Antti Karjakin

Storing Data to Persistent Memory of Mobile Phone

Helsinki Metropolia University of Applied Sciences

Master’s Degree

Information Technology

Master’s Thesis

8 June 2015

Helsinki
Metropolia
University of Applied Sciences

Abstract

Author(s) Title Number of Pages Date	Antti Karjakin Storing Data to Persistent Memory of Mobile Phone 83 pages + 1 appendices 8 June 2015
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Mika Ruottinen, Product Development Manager Ville Jääskeläinen, Head of Master's program in IT

The performance and visual appearance of mobile phones have been and still are developing fast. This development has made it possible to consider a smart phone as a daily tool for many people. Mobile phones are usually carried along everywhere, even for places where network connection is not available.

A native mobile application is capable of showing the data stored to the mobile phone even when the network connection is not available. Browser based software was not able to do the same effectively before the HTML5 standard.

Granlund Oy is Finnish company specialized in design, consultancy and software services. In the company a maintenance management application called Granlund Manager is being developed and distributed as a SaaS solution. The Granlund Manager needed a mobile application that could be used without network connection.

The objective of this thesis was to get acquainted with an offline data support in the mobile applications. The application can be native, hybrid or HTML5 based. The thesis focused on storing and viewing the data and it researched how the offline capability can be implemented when working with HTML5 based applications.

The study started by defining the requirements of the application that needs the offline capability. From the offline capability point of view storing the data and securing HTML5 application logic were covered. Also a brief lookup for detecting the network connectivity of a mobile phone was studied. The data storing mechanism that existed before the HTML5 standard was researched. Then solutions from the selected platforms were covered before covering the options offered by the HTML5 standard.

The research indicated that the mobile market in Finland is divided slightly differently than globally in the world. The market is divided so that the three most considerable options are Google Android, Apple iOS and Windows Phone in that order.

The thesis indicated that it is possible to store data into the persistent memory of a mobile phone and that data can be viewed without a network connection in all of the selected mobile platforms. The thesis also proved that an HTML5 based application can be as offline capable as a native mobile application. A lot of knowledge on platform specific and HTML5 specific solutions was gained during the research process.

Keywords	HTML5, persistent memory, mobile application

Contents

List of Figures

List of Listings

List of Abbreviations

.NET	.NET Framework developed by Microsoft
API	Application Programming Interface
C#	a programming language created by Microsoft
CMSS	Computerized Maintenance Management System
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HVAC	Heating, Ventilation and Air Conditioning
IE	Internet Explorer
JNLP	Java Network Launch Protocol
LINQ	Language Integrated Query
SaaS	Software as a service
SQL	Structured Query Language
SQLite	relational database management system
SQL CE	SQL Server Compact Edition

1 Introduction

Currently real estate companies that have many buildings and facilities rely on computerized maintenance management systems (CMSS). The other option is to use some sort of facility management software. Usually these software systems are desktop or internet applications.

On the application background there is a data storage which can hold data in many various forms. That data is collected and reported in many different ways. Some of the data are collected and stored automatically but the main data is collected by users from all around the facilities.

These facilities and buildings are having different kind of machines which take care of HVAC (heating, ventilation and air conditioning) needs. Those machines contain different kind of data which is read and stored to CMSS or to facility management software. Modern HVAC machines contains an interface to collect that data automatically. However many current facilities and buildings especially old ones do not have modern HVAC machines and all data needs to be collected manually.

People collecting that data to CMSS do not carry a laptop with them while working because laptops are too heavy, so another solution is needed. The solution could be a handheld device with an application for storing data to the server and that could view charts and reports from that data. Making a mobile user interface of the Internet application for the handheld device is not a solution because there is no connectivity to Internet in all places of the facilities.

The solution has a special need. The solution should be able to work both while it has a connection to the Internet and while it has not. The main technological problem is how to store data while the user is out of the network coverage area. This scenario is very common, because many of the HVAC machines are located at the basement level or on another area where it is not possible to use networks.

The solution should act as a network independent data collector. So people working on buildings could go to a worksite and read the data and input it using a handheld device. The Application should be able to handle the offline situation and store the data until the device is back on the online status and ready again to download the collected data to the server.

1.1 Company and Project Background

Granlund is a Finnish company specialized in design, consultancy and software services. Granlund is the leading expert in all its specialty areas in Finland. The software development department consists of two development teams. One of the teams has the responsibility to develop a tool for maintenance management. That tool is known as Granlund Manager.

Granlund Manager is a browser-based flexible maintenance management application. It offers solutions for the following areas of maintenance: Maintenance activities and maintenance manuals, energy management, long term planning and service requests.

Granlund Manager controls the life cycle, utilizes data models, motivates the users and is able to monitor the quality of maintenance in real time. Granlund Manager is distributed mainly as a SaaS (Software as a service) solution and it serves nearly 300 organizations in Finland and about 100 organizations abroad.

Granlund Manager was designed and developed to be used on the desktop browsers. In the company the need of a mobile capable solution was detected. The needed mobile application has to be capable of showing and storing data at places where there is no network coverage available.

1.2 Objective, Scope and Structure of Thesis

The objective of this thesis was to research the mechanisms to store data to the persistent memory of mobile phones. It covers data storing mechanisms of native and HTML5 based applications. Offline capability in an HTML5 based application is also an important object as the new standard gives tools to solve the need of an offline application.

An overview on how to save data to mobile phones on selected mobile platforms is covered along the research. Tools to manage browser based application resource usage are researched from the developers' point of view. Mechanisms to recognize network availability and HTML5 standard solution to save application logic to the mobile phone are also researched.

The research question of this thesis is how to store data to persistent memory of mobile phones, making it possible to use the application without network connection.

The research begun by inspecting the mobile operating systems market in Finland. With that research the company wanted to be sure that the correct mobile platforms are being covered during the research. Then a research about different mechanisms to save data to mobile phones was done by using information from the literature and from the internet. After the platform specific solutions were researched the possibilities of the HTML5 solutions were studied. Based on the results gained from this research the decision about implementing the offline capability will be made.

This thesis is written in eleven sections. Section 2 describes the research that was done earlier to define the requirements for the offline capable application. Section 3 gives an overview of the theory and briefly introduces the options used before HTML5. Section 4 presents tools to investigate resource usage in the browsers for developers.

Section 5 investigates how to identify the network availability on each platform. Section 6 researches the application manifest of HTML5. Section 7 and 8 examine solutions to save different kind of data to a mobile phone and researches the database versioning feature that is common for a few offline data solutions.

Section 9 presents the summary of the study and future steps for research and development and also summarizes the study with conclusions.

2 Mobile Applications

This section describes what mobile operating systems are used in the world and in Finland more particularly. It also explains how web applications work when compared to mobile applications and the requirements given to the application are presented in this section.

2.1 Mobile Platforms in Finland

In order to meet the challenges described in Section 1 the company decided that a new mobile application is needed. One of the first questions was which platforms the application should support. The company has not developed any mobile applications in the past so all the research data was needed to help deciding the supported mobile platforms.

When examining the smart phone mobile operating system usage worldwide it can be clearly see that two operating systems (Google's Android and Apples iOS) share almost four-fifths of the mobile market. This is shown in Figure 1 [26]. The Series 40 operating system does not have all the capabilities that modern smart phones have so technically Windows Phone is the third popular smart phone operating system.

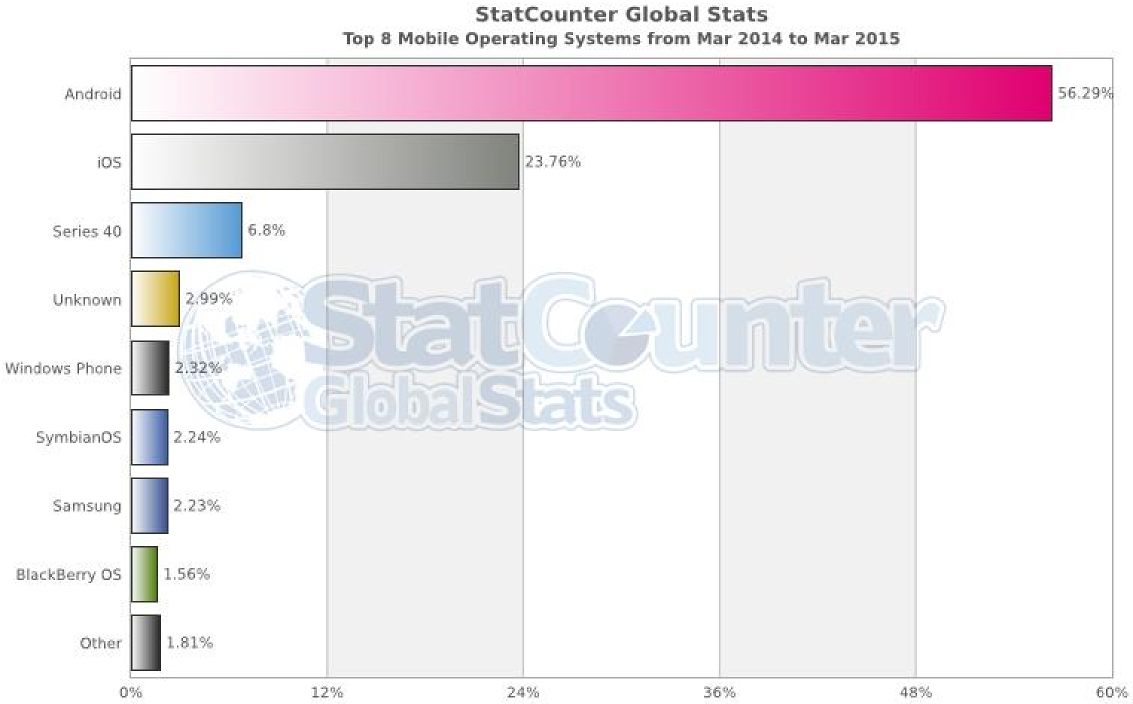


Figure 1: Top 8 Mobile Operation Systems from Mar 2014 to Mar 2015 [26]

In the Finnish technology industry Nokia show the way a long time. This has affected the Finnish mobile market and customer behaviour. Many Finnish companies still offers Microsoft’s Lumia phones for the employees as phone benefit. This can be clearly seen in the Finnish mobile operating system market as the Windows Phone is clearly in the third place, as shown in Figure 2 [26].

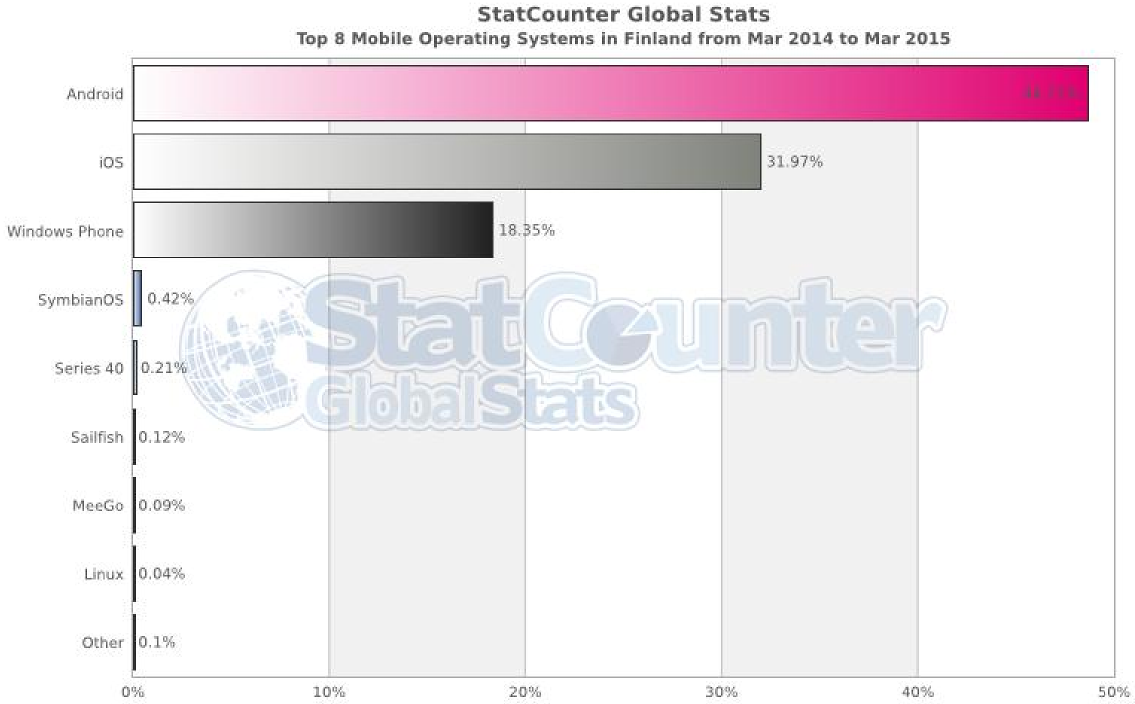


Figure 2: Top 8 Mobile Operating Systems in Finland from Mar 2014 to Mar 2015 [26]

Being aware of the mobile operating systems market in Finland, designing and developing the mobile application to be compatible with the Android and iOS platforms would not be enough. If the Windows Phone platform is not supported, almost one-fifths of the mobile users could not use the developed application.

Devices with Android or Windows Phone operating system also tend to be cheaper than iOS platform devices. So at the beginning it was stated that the application development should be prioritised to work in the Android and Windows Phone platforms rather than in the iOS.

When developing an application that is being piloted and tested in the Finnish market it was quite clear that the application should work in all of the three major mobile operating systems.

2.2 Native, Hybrid or HTML5

The mobile application development applications can be divided into three categories, native, hybrid and HTML5. Native and hybrid applications are distributed through platform specific marketplaces such as Google Play, Apple Store and Windows Phone Store. Native or hybrid applications that are not distributed through platform specific marketplaces should not be installed to the mobile phones because of security risks.

When trying to choose the right approach for the new application a few key elements have to be considered. First, for which platforms the application is being targeted. Native applications have to be written separately to each platform. If there are more than one targeted platform, the native approach means that as many applications have to be developed. If the supported platforms cannot be restricted only to a few it could lead into a lot of work: If the development team is not having enough experience on multiple platforms learning them will take time.

Hybrid applications are share the same code base and usually some software is used to compile the application to each platform. Because they share the code base there could be some limitations with the features compared to the native applications [5].

Hybrid applications can also have a platform specific code. To gain advantage from this feature there should be enough understanding on each platform where the application is being developed. Just like when developing native applications.

Secondly, one should study how many native application features are needed and whether there are alternative solutions to them. A native application feature can be recognized for the need of using some hardware component. For example does the application having support for the phones camera, so the application could ask the user to take a picture and that picture could then be easily added to the application?

The third main element is the performance. Are there heavy operations that have to be performed in the mobile phone? Or are there heavy animations presented in the application? Native applications have more processor and memory available than the applications running in the mobile phones browser. Those two things have quite a significant role what comes to animation smoothness.

The application monetization is also an important element. Before starting application development the developers and the business people should share the idea how the money can be collected from the developed application and how the application is being distributed to the clients.

Very early in the development it came evident that the main features of the application have to work on all main mobile platforms in Finland (Windows Phone, Google Android and Apple iOS). Sharing the application through marketplaces would mean a lot of administrative work and some extra expenses yearly. When the application is in the marketplace the company will not have a full control when the application updates are being distributed to the end users. The HTML5 solution does not have any of these downsides. It can be updated every day and the company has full control when the updates are applied.

From the developers' point of view, the development tools are rather important to be considered. The development team was currently working with the Visual Studios and using the C# language. Adapting new tools and techniques will lead into a longer development period. It would be quite a steep learning curve if they had to start develop a native iOS application.

High usability is also an important point in the application design process. Knowing the fact that native applications can utilize the resources of the phone more effectively than the browser based application a few HTML5 proof of concepts were made. With the proof of concepts it could be substantiated that the HTML5 application can have sufficient performance without too much impact on the usability. On each platform a link to the home screen can be made to ease the access to the application and the application seems more like a native application.

With these requirements and having the need that the application should work on all three platforms there were enough reasons to choose the HTML5 application for the project.

2.3 Application Structures

A common structure of a modern web application is that the business logic and data is stored in the server and the view which the user is able to see is constructed combining information from these two. This kind of simplified structure can be seen in Figure 3. The view contains events that call business logic to make changes to the data. Connection between the user's browser and the server has to be available every time when the user makes any action.

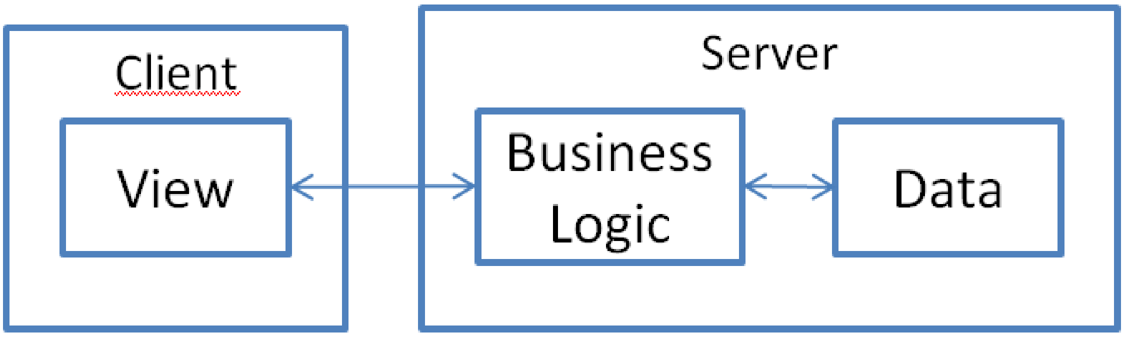


Figure 3: Simplified web application command chain

If the web browser is closed and opened again while there is no network connection the page cannot be opened. If the page is in the browser cache it might be possible open the page but it would be empty because the browser is not caching the data that the page is viewing.

Mobile applications can use the same business logic and data as the web application. The mobile application has its own view and events. The mobile application can be implemented in the same way as the browser's view but because mobile phones have a lot smaller display than computers usually do another approach is used. The data can be stored to the mobile application and then be viewed on smaller chunks. Application network usage can also be lowered when the data is transferred to the mobile application only when needed and only the necessary data changes are transferred through the network.

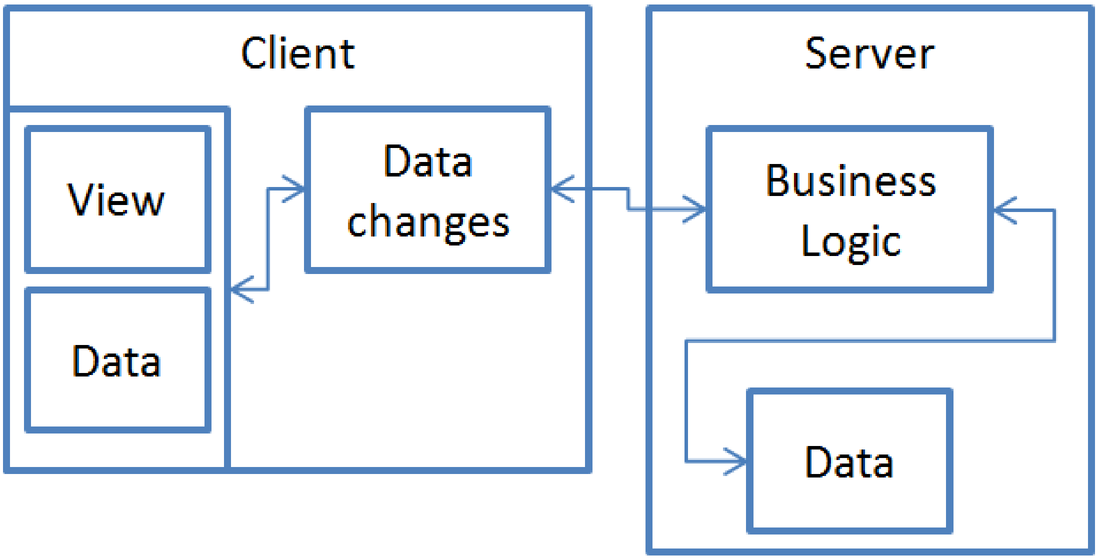


Figure 4: Simplified mobile application command chain

After the data is transferred from the network to the mobile phone the application can be closed and opened again. If the data is stored to the mobile phone the application is able to show the data that is stored to the mobile phone immediately. Even if there is no network coverage available the data can be viewed because all the necessary components are stored to the same device. A simplified structure of a mobile application can be seen in Figure 4.

In every mobile platform it is possible to develop a mobile application that does not need network connections at all. All the data that this type of an application shows is produced within the same mobile phone. For example a simple game can contain all of its logic and data in the application itself that is installed to the mobile phone.

Options to save data to the user's web browsers were very limited before the HTML5 standard was introduced. In the mobile platforms saving data to the actual device has been possible a lot longer. With the mechanisms introduced in the HTML5 standard it is now possible to store data to the user's browser and build applications that could work without network connection.

3 Storing Data Permanently to Client

The most common way how internet applications work is that most of the data is located on the server and only the data that can be seen in the view is transferred to the user's internet browser (client side). When an application has to be usable without internet connection it cannot rely on data or logic that is stored outside the application.

Without internet connection the application behaviour is limited. Resources that can be accessed while a user is using the application without network coverage are limited, too. The key data should be available and stored somewhere on the client side to ensure that the user can still work with the application while he or she has no internet connection. Also the key components of the application logic should be available.

From a security point of a view it should be always considered more than once what kind of data is saved into the actual devices memory. Very sensitive data should not be saved without encrypting to devices that can be lost or stolen [6, 37].

3.1 HTTP

HTTP Cookies is the oldest technology that can save data from the server to the user's web browser. Support for HTTP cookies was introduced in the Internet Explorer 2, back in the year 1995. A Cookie is a small data part that is sent from website to user's network browser.

When considering using cookies, the developer should know the limitations of cookies. A cookie should not be larger than 4 096 bytes (as measured by the sum of the length of the cookie's name, value and attributes [18]), because that is the size that is set on the specification. So if a developer makes cookies larger than 4 096 bytes there is a risk that some browsers deny them.

The specification also states that every browser should have a capability to handle 50 cookies from each domain and also be able to handle at least 3 000 cookies in total.

Cookies are transferred between server and client on the every request that the browser makes to the server [18]. They are also transferred inside HTTP headers so using a lot of them will slow down the communication between the server and the client.

If there are a lot of requests between a browser and a server, a huge number of cookies will result a huge bandwidth usage. It will affect the network availability and running costs for both participants. To the one that is serving the data and to the one that is querying the data. A user must also have a very fast connection to ensure the smooth data transitioning.

Cookies can also store some data to the client. They are counted as a possible offline storage even if the amount of data they can store is not much.

3.2 Browser Plug-ins

Browser plug-in is a small component that will extend the web browsers capabilities. Often the user has to install them separately. There are a lot of plug-ins available to the desktop internet browsers, but the two most popular plug-ins are Adobes Flash and Oracles Java. Google was developing their own plug-in called Gears from 2007 to 2010. Microsoft has been developing the Silverlight plug-in. Plug-ins has been available mainly for the desktop browsers only.

Some browser plug-ins have a mechanism to save data to the user’s computer and also to read the data from the disk. Some plug-ins can read and write to separate files too. These files can be used with other applications as well.

Adobes Flash introduced a mechanism called local shared object. It has also been called as flash cookies because of their similarity to http cookies. There have been also a lot of criticisms about their security. Mostly they were used to save some data of the application to the user’s computer not all of it. For example a flash based game could have a separate leader board for the local users only at the user’s computer.

With the Oracles Java it is possible to write Java applets that are placed on the web page. These applets can have access to the user’s local files. The applet must request permission from the user before it can access data on a user’s disk.

"*The security model behind Java applets has been designed with the goal of protecting the user from malicious applets.*" [20]

In order to get access to the user's local files the Java applet has to be started using Java Network Launch Protocol (JNLP). The applet still needs the user's permission before it can really access the user's local file system and read and write data to the files.

Microsoft Silverlight also has a mechanism to read and write to local files. This feature was introduced with the Silverlight 5. The user has to approve the Silverlight 5 application as a trusted application before it can have access to all of the user's files.

"*Trusted applications are applications that you configure to require elevated trust. These applications have special installation requirements, but can bypass some of the restrictions of the default Silverlight security sandbox. For example, trusted applications can access user files and use full-screen mode without keyboard restrictions.*" [21]

Previous versions of the Microsoft Silverlight could access only files located under the user folder (also known as C:\Users). The Silverlight 3 could save files to anywhere on the users local file system by using the save file dialog. It can be used only for saving files because the security model that Silverlight was obeying meant that the path where the file was saved was never transferred back to the actual application.

In 2007 Google released their own browser plug-in called Gears. It extended the browser's capabilities to the new levels. It offered a way to store some of the files to a local file system and they could be retrieved while there were no network connections. As the HTML5 standard was developed Google decided to stop working with the Gears plug-in and they ended its development in February 2010.

The plug-in was earlier preinstalled on the Google's Chrome browser. It was removed from the base install in the release of version 12.0.742.91 in June 2011.

3.3 Mobile Platforms

Mobile applications can be used without internet connection after they are installed to the phone. The difference to the HTML5 applications is huge because only the data that is located in the network is inaccessible. Applications different views and logic is always accessible and can be used for smooth user experience even while working without a network connection.

Many of the current html applications contain only the data they show and only small part of the actual logic. Main business logics are held on the server side of the application and never transferred into the user's browser.

3.3.1 Google Android

The Android development environment contains multiple frameworks to store data to the actual device. These are known as Shared preferences, Internal and External storage and also as SQLite Database. The developer should handle them all and also have knowledge which framework will give the best support and performance for the application.

Shared preferences framework in the Android environment is a general framework that allows saving and retrieving persistent key-value pairs of primitive data types. So it is a lightweight mechanism to store a known set of values [1]. It is very handy when saving application settings or user's preference. For example if there is a possibility for a user to decide if he wants to use metric or imperial units in the application the decision can be saved into the shared preferences.

If the application needs to work with several files, for example if the main focus of the application is on to handle media files (music, photos etc.) the developer has two options. To use the internal storage or external storage framework to gain access to the files in the device.

By default, files saved to the internal storage are private to that application and other applications cannot access them (nor can the user). When the user uninstalls the application, these files are removed [2].

External storage is also used to store files, but unlike in the internal storage, files in the external storage can be accessible outside of the application. It can be used even from a desktop computer if the phone is plugged into the computer. External storage can be located at the removable media or non-removable media (memory card in the mobile phone).

Extra caution is needed when using external storage that is located at the removable media. The user could somehow make the removable external storage inaccessible at any point of the applications flow by removing the removable media from the phone.

In the previous version of Android applications to read or to write to the external storage the application manifest must have a permission to read or write to external storage. At this point the version 4.4 of Android does not need the permissions if the application uses external storage files only for inside the application. Permissions must be set in the application manifest if there is a reason to use the files which the application was made outside the actual application. This is a good example how fast things develop in the mobile field.

The database support on Android devices is built using SQLite framework. Any database that is created in the application is accessible by any component inside the application, but not outside the application.

3.3.2 Apple iOS

Apple has developed a few options to save data into the iOS devices. To save the simplest data a property list can be used. Property lists holds data as key value pairs. Property list is an xml file that is located in the applications resources folder.

The need of storing more complex objects than key value pair objects can be done using the NSCoding protocol. With the NSCoding protocol objects can be serialised and deserialized into the persistent memory. It is a good mechanism when there are not many objects and the device performance is not affected too much.

When storing multiple objects the Apples solution is The Core Data Framework. “*The Core Data framework provides generalized and automated solutions to common tasks associated with object life-cycle and object graph management, including persistence.*” [16]

Core Data framework is very powerful tool for iOS developer and every iOS developer should be familiar with it. It is not a relationship database, it is more. With the help of the core data model there is no need to glue code between the applications user interface and its data model, because Interface Builder tool provides pre-built Code Data controller objects.

Apple iOS includes also the SQLite library and it has the same limitation as in the Android environment. Another applications SQLite database cannot be used with another application. It can be found in the library frameworks and it is called libsqlite3.dylib.

3.3.3 Windows Phone

Isolated storage in the Windows Phone platform is a piece of space at the phone hard disk. It offers unlimited data for every application (of course in the limitations of the available data in the phone). Data inside an isolated storage is saved using the application id, so the data belongs to a single application. Data in the isolated storage cannot be shared between two applications.

When updating the developed application one considerable thing is that files that are saved into the isolated storage are not updated from the marketplace. So on the every application update the developer needs to manage the situation inside the application logic if files are needed in the application constantly.

Database support in the Windows phone platform is done by using SQL CE (SQL Server Compact Edition). It has also a support for LINQ to SQL. With the help of the community there is also a possibility to use SQLite database with the windows phone 7 and 7.5.

The Windows Phone platform has its own database framework and it is more than recommended to use it. Community libraries can be strong when they have support by the community but often they cannot compete with built in frameworks.

3.4 HTML5

The offline capabilities of HTML5 can be divided into two different logical components. The first logical component focuses to store data into the user's browser (and to the user's mobile phone at same time). The second logical component is the Application cache that focuses to ensure that the application logic is available even when there is not network coverage.

"*The distinction is core application logic versus data. Application caching involves saving the application's core logic and user-interface. Offline storage is about capturing specific data generated by the user, or resources the user has expressed interest in.*" [3]

While native mobile application logic is always available in the phone's memory, web based application logic is not available in the browser by default. The HTML5 enables control for the developers to choose which files should be available always for the user's browser. Every logic part that is needed while the user uses the application without network connection has to be in the application cache manifest. This ensures that the application logic is saved into the user's browser and can be used while the network connection is not available.

The first offline data storage method that the HTML5 standard introduces is called web storage. It is similar to Android's shared preferences since it is also a key/value pair mechanism. A big difference comes from the value types since the web storage supports only string values. So every object or a value that is stored to the web storage must be serialized first.

The web storage is further divided into two separate parts, to the local storage and to the session storage. Values that are stored into the local storage can be accessed from every session that is opened from the current browser. So the value saved today can be obtained tomorrow or later. Values that are stored into the session storage can be accessed only inside the same session where they were added.

When the session ends (for example the user logouts from the service) the local data is cleared. It is also cleared when the user closes the window where the HTML5 application runs (even the actual session is still alive).

Web storages do have a drawback, they lack of transactions. It means that if the users work with the application simultaneously using multiple browser tabs the data changes are not synchronized between the tabs. When the user modifies data in the first tab there is no mechanism to ensure that the second tab will have the same changes. That could lead into confusing situations for the user.

How much disk space can a single site consume when using web storages from the user's disk where the browser runs? Current specification of web storage states that "*User agents should limit the total amount of space allowed for storage areas, because hostile authors could otherwise use this feature to exhaust the user's available disk space. User agents may prompt the user when quotas are reached, allowing the user to grant a site more space.*" [9].

If the application needs more data than is reasonable and effective to store into the persistent memory using web storages developer should consider other HTML5's off-line data solutions.

For example, if the developer needs to save multiple instances of objects and view them all to the user using some sort of a list. It would be very frustrating to store every object with a key to local storage and then trying to retrieve every object by using the key from the storage. The database like solutions are better because it is possible to retrieve multiple items with a single query using *where* clauses etc.

In the beginning of the HTML5 standard the Web SQL Database was introduced as part of HTML5 standard. The Web SQL Databases purpose was to offer a client-side database that could be manipulated using already known and familiar SQL language.

The Web SQL Database was implemented with all the good and bad things from the ordinary relation database. The schema must be defined upfront and every object in the single table must match the table structure or the object cannot be inserted or updated in the table. Looking from another angle the rigid data structure means that it is easier to maintain the integrity of the database objects.

Web SQL supports indexes like an ordinary SQL database so search performance is usually fairly good and it can be improved by adding indexes. The Web SQL API was developed as an asynchronous API so queries and data manipulation is not locking the user interface. The user is able to interact with the user interface while the queries are made. Not a single user wants to use an application that is constantly unresponsive.

The development of Web SQL database specification has been stopped in November 2010 because all of the interested implementers used the same SQL backend. Standardisation could not be made when everyone was using the same backend. As the Web SQL database standardisation document states: "*This document was on the W3C Recommendation track but specification work has stopped. The specification reached an impasse: all interested implementors have used the same SQL backend (Sqlite), but we need multiple independent implementations to proceed along a standardisation path.*" [12]

The developers should not consider Web SQL as an option to store client-side data because the working group states that: "*Implementors should be aware that this specification is not stable.*" [12]

The competing solution for the Web SQL standard was introduced in 2009 and it is called Indexed Database API. The first public draft was developed by Oracle [24] and first it was called WebSimpleDB API. In present day it is mainly known as Indexed Database API [13].

Indexed Database API development started after the first versions of web storage and Web SQL Database was completed. With gained knowledge the working team could focus to implement it using strengths from the two earlier solutions. In the development they have also been focusing to solve the weaknesses that were found out from the earlier solutions.

The Indexed Database contains collections of objects in stores. These object stores have not constrains between each other. Because there are no constraints between stores object structures does not need reservations to pair objects together. This also means that there is no need to define object stores upfront.

When the database can be used without defining its structures upfront the database can be taken to use much faster than the ordinary relation database. Making simple offline demos or proof of concepts can be set up very fast and changes can be implemented fast because the databases agile model.

Web storages and Indexed Database solutions are effective when the data is mainly structured data or text. When there is a need to save large binary content for example files or images the developer should consider using the HTML5 standards FileSystem API.

In the FileSystem API development the working team has focused on solving five different use cases that are: persistent uploader, application with lots of media assets, audio/photo editor, offline video viewer and offline mail client. The working team focuses also on the ability to share data with other applications outside of the browser. [25]

The use cases can be summed together with one common feature. Every one of them stores large amount of data to the user’s disk. That data is then utilised in the application for some heavier operation. For example the user’s emails are stored to disk so the user can view them when he or she has no network coverage. The user could also store a large video file to the disk and then view it later with some another application.

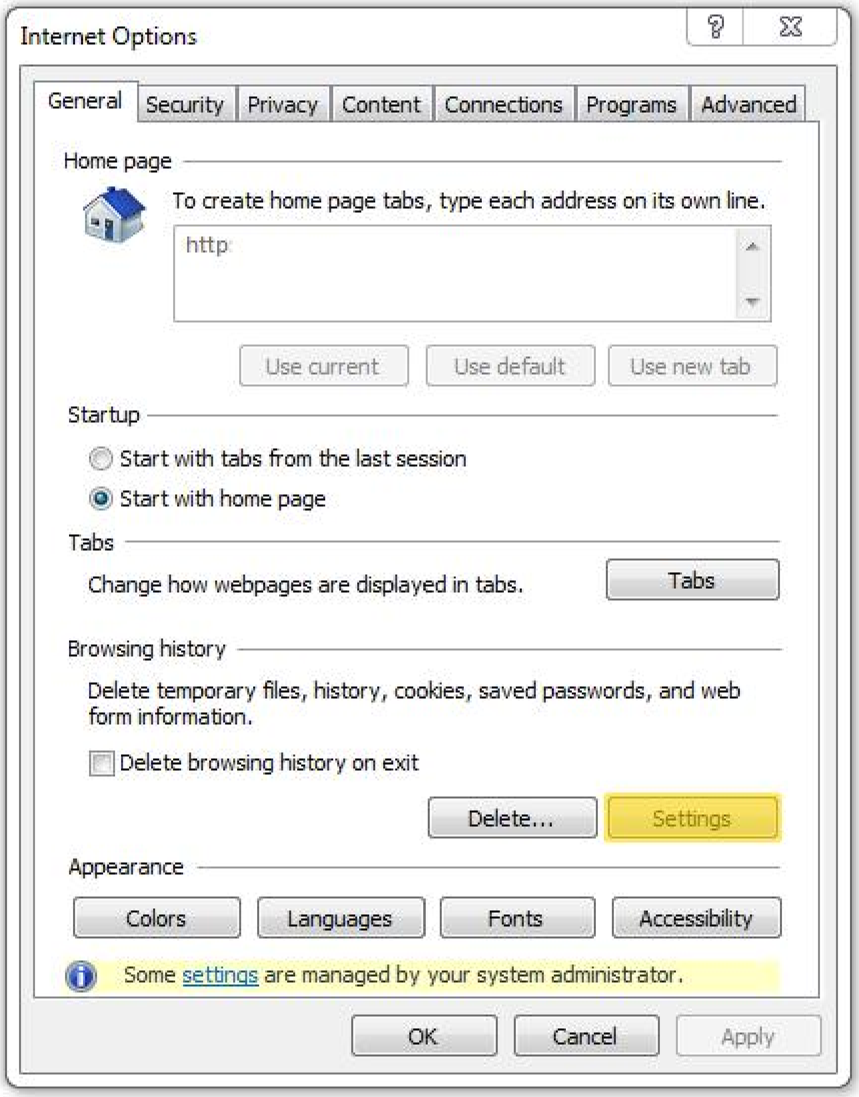
4 Inspecting HTML5 Storages Usage

Some of the browsers have preinstalled tools to inspect the usage of the sites. This chapter is dedicated to browser features. From the browser features the research was looking out the possibilities to inspect the usage of Application Cache, IndexedDB, WebSQL, LocalStorage and SessionStorage from the browser itself without using other development tools.

4.1 Internet Explorer

Internet Explorer has a minimalistic view for the offline data that websites have stored into the device's disk. How much space sites have allocated and the settings for the website cache and databases sizes can be inspected and set in the Website Data Settings dialog Caches and databases tab.

Website Data dialog can be found accessing option “Internet Options” under browsers Tools menu. From the opened Internet Options dialog one chooses the Settings button under Browsing history title. This can be seen in Figure 5.



Home page

To create home page tabs, type each address on its own line.

http

Startup

○ Start with tabs from the last session

● Start with home page

Tabs

Change how webpages are displayed in tabs.

Browsing history

Delete temporary files, history, cookies, saved passwords, and web form information.

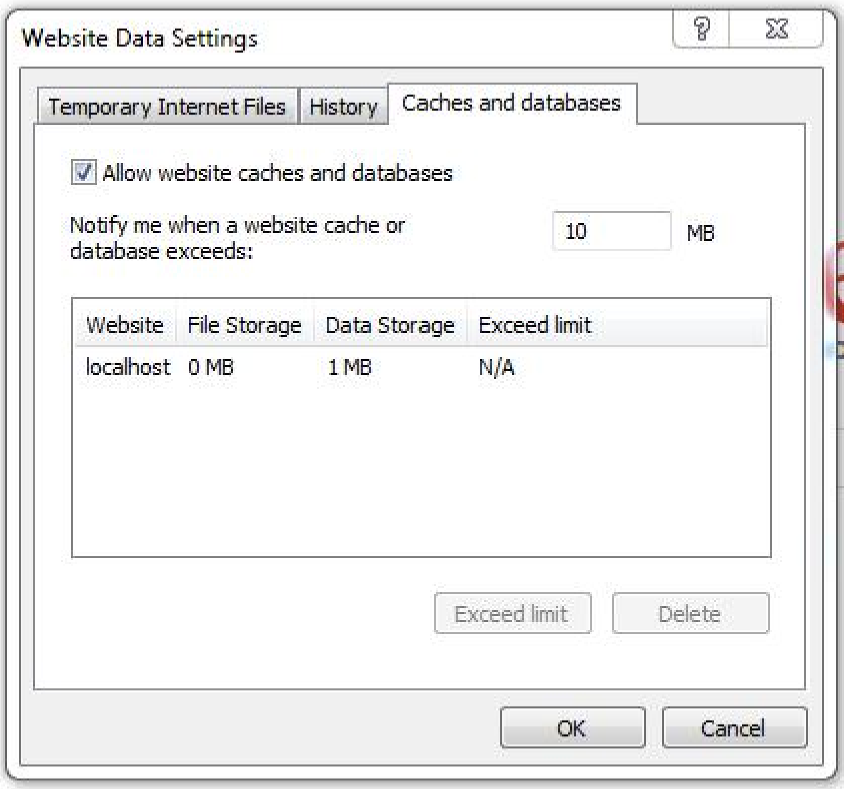
□ Delete browsing history on exit

Appearance

Some settings are managed by your system administrator.

Figure 5: Internet Explorers Internet Options dialog

Figure 6 presents the opened “*Website Data Settings*” dialog. From the opened dialog one would select “*Caches and databases*” tab to inspect usage of sites and setting limit how much space sites can allocate before user has to accept the disk usage. The default setting value for Internet Explorer is 10 MB.



Website Data Settings

☑ Allow website caches and databases

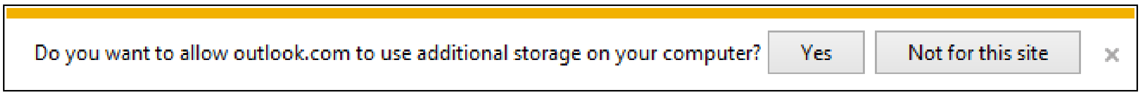
Notify me when a website cache or database exceeds: 10 MB

Website	File Storage	Data Storage	Exceed limit
localhost	0 MB	1 MB	N/A

Figure 6: Caches and databases settings tab under Website Data Settings dialog

Currently Internet Explorer does not offer any tools to inspect the data that is stored in the browser. It can only show how much data has been stored and offers a single solution to delete it and that is all.

When disk usage is exceeded a prompt is displayed that will request the user input to allow or disallow extra disk usage. Figure 7 presents the prompt that the Internet Explorer will show.



Do you want to allow outlook.com to use additional storage on your computer?

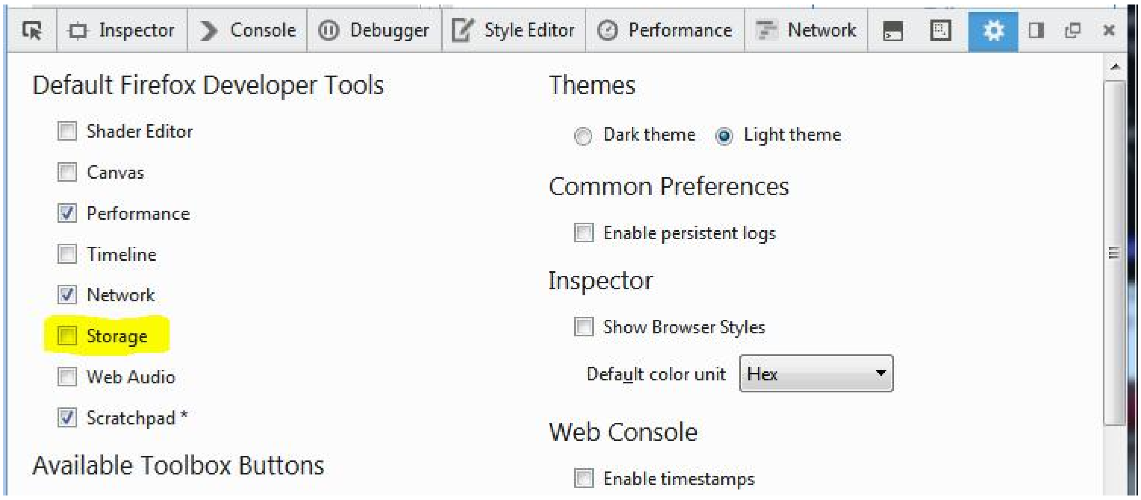
Figure 7: additional storage prompt in the Internet Explorer

After the user has allowed the site to exceed the usage the prompt is not displayed again. The user should always consider why the site tries to save more data than the default quota is if it is not clearly explained in the application.

4.2 Mozilla Firefox

Firefox introduced developer storage tools in its release of version 34. This storage tool allows inspecting the data that the browser has been saved from the websites. In the older Firefox it is possible to use a plug-in that offers tools to inspect the browser storages.

In the version 34 there is a dedicated Storage panel in the developer tools. The default setting for the panel in the developer tools is that the storage tab is not active. To activate it the user has to first open the developer tools (F12) and then open the settings panel from the icon shown in Figure 8.



Default Firefox Developer Tools

□ Shader Editor
□ Canvas
☑ Performance
□ Timeline
☑ Network
□ Storage
□ Web Audio
☑ Scratchpad *

Available Toolbox Buttons

Themes

○ Dark theme ● Light theme

Common Preferences

□ Enable persistent logs

Inspector

□ Show Browser Styles

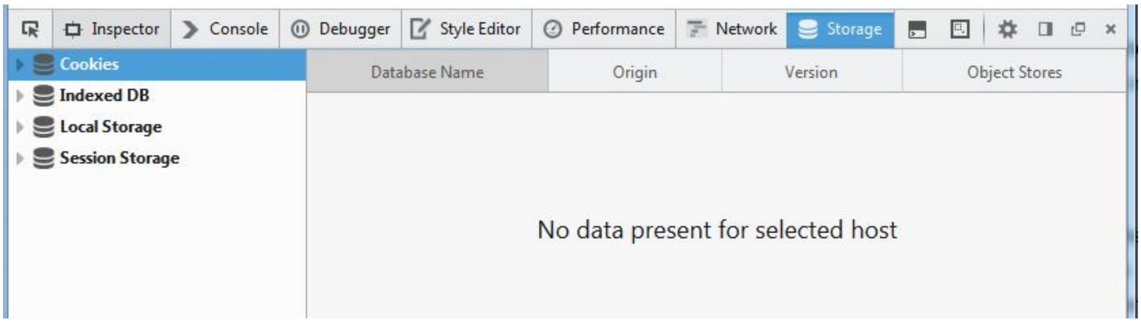
Default color unit Hex

Web Console

□ Enable timestamps

Figure 8: Firefox developer tools settings

From the settings tab the storage must be checked. Immediately after checking the storage setting a Storage tab is presented in the developer tools. A view of the storage tab is presented in Figure 9.



- Cookies
- Indexed DB
- Local Storage
- Session Storage

Database Name	Origin	Version	Object Stores

No data present for selected host

Figure 9: Firefox developer tools Storage tab

It is possible to inspect all of the storages listed in the storage tab. With the default storage inspector it is not possible to clear the storages. All but the indexed database storages can be cleared by removing all the data from the browser history.

To permanently delete the Firefox's indexed database it can be done by deleting the .sqlite files from the disk. These files are located in the users Firefox folder in the computers application data folder. In Windows 7 by default these files are located in the "C:\Users\USERNAME\AppData\Roaming\Mozilla\Firefox\Profiles\*.default-*\storage\persistent\" path. Where the USERNAME- is the current user's name and the profile is default. Every site that uses indexed database has its own folder under the persistent folder.

4.3 Google Chrome

With Google Chrome it is possible to inspect the saved data from the Application cache, IndexedDB, WebSQL and Local- and SessionStorages. They can all be inspected from the resources tab in the Chromes developer tools. Chomes resources tab is presented in Figure 10.

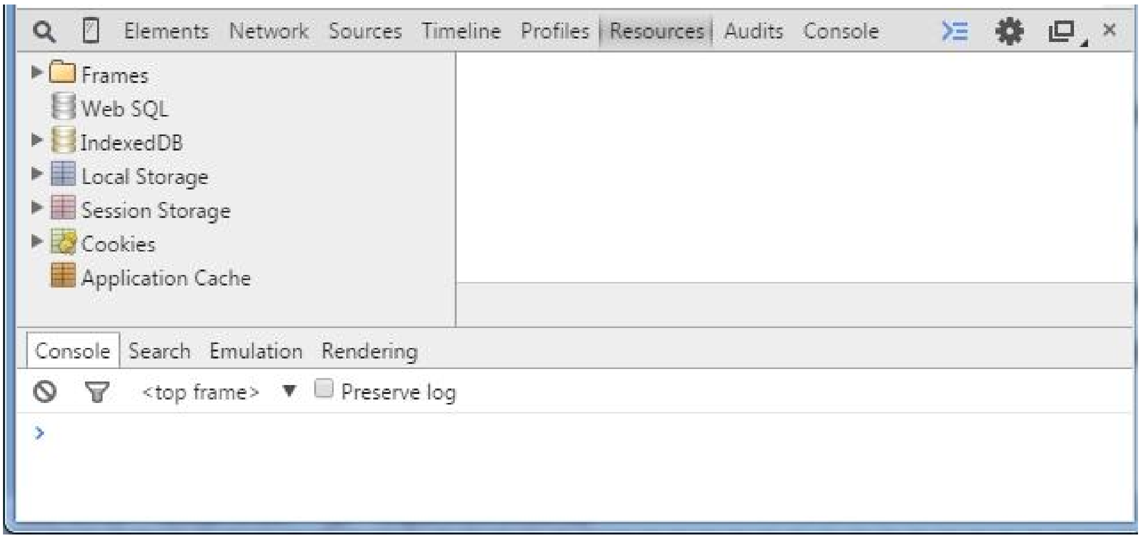


Figure 10: Google Chromes Developer tools resources tab

The disk availability in the Chrome is superior compared to the disk availability in the Internet Explorer. "*Each app can have up to 20% of the shared pool. As an example, if the total available disk space is 50 GB, the shared pool is 25 GB, and the app can have up to 5 GB. This is calculated from 20% (up to 5 GB) of half (up to 25 GB) of the available disk space (50 GB).*" [22]

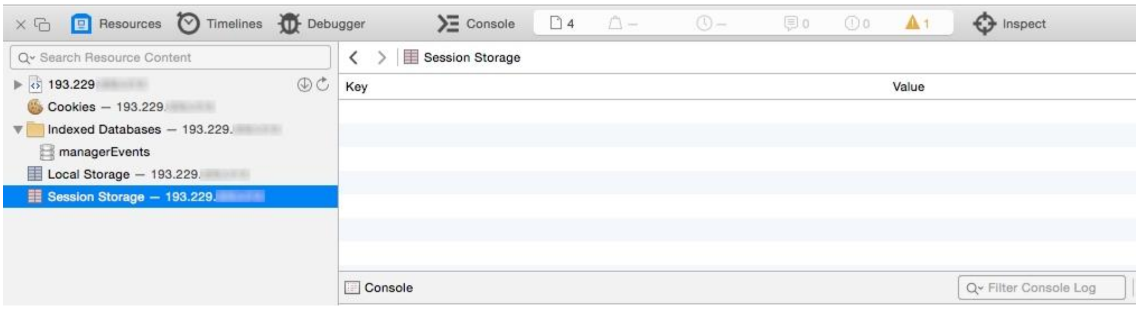
From the indexed databases it is possible to clear the object stores one by one using the right click on the mouse from the top of the indexed database store. The right click will open a small menu with a single command clear. It will clear the selected indexed database store.

From the Local and Session storage and from the Cookies it is possible to delete saved data by one row a time. After selected the site from the left panel the right panel will show every key-value pair that the site is stored. Unwanted key-value pair can be selected and pressing the delete button from the keyboard or the delete icon from the bottom of the right panel will delete the selected row.

Using only the default tools that are built in the browsers the Chrome offers currently the most advanced tool to monitor and control the data that is stored inside the browser.

4.4 Apple Safari

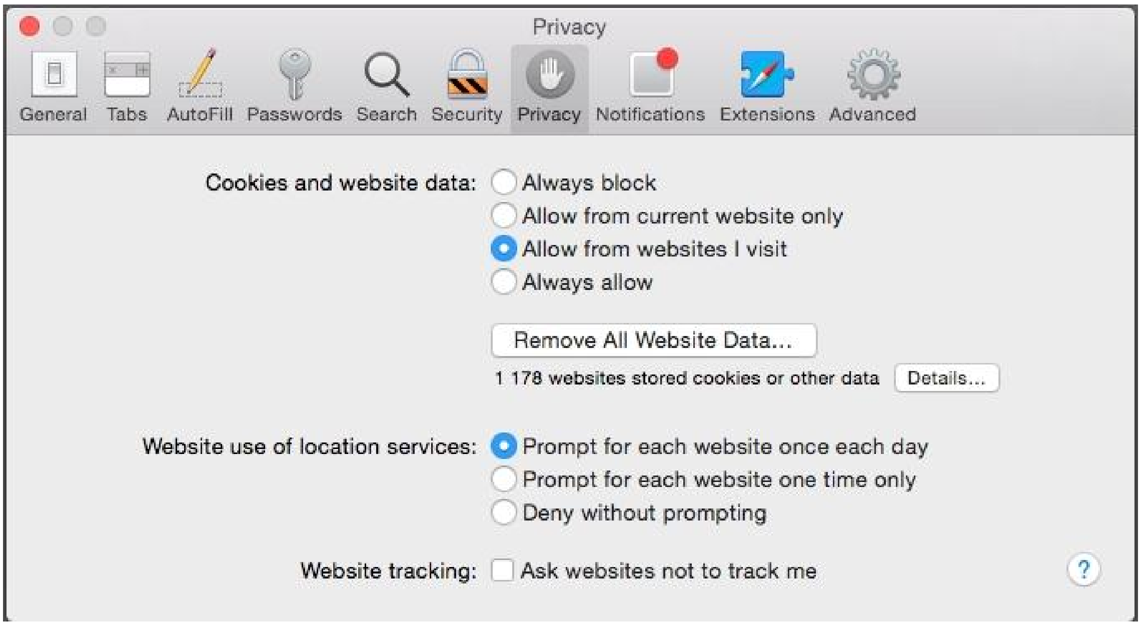
Apples desktop safari has similar developer tools than Firefox and Chrome have. The developer resources view can be opened from the Develop menu by choosing the "Show Page Resources". Safaris developer tools are able to show data from the Cookies, Indexed databases, local- and session storages. Safaris resource view is presented in Figure 11.



- 193.229[redacted]
- Cookies — 193.229[redacted]
- Indexed Databases — 193.229.[redacted]
  - managerEvents
- Local Storage — 193.229.[redacted]
- Session Storage — 193.229.[redacted]

Figure 11: View from the Safaris Resources tab in the developer tools

The Safaris developer tool is only capable to view the saved data. The entire saved data from a single page can be removed using the preferences menu. To do this, one would open the Safari preferences (from the Mac's top menu Preferences is found under the Safari title). From the opened preferences dialog the Privacy tab is chosen. There is a possibility to remove all data of the website using the button that is in the dialog. The preferences privacy tab is presented in Figure 12.



Cookies and website data:
○ Always block
○ Allow from current website only
● Allow from websites I visit
○ Always allow

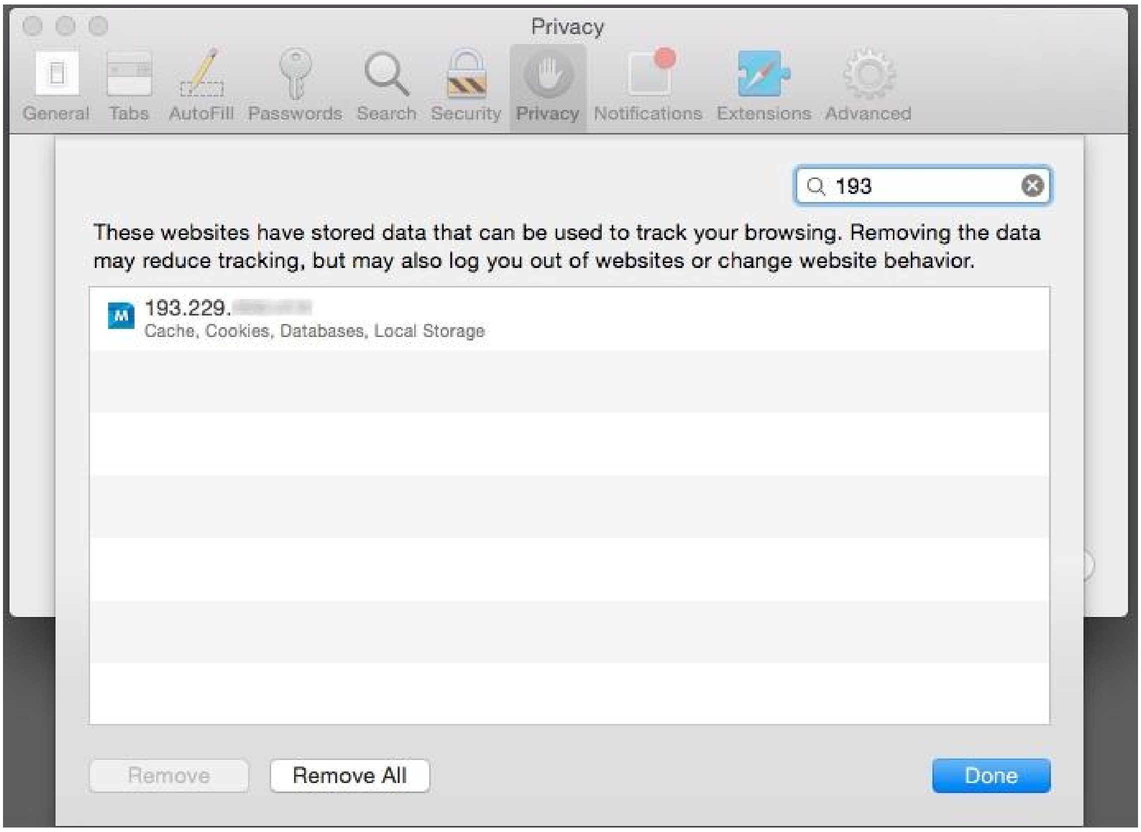
1 178 websites stored cookies or other data

Website use of location services:
● Prompt for each website once each day
○ Prompt for each website one time only
○ Deny without prompting

Website tracking: □ Ask websites not to track me

Figure 12: Safaris settings dialog with Privacy tab selected

Clearing data only from a single page can be done by clicking the “Details...” button from the dialog. This dialog is presented in Figure 13. From the opened dialog there is a search field where it is possible to type the pages domain which data is wanted to be removed. When the appropriate domain is in the list it can be selected and then the Remove button must be clicked. The data that domain has saved is being deleted. All the saved data from the cache, from the cookies, from the databases and from the storages have been removed.



These websites have stored data that can be used to track your browsing. Removing the data may reduce tracking, but may also log you out of websites or change website behavior.

193.229.[redacted]
Cache, Cookies, Databases, Local Storage

Figure 13: Deleting saved data from a specific site in the Safari browser

Tools to inspect and manipulate data that is stored to the browsers memory are handy when developing applications that run in the browser. There is no reason why normal user should normally inspect or manipulate data that is saved from the different websites.

5 Detecting Internet Connection in Offline Capable Application

Applications that have offline capability also need a mechanism to detect the state of the internet connection, whether the user's handheld device have the internet connection or not.

The internet connection detection is almost as important a feature in the offline capable application as the data handling. Although if in a certain point there is a network connection it does not mean that it is available in the next application step.

It should be considered when there is enough information that the phone has a network connection. Is it enough when the phone has a connection to the network or should also be checked that the internet connection can be made. Nowadays mobile applications transfer data from the internet and to the internet quite a lot more than for example five years ago. It must be also remembered that when the mobile phone is connected to the network there could be some cases that internet connection is not set up or the internet connection speed is low.

One commonly used way to identify the internet connection inside a mobile application is to ping some host at the internet and check if that succeeds. When developing application which is entirely in the hands of single company it would be reasonable to ping one of addresses that belongs to that company.

If the application collects data from numerous locations then it would be reasonable to test if the ping can reach some widely used and trusted service like for example the Google. Keeping in mind that there are nations where it cannot be reached even if the internet connection works normally.

5.1 Google Android

In the Google Android platform, network connection can be checked using ConnectivityManager class. That class contains method getActiveNetworkInfo. It will return the first found connected network interface or null. Also the ACESS_NETWORK_STATE permission has to be added to the Android applications manifest. Example of testing network status in the Android application is presented in Listing 1.

```
ConnectivityManager connManager
= (ConnectivityManager) getSystemSer-
vice(Context.CONNECTIVITY_SERVICE);
NetworkInfo activeNetworkInfo = connManager.getActiveNetworkInfo();
boolean isNetwork = activeNetworkInfo != null && activeNetwork-
Info.isConnected();

//to the application manifest
<uses-permission an-
droid:name="android.permission.ACCESS_NETWORK_STATE" />
```

Listing 1 - Sample of testing network status in the Android [28]

Listing 1 presents the actual test of the network interface and the permission that should be added to the Android application manifest.

5.2 Apple iOS

Testing the network connection in the Apples iOS platform is slightly more complex than in the other platforms. Apple distributes a sample application that demonstrates how to check and monitor the network state of an iOS device [15].

With the help of that sample project it is possible to make a simple network status check. First a reachability-object is needed. That object contains a method currentReachabilityStatus which will return a NetworkStatus object that contains the information about current connection. Example of testing network status in the iOS can be seen in Listing 2.

```
Reachability *networkReachability = [Reachability reachability-
ForInternetConnection];
NetworkStatus networkStatus = [networkReachability currentReachabili-
tyStatus];

if (networkStatus == NotReachable) {
    NSLog(@"There IS NO internet connection");
} else {
     NSLog(@"There IS internet connection");
}
```

Listing 2 - Sample of testing network status in the iOS [15]

As presented in Listing 2 the returned networkStatus object can be compared to the NotReachable enumeration.

5.3 Windows Phone

Windows phone platform offers NetworkInformation class for identifying the network access. NetworkInformation-class has a method called GetIsNetworkAvailable. It does not have any parameters and it will return a boolean value indicating if the phone is connected to the network or not. Network availability test in the Windows Phone application can be seen in Listing 3.

```
var isNetWork = NetworkInterface.GetIsNetworkAvailable();

if (isNetWork)
{
    Debug.WriteLine("There IS network connection");
} else {
    Debug.WriteLine("There IS NO network connection");
}
```

Listing 3 - Sample of testing network status in the Windows Phone

The Windows Phone’s GetIsNetworkAvailable inspects only the network availability. It is possible that the user can dial with the phone but the internet connection is not available, so the check is not totally accurate.

5.4 HTML5

In HTML5 or in the browsers the online can be tested using JavaScript and the navigator object. The navigator object contains information about the browser. It has a property called onLine. Example of using the property is presented in Listing 4.

```
function updateOnlineStatus(event) {
    if(navigator.onLine) {
    //application has network connection

    } else {
    //application does not have network connection

    }
}

document.body.addEventListener('online',  updateOnlineStatus);
document.body.addEventListener('offline', updateOnlineStatus);
```

Listing 4 – Adding event listeners to the online and offline events [23]

It is possible to hook the application code to monitor the network status using JavaScript. The application should add event listeners to the online and offline events. These events occurs when the onLine-property changes. This can be made using basic JavaScript. The application should add event listeners to the document body or to the window for these events with an event handler. Example of monitoring the events is presented in Listing 4.

As researched it is possible to test the network availability in the selected mobile platforms as well as in the HTML5 based solutions. Testing the network availability does not give the accurate details for example about the speed of the current network.

When developing a native application it would be possible to benefit from the extra features available in the native platform. For example in the iOS environment it is possible to identify accurately what kind of network interface have connection. The Wi-Fi interface or some other interface which speed is not as fast as Wi-Fi interface’s.

The information about the connected interface could be utilised in the application development. For example images transferred to the user could have better resolution or if the application transfers and shows video the connection speed is affecting to the result even more. In the HTML5 development it is not possible to get details about the connected interface so another solution should be researched.

The HTML5 based solutions needs also more work what comes to the application logic while the network is not available. The next chapter researches the HTML5 solution to secure the application logic into the user’s browser.

6 HTML5 Application Cache

Implementing HTML5 application with offline support has two key parts. The first one can be called as data storage. That is how and where the application data is stored and how it is handled. Different data storage possibilities have been covered in the previous chapters. The second key part is how to ensure that all the necessary files are available for the application even when there is no internet connection.

HTML5 has a way to extend the traditional browser caching significantly. The traditional browser cache stores only a few of the latest pages the user has visited into the browsers memory. Those files have a time stamp and they are requested again if the timestamp is too old. In the traditional cache developers had no control over what files were stored and what were not.

The application cache manifest of HTML5 makes it possible to retrieve files beforehand into the browsers cache. With the help of application cache it is possible to store all necessary files into the user's browsers and application can operate normally without the internet connection.

The application cache manifest file is a file which lists all the files the application needs to be operational without internet connection. The user's browser will keep a copy of those files to make the offline usage possible [17].

The simplest manifest file contains the title "CACHE MANIFEST" so browsers can identify that the file is the cache manifest file. The files in the manifest are listed with an absolute URL or with absolute path to the actual files. Each file that the application needs to operate without connection is listed on its own separate line. A sample file is presented in Listing 5.

```
CACHE MANIFEST
todo.html
todo.css
todo.js
```

Listing 5 – Simple cache manifest file

If the first line of the file contains other than spaces, tabulars and text “CACHE MANIFEST” the browsers will ignore the manifest file. Even if the space between the cache and manifest is something else than a single white space the file is ignored.

From the next lines it is possible to use comments in the manifest file also. If the developer wants to do so a new line has to be started with a number sign (#). A common trick is using it to versioning the manifest file. Simply by writing a number on the comment line or being a little bit more specific and writing it after a version word. Listing 6 presents the use of a version comment.

```
CACHE MANIFEST
#version 0.9
todo.html
todo.css
todo.js
```

Listing 6 – Simple cache manifest file with an additional comment line

The manifest file can have any name or file extension. What is important is that the file is a text file and encoded using UTF-8. The file has to be served using a MIME TYPE text/cache-manifest.

The application manifest files mime type may need to be added to the server’s configuration file. In the Apache environment the mime type can be added using the .htaccess file. In the Windows Internet Information Server the mime type can be added using the applications web.config file. Adding the mime type to the server configuration file is presented in Listing 7.

```
//in the .htaccess
AddType text/cache-manifest .manifest

//in the IIS web.config
<system.webServer>
    <staticContent>
      <mimeMap fileExtension=".appcache" mimeType="text/cache-
manifest"/>
    </staticContent>
</system.webServer>
```

Listing 7 – Adding a mime type on Apache and on the Windows environments

The manifest file is marked inside the page's html tag using the attribute manifest. If a wrong location is set or there is a type error in the manifest file name the browser will get an http 404 error. If the 404 error occurs when the browser tries to read the manifest file the application cache for this page will be deleted. Adding the manifest attribute to the html tag is presented in Listing 8.

```
<!-- todo.html -->
<!DOCTYPE HTML>
<html manifest="todo.appcache">
 <head>
  <title>Todos</title>
  <script src="todo.js"></script>
  <link rel="stylesheet" href="todo.css">
 </head>
 <body>
...
 </body>
</html>
```

Listing 8 – Cache manifest file attached to simple html page

There are several important things to remember when using application cache. The first is that all the listed files are always served from the application cache not from the browser cache [19].

This means that the browser will first render the page from the cached files. When the rendering is completed the browser then starts to look out if there is an updated manifest or if the files listed in the manifest have been updated.

This may sound confusing but there is a reason why it works like that. If the user's connection drops during the load the browsers can finish the page rendering when the data comes from the application cache.

With the help of the application caches Javascript-API the developer could notify the user to refresh the page when there are newer files available. This has been already seen in parts of the Googles web-solutions (Drive, mail etc). The solutions notify the user to refresh the page because there are updates available.

Manifest structure and behaviour can be improved using CACHE, NETWORK and FALLBACK attributes.

Files that are listed under the CACHE attribute are being cached. The cache attribute is not mandatory and files that have to be cached can be listed also after the CACHE MANIFEST title. Using the extra cache attribute could make the manifest file easier to read.

Under the NETWORK attribute there are listed files that may come from the network if they are not in the cache. Most applications uses an asterisk (*) here. So the application developers do not need to list every file that the application would use [4, 146].

Under the FALLBACK-attribute it is possible to reroute the user to a specified URL if the application is not have a network connection. For example it would be possible to cache a single image and use that image on every place where an image would be shown if the application would have network connection. There is a limitation as to how much data can be saved so using a single image rather than each of the images would save a lot of space from the memory.

7 Accessing and Saving Key Value Pair Data

As reported in Chapter 3 there are many different solutions and ways to save data to the mobile phones persistent memory depending on the platform that developers work with. This chapter introduces different mechanisms to save simple key value pair data to the persistent memory of mobile phones.

7.1 Google Android

Androids option to save key value pair data to the persistent memory of a mobile phone is called the SharedPreferences. SharedPreferences allows saving and retrieving primitive data types using the SharedPreferences class that provides a general framework to work with.

First using the getSharedPreferences- or getPreferences- method the SharedPreferences object is initialized. The guideline from Google states that getSharedPreferences method should be used when the application is using multiple files to store the applications preferences [27]. The getSharedPreferences-methods the first parameter is the name of the settings file and the second parameter is the mode.

The getPreferences method should be used when one settings file is enough for the activity. All the settings are stored into the single file and that file can be accessed only from the current activity. The getPreferences method needs a single argument that is a mode. Current documentation states that MODE_PRIVATE (equals integer 0) should be used and it is the only not deprecated value for the argument.

Now when the SharedPrefences object is created key values can be read from the memory. To write key value pair data an editor is needed. The editor (SharedPrefences.Editor) can be created by calling the edit method under the SharedPreferences object.

With the editor it is possible to write values. It has to be remembered to use correct method for each primitive data type such as putBoolean and putString. These put methods have two parameters. The first one is the name of the settings and the second one is the actual value of the setting.

When the putBoolean or putString method is called the setting values are stored first to the SharedPreferences.Editor only. To save the values to the persistent memory commit- or apply-method have to be called from the editor. Commit-method writes changes atomically into the persistent memory unlike the apply-method writes changes first to the in-memory SharedPreferences and then performs asynchronous commit to the persistent memory. Getting the sharedPrerences and the editor objects to save a boolean type of setting to the persistent memory is presented in Listing 9.

```
// Restore preferences
SharedPreferences sharedPreferences = getPreferences(0);´

//get the editor by calling the edit
SharedPreferences.Editor editor = sharedPreferences.edit();

//add value to the shared prefenreces
boolean useThousands = true;

editor.putBoolean(“useThousands”, useThousands);

//apply the modification
editor.apply();
```

Listing 9 - Sample of adding a setting to the SharedPreferences

If there are multiple changes to the same preferences file with multiple editors the last editor to commit will get its changes saved into the persistent memory. Commit-method returns true if it successfully saved changes and the apply-method is just a void method. The Android developer guide suggests that if the commit methods return value is not used the developers should always use the apply-method instead.

Reading settings from persistent memory is a slightly straight forwarded than the writing. First the SharedPreferences object is needed, but after that primitive data-type get methods can be called such as getBoolean or getString or equivalent. These get-methods have also two parameters. The first parameter is the name for the setting and the second is the default value. Default value is used if there is no value in the persistent memory with the queried setting name. Restoring a setting value using SharedPreferences is presented in Listing 10.

```
// Restore preferences
SharedPreferences sharedPrefences = getPreferences(0);
boolean useThousands = sharedPrefences.getBoolean("useThousands",
false);

if(useThousands) {
    ...
}
```

Listing 10 - Sample of reading a SharedPreferences

The SharedPreferences of Android platform gives the developer a reasonable toolbox to work with. It is designed to hold primitive values but complex objects could be saved to it by serialising them first and saving the data using the string primitive data type.

7.2 Apple iOS

Apples iOS platform uses property lists as a method to save simple data from the application to the persistent memory of a mobile phone or tablet. These property lists are xml based files that are located in the applications resources folder. When the user closes the application the developer should make sure that all the necessary settings are written into the xml file.

Stored data in the property list can be from any objective-c primitive data type. The Apple developers are however implemented methods to read and write into the property list only for the NSArray and the NSDictionary classes. Both of these classes have writeToFile:atomically and writeToURL:atomically-methods.

Keeping that in mind it would be reasonable and easier to store all the application settings into single NSDictionary object that is saved into the applications persistent memory using property lists.

When dealing with property lists it is always good to remember that result in the disk is an actual xml file. That is the reason why saving simple settings have more steps than in the compared platforms.

First the path to the actual xml file is needed and before it can be determinate application's root path is needed. That can be found using NSSearchPathForDirectoriesInDomains object with the NSUserDomainMask parameter.

When the rootpath is discovered it has to be appended with a property list name using NSStrings stringByAppendingPathComponent method.

It is the developer’s responsibility to ensure that file exists on that location. If not, the file should be made programmatically before the program can continue any further.

When the file is found and located from the disk the property list xml can be read out as a NSData object. That NSData object can then be deserialized to NSDictionary and the developer may now access its properties to read out the actual setting values. The example of reading from property list can be seen in Listing 11.

```
// discover the root path
NSString *rootPath =
[NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES) objectAtIndex:0];

//append property list name to rootpath to get propertyLists path
NSString *plistPath = [rootPath
stringByAppendingPathComponent:@"AppSettings.plist"];

//read the actual file as NSData
NSData *plistXML = [[NSFileManager defaultManager]
contentsAtPath:plistPath];

//deserialise the xml file into NSDictionary
NSDictionary *temp = (NSDictionary *)[NSPropertyListSerialization
propertyListFromData:plistXML
mutabilityOption:NSPropertyListMutableContainersAndLeaves for-
mat:&format errorDescription:&errorDesc];

//retrieve setting value
NSString *programName = [temp objectForKey:@"ProgramName"];
```

Listing 11 - Sample of reading from property list.

When the user closes the program the current setting values have to be saved into property list file or when the user closes the application settings view. Saving can be implemented by the help of the NSDictionary and NSData classes.

Before saving can be done the property list file path must be discovered and appended with the name of the property list file. With the correct path the settings, dictionary can be serialized using the NSDictionary classes dataFromPropertyList-method. DataFromPropertyList will serialize the NSDictionary to NSData object and that NSData object can be written into actual file that locates in the disk. The writing is done by using NSData class’s writeToFile-method.

```
//we are about to save the NSDictionary *settingsDict
NSString *error;

//get the applications root path
NSString *rootPath = [NSSearchPathForDirectoriesInDomains(NSDocumentDirectory, NSUserDomainMask, YES) objectAtIndex:0];

//append it by adding our settings list name
NSString *plistPath = [rootPath stringByAppendingPathComponent:@"AppSettings.plist"];

//make NSData from our NSDictionary
NSData *plistData = [NSPropertyListSerialization dataFromPropertyList:settingsDict format:NSPropertyListXMLFormat_v1_0 errorDescription:&error];

//write that file to disk
[plistData writeToFile:plistPath atomically:YES];
```

Listing 12 - Sample of saving the AppSettings property list.

The Apples developer library has more complex and detailed information about property lists in the “Property List Programming Guide” [8]. An example of saving property list to the disk can be seen in Listing 12.

7.3 Windows Phone

On the Windows Phone platform saving key value pair data can be done using the IsolatedStorageSettings class. This class is a Dictionary(TKey, TValue) that stores its data into the persistent memory.

Before using the isolated storage and the IsolatedStorageSettings class a reference to System.IO.IsolatedStorage library needs to be added to the program. To get access to the applications settings file a call to the system class ApplicationSettings has to be made. That system class can be found under IsolatedStorageSettings.

Adding key value pairs into the application settings can be made using the ApplicatoinSettings class’s add-method. If the key that is added already exists in the application settings an exception will be thrown. When keys are added there should be always a check that the key is not already found in the application settings dictionary. Adding a setting to the IsolatedStorage is presented in Listing 13.

```
//get settings
var appSettings = IsolatedStorageSettings.ApplicationSettings;

//aad value of useThousands setting
appSettings.Add(“useThousands”, true);
```

Listing 13 - Sample of adding a setting to the IsolatedStorage [7]

When reading from isolated storage there is a possibility that value is never added before trying to read it. So there should be always a check that queried key is in the settings before trying to read it. And there should also be some kind of default value if the read setting is not in the isolated storage. Reading a setting value from the IsolatedStorage is presented in Listing 14.

```
//get application settings
var appSettings = IsolatedStorageSettings.ApplicationSettings;

//read the value of useThousands setting
var valueInSettings = appSettings[“useThousands”];
```

Listing 14 - Sample of reading an IsolatedStorage [7]

Even the Microsoft suggests that some sort of helper class should be used when dealing with settings in the isolated storage. And they have written a very basic but a good helper class for everyone to start with [8].

7.4 HTML5

HTML5 offers web storages to use with key value pair data. When dealing with the offline requirement key value pairs has to be saved somewhere so they are available even after the solution is closed and opened again. When testing the application closing scenario it should be tested by closing the browser executable. Not by only closing the tab where the application runs.

Web storages persistent memory option in the HTML5 standard is called as a local storage. Comparing it to the mobile platforms solutions it has a slight limitation. It is possible to save only string type values to the local storage.

Before using local storages the application should test if the user’s browser supports local storages. Local storage API is stored to the localStorage variable that is found from the window. This test can be done using pure JavaScript or by using some 3rd party library like for example modernizr. Example of simple localStorage tester can be seen in Listing 15.

```
//simple local storage tester
function IsSupportingLocalStorage()
{
    try {
        return 'localStorage' in window && window['localStorage'] !==
null;
    } catch (e) {
        return false;
   }
}
```

Listing 15 - Sample of simple local storage tester

When the local storage test passes and there is a proof that the browser has a support for it the developer can read the setting values from the local storage by calling getItem-method. The first parameter of the method is the name of the setting. Because local storage is a JavaScript object like any other JavaScript object it is possible to use it as an associative array also.

With the help of square brackets getItem-method can be written like *localStorage[“settingName”]*. Again it must be remembered that all the read values are type of strings. Sample of reading a value from the local storage is presented in Listing 16.

```
//use test method to ensure compatibility
if(IsSupportingLocalStorage())
{
 //using the getItem method to read value
 var appSetting1 = localStorage.getItem("appSetting1");

 //using the square brackets to read value
 var appSetting2 = localStorage["appSetting2"];
}
```

Listing 16 - Sample of reading the browsers local storage

If key that is requested is not present in the local storage, a null is returned and there are not any exceptions thrown.

Saving values into the local storage is as easy as reading. Again after testing the support for the local storage setItem-method can be called or with the help of square brackets the string value can be written into the local storage. Of course it is possible to write any value into the local storage once it is transformed to a string with an equivalent method. Saving a value to the local storage can be seen in Listing 17.

```
//use test method to ensure compatibility
if(IsSupportingLocalStorage())
{
 //using the setItem method to read value
 var appSetting1value = “foo”;
 localStorage.setItem("appSetting1", appSetting1value);

 //using the square brackets to read value
 var appSetting2value = “bar”;
 localStorage["appSetting2", appSetting2value];
}
```

Listing 17 - Sample of writing into the browsers local storage

If the key already exists in the local storage with a value the previous value is overwritten in the background. If the key does not exist it will be added.

If there is a need to delete values in the local storage it can be made using the removeItem-method which has the setting key as its parameter. If the setting name is not given the method will do nothing. There is also a possibility to delete every setting instance in the local storage by calling the local storages clear-method. Example of clearing local storage and single item removing can be seen in Listing 18.

```
//this does nothing
localStorage.removeItem();

//this removes value by the key appSetting1
localStorage.removeItem(“appSetting1);

//this clears every key and value from the local storage
localStorage.clear();
```

Listing 18 - Sample of deleting local storage values

8 Accessing and Saving Large Amount of Structured Data

First of all, what is structured data? Structured data can be explained using a weather forecast as an example. The weather forecast is given to a specified location. That means that there are two elements, the forecast and the place. They are tied together because the weather forecast is given to the exact place for example the weather forecast of Helsinki. This kind of data structure is presented in Figure 14.

In the application there could be a selection field where all the places are listed. The weather forecast could be queried from all the places in the list. When the user chooses the place from the selection the weather forecast is shown. Underneath the user interface there will be weather forecasts to different places.

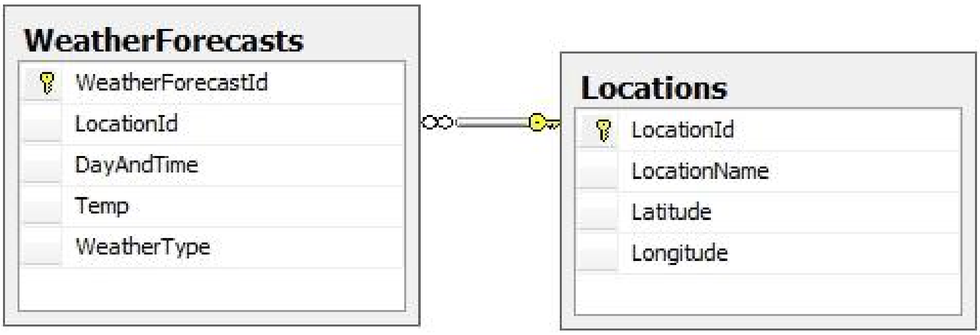


Figure 14: Simple weather forecast and locations structured data sample

The application could store the queried weather forecasts to the phone and query only the latest weather forecast from the internet. That would make possible to view the queried forecasts when there are no network coverage. This chapter studies different mechanisms to save structured data to persistent memory of a mobile phone.

8.1 Google Android

Saving large amount of structured data in the Android application can be done using the SQLite databases. The Android developer guide states that “*Android provides full support for SQLite databases.*” [10].

The developer guide suggest that when implementing the Android applications SQLite database a subclass of SQLiteOpenHelper should be created and its onCreate-method should be overridden. Example of a simple subclass can be seen in Listing 19.

The SQLiteOpenHelper is a helper class written by Android developers. It is designed to manage database creation and version management. It has a monotonic version number to ensure that exact database version is used with an exact version of the application.

In the subclasses constructor a super-method should be called using the database name and version as parameters. The version number parameter helps the platform to decide if a new database is needed or could it use the previously created one. With the database name it would be possible to divide the applications data into multiple databases.

```
public class DbHelper extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "db.testDb";
    private static final int DATABASE_VERSION = 1;
    //constructor of DbHelper class
    public DbHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }
}
```

Listing 19 – Example of subclass from the SQLiteOpenHelper class

In the overridden onCreate-method the Android developer guide suggests to call execSQL-method to create tables. Table creation scripts are written with transact SQL and put into the string variable. The SQLite database supports primary keys with automatically incrementing values and other common transact SQL column properties. If the application needs some default values they could be written into the database inside the onCreate-method. Example of overriding the onCreate method can be seen in Listing 20.

```
public class DbHelper extends SQLiteOpenHelper {
    public static final String TABLE_TODO = "TODO";

    public static final String COLUMN_ID = "id";
    public static final String COLUMN_NAME = "name";

    public static final String CREATE_TABLETODOS = "create table "
         TABLE_TODO + "(" + COLUMN_ID
        + " integer primary key autoincrement, " + COLUMN_NAME
        + " text not null);";

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(CREATE_TABLETODOS);
    }
}
```

Listing 20 – Example of onCreate-method in the SQLiteOpenHelper subclass

If the application's SQLite database has to be updated during applications lifecycle the version number should be changed. Also an overridden onUpgrage-method should be implemented in the subclass of the SQLiteOpenHelper class. Example of overridden onUpgrade-method can be seen in Listing 21.

The simplest way to handle the database updating is to get rid of the old tables in the database and then call the onCreate-method.

The Android developers guide states that database update should not be run in the main thread since it could take some time to finish. The application users are not used to wait long times.

```
public class DbHelper extends SQLiteOpenHelper {
    public static final String TABLE_TODO = "TODO";

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int new-
Version) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_TODO);
        onCreate(db);
    }
}
```

Listing 21 – Simple onUpgrade-method in the SQLiteOpenHelper subclass

The database is not actually created to the disk yet. It is located in the phones memory until getWritableDatabase or getReadableDatabase-method is called. The getWritableDatabase-method opens or creates a database which can be then read and write. The getReadableDatabase-method creates or opens a database which can only be read. However, when the database is opened the onCreate or the onUpgrade-method is called in the SQLiteOpenHelper subclass.

The database is now cached and will stay cached until the close-method is called. The close-method should be called as soon as the database is not needed anymore. For example if the database is opened when the application starts and some data is read and displayed at the applications main screen. As soon as all the displayed data is read out from the database the connection should be closed.

Inserting data into the Androids SQLite database can be done using the insert-method. The method has three parameters, the name of the table where the values are inserted, the nullColumnHack parameter and the actual values parameter. The method will return the row id for the inserted row, or “-1” if an error occurred. Error could occur for example when the phone does not have enough free space available to store the inserted data.

The SQLite database does not allow an empty row insertion without extra work. If there is a need to insert an empty row then the nullColumnHack parameter is needed. The values parameter is left null but the nullColumnHack parameter is used to determinate the column where the null value is stored.

The actual values are stored as objects of ContentValues-class. The class stores values as key value pairs. Every column and their values are added to the ContentValue object using the put-method. The put-method has two parameters, the key of the value and the inserted value. In the database scenario the ContentValues object keys are the names of the table columns. Listing 22 presents a simple example of adding values to the SQLite database.

```
public static final String TABLE_TODO = "TODO";
public static final String COLUMN_NAME = "name";

…

ContentValues todoItemValues = new ContentValues();
todoItemValues.put(DbHelper.COLUMN_NAME, "fill travel bill");

db.insert(DbHelper.TABLE_TODO, null, todoItemValues);
```

Listing 22 – Example of inserting values into the SQLite database

Querying objects from the SQLite database can be done using the SQLiteDatabases query-method. The method will return a Cursor-class object that point to the results of a query. The cursor can be used to read all the rows from the result. The query-method has seven parameters but for making simple query only the first two is needed.

The first parameter is the name of the table where the query is made and the second parameter is the columns array which defines the table columns where data should be returned. Example of data query can be seen in Listing 23.

```
private String[] allColumns = { DbHelper.COLUMN_ID,
DbHelper.COLUMN_NAME };

Cursor queryCursor = db.query(DbHelper.TABLE_BACKLOG, allColumns,
null, null, null, null, null);
```

Listing 23 – Example of querying values from the SQLite database

To simplify even more the column parameter could also be null, so it would return every column from the table. But the Android developer guide does not recommend using it so. The guide states that "*Passing null will return all columns, which is discouraged to prevent reading data from storage that isn't going to be used*" [10]. Mobile phones have limited resources therefore every query should be done using a minimum viable query.

The third parameter in the query-method is the selection parameter. It is used to form a where-clause in the SQL-query. The selection should be written without the actual where word. If the parameter is left null every row from the table is returned. Using where clause in the query can be seen in Listing 24.

```
private String[] allColumns = { DbHelper.COLUMN_ID,
DbHelper.COLUMN_NAME };
private String whereClause = DbHelper.COLUMN_ID + “ < 10”;

Cursor queryCursor = db.query(DbHelper.TABLE_BACKLOG, allColumns,
whereClause, null, null, null, null);
```

Listing 24 – Example of using where clause in the SQLite database query

The fourth parameter is the arguments of the selection. It is used together with the selection parameter. In the selection parameter it is possible to write the where clause using question marks. Question marks are replaced with provided selection arguments when the query runs. It will also help to escape the values (escaping a word value: ‘value’) so it prevents unintended escaping. This is also an easy way to set user inputted values into the SQL query. This kind of query is presented in Listing 25.

```
private String[] allColumns = { DbHelper.COLUMN_ID,
DbHelper.COLUMN_NAME };
private String whereClause = DbHelper.COLUMN_NAME + “ = ?”;
String[] userInputs = { "fill up a gas tank" };

Cursor queryCursor = db.query(DbHelper.TABLE_BACKLOG, allColumns,
whereClause, userInputs, null, null, null);
```

Listing 25 – Example of using where clause with arguments in the SQLite-query

The fifth parameter is the group by parameter. It is used to form a SQL Group By clause. Like the selection parameter the actual group by words are excluded and if the parameter is left null, rows are not grouped. Example of using group by clause in the query can be seen in Listing 26.

```
private String[] allColumns = { DbHelper.COLUMN_ID,
DbHelper.COLUMN_NAME };
private String groupClause = DbHelper.COLUMN_NAME;

Cursor queryCursor = db.query(DbHelper.TABLE_BACKLOG, allColumns,
null, null, groupClause, null, null);
```

Listing 26 – Example of using group by clause in the SQLite-query

The sixth parameter is the having parameter. It is tied together with the group by parameter because when the group and having parameters are both used the having parameter is formatted into SQL having clause. When using the having parameter the actual having word is excluded like in the selection and group by parameters.

The seventh parameter is the order by parameter. It also excludes the order by words from the query when the actual query is formatted. Order by parameter can be left null but then default sort order is used. Using a lot of queries without order clauses could lead into mysterious problems and different behaviour because the order of retrieved items could vary. Using order by clause is presented in Listing 27.

```
private String[] allColumns = { DbHelper.COLUMN_ID,
DbHelper.COLUMN_NAME };
private String orderByName = DbHelper.COLUMN_NAME;

Cursor queryCursor = db.query(DbHelper.TABLE_BACKLOG, allColumns,
null, null, null, null, orderByName);
```

Listing 27 – Example of using order by clause in the SQLite-query

When the query runs and the cursor object is returned the Cursor is positioned before the first entry. To start working with the objects that exists in the database the cursor must be moved. It can be moved to a desired object or it can be moved to point to the first item or to the last item in the result. This moving can be done by using cursor objects moveToFirst and moveToLast-methods.

These two methods have a handy extra feature. They will both return false if the query result did not return any objects. Information could be used to end the application query result handling as early as possible. Simple test if query returned any objects can be seen in Listing 28.

```
Cursor cursor = database.query(DbHelper.TABLE_BACKLOG, allColumns,
null, null, null, null, null);

if(cursor.moveToFirst()) {
…
}

cursor.close();
```

Listing 28 – Testing if query returned any objects

When the cursor is moved to the first item the item can be read out to as an object. A new object has to be created. To set the created objects property values from the database object the values are queried out using the cursor object.

Reading the values can be done by using the primitive data type get-methods. For example getInt-method will return a value from integer column. It has a single parameter that is the index of the column where the integer value is read out. The column parameter is zero based so if the first column has the items unique id it can be read out by calling getInt-method with column index zero. Reading database row values and constructing a new object is presented in Listing 29.

```
private ToDo cursorToToDo(Cursor cursor) {
    ToDo t = new ToDo();
    t.setId(cursor.getInt(0));
    t.setName(cursor.getString(1));
    return t;
}
```

Listing 29 – Transforming database row to actual object

When the database object handling completes the cursor can be moved to the next result item using the moveNext-method. The moveNext-method has the same feature as the moveToFirst- and the moveToLast-methods. It will return false if there is no more results. Information can be used again to complete the database object handling as soon as every item is handled. Listing 30 presents the iterating through result items.

```
cursor.moveToFirst();

while (!cursor.isAfterLast()) {
   ToDo toDoItem = cursorToToDo(cursor);
   //do something with the actual object

   cursor.moveToNext();
}

cursor.close();
```

Listing 30 – Using cursor and reading items out from the SQLiteDatabase

Updating objects in the database can be done with the SQLiteDatabases update-method. This method has four parameters. The first one is the table where the updated item is located. Second parameter is the ContentValue container. The container has key-value pairs to map the database column and the new value. Multiple columns can be updated with a single update.

Third parameter is the where clause, it defines which of the table rows are updated. If the parameter is left null every row in the table is updated so extra attention is needed. Updating values with the where clause is presented in Listing 31.

```
public void updateToDo(ToDo t) {
    ContentValues values = new ContentValues();
    values.put(DbHelper.COLUMN_NAME, t.getName());

    database.update(DbHelper.TABLE_TODO, values, DbHelper.COLUMN_ID +
"=" + t.getId(), null);
}
```

Listing 31 – Updating item in the SQLiteDatabase using items Id

Fourth parameter is the values of the where clause. When dealing with different where clauses and multiple columns in the where clause using the where parameter with the where values parameter leads into more maintainable and readable code.

On the other hand, if the where clause has only a single column, like for example items are updated based on their id-property. Then it would be easier to use only the third where parameter with an injected value. The where clause and the where arguments parameter work same way as in the query and update methods.

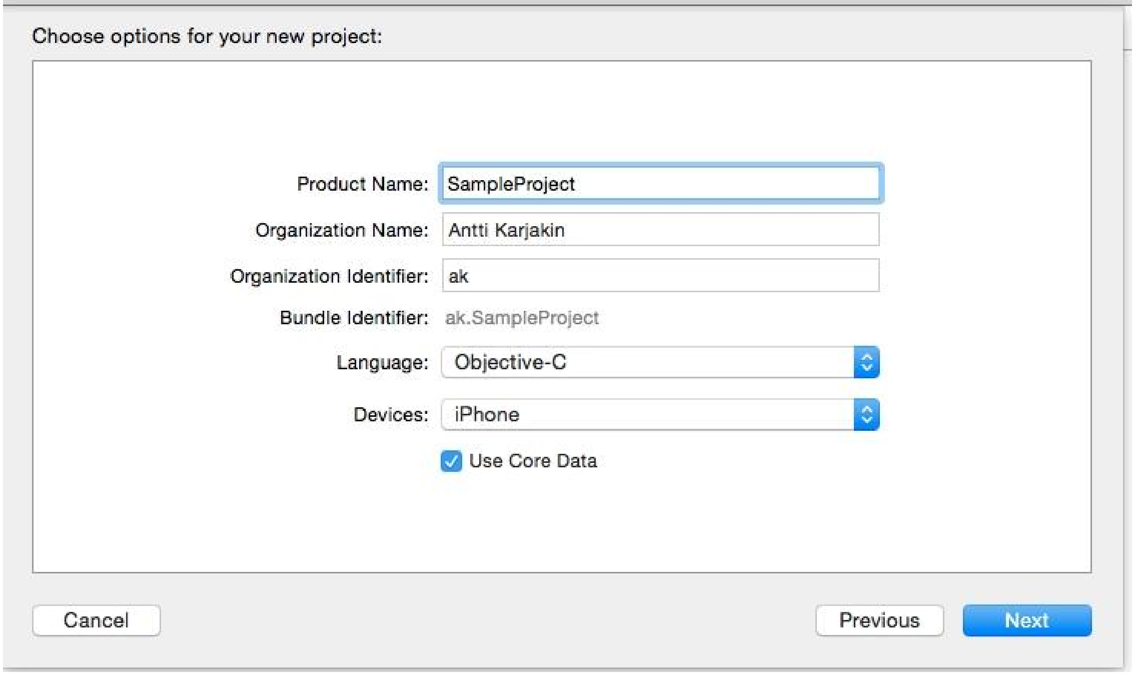
Deleting rows in the SQLiteDatabase can be done using the delete-method. The delete-method has three parameters. The first one is the table, the second one is the where-clause and the third one is the where clause arguments.

This method will return a number of rows that it deleted from the database with couple of exceptions. If no where clause is used all the table rows are deleted and the method will return a zero. If the developer wants to know how many rows there was it can be done by using a “1” in the where clause. When it is used the method will return a number of deleted rows.

8.2 Apple iOS

In this chapter the focus is in the Core Data framework. The developer should be familiar with the Objective-C language to understand given examples. Also knowledge about iOS memory management will help to understand the Core Data framework solution.

The easiest way to do a simple Core Data application is to start with a new project and on the project creation the use core data setting should be applied. Applying the core data setting on a new project creation dialog is presented in Figure 15. With the core data setting on a new project creation the project will be created with core data framework built into it. Project will also contain an empty data model and some default code in the project’s AppDelegate and on its view controller classes.



Choose options for your new project:

Product Name: SampleProject

Organization Name: Antti Karjakin

Organization Identifier: ak

Bundle Identifier: ak.SampleProject

Language: Objective-C

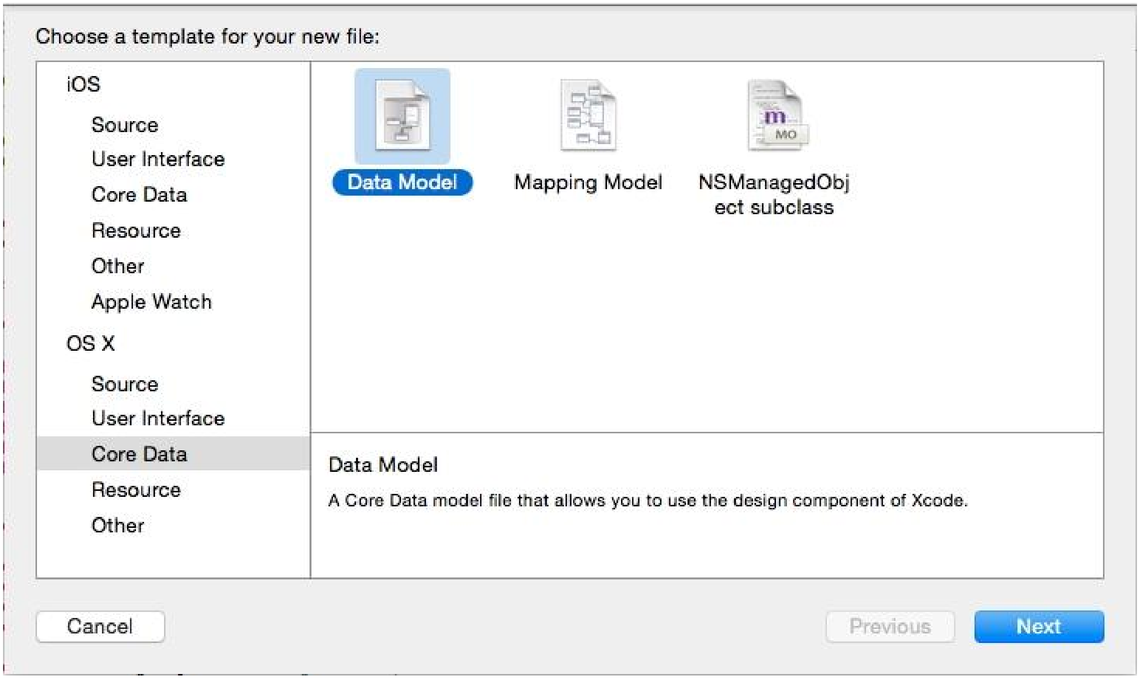
Devices: iPhone

☑ Use Core Data

Figure 15: Creating a new project in the Xcode using Core Data

The Core Data Framework can be also added to existing project, but all the necessary pats has to be added step by step what is included when creating a new project using the setting “use core data”.

Adding Core Data Framework to existing project can be started by adding the Core Data Framework to the project. After the framework is added the empty data model can be added. The data model can be added through File, new, file menu (shortcut command+N). From the opened “*choose template for you new file*” dialog one should select Core Data under iOS. In that group is a data model file. The example of a data model as a file type can be seen in Figure 16.



Data Model

A Core Data model file that allows you to use the design component of Xcode.

Figure 16: Adding empty data model to existing project in the Xcode

After the data model is added the next step is to add the Core Data basics code. One approach to this is to make empty Core Data project and copy the necessary parts from it to the new project. In the empty project there are three properties in the AppDelegate header file. The first one is the managed object context which is returned to use after it is tied to the persistent store coordinator.

The second property is the managed object model. It is application’s model which the controller uses. The third property is the persistent store coordinator which is used to save the model state to the persistent memory.

Also in the AppDelegate.h there should be the actual Core Data import and two private methods. The first method is the saveContext method which is called in the app delegate when the application will terminate. The second method is the applicationDocumentsDirectory method. This method will return the directory which the application uses to store the core data store file.

The core data basics code in the AppDelegate class is presented in Appendix 1.

The Core Data framework will now handle the database opening and closing and saving it to the persistent memory of a mobile device. When the developer updates the application and changes the model users who had already the previous version of the application will need a migration code to traverse their data through the update.

When inserting objects with the Core Data Framework the object is created using the NSEntityDescription class's insertNewObjectForEntityForName method. This method has two parameters. The first parameter is the name of the model which will be inserted and the second parameter is the managed object context. Inserting new object using the managed object context is presented in Listing 32.

```
//helper class which contains the save method has a pointer to the
managedObjectContext
NSManagedObjectContext *context = [self managedObjectContext];

NSString *thingTodo = @"Todo item 1";

//creating the new entity
Todo *todo = (Todo *)[NSEntityDescription insertNewObjectForEntityFor-
Name:@"Todo" inManagedObjectContext:context];

//set the text to the created object
[todo setThingTodo:thingTodo];

//saving the object to the context
if(![context save:&error])
{
    NSLog(@"Unresolved error %@ %@", error, [error userInfo]);
}
else
{
    NSLog(@"added a new todo: %@", thingTodo);
}
```

Listing 32 – Inserting new object to the core data managed object context

Core Data framework can be used with multiple context, but the developer should consider carefully when to move from one context to multiple contexts. Using multiple contexts will heavily effect the development if the developers are not familiar with it because it will make debug harder and more complex.

Although if the application is used to export or import large amount of data. It is better to have multiple contexts. The first context is used with the application's main thread and the second context is used to handle the data. This approach will ensure that the user's actions are still performed in the application. In this thesis only the single context use case is covered with the Core Data Framework.

After the new empty object is inserted the object properties can be set. To make sure that the new object contains the added properties the save-method should be called from the managed object context.

Querying objects from the Core Data frameworks managed object context can be done using the executeFetchRequest-method. The executeFetchRequest-method has two parameters that are the NSFetchRequest and the NSError. To create an NSFetchRequest object a help from the NSEntityDescription and from the NSPredicate class is needed.

It does not matter what is the order when creating all the necessary objects. In the example first the NSEntityDescription object is created using the entityForName-initializer. The entityForName has two parameters, the name of the entity and the managed object context.

The NSPredicate object works like a where clause in the SQL environments. The NSPredicate object is initialized using its predicateWithFormat. It has a single parameter and that is the where clause. In this example a Todo items entry date is compared to the date that the user is given. The comparing clause is given as NSString object.

When the NSEntityDescription and the NSPredicate objects are initialised the NSFetchRequest object can be initialized. The NSFetchRequest objects entity and predicate properties are set using the setEntity and setPredicate-methods. They both have a single parameter and that is the object that will be placed to the property. Previously initialized NSEntityDescription object is set to the NSFetchRequest class’s entity and the initialized NSPredicate object is set to the NSFetchRequest’s predicate.

Before the actual fetchRequest is executed, a nil (objective-c’s null is nil) NSError object is needed for the second parameter of the contexts executeFetchRequest method. With the NSError object the executeFetchRequest method can be called and it will return the NSArray object as a result. It can contain single or multiple objects depending of the used NSPredicate object. Making a query using the managed object context is presented in Listing 33.

```
//a class method is beign used when querying objects from the context
+(Todo *)getTodoWithContext:(NSManagedObjectContext *)context withEn-
tryDate:(NSDate *)date {

    NSEntityDescription *entityDescription = [NSEntityDescription en-
tityForName:@"Todo" inManagedObjectContext:context];

    NSPredicate *predicate = [NSPredicate predicateWithFor-
mat:@"entryDate == %@", date];

    NSFetchRequest *request = [[NSFetchRequest alloc] init];

    [request setEntity:entityDescription];
    [request setPredicate:predicate];

    NSError *error = nil;

    //the actual item request
    NSArray *array = [context executeFetchRequest:request er-
ror:&error];

    if(array == nil) {
        NSLog(@"getTodoWithContext:withEntryDate fetch failed");
        abort();
    }

    //handle the result
    ...
}
```

Listing 33 – Querying object using from the Core Data Framework

Updating and deleting items in the Core Data Framework can be done after the queried objects are handled. In the update the appropriate item is fetched from the managed object context. The properties of the item are changed using the set-methods for the entity properties. When the properties are changed managed object context save-method should be called. The save-method has a single parameter and that is the object of NSError class.

Deleting items from the Core Data Framework context can be done using the deleteObject-method from the managed object context. It has a single parameter and it is the object that is deleted. After the appropriate item is fetched and forwarded to the deleteObject-method the context must be saved by calling the already mentioned save-method again.

8.3 Windows Phone

In the Windows Phone platform structured data can be saved into the persistent memory using the support of Local Databases. The application’s local database file is located in the application’s own local folder in the memory of a mobile phone. Windows phone applications uses LINQ to SQL for all database operations [11]. And that is really for all the operations and all the operations only. Transact-SQL is not supported in the Windows Phone environment at all.

The Windows Phone application and the application’s local database share the same process. This limitation effects the database connections when the application tombstones. Every underlying database connection is closed at that very same moment. After returning the application back to use every suddenly closed query has to be made again.

When implementing the local database a subclass of DataContext-class is needed. That DataContext can be found in the System.Data.Linq library and the library should be at the project. If the project does not have the Linq-library a reference must be added. The database can be created in the main application constructor using the implemented subclass of DataContext.

In the main application constructor a new instance of the subclass can be created. And that subclass object should have a method DatabaseExists. If that method returns false the application database has been never created and it should be created now. Database creation is done by calling CreateDatabase-method from the instance of a subclass.

Tables of the database are defined in the DataContext-subclass. Local database should always have at least a single table. Table definitions are made using properties inside of the subclass. Properties are stamped with a type Table. Defining subclass of DataContext class with a single table is presented in Listing 34.

```
public class ToDoDataContext : DataContext
{
    ...
    // Specify a single table for the to-do items.
    public Table<ToDoItem> ToDoItems;
}
```

Listing 34 – Table of ToDoItems inside subclass of DataContext

These table properties have an entity type. These entities are also classes that represent tables in the underlying database. To make a database entity class a new class should be implemented. That class has to have an attribute TableAttribute which is in the System.Data.Linq.Mapping library.

Table columns are properties of a table class with a Column-attribute attached. These properties can be integer or string or other strong typed variables. Example of a class that represents table with a column can be seen in Listing 35.

```
[Table]
    public class ToDoItem
{
...
        [Column]
        public string ItemName
        {
            get;
            set;
        }
}
```

Listing 35 – Class with a Table attribute attached and single string column

Adding a new object to subclass of DataContext is done by using data context tables InsertOnSubmit-method. It has a single parameter that is the object which will be added. The added object has to be type of the collection inside the table collection. In this example it is typed as ToDoItem. So a new ToDoItem has to be created and then it can be added to the table.

Before the added object can be queried from the database table SubmitOnChanges-method has to be called. This method computes which of the objects are added, updated or deleted and executes the changes as a single command. With the help of SubmitOnChanges-method multiple changes can be made before actually changing the data in the actual database table. This can give performance boost if it is used right. There is rarely a need to call SubmitOnChanges-method after every change. Listing 36 presents how to add an object to the local database.

```
//instance of the database
var toDoDB = new ToDoDataContext(ToDoDataContext.DBConnectionString);

// Create a new to-do item based on the text box.
ToDoItem toDo = new ToDoItem { ItemName = “Todo thing nro 1” };

// Add a to-do item to the local database.
toDoDB.ToDoItems.InsertOnSubmit(toDo);

// Save changes to the database.
toDoDB.SubmitChanges();
```

Listing 36 – Adding an object to desired table

Querying objects from the local database tables is done by using Language-Integrated Query (LINQ). “*Language-Integrated Query (LINQ) is an innovation introduced in Visual Studio 2008 and .NET Framework version 3.5 that bridges the gap between the world of objects and the world of data.”* [14]

Explaining it would need at least its own chapter or a detailed introduction in this thesis so it is not covered. There are lot of good examples out there that explain how to use the LINQ. Almost every developer who works with Microsoft development tools is familiar with it.

Updating objects in the local database can be done by querying the actual objects from the database. Then changed their properties and called the SubmitOnChanges-method. When the Windows Phone application is implemented using the preferred guide lines, like the binding object values to the user interface the actual update method does not contain anything else.

Deleting objects from the local database is done by using the tables DeleteOnSubmit-method. It has only a single parameter that is the object to be deleted. Again it has limitation that it has to be type of the collection where it is deleted from. Deleting an object from the local database is presented in Listing 37.

```
private void RemoveToDoItem(ToDoItem toDoForDelete)
{
      // Remove the to-do item from the local database.
      toDoDB.ToDoItems.DeleteOnSubmit(toDoForDelete);

      // Save changes to the database.
      toDoDB.SubmitChanges();
}
```

Listing 37 – Deleting an object from the desired table

The Local Database shares many features and development features from the .NET based internet application development. A development team with knowledge about traditional .NET development could get familiar with the local database development.

8.4 HTML5

As already explained in Chapter 3, in the beginning of HTML5 standard there was two ways to support large data masses in the memory of a mobile phone browser. In this chapter first the preferred Indexed Database API is covered and the older Web SQL Databases then. If the HTML5 solution needs to support older phones alternative data storage could be implemented using Web SQL Databases but it should be avoided.

8.4.1 Indexed Database API

The Indexed Database API is the newest developed feature what comes to storing large numbers of objects locally in to the browsers. It is pretty easy to start using it. Adding, updating and deleting records are a quite straight forward procedure. The difficult part comes from the indexes which provide query functionality to the database. Database efficiency can be dramatically improved by using correct indexes.

The developers must adopt a little what comes to storing objects into indexed database when compared to the transient SQL database. Tables are called object stores. Single database can have multiple object stores like the relational database can have multiple tables. When creating the actual indexed database it does not contain any object stores, like when creating the SQL database it does not contain any tables.

Object stores can be changed later. But it can be made only by using a versionchange-transaction. As the indexed database is created or opened the method contains a version number. With the help of that version number the onupgradeneeded-method is called. Inside the onupgradeneeded method it is possible to change the database structure.

It should be heavily considered what are tried to achieve in the version change. There is always a possibility that the user leaps through a version, because he has been using the software only occasionally. So making very strict steps on the database upgrades could lead very shortly to lots of boilerplate code that could have lot of special problems with such of combinations that are more than money worth to figure out and then possibly fixed.

Before trying to perform any indexed database operations the application should test if the user’s browser is capable to use the indexed database API. This can be achieved with a help of third party library or with a few simple tests. The most basic test is to test that current window has indexedDB variable defined. The suggested next step (also very recommendable) is to check that the window has an IDBTransactionType variable set to the correct one (depending from a browser that is used). A simple check for browsers indexed database support is shown in Listing 38.

```
//testing if browser has indexedDB defined
if (!window.indexedDB) {
    //browser does not support indexedDB!
}

//android browsers handle things a bit different
if (window.webkitIDBTransaction != null) {
        if (window.webkitIDBTransaction.READ_WRITE != null) {
           //set the old version of transctionType to use
            idbClass.indexedDB.transactionType = win-
dow.webkitIDBTransaction.READ_WRITE;
        }
}
```

Listing 38 – Check if browser has support for indexed database api

The indexed database is created or opened using asynchronous open-method under indexDB-class. When the open-function is used for the first time the database is created. Otherwise previously created database will be opened. The open-method will return an IDBRequest object. With the help of the IDBRequest object it is possible to implement specified tasks to different events such as onsuccess and onerror. Example of opening indexed database can be seen in Listing 39.

```
//define the database version
var dbVersion = 1;

//call the asynchronous open-method
var request = indexedDB.open("Todo", dbVersion);

//setting the onsuccess method to notify that indexed database is
ready
request.onsuccess = function(e) {
    alert(“Database ready to use!”);
}

//setting the onerror method to notify that there was a problem
request.onerror = function() {
    alert(“Problem opening indexed database!”);
}
```

Listing 39 – Opening indexed database

After the database is created the object stores can be created to the database. Object stores can be added only inside the onupgradeneeded-event that can be found from the IDBRequest-object. This event is called when the database is created or when the database version is changed. For example if a new index is needed for an object store the database version must be changed to ensure that the onupgradeneeded-event is raised when the user opens the application next time.

The actual object store is created using createObjectStore-method. That method has two parameters. The first parameter is the name of the object store and the second parameter is the keypath to the objects in the objects store. The keypath is explained as: "*Defines where the browser should extract the key from in the object store or index. A valid key path can include one of the following: an empty string, a JavaScript identifier, or multiple JavaScript identifiers separated by periods. It cannot include spaces*" [29].

The createObjectStore-method creates and returns the object store that it created inside the connected database. The createObject-Store-methdod can be called only within a versiochange-transcation. Otherwise the InvalidStateError-exception is thrown. An example of object store creation can be seen in Listing 40.

```
//using the same request object than in the previous listing
request.onupgradeneeded = function (e) {
//the opened database can be fetched from the parameter e
var db = e.target.result;

// A versionchange transaction is started automatically so define the
onerror-method
e.target.transaction.onerror = function() {
    alert(“Problem inside versionchange transaction!”);
}
//if the object store is already defined remove it before adding it
back
if (db.objectStoreNames.contains("TodoStore")) {
    db.deleteObjectStore("TodoStore ");
}

//create the object store using the Todo-objects timestamp as the key
var store = db.createObjectStore("TodoStore", { keyPath: "timeStamp"
});
```

Listing 40 – Creating an object store to the indexed database

A big difference here compared to the usual databases is that table columns or object properties are not defined. The object store inside the indexed database is interested only that stored objects have the keypath values.

When the createObjectStore-method returns the created object store, it could be filled with default values using put-method. It is also possible to define more indexes to the object store to achieve better performance or make indexes that would help making effective searches to the object store.

Inserting objects into the object store can be done by using add or put-methods. Both of them have one required parameter and that parameter is the actual object that the developers want to save to the object store. If the object already exists in the object store and the developer is using add-method then a ConstrainError event is rised. Again the object store only compares values that are defined for the keypath for it. Inserting objects to the indexed databases object store is presented in Listing 41.

```
//using the same request object than in the previous listing
request.onsuccess = function(e) {
    var db = e.target.result;

    var tx = db.transaction([“Todos”], IDBTransaction.READ_WRITE);

    var store = tx.objectStore("TodoStore");

    store.add({title: "Do thing 1", timestamp: 201503311200});
    store.add({title: "Do thing 2", timestamp: 201503311201});
    store.add({title: "Do thing 3", timestamp: 201503311202});

    tx.oncomplete = function() {
    // All requests have succeeded and the transaction has committed.
    };
}
```

Listing 41 – Adding objects to the indexed database

When using the put-method it will update the existing object in the database or add a new one if it does not exist. The database can generate index keys by itself or the developer could specify each objects key using the second optional parameter in the put and in the add methods.

Deleting objects from the object store can be made using the delete method. It has a single parameter and that is the key of the object. For example objects are stored into the object store using the timestamp as a keypath. Deleting objects can be done by calling the delete-method and giving the appropriate timestamp as a parameter.

An entire object store can be cleared using the object stores clear-method. The object store will clear itself on a separate thread. So it will not lock the user interface while it empties itself.

Querying objects in the indexed database object store can be made using the get-method. It has a single parameter and that is the value of the queried object key in the object store. It is a handy method if the developer needs only a single object from the object store.

When the indexed database is used more like ordinary database and there are lot of objects and the developer needs a lot of objects as a result then the indexed database cursor (IDBCursor) must be used.

To control what values the cursor returns the developer must be familiar with the key range (IDBKeyRange) of the indexed database. This is the biggest difference what comes to many ordinary relational databases. It can be a little confusing in the beginning but when starting to study the indexed database an extra attention should be given to KeyRange mechanism.

The get-methods result of indexed database object can be also achieved by using the key range and key range method only. The difference here is that the cursor may return multiple objects if it is used with an index that is not used as key path.

For example, if there are categories in the todo list. For example the home and the work categories are made and the todo items are divided into these categories. It would be possible to query only work related todo items using single query. That query could be made using an index that points to the selected group and then fetch the items from the indexed database store using the only-method.

Adding new indexes to the object store can be made using the createIndex-method in the object store. The method has three parameters. The first one is the name of the index, the second one is the keyPath for the index to use and the third one is the optionalParameters.

Now it must be remembered that every object in the database must have the keyPath values defined in the store creation. Indexes rely on the data that is on the object store. Their keyPath values are not mandatory but indexes cannot return objects which does not have the keyPath properties.

Index keyPath can contain multiple properties from the objects. For example it would be possible to create index to the todo object store using the name and timestamp.

Under the research process of this thesis a problem was encountered. The problem was that Microsoft's Internet Explorer (not even the latest version, the IE 11) could not handle the multidimensional indexes under indexed databases. When creating the multi column indexes in the Internet Explorer using the object stores createIndex-method everything looks working normally. Any indications (unwanted errors or events) that would alert the developer are not shown.

Adding new objects to the object store was also working fine. Objects were added to the object store and there were not any errors shown or any error events fired.

Problems start to arise when the program was written so it would use the index as part of a query that fetches objects from the object store to the user interface. When the query was processed the browser invoked a data error event.

Multi column indexes were not implemented to the Internet Explorer 10 at all and this thesis research indicated that the Internet Explorer 11 does not work properly.

Web browser in the Windows Phone Lumia product family inherits from the desktop Internet Explorer browser. Multi column indexes problem was pretty severe in the research process because Lumias are so popular in Finland and the application should work on the Windows platform also.

When writing this thesis it was not possible to install better alternate browser to the Lumia phone. There is an alternate browser in the Windows Phone Marketplace, but it does not support indexed database at all.

The risen problem was resolved by adding a property to the object store objects. The new property contains the values from the both property that would be normally used with the compound index. Then a new index had to be made. An index that uses only the created property which have both of the values what the compound index would be used normally.

This will slightly affect the performance of the indexed database queries and there are reasons why indexed database was designed to support multi column indexes.

When the indexed database has an appropriate index a query can be made. A simple query is executed by using the indexed databases transaction, key range and cursor opening.

First a transaction is needed to the correct indexed database using at least read access right. From the indexed databases transaction an object store where the queried data exists is opened. The second step is to get the appropriate index from the object store using index-method. Index-method has a single parameter and that is the name of the queried index.

The third step is to create the IDBKeyRange. For example the developer wants to query Todo items only from the precise moment. There should be an index to the Todo object store that points to the timestamp-property. By using IDBKeyRanges only-method it is possible. The only-method has single parameter and that is the value which is used in the query.

From the selected index the openCursor-method is called. This method has single parameter that is the IDBKeyRange which is used on the query. Querying data from the indexed database is presented in Listing 42.

```
var trans = db.transaction(["ToDos"], IDBTransaction.READ);
var objectStore = trans.objectStore("todoStore");

var index = objectStore.index("timeStamp");

var rangeTest = window.IDBKeyRange.only(201503311200);

index.openCursor(rangeTest).onsuccess = function (e) {
```

Listing 42 – Querying data from the indexed database

The onsuccess events implementation function can contain a parameter that will be initialized as an object of the success event. From the event the returned data can be found under the events target property. The target property contains a result property. If the result property exists then it has a value property that contains the actual object from the Indexed Database.

After handling the item that is returned from the Indexed Database requests continue-method can be called and the cursor will move to the next item. When all the items in the result are handled a return should be called to end the onsuccess-event. An example of handling items can be seen in Listing 43.

```
index.openCursor(rangeTest).onsuccess = function (openEvent) {
    var result = openEvent.target.result;

    if(!!result === false) {
        //no result / all the lines read
        return;
    }

    var todoItem = result.value;

    //do something with the item
    todoitemHandler(todoItem);

    //continue reading results
    result.continue();
}
```

Listing 43 – Handling the data queried from the indexed database

8.4.2 Web SQL Database

Web SQL queries remresembleind a lot transact-SQL queries. That is because the browsers use SQLite as their backend implementation.

Before using Web SQL, the application should inspect if the current browser that accesses the application supports Web SQL databases. In the simplest way this can be tested by checking if JavaScripts openDatabase-variable is specified. There are also third party libraries that provides methods to tests the browsers Web SQL capabilities.

After ensuring the support for the Web SQL databases the actual database can be created. To create the Web SQL database the OpenDatabase-function should be called. The OpenDatabase-function returns a handle to the actual Web SQL database. If this is the first time when this database is opened it will be created. Otherwise it opens the previously created database.

The OpenDatabase-function has five parameters that are the name of the database, version of the database, the display name of the database (also known as human readable database name), estimation of the database size and lastly the callback-method what will be called when the database has been successfully created.

The OpenDatabase callback-method is called only when database is created not when it is opened. So for example if the database must have some data before using it this could be the place where to determinate if the database was recently created or not.

The database size is given in bytes. So for example when trying to estimate that the application's database could consume up to 5 megabytes it has to be transformed to bytes. Because units used in the information technology the five has to be multiplied with a square of 1024. Example of opening a Web SQL database can be seen in Listing 44.

```
//using local variable to help estimating database size
var dbSize = 5 * 1024 * 1024;

//open the database
var testDatabase = openDatabase("Todo", "1", "Simple Todo planner",
dbSize, todoDbCreatedSuccesfully);

//the callback method for the openDatabase
function todoDbCreatedSuccesfully() {

}
```

Listing 44 - Sample of opening the browsers local storage

After the database has been created or opened it can be used. Database needs table or tables where the data can be put into. After the database is created or opened every action to database is made using the database handles transaction-method. That method can be used with one to three parameters.

The first parameter and the most important in the transaction-method is the SQLTransactionCallback. The second parameter is SQLTransactionErrorCallback for handling errors when errors occur. The third parameter is SQLVoidCallBack. All these parameters are inherited from the SQLTransaction interface.

The SQLTransaction interface has an executeSql-method which is used for all the data manipulating in the Web SQL database. The executeSql-method can be used with one to four parameters where the first parameter is the actual sqlStatement. The sqlStatement is the only required parameter.

The second parameter is the SQL statements arguments in an object array. The third parameter is optional call back for success call back and the fourth one is used as error call back method.

The created database will need some tables to work with. Creating tables to the opened database can be done with a simple create table transact-SQL statement. There can be only one table with a desired name in the database so there is a possibility to exception when the database is opened next time. A good way to avoid this is have table exist check before the actual table create transact SQL. Adding a table to the Web SQL database is presented in Listing 45.

```
//testDatabase variable is the handle for the opened database

//add new table to database if it is not already existing
testDatabase.transaction(function(tx) {
  tx.executeSql(“CREATE TABLE IF NOT EXISTS Todo(ID INTEGER PRIMARY
KEY ASC, Task TEXT)”);
}
```

Listing 45 - Sample of adding new table to Web Sql Database

After the tables are created data can be added to them. For example some default data what is needed in the application. To add data to the Web SQL database table it can be done by using same executeSql-method. The default data should be inserted only when the table is created not on every application opening.

It would be possible to check if the table does not exist and then continue forward from there with some flag that indicates that default data must be added. Another type of solution can be made using the transaction-method from the webSql database handle.

As already mentioned, the transaction function has one mandatory and three optional parameters. Those parameters are the SQLTransactionCallback, SQLTransactionErrorCallback and SQLVoidCallBack. The decision about adding default values could be made inside a single transaction. That transaction could be implemented as a separate call back method and that method could be called when the database is opened.h

Inside a single transaction it is possible to make a single SQL command or multiple commands. If every SQL command inside a single transaction only reads the data readTransaction-method could be used. When using the readTransaction-method the database mode has to be read-only. And when using the plain transaction-method the mode must be read/write. Example of using multiple SQL clauses in single transaction is presented in Listing 46.

```
var addTableAndData = function(db) {
    db.executeSql(‘CREATE TABLE IF NOT EXISTS Todo(ID INTEGER PRIMARY
KEY ASC, Task TEXT)’);
    db.executeSql('INSERT INTO LOGS (id, log) VALUES (1, "Todo sample
1")');
    db.executeSql('INSERT INTO LOGS (id, log) VALUES (2, "Todo sample
2")');
};

//testDatabase variable is the handle for the opened database
testDatabase.transaction(addTableAndData);
```

Listing 46 - Sample of adding new table and data using single transaction

Reading objects from the webSQL database is done by using appropriate transact SQL clauses as executeSql statements. For queries without where clause executeSql-method can be called with a single parameter. If there is need for single or to multiple where clause then the values of the where clause should be added using the second statement arguments parameter. Reading data from web sql database is presented in Listing 47.

```
//do something with the Todo items
var readAllFromTodoComplete = function(db, result) {
    if(result.rows && result.rows.length > 0) {
        ...
    }
}

//actual transact sql statement
var readAllFromTodo = function(db) {
    db.executeSql(‘SELECT * FROM Todo’, [], readAllFromTodoComplete);
};


//testDatabase variable is the handle for the opened database
testDatabase.transaction(readAllFromTodo);
```

Listing 47 - Sample of reading data from Web Sql Database

The where clause is written normally but the arguments of the where clause should be converted to a question marks as placeholders. SQL injection is a truly security risk even in the actual web applications and it could be used to break the HTML5 application as well. Even the standard states that: “*Authors are strongly recommended to make use of the ? placeholder feature of the executeSql() method, and to never construct SQL statements on the fly.*” [12]. Listing 48 presents the use of a where clause in the query.

```
//actual transact sql statement
var readSingleFromTodo = function(db, todoId) {
    db.executeSql(‘SELECT * FROM Todo WHERE ID = ?’, [todoId]
};


//testDatabase variable is the handle for the opened database
testDatabase.transaction(readSingleFromTodo(testDatabase, 1);
```

Listing 48 - Using where clause while reading data from Web Sql Database

Like other actions in the Web SQL database also deleting objects from the database is done by using the executeSql-method. Deleting objects is done by using the delete from statement, familiar from the transact SQL. It is important to write the correct where clause if there is no need to delete every item in the database table. Example of deleting objects is presented in Listing 49.

```
//actual transact sql statement
var deleteFromTodoById = function(db, todoId) {
    db.executeSql(‘DELETE FROM Todo WHERE ID = ?’, [todoId]
};


//testDatabase variable is the handle for the opened database
testDatabase.transaction(deleteFromTodoById(testDatabase, 3);
```

Listing 49 - Deleting object by id from the Web Sql Database using where clause

8.5 Database Versioning and Migration in Mobile Environment

Different mobile platforms have different solutions to save lots of data but the solutions have a common part, the database versioning. This mechanism can be found in the SQLite solutions (Android SQLite and Web SQL) and it can be also found in other (Windows Phones SQL CE and Indexed Database API) solutions.

The principle of the database versions is that when the database is opened its version is compared to the version that was given in the opening request. If the versions match the database is opened and given to use. If the requested version is newer than the one that is stored to the devices memory something has to be done.

When starting the application development this is not important and the developer can quite freely update the version number while the features are developed to the application. Things get more complicated when the first version of the application has been delivered to the clients.

After the first version is released and first users have been using the application the database version changes are important on every update. If the application is updated frequently and the database change on each update it is possible that users leap through a version. The code implementing the database version change should be capable not only to do a single update but to do multiple updates on a single run.

What should and can be done on the database version change? Adding new tables or some fields is not a big deal. But if a large update with data manipulation is done it is possible that the update could take more time than the user is willing to wait. Android suggests that the migration should be done on a separate thread so the user is not disturbed if the migration takes more time than wished.

Migration situations should be designed so that they are easy to maintain and that they perform in a reasonable time. It could be taken as far as when the database version is changed the old database is deleted and the new one is constructed from the beginning. This approach has a drawback if there is data that cannot be obtained from other sources.

9 Summary and Conclusions

The research demonstrated that every mobile platform has quite a unique way of saving data to the persistent memory of the mobile phones. If a native application is being developed the guidelines given by the operating system development team should be obeyed. The frameworks that are shipped alongside the operating system are more advanced and usually have better performance than the community libraries.

Mobile applications are usually developed using agile software development methods. The application is developed piece by piece and often shipped feature by feature. When working with data that is saved into the persistent memory of the phone, the database versioning should be taken seriously. How to handle the different cases, how to traverse through multiple versions and how to perform the database changes so that the user is not disturbed too much? With native applications the developer has slightly more advanced techniques available to perform long and heavy operations compared to the HTML5 based applications.

Developing a mobile application that supports multiple platforms is not an easy case. With HTML5 it is possible and fairly easy to have a native application kind of feel, user experience and usability. The finished application and the updates can be shipped to users instantly. Developing the mobile application with the HTML5 is not all the time as easy as developing a native application. When the application also has to fulfil special needs that rely on hardware components things get complicated.

During the research it was noticed how fast mobile phones are developing. For example in a case where a user adds photos to an internet application. The photos were taken with the camera of a mobile phone. HTML5 standard introduces an input control attribute that enables the camera in the phone to be used as a source of a photo. This feature did not work at all in the Windows Phone environment when the research began. But the Cyan update that was released by Microsoft added the support for the Windows Phones.

During the research it was found out that even using the latest standard to store the data can have pitfalls. Again it was the Windows Phone environment that has problems since the support for the indexed database does not support all the features.

When comparing two mobile phones that have almost identical hardware specification the one with the Android operating system feels smoother to use and for example scrolling long lists is faster than in the phone running the Windows Phone operating system.

Alongside the research of this thesis a HTML5 based application was developed to the Android, iOS and Windows Phone platforms. The knowledge gained from the research will be used to add the offline support for the application. The key role of HTML5's Application Cache must be taken into account when adding the offline support.

The objective of this thesis was to get acquainted with offline data support in the mobile applications. The application could be native, hybrid or HTML5 based. Different mechanisms to store data to the user's device were covered in the theory section. The old techniques from the html based solutions were also covered. Specific solutions from current mobile platforms were inspected and also new possibilities offered by the HTML5 standard were researched.

Actual mobile environments (Googles Android, Apples iOS, Microsofts Windows Phone) each have their own way to save actual data into the persistent memory of handheld devices. They each have a preferred solution given by the operating system developer. The HTML5 offers the same behaviour using only the internet browser in the mobile phone.

The HTML5 standard is developing fast and it should be taken seriously as a competitor for the native mobile applications. When storing data to the user's browser only the preferred and still approved mechanisms should be used.

The objective of the thesis was fulfilled. The notable part being the indication that offline data can be stored using only the HTML5 based solution. With the gained knowledge the offline capability will be added to the existing HTML5 based application at Granlund.

References

1 Jain, Chetan K. **jQuery Mobile Cookbook**. Olton Birmingham, GBR: Packt Publishing Ltd; 2012.

2 Smutny P. **Mobile development tools and cross-platform solutions** in: 13th International Carpathian Control Conference (ICCC), 2012.

3 Olson S, Hunter J, Horgen, Turid H. **Professional Cross-Platform Mobile Development in C#**. Hoboken, NJ, USA: Wiley; 2012.

4 Chuan S. **HTML5 Mobile Development Cookbook**. Olton Birmingham, GBR: Packt Publishing Ltd; 2012.

5 Xamarin Development Center Documentation - **Building Cross Platform Applications**. URL: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/. Accessed 17 January 2015.

6 Renegar B D, Michael K, Michael M G. **Privacy, Value and Control Issues in Four Mobile Business Applications** on Mobile Business, 2008. ICMB '08. 7th International Conference on; July 2008

7 The Microsoft Developer Network - **How to create a settings page for Windows Phone 8**. URL: http://msdn.microsoft.com/en-us/library/windows/apps/jj657972.aspx. Accessed 8 February 2015.

8 iOS Developer Library - **Property List Programming Guide**. URL: https://developer.apple.com/library/ios/documentation/Cocoa/Conceptual/PropertProper/QuickStartPlist/QuickStartPlist.html#//apple_ref/doc/uid/10000048i-CH4-SW5. Accessed 9 February 2015.

9 The World Wide Web Consortium (W3C) - **Web Storage**. URL: http://dev.w3.org/html5/webstorage/. Accessed 8 February 2015.

10 Android Developers Guide – **Storage Options**. URL: http://developer.android.com/guide/topics/data/data-storage.html. Accessed 15 February 2015.

11 The Microsoft Developer Center - **Local database for Windows Phone 8**. URL: https://msdn.microsoft.com/en-us/library/windows/apps/hh202860%28v=vs.105%29.aspx. Accessed 15 February 2015.

12 World Wide Web Consortium – **Web SQL Database**. URL: http://dev.w3.org/html5/webdatabase/. Accessed 15 February 2015.

13 World Wide Web Consortium – **Indexed Database API**. URL: http://www.w3.org/TR/IndexedDB/. Accessed 16 February 2015.

14 The Microsoft Developer Center – **Introduction to LINQ**. URL: https://msdn.microsoft.com/en-us/library/bb397897.aspx. Accessed 8 March 2015.

15 iOS Developer Library – **Reachability**. URL: https://developer.apple.com/library/ios/samplecode/Reachability/Introduction/Intro.html. Accessed 15 March 2015.

16 Mac Developer Library - **Core Data Programming Guide**. URL: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/CoreData/cdProgrammingGuide.html. Accessed 16 March 2015.

17 World Wide Web Consortium - **Offline Web applications**. URL: http://www.w3.org/TR/2011/WD-html5-20110525/offline.html. Accessed 20 March 2015.

18 HTML5 Rocks by Google - **"Offline": What does it mean and why should I care?.** URL: http://www.html5rocks.com/en/tutorials/offline/whats-offline/. Accessed 10 January 2015.

19 A List Apart - **Application Cache is a Douchebag** by Jake Archibald . URL: http://alistapart.com/article/application-cache-is-a-douchebag. Accessed 27 March 2015.

20 Oracle Java Documentation - **What Applets Can and Cannot Do**. URL: http://docs.oracle.com/javase/tutorial/deployment/applet/security.html. Accessed 28 March 2015.

21 The Microsoft Developer Center – **Silverlight Trusted Applications**. URL: https://msdn.microsoft.com/en-us/library/ee721083(v=vs.95).aspx. Accessed 28 March 2015.

22 Google Developers - **Managing HTML5 Offline Storage**. URL: https://developer.chrome.com/apps/offline_storage. Accessed 29 March 2015.

23 Mozilla Developer Network – **Online and offline events**. URL: https://developer.mozilla.org/en-US/docs/Online_and_offline_events. Accessed 29 March 2015.

24 World Wide Web Consortium - **WebSimpleDB API**. URL: http://www.w3.org/TR/2009/WD-WebSimpleDB-20090929/. Accessed 30 March 2015.

25 W3C Editor's Draft – **File API**. URL: http://dev.w3.org/2009/dap/file-system/file-dir-sys.html. Accessed 30 March 2015.

26 StatCounter – **Global Stats**. URL: http://gs.statcounter.com. Accessed 5 April 2015.

27 Android Developers Guide – **Saving Key-Value Sets**. URL: http://developer.android.com/training/basics/data-storage/shared-preferences.html. Accessed 17 February 2015.

28 Android Developers Guide - **Determining and Monitoring the Connectivity Status**. http://developer.android.com/training/monitoring-device-state/connectivity-monitoring.html. Accessed 18 February 2015.

29 Mozilla Developer Network - **Basic concepts of Indexed Database**. URL: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Basic_Concepts_Behind_IndexedDB#gloss_keypath. Accessed 20 February 2015.

Core Data basics

```
//
//  AppDelegate.h
//  testCoreData
//
//  Created by Antti Karjakin on 19/03/15.
//  Copyright (c) 2015 Antti Karjakin. All rights reserved.
//

#import <UIKit/UIKit.h>
#import <CoreData/CoreData.h>

@interface AppDelegate : UIResponder <UIApplicationDelegate>

@property (strong, nonatomic) UIWindow *window;

@property (readonly, strong, nonatomic) NSManagedObjectContext
*managedObjectContext;
@property (readonly, strong, nonatomic) NSManagedObjectModel
*managedObjectModel;
@property (readonly, strong, nonatomic) NSPersistentStoreCoordinator
*persistentStoreCoordinator;

- (void)saveContext;
- (NSURL *)applicationDocumentsDirectory;


@end
```

```
//
//  AppDelegate.m
//  testCoreData
//
//  Created by Antti Karjakin on 19/03/15.
//  Copyright (c) 2015 Antti Karjakin. All rights reserved.
//

#import "AppDelegate.h"

@interface AppDelegate ()

@end

@implementation AppDelegate


- (BOOL)application:(UIApplication *)application
didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    // Override point for customization after application launch.
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application {
    // Sent when the application is about to move from active to inactive state. This can
occur for certain types of temporary interruptions (such as an incoming phone call or
SMS message) or when the user quits the application and it begins the transition to the
background state.
    // Use this method to pause ongoing tasks, disable timers, and throttle down
OpenGL ES frame rates. Games should use this method to pause the game.
}
```

```
- (void)applicationDidEnterBackground:(UIApplication *)application {
    // Use this method to release shared resources, save user data, invalidate timers, and store enough application state information to restore your application to its current state in case it is terminated later.
    // If your application supports background execution, this method is called instead of applicationWillTerminate: when the user quits.
}

- (void)applicationWillEnterForeground:(UIApplication *)application {
    // Called as part of the transition from the background to the inactive state; here you can undo many of the changes made on entering the background.
}

- (void)applicationDidBecomeActive:(UIApplication *)application {
    // Restart any tasks that were paused (or not yet started) while the application was inactive. If the application was previously in the background, optionally refresh the user interface.
}

- (void)applicationWillTerminate:(UIApplication *)application {
    // Called when the application is about to terminate. Save data if appropriate. See also applicationDidEnterBackground:.
    // Saves changes in the application's managed object context before the application terminates.
    [self saveContext];
}

#pragma mark - Core Data stack

@synthesize managedObjectContext = _managedObjectContext;
@synthesize managedObjectModel = _managedObjectModel;
@synthesize persistentStoreCoordinator = _persistentStoreCoordinator;
```

```
- (NSURL *)applicationDocumentsDirectory {
    // The directory the application uses to store the Core Data store file. This code uses
a directory named "ak.testCoreData" in the application's documents directory.
    return [[[NSFileManager defaultManager] URLsForDirectory:NSDocumentDirectory
inDomains:NSUserDomainMask] lastObject];
}

- (NSManagedObjectModel *)managedObjectModel {
    // The managed object model for the application. It is a fatal error for the application
not to be able to find and load its model.
    if (_managedObjectModel != nil) {
        return _managedObjectModel;
    }
    NSURL *modelURL = [[NSBundle mainBundle] URLForResource:@"testCoreData"
withExtension:@"momd"];
    _managedObjectModel = [[NSManagedObjectModel alloc]
initWithContentsOfURL:modelURL];
    return _managedObjectModel;
}

- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
    // The persistent store coordinator for the application. This implementation creates
and return a coordinator, having added the store for the application to it.
    if (_persistentStoreCoordinator != nil) {
        return _persistentStoreCoordinator;
    }
// Create the coordinator and store

    _persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc] initWithManagedObjectModel:[self managedObjectModel]];
    NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppendingPathComponent:@"testCoreData.sqlite"];
    NSError *error = nil;
    NSString *failureReason = @"There was an error creating or loading the application's saved data.";
    if (![_persistentStoreCoordinator addPersistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL options:nil error:&error]) {
        // Report any error we got.
        NSMutableDictionary *dict = [NSMutableDictionary dictionary];
        dict[NSLocalizedDescriptionKey] = @"Failed to initialize the application's saved data";
        dict[NSLocalizedFailureReasonErrorKey] = failureReason;
        dict[NSUnderlyingErrorKey] = error;
        error = [NSError errorWithDomain:@"YOUR_ERROR_DOMAIN" code:9999 userInfo:dict];
        // Replace this with code to handle the error appropriately.
        // abort() causes the application to generate a crash log and terminate. You should not use this function in a shipping application, although it may be useful during development.
        NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
        abort();
    }

    return _persistentStoreCoordinator;
}
```

```
- (NSManagedObjectContext *)managedObjectContext {
    // Returns the managed object context for the application (which is already bound to
the persistent store coordinator for the application.)
    if (_managedObjectContext != nil) {
        return _managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoordinator];
    if (!coordinator) {
        return nil;
    }
    _managedObjectContext = [[NSManagedObjectContext alloc] init];
    [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    return _managedObjectContext;
}

#pragma mark - Core Data Saving support

- (void)saveContext {
    NSManagedObjectContext *managedObjectContext = self.managedObjectContext;
    if (managedObjectContext != nil) {
        NSError *error = nil;
        if ([managedObjectContext hasChanges] && ![managedObjectContext
save:&error]) {
            // Replace this implementation with code to handle the error appropriately.
            // abort() causes the application to generate a crash log and terminate. You
should not use this function in a shipping application, although it may be useful during
development.
            NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
            abort();
        }
    }
}

@end
```