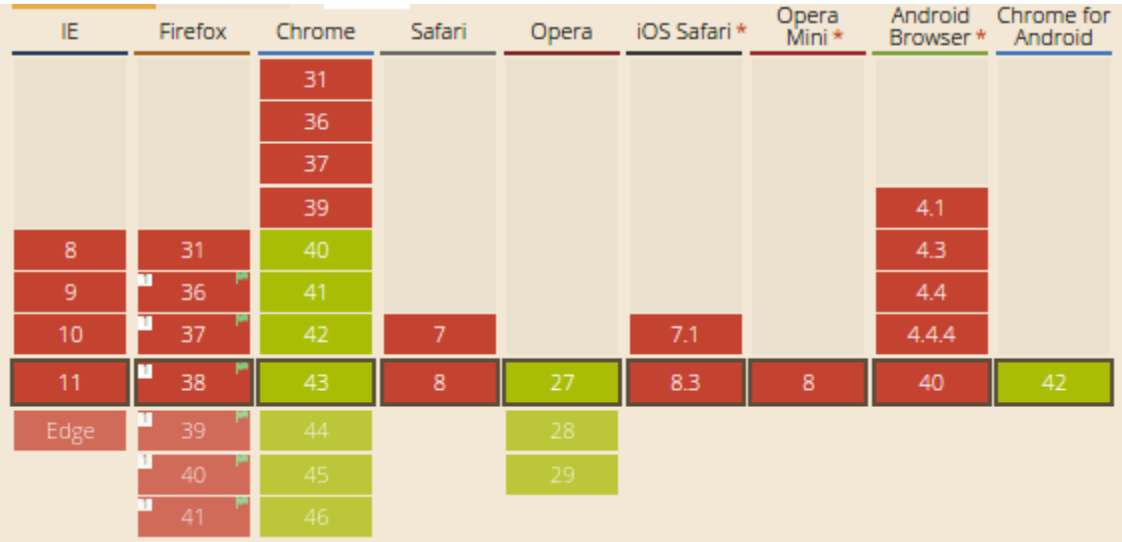

Figure 24: ServiceWorker API browser support

Currently the only browsers that support SW API are the Chrome version from 40, Opera 27, Chrome for Android and Firefox 36 with configuration flag *dom.serviceWorkers.enabled* (Figure 24). Due to the small number supporting browsers this technology is not yet widely used and should also be combined with fall-back to the Application Cache API.

5 Evaluation

It is important that the offline support should be implementing only the basic functionality. This should be done because of the security perspective as the personal user data is not needed to be exposed. By adding offline support and isolating the main functionality, allows the application to work even when the centralized synchronization server is down. From the development point of view adding offline support for application increases the complexity.

In the related work search and as a result of analysis the Swarm.js solution were chosen. The Swarm.js is in early stage but there is huge potential to represent different solution among the version control like data synchronization. With the recent two years improvements in browser technology mostly by the ServiceWorker API which enables more features for background processing. From the development side the developers should prefer modular solutions instead of the single monolithic solution, which does only one thing and could not be replaced when needed.

The contact with the Swarm.js members and the lead developer, researcher of Swarm.js PhD Victor Grishchenko has helped a lot. Although the documentation of library is not ready Victor has introduced the concept and the usage behind Swarm.js.

5.1 Further development

The further development for the application and more specific to Swarm.js.

The evaluation of the Swarm.js library:

- Add the support for the RDBMS for example PostgreSQL, MySQL etc. This allows storing the data in a more structured way, with the benefits of searching and additional metrics.
- Peer to peer network, instead of using the central server use other devices to synchronize the data.

- Operation log data compaction and compression, current approach allows growing the log size without any limits.
- Scalability is another large topic, both vertical and horizontal, the major bottlenecks are data transfers between database and the client. One single master database has to handle a lot of inserts so this would be one of the requirements when choosing a database system.

Developed application evaluation and further development of application:

- User metrics and analytic support, for additional the examples of how many users have used the application and the timings for each mode offline and online.
- History support, with undo and redo possibility. A more advanced feature would be timeline based undo and redo which enables to see and roll-back to a specific event in the past.
- Common shape library support, for example the basic shapes of UML or flow chart. Besides the basic shape support it would be possible to use machine learning algorithms to recognize user drawn shapes [8][14].
- Improve the rendering by using the WebGL technology instead of the HTML5 canvas which has better GPU usage on mobile devices. The HTML5 canvas is good for prototyping and some smaller scale solutions. WebGL uses quite a different API compared to the HTML Canvas and allows lower level control over the rendering.

6 Summary

Within the thesis the offline-capable collaborative web application solutions were analysed and implemented. The main occurring problems are caused by the bad network connectivity that affect all of us. Even though the mobile network coverage in Estonia is good there are still many places, indoors and countryside where connectivity is limited or not working at all. The situation in the world is more chaotic, the new generation 3G and 4G technology coverage is lower. Besides the connectivity there are also mobile device and network stack caused problems. That is the reason why the offline data synchronization is needed.

The main categories where offline data synchronization should be implemented are: collaborative whiteboard, email client, chatting, collaborative document writing applications etc. Some use cases were written about in detail to understand the specific needs.

In search for related work four quite different solutions were found and analysed. The analysed solutions include StrongLoop LoopBack, CouchDB + PouchDB, Firebase, Swarm.js. The positive and negative sides of the details of the solutions were analysed. Based on an analysis the best match was Swarm.js which uses the concept of conflict-free replicated data types(CRDT). CRDT-s lead to the analysis of different studies and use cases in distributed systems and networks.

Understanding the methodology and technology behind the Swarm.js CRDT is not enough. Based on the chosen solution for the development of a collaborative prototype application the author began to see possible issues and shortcomings. As the Swarm.js is in an early state there is no good documentation which made the development more time consuming and complex.

One of the result is prototype developed virtual whiteboard web application, which allows multiple users to work and sketch on the same board. The data synchronization is done using the Swarm.js library. The offline enabled mode works with the latest browsers where the ServiceWorker API has enabled, which is used for caching the resources.

There are many subtopics that could be further researched like the server side's scalability, security on the client side's, synchronization optimisations.

Kokkuvõte

Antud töö raames analüüsiti ja arendati kollaboratiivse veebirakenduse lahendust, mis töötab ka võrguühenduse puudumisel. Kehva või puuduva interneti ühendusega on kokkupuutunud igaüks. Kuigi Eestis on mobiilse interneti ühenduse kvaltiteet heal tasemel on siiski paljusid kohtasid, kus interneti ühendus on limiteeritud või puudub täielikult. Näiteks esineb mobiilse võrguühendusega probleeme ruumides ja maakohtades. Olukord maailmas on veelgi kaootlisem ning uuemate mobiilsete tehnoloogiate 3G ja 4G kasutatavus on madalam. Lisaks ühendusele on ka probleem mobiilsete seadmete ja interneti protokolliga seotud piirangutega. See on põhjus, miks on vaja leida jätkusuutlik lahendus andmete sünkroniseerimiseks piiratud võrguühenduse korral.

Põhilised kasutusalad kus kasutada võrgühenduseta andmete sünkroniseerimist on: kollaboratiivse virtuaalse tahvli rakendused, e-maili kliendid, suhtlus rakendused, kollaboratiivsed dokumendi kirjutamise rakendused jne. Probleemi mõistmiseks on detailsemalt lahtikirjutatud kasutuskohad.

Sarnase töö otsingul leiti neli erinevat lahendust, mida põhjalikumalt analüüsiti. Sarnase töö lahenduste alla kuulus StrongLoop LoopBack, CouchDB + PouchDB, Firebase, Swarm.js. Iga lahendus juures analüüsiti põhjalikult positiivseid kui ka negatiivseid külgi. Analüüsi põhjal selgus, et kõige paremini sobiv lahendus on Swarm.js, mis kasutab konflikti-vaba replikatiivseid andmetüüpe (KVRA). KVRA lahendusi kasutatakse hajutatud süsteemides ja võrkudes, mis sobib sarnaselt andmete sünkroniseerimiseks mobiilsetes seadmetes.

Teoreetiline arusaam Swarm.js tehnoloogiast ning KVRA metoodikast pole piisav. Välja valitud lahendus põhjal arendati kollaboratiivne prototüübi rakendus, mille käigus tuli välja probleeme ja puudusi. Swarm.js on alles algusjärgus olev lahendus ning antud hetkel puudub selge dokumentatsioon, mis muudab arenduse ajaliselt pikemaks ja keerulisemaks.

Töö tulemusena valmis veebipõhine virtuaalne tahvlirakendus, mis lubab mitmel inimesele samal ajal töötada ja joonistada. Andmete sünkroniseerimine põhineb Swarm.js teegi lahendusel. Rakenduse kasutamine võrguühenduseta on võimalik ainult uuemate

veebibrauseritega.

Tööst tuli välja mitmeid teemasid, mis vajavad põhjalikuma edasiuurimist näiteks: serveri poolne skaleeruvus, kliendi poolne turvalisus, andmete sünkroniseerimise optimeerimine.

References

1. Anant Narayanan, Where does Firebase fit in your app? [WWW] https://www.firebase.com/blog/2013-03-25-where-does-firebase-fit.html (25.04.2015)
2. Eiji Kitamura, Working with quota on mobile browsers, 2014
3. Greg DeMichillie, Welcome Firebase to the Google Cloud Platform Team [WWW] http://googlecloudplatform.blogspot.com/2014/10/welcome-firebase-to-google-cloud-platform.html (24.03.2015)
4. Ian Hickson, Web SQL Database [WWW] http://www.w3.org/TR/webdatabase/ (11.04.2015)
5. Ian Hickson, Web Storage [WWW] http://www.w3.org/TR/webstorage/ (11.04.2015)
6. Ilya Grigorik, High Performance Browser Networking, 2013
7. Jeff Cross, Ritchie Martori, Anant Narayanan, AngularJS 2.0 Data Persistence Design Doc - draft, 2015
8. Liam Don, Ioannis Ivrissimtzis, Multi-pen Sketch Recognition in a Learning Environment, 2009
9. Marc Shapiro, Nuno Pregui¸ca, Carlos Baquero, Marek Zawirski, A comprehensive study of Convergent and Commutative Replicated Data Types, 2011
10. Mikito Takada, Distributed systems: for fun and profit, 2013
11. Nabhendra Bisnik, Protocol Design for Wireless Ad hoc Networks:The Cross-Layer Paradigm, 2005
12. Nuno Pregui¸ca, Joan Manuel Marqu`es, Marc Shapiro, Mihai Leia, A commutative replicated data type for cooperativeediting, 2010
13. Paul Krzyzanowski, Clock Synchronization, 2002
14. Shikha Garg, Gianetan Singh Sekhon , Shape Analysis and Recognition Based on Oversegmentation Technique , 2012
15. Stephane Weiss, Pascal Urso and Pascal Molli, Logoot: a Scalable Optimistic Replication Algorithmfor Collaborative Editing on P2P Networks, 2009
16: The National Audit Office, Effectiveness of the development of a broadband network or high-speed Internet, 2015
17. Victor Grishchenko, Swarm/Lamport timestamps [WWW]

http://swarmjs.github.io/articles/lamport/ (17.04.2015)
18. Background synchronization explained [WWW] https://github.com/slightlyoff/BackgroundSync/blob/master/explainer.md (18.05.2015)
19. Browserify [WWW] http://browserify.org/ (17.04.2015)
20. Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update 2014–2019 White Paper [WWW] http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.pdf (31.03.2015)
21. DokuWiki [WWW] https://www.dokuwiki.org (02.05.2015)
22. EMT 4G coverage area [WWW] https://www.emt.ee/en/firmast/leviala (31.03.2015)
23. Facebook [WWW] https://www.facebook.com/ (01.05.2015)
24. Firebase - Offline Capabilities [WWW] https://www.firebase.com/docs/web/guide/offline-capabilities.html (23.03.2015)
25. Firebase - Open Data Sets [WWW] https://www.firebase.com/docs/open-data (22.03.2015)
26. Gartner Says Global Devices Shipments to Grow 2.8 Percent in 2015 [WWW] http://www.gartner.com/newsroom/id/3010017 (21.04.2015)
27. Google Docs [WWW] https://www.google.com/intl/en/docs/about/ (01.05.2015)
28. Indexed Database API [WWW] http://www.w3.org/TR/IndexedDB/ (24.03.2015)
29. Instagram [WWW] https://instagram.com/ (01.05.2015)
30. Linux Foundation - netem [WWW] http://www.linuxfoundation.org/collaborate/workgroups/networking/netem (12.04.2015)
31. List of countries by number of mobile cellular subscriptions [WWW] http://en.wikipedia.org/wiki/List_of_countries_by_number_of_broadband_Internet_subscriptions (21.04.2015)
32. localForage - Offline storage, improved [WWW] http://mozilla.github.io/localForage (26.03.2015)
33. Loopback: Advanced topics - sync [WWW] http://docs.strongloop.com/display/public/LB/Advanced+topics+-+sync (11.04.2015)
34. MediaWiki [WWW] https://www.mediawiki.org (02.05.2015)

35. Mobiilse interneti andmesidekiirused Eestis [WWW]
http://www.tja.ee/public/documents/Elektrooniline_side/Sideteenused/Mobiilse_interneti_andmesidekiirused_Eestis_jaanuar_2015_F.pdf (31.02.2015)
36. , Offline Web applications [WWW]
http://www.w3.org/TR/html5/browsers.html#offline (11.04.2015)
37. Redis [WWW] http://redis.io/ (06.05.2015)
38. Riak [WWW] http://basho.com/riak/ (05.05.2015)
39. Riak - Data Types [WWW] http://docs.basho.com/riak/latest/theory/concepts/crdts (08.04.2015)
40. SoundCloud - roshi [WWW] https://github.com/soundcloud/roshi (25.03.2015)
41. The LoopBack framework [WWW]
http://docs.strongloop.com/display/public/LB/LoopBack (24.04.2015)
42. Webpack - module bundler [WWW] http://webpack.github.io/ (17.04.2015)
43. WebRTC 1.0: Real-time Communication Between Browsers [WWW]
http://w3c.github.io/webrtc-pc/ (04.05.2015)
44. Wiki [WWW] http://en.wikipedia.org/wiki/Wiki (02.05.2015)