Bachelor Degree Project

Synchronization and data merging between iOS, server and database

- Solution for setup of synchronized offline capable ***crud*** *functionality between iOS client and server*

Author: Mikael Melander
Supervisor: Johan Hagelbäck
Semester: ST 2018

Abstract

Keywords: iOS, Node js, MySQL, crud, offline synchronization, self-hosting, GDPR, Firebase, Azure

Contents

1 Introduction

This degree project will study and develop a solution for an iOS framework with server integration to handle offline data synchronization and data merging.

The study aims to examine the best way to develop a free self-hosting solution that implements rules for data merging as well as data structures that can be handled by both iOS and an online server database solution.

1.1 Background

The world of mobile applications has exploded in the last couple of years, a research conducted in Sweden[1], show that the average person above 12 years' old who use internet on their mobile phone everyday is up to 76% by 2017. And companies increasingly adopt the functionality of mobile devices, everything from mail to checking train tickets, what used to be done by pen and paper or only by sitting down at a computer is now possible with mobile applications.

Because mobile devices can be taken anywhere, work should be able to follow, but most applications demand an external database to store data across users, this, in turn, requires a mobile data connection or WIFI by the device itself.

This is a problem when working in places where cellular reception is limited or unavailable. When this occurs it will greatly limit the productivity in the work being preformed. The solution to this is to be able to add, see and use the data that already exists within the application regardless of available cellular reception.

1.2 Related work

Research around project that solve the iOS, server and data integration with offline capabilities have been conducted and there are some big and famous companies that have created some type of solutions to the problem. These are solutions that work really well and integrates the solution that this project aims to solve. They have some drawbacks and exactly how they solve it is not disclosed.

Firebase is Googles database service that allows the creation of a mobile/web application connected to the database within a couple of easy steps. They have a complete framework for iOS, android, windows phone and web solutions like JavaScript and more. They also have a set of tools to

verify data, handle security, analyze usage, send push notifications and many more. Firebase as a Google solution locks the user down to their own way of thinking, to use it is means to use it on their terms or not at all. It is only possible to store data in their database and build the solution around them[2].

Microsoft Azure also supports the ability for offline client synchronization, but as well as Firebase it relies on the use of their cloud services[3].

Within this area, there are a lot of different solutions that solve the problem, but they solve the problem within their particular code language and platform. Examples of this are solutions that work for frameworks such as **Xamarin** that is a cross-platform .NET solution, **React-native** that is a cross-platform JavaScript solution and **Ionic** that is a JavaScript cross-platform solution that works with html5 instead of native as react-native does [4][5][6]. These solutions aim to solve the problem specifically for their platforms, locking the user down to their client and backend framework. For companies that want to have a self-hosted environment to be able to protect their own data, either for security reasons or because of laws this will become a problem.

An article written by Mohammad Faiz and Udai Shanker[7] explains different solutions for merging situation by bringing up, data synchronization techniques, like the Timestamp synchronization that uses timestamps to keep track of when the data was last modified. They also mention the fact of the conflict possibilities when working with offline capable solutions. If changes is made to an object when a user is offline, and that user in turn changes the data while offline, there will be a conflict that needs to be solved.

1.3 Problem formulation

The goal is to find a suitable solution that is a reusable starting point for developing a mobile application within iOS that connects to a server and database backend. A project that already has the functionality of connecting the server, database and iOS frameworks together.

The entire solution needs to be self-hosted to address the data privacy perspective of companies, because when using cloud services, one can never be sure exactly where or how the data is stored and who has access to it. This is especially important since the new GDPR law (General Data Protection Regulation[8]) that demands both higher security and responsibility for data protection, as well as with higher penalties than the DPD law it replaces explained by Katrin Schaar[9].

The finished project should be able to query, edit and upload data locally/offline and automatically handle the upload/query to the online server

database. This requires developing and defining a database structure that correctly keeps track of data versions, to allow the system to handle different data versions being received and sent by different users/devices that have edited data while offline. To be able to handle these data merges, it is also required to develop a set of rules or algorithms to handle the merges correctly. The rules need to support multi-tenant usage and should not corrupt any data.

1.4 Motivation

Today there exists online solutions to setup databases, server and application frameworks to handle all of the server and client communications, including offline capabilities. For example, Firebase and Azure, that also provide extensive functionality surrounding the complete solution, but using these means the need to handing sensitive data to a third-party.

Many companies want a solution for mobile applications associated with their company that will not allow the data to be stored anywhere other than in their own control. This is to protect any and all company or users information from a third party, as well as to follow laws and regulations of data privacy, the new GDPR law for example.

To create a solution that would function well for work the offline capabilities is essential as no reception should not be in the way of any productivity. This coupled together with the fact that it would be self-hosted, giving the company themselves control over the data, where it is stored, how it is hosted, what firewalls it is behind and how the backups are conducted. This would give strong motivations to support and use a solution like this, in a time when laws regarding data privacy is tightened and more regularly enforced.

If you as a company that creates IT solutions for other companies, can offer a re-useable solution that can be further developed on with more functionality over time, that is both time and cost efficient, it will offer you a great new selling point.

1.5 Objectives

O1	Research and determine server platform, data structure and language
O2	Implement connection and crud functionality to the server database
O3	Implement local storage iOS and crud functionality locally
O4	Implement automatic synchronization between local and server database
O5	Implement methods for querying data given specific arguments that handle both local/server database
O6	Implement functionality to keep track of data versions
O7	Implementation of data merge rules

The big goal of the project is to find out if it is possible to create an offline synchronization solution between server and iOS device that could be used as a replacement to the big corporation solutions, a solution that gives control of all the aspects of the data. As well as find the suitable implementations of merge rules for the data to be able to build a starting point of a solution that is actually usable as a real replacement.

To be able to achieve this we require a server framework that is free to use and can handle at least one of the same database languages that Xcode, Apple development tool for iOS applications, can handle to be able to implement a seamless integration of the server and local database.

With that in place, the implementation of crud functionality on the server side needs to be added. The implementation of the local database can then be implemented to be able to reflect the server database.

After the database structure is created, the implementation of the functionality to let the framework automatically keep track of the data and update both the local and server database can be made.

When all of this is working the server side implementation of merging data versions should be determined and implemented.

1.6 Scope/Limitation

Within the scope of this project, the solution should include a server implementation as well as an iOS framework implementation. The solution should be a starting point for projects, that is reusable.

The solution should have full crud functionality, meaning to be able to create, read, update and delete data on both the local and server database and be able to keep the databases synchronized as long as the devices has an internet connection.

The solution is to be an open sourced project the dependencies used within the development should all be free, this not including the iOS development certificate needed to deploy applications on the AppStore.

Because of the time limitations of the project, it will not take into consideration the security aspects meaning that it will not have a solution for HTTPS or that it will have any users or data access lists.

The iOS code should be written in Obj-C within Xcode and support iOS latest three versions.

1.7 Target group

This project can be of interest to companies, organizations or persons wanting to be in control of their own data and host their own solutions that integrate with iOS in a cost-efficient way.

The solution should be considered as an open starting point to keep building upon, that already has the important implementations for server integration and offline data support.

1.8 Outline

The next chapter will present the **methods** that are used to execute the different objectives that are presented above.

The **implementation** chapter will be a more detailed explanation of how the project will be implemented and how the solution itself works, how the merge rules will be executed and how the automatic syncing is handled.

The **result** chapter will cover what came of the project, what the resulting structure of the solution became.

The **analysis** will cover an overall analysis of the concluded results.

The **discussion** will deeper discuss the analysis and results.

Chapter seven will include **conclusions** that are based on the results as well as present future work and recommendations.

2 Method

The method used to conduct this project will be verification and validation.

The project is not created in any collaboration with a company, meaning there is not a given outline from an external source to create these requirements. So to get the requirements for this method, the defined problems for the project will be converted into requirements.

This project does not build upon already existing code or will not use any existing code that will have to be collected (This does not include the frameworks, platforms and dependencies that will have to be used). This means that the functionality of the project will be based upon written code for the functions that need to be implemented, so the requirements will make sure that the implementation and functionality works as intended and in that case are fulfilled.

By using the verification and validation method, it is possible to see if the project supports the functionality that is required by verifying and validating the requirements with different manual tests that are connected to the required part of the implementation.

2.1 Requirements

The requirements are converted from the problem formulation and the test will be conducted manual running on a Mac with an iOS simulation and a Docker environment running the server. The requirements are presented Table 2.1 below.

Requirement	**Description**	**Test**
1	Dependencies are free	Manual
2	Saves data **offline**	Manual
3	Query data **offline**	Manual
4	Edit data **offline**	Manual
5	Remove data **offline**	Manual
6	Save data **online**	Manual
7	Query data **online**	Manual
8	Edit data **online**	Manual
9	Remove data **online**	Manual
10	A working implementation of merge rules	Manual
11	Synchronize data between local and online database	Manual

Table 2.1 Requirements

2.2 Reliability and Validity

To be able to use the verification and validation method correctly and be sure that the results are reliable, the requirements that were create needs to be measurable and objective. This means that it is necessary to make sure that the requirements can not be subjective, if they would to be subjective, different people can interpret the requirements in different ways. If this happens the reliability of the results could be compromised.

The requirements created from the problem definitions will be broken down in small pieces that will be easier to understand and phrased in a way that should be easy to confirm or deny if it is fulfilled or not. For example, “Is the data saved locally if no internet connection is available?”, it will be a simple yes or no answer. Conducting it in this way will help to ensure the reliability of the method and that the verification and validity are correct.

The verification and validation method is most often used to confirm the results of a working project to a customer, but because the project is conducted by only one person the validity of the results might be questioned. Therefore, there is an even bigger reason for the requirements to be as simple and direct as possible. This project aims to create a solution that will continuously be developed after this project. There would be no reason for the results of the verification and validity to not be correctly conducted.

2.3 Ethical Considerations

The project goal is to create an deployable server, iOS and offline data synchronization solution. The solution should be open sourced and has a potential to be worked on more to create extra functionality and widen the scope. By conducting this project there should not be any reason for any ethical issues to come to light.

3 Implementation

This chapter will explain how the solution was conducted and implemented. It aims to explain a bit of the technicality surrounding the project.

3.1 Goal

The plan for the complete solution is an iOS client application working with the built framework, the framework needs to be able to have full CRUD functionality with the local database, to return the result when saved locally to the user for faster loads and offline capability. Then depending on internet access send the CRUD request to the server.

The server then validates the data version and executes the merge rules, saves the **correct** data to the database and sends a response back to the framework, the framework now updates the local database if needed and sends the final response back to the client. An overall design of the framework and the server can be seen in Figure 3.1

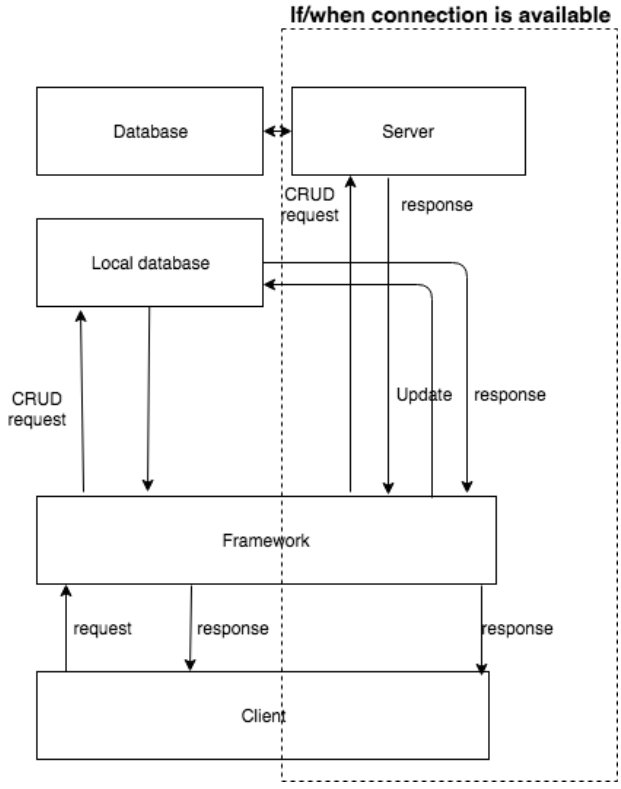

Figure 3.1 Design

3.2 Database

Before the server framework could be determined the database language had to be decided. The choice was set between MySQL and MongoDB and decided based on the pros and cons in Table 3.2 below[10].

MySQL	**Pros:**
	Mature
	Support Join
	Privilege and password
	Native iOS support
	Cons:
	Stability concerns
	None community driven
MongoDB	**Pros:**
	Integrated storage engines
	Dynamic schemas
	Cons:
	No native iOS support
	Younger solution

Table 3.2 Relational database

The decision for this solution landed on MySQL. MySQL has been around for a long time and a lot of people are familiar with it, but the biggest reason for the choice was that the native support already exists within iOS which would grant better integration and only require dependencies from Xcode and apple themselves.

3.3 Server

To determine the server framework to use, there were some restraints.
The framework had to be free, and support the database type that was decided earlier in Table 3.2.

Below in Table 3.3 are the considerations and the important pros and cons of the server framework [11]:

Node js	**Pros:**
	Fast
	Full Stack
	Lightweight
	Big open source library
	Cons:
	Unstable API
	Less fitting for CPU intensive tasks
Ruby on rails	**Pros:**
	Flexible
	High quality (because of set standards)
	Evolved framework with a lot of tools
	Consistent
	Cons:
	Slow
	Large stack frame
	Depends on Apache/Nginx or something similar.

Table 3.3 Server framework

The decision landed on Node js. Node js is a free solution written with JavaScript and is widely used across the world and supports Npm. It is also known to have good performance as can be supported by more than one article [12],[13]. Npm is used as a collaborative community that gives developers free access to libraries and components of re-usable code. This in term is optimal for this project since the goal is to make it open source, that gives the possibility for people to keep building upon it.

The complete server solution is built as a REST API with full CRUD functionality, integrated with the MySQL database. The server receives an http call with the CRUD request from the client and compares the data received with the data currently in the database, if it exists, then updates the database according to the merge rules. It then sends back a response to the client framework as seen in Figure 3.4 below.

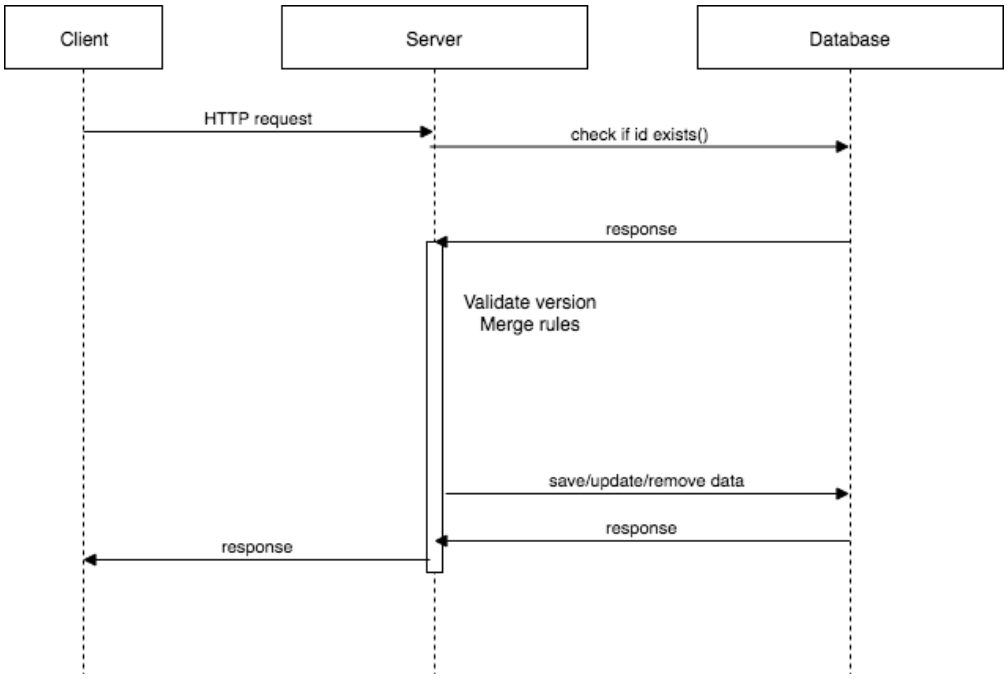

Figure 3.4 Server framework

3.4 iOS

The goal of the iOS framework is to more or less work as a middle hand that handles all the logic between the server/databases and the user.

It provides a set of ready to use functions to query, save, update and delete data, meaning full CRUD functionality.

The framework should always query the local database first, to keep the query as fast as possible. The client is served with the local database data that can be used while offline and for example to pre-render the UI, then the framework should check the online database and return the response data to the client.

The same goes for when the framework saves data, the data is saved locally first and returned to the user, then in the background sent to be saved to the server. If there is no connection available, the background request will be paused until a connection returns, then it is sent to the server. This process is shown in Figure 3.5 below.

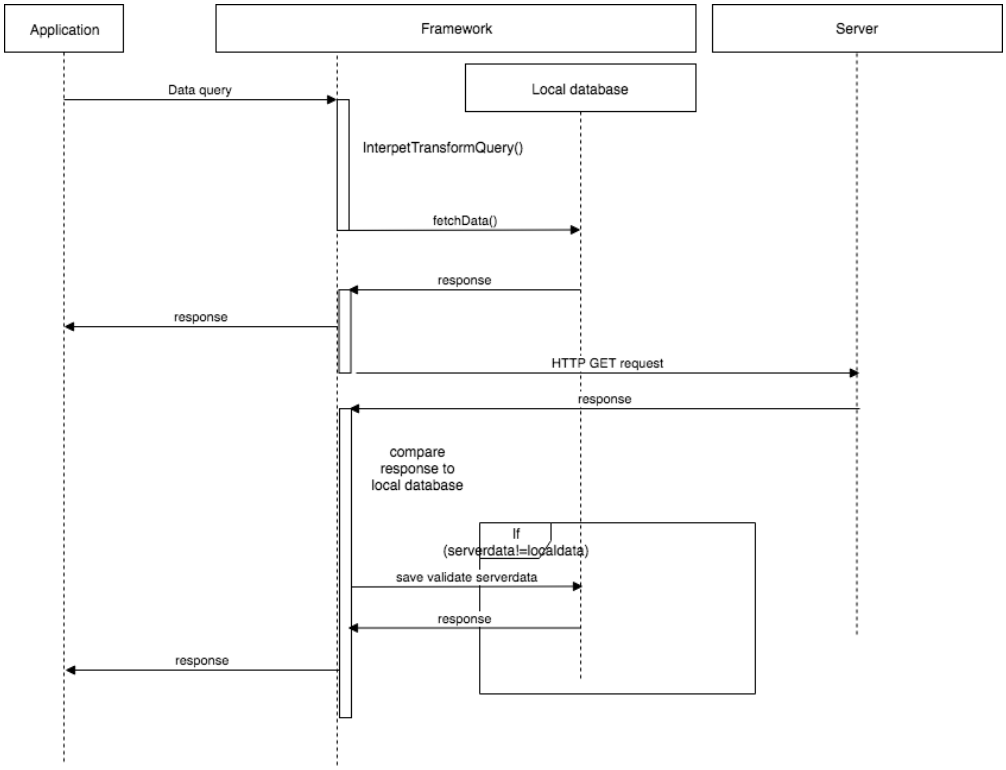

Figure 3.5 iOS framework data query

3.5 Validation and merge rules

The validation and merge rules are the rules the server applies to decide what data is the correct one to keep in the online database. Because several users or the same user with different devices might manipulate, update and delete the same data while some are offline and some are online the data might exist in several versions at the same time on different devices. To decide what data is the one to keep in the database there needs to be some rules the server can use. The rules considered and implemented follows in the Table 3.6 below.

Name	Description	Comment
Newest wins	The most recent data timestamp always wins.	This might become a problem because the updated time will be set on the device. If the user decides to change the time manually on the phone. Example if the user locally changes Year, month or day.
Server wins	Synching offline data (data older than current time) to the server will be disregarded.	For applications wanting to use the framework without the offline update version.
Client wins	Ignores the conflict and changes are overwriting any value in the online database.	This approach might lead to the loss of important data changes
User decides	The user is told about the conflict and gets to decide	This approach was **removed** from consideration because the framework should work with as little effort as possible from users.

Table 3.6 Validation and merge rules

The validation and merge rules are implemented on the server side before the save occurs, the choice of what rules to choose is set as parameters on the server.

Some columns in the data structure is required for the server to be able to handle the rules, this is implemented on all the data by the framework on the client side before it is sent to the server. These can be seen in Table 3.7 below.

Column	Description
createdAt [timestamp]	Locally creates a timestamp when creating a object
updatedAt [timestamp]	When the object is created the timestamp is the same as createdAt but this is the one that updates on each change

Table 3.7 Required object data

3.6 Problematic use cases

Whit regard to offline capabilities and synchronization, to not corrupt any data by accident, some rarer use cases need to be take in to consideration. Below in Table 3.8 use cases related to this is listed.

Use case	Cause
Application is terminated by user before the data is sent to the server	The none sent local data have not been synchronized to the server and will be overwritten when the applications fetches server data, causing corruption.
Application device losing internet connection in the middle of synchronisation	
Local changes have not been synchronized to the server when the application was terminated. On start-up the application fetches new data from the server database.	

Table 3.8 Problematic use cases

To be able to solve these problems Xcode has some delegate methods that can help the framework address this issue, further explained by Apple[14], and can be seen in Table 3.9 below.

Method	Action
applicationDidEnterBackground	Notifies when the application did become inactive
applicationDidBecomeActive	Notifies when the application did become active
applicationWillTerminate	Notifies when the application is about to terminate

Table 3.9 Delegate methods

By implementing this, the framework will know when the application is about to be deactivated, terminated or started and can push the none synchronized changes to the server before doing anything else, removing the risk of overwriting the local changes.

4 Results

The result chapter is based on the tests of the final product, according to the requirements presented in Table 2.1.

4.1 Server and database

The frameworks that were selected to run as the server and database, are free to use, according to requirement 1, can be seen in the Table 4.1 below.

Platform	**Choice**
Server	Node js
Database	MySQL

Table 4.1 Sever and database results. ***Requirement 1***

4.2 Save and query data

To save data, the framework will convert the input data to a MySQL insert as can be seen in A Appendix 1 and will automatically add two columns, as seen in the response, the columns represent timestamps for createdAt and updatedAt as well as the ID column that will always be represented as a 1 in these examples for simplicity.

The Table 4.2 below is based on the example user input of for a save action

User Input	**Framework conversion**	**Local response**	**Server response**
(Växjö-Kalmar, 0,0)	(1, Växjö-Kalmar, 0,0, 2018-04-29 11:21:01, 2018-04-29 11:21:01)	(1, Växjö-Kalmar, 0,0, 2018-04-29 11:21:01, 2018-04-29 11:21:01)	(1, Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:21:02)

Table 4.2 Save data responses. ***Requirements 2, 6 and 11*** *(No merge rules on save data)*

When querying data, the framework supplies a given function, using this function that takes a MySQL parameter the framework first returns local data, then online data, this is if they both exist, otherwise it will return nil. The example in Table 4.3 shows the responses for data that exists in both the local and online database. The data is the same because the local data was overwritten by the servers on the initial response in Table 4.2. This can be further explained when looking at Figure 3.5.

User input	**Local response**	**Server response**
Select * from exampleTable	(1, Växjö-Kalmar, 0,0, 2018-04-29 11:21:02, 2018-04-29 11:21:02)	(1, Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:21:02)

Table 4.3 Query data responses (only one object in table). ***Requirements 3, 7 and 11***

4.3 Merge rules

The results below will be shown for the different merge rules implemented. Merge rules is only applicable to updates to an object, this is why the current data already has the required columns.

The Table 4.4 and Table 4.5 will show the different responses for an update example according to the merge rules, the server response is always the **winning** data, that **always** will be saved as local as well.

Examples for >updatedAt data **All input data sent 11:30:00**		
Current data in database online and offline	(1, Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:21:02)	
Input data	(1, Växjö-Kalmar, 1,0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	
Rule	**Local response**	**Server response**
Newest wins	(1, Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1, Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:01)
Server wins	(1, Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1, Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:21:02)
Client Wins	(1, Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1,Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:01)

Table 4.4 Edit data responses. ***Requirements 4, 8, 9 and 11***

Examples for >updatedAt data All input data sent 11:30:00		
Current data in database online and offline	(1,Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:45:05)	
Input data	(1,Växjö-Kalmar, 1,0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	
Rule	**Local response**	**Server response**
Newest wins	(1,Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1,Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:45:05)
Server wins	(1,Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1,Växjö-Kalmar, 0, 0, 2018-04-29 11:21:02, 2018-04-29 11:45:05)
Client Wins	(1,Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:00)	(1,Växjö-Kalmar, 1, 0, 2018-04-29 11:21:02, 2018-04-29 11:30:01)

Table 4.5 Update data responses for offline data edited earlier than server stored data. ***Requirements 4, 8, 9 and 11***

Table 4.6 below will show and explain some alternative scenarios and the result of the situation.

Rule	Scenario	Local response	Server response
Newest wins/ Server wins	Attempting to remove a local object that has been updated more recently on the server.	Removes object	Returns the updated object
Client Wins	Attempting to remove a local object that has been updated more recently on the server.	Removes object	Removes object
All	Attempting to update an object that has already been deleted from the server	Updates object	Returns error, object still removed

Table 4.6 Alternative scenarios. ***Requirements 5, 9 and 11***

5 Analysis

The goal of the project was to create a server, online database and iOS framework to handle automatic synchronization between data and support offline capabilities within the application. When looking at the results in chapter 4 of the report they support the knowledge that this is feasible within the scope of the project that was created. The result shows that the solution supports the main use cases that the merging rules are designed for.

The local data will always first conform to the users input data, it will be able to edit, save and delete data so that the framework is always usable, even if the device does not have any reception. Because the createdAt/updatedAt local data sent to the server will conform to the devices own time settings this can become somewhat of a problem. If someone would to manually change the time on the device the data sent to the server would be handled with that timestamp.

Then depending on the selected merge rules the data change accordingly when an internet connection and data synchronization is initialized. This shows that the project would be able to replace the bigger cloud solutions if it requires offline synchronization capabilities for iOS, but still want control of all data. And even the offline capabilities is not required, it will still be an open sourced project that already implements a functioning crud database solution for iOS.

Because it is running on a Node js server and is open source this project could expand in the open source community and create several additional functions and purposes which was one of the final goals of the project.

6 Discussion

The biggest reason for this project, was the possibility to show that having an open sourced backend to handle the situations for when companies and individuals normally would turn to the big companies for their ready to go solutions is not as needed as most people think. The biggest reasons to use their backends is the functionality that they have ready to go, stability, scalability, continues updates, and simplicity.

The functionality aspect is why the project chose to use be open sourced and run on Node js, this in turn would give the project a chance to have functions continuously implemented. At the same time Node js, Npm already has a big library of components that are able to be integrated to give this functionality without any to big investments of time.

The stability and scalability might be seen as a trade of for having control of your own data, but at the same time having a self-hosted solution gives you the choice of where the server are placed and what hardware the servers are run on. This is obviously more work than to just set up the solution within a couple of steps, the way the big companies solutions more commonly works, but it gives you the power of how it is done. This also means that you are able to create all the code and implementation and when it is done, you can handle it off. Letting customers or individuals handle the hosting themselves. This is one of the biggest reasons for this project to get traction in the open sourced community, the self-hosting and full control of data is in this time, with the new GDPR law, essential for a lot of companies handling and storing personal data.

The simplicity of the big companies is hard to compete with, you can by logging in to their website be up and running within moments. Their solutions give you some chances to manipulate the built in functionality but far from everything. Firebase for example has a real-time database with offline capabilities but no chance to directly change the merging rules, although this can be managed by adding a self-hosted server, or one from googles app engine. Then the simplicity is not as simple anymore. Because this project gives you full access it is easy use one of the already implemented merge rules or almost as easy to implement your own. It gives you more control at the cost of simplicity.

6.1 Validity of results

The test comes from the requirements of the project and is seen in chapter 4. They were conducted by manually inputting the requests in the code on the iOS client side and running it. It was tested several times and with different data structures and the results was as expected.

The part that can question this data is that it has only been tested on a Node js Docker enviorment, running on the same computer as the iOS simulator. This should not compromise any of the result, but the solution has not been tested on a self-hosted server.

7 Conclusion

In the thesis, the goal was to try to find out if it was possible to create a self-hosted replacement for current solutions like Firebase or Azure.

The scope of the project did not for obvious reasons include all the functionality these expanded solutions already support, so the final product of this project is not meant to be a full replacement, but a proof of concept.

The basic implementation of the project includes to be able handle the part of offline data synchronization, a part that Firebase and Azure already provides, as well as handling data merge rules, in a self-hosted environment. As the results in chapter 4 shows, this has been successfully implemented and tested according to the set requirement.

There is a lot more aspects of a complete solution like Firebase and Azure, more than is brought up in this thesis, performance, stability and scalability for example. All areas where Firebase and Azure perform really well, but the data is entrusted to a third party. To solve that problem, the solution has to be self-hosted, and in that situation, this solution could be a feasible one. The hope of this project is that it will over time get more functionality implemented and become a real replacement over Firebase and others.

Whit all the requirements implemented successfully the conclusion is that although it is not a full replacement, it is a proof of concept that it could become one.

7.1 Future work

The time limitations of the project made the scope narrow against what could have been possible.

The big picture of the project would include an android framework, more functionality of the backend, like push notifications, file storage, data analytic tools, more merge rules or even algorithms. There is close to no limitations regarding implementation to a open sourced system like this, so the possibilities are almost endless.

In the more current picture, the security aspect of the system would have been implemented in regards to user access lists and https to protect the data. It would as well have been interesting in a user friendly perspective to have a script based setup of the entire project.

References

[1] **Svenskar och internet**
http://www.soi2017.se/allmant-om-utvecklingen/internet-i-mobilen/
Accessed: *2018-02-30*

[2] **Firebase: Offline capabilities on iOS**
https://firebase.google.com/docs/database/ios/offline-capabilities
Accessed: *2018-02-29*

[3] **Azure: Enable offline syncing with iOS mobile apps**
https://docs.microsoft.com/en-us/azure/app-service-mobile/app-service-mobile-ios-get-started-offline-data
Accessed: *2018-02-29*

[4] **OpenSoure: Does your app work without an internet connection**
http://opensourceforu.com/2017/01/mobile-app-without-internet/
by Minni Arora, 2017
Accessed: *2018-03-01*

[5] **Github: SQlite-Sync.com version 3**
https://github.com/sqlite-sync/SQLite-sync.com
Accessed: *2018-03-18*

[6] **Frontmag: Offline data synchronization in Ionic**
https://frontmag.no/artikler/utvikling/offline-data-synchronization-ionic
Accessed: *2018-03-18*

[7] **“Data synchronization in distributed client-server applications”**
by Mohammad Faiz and Udai Shanker, 2016.

[8] **General Data Protection Regulation**
https://en.wikipedia.org/wiki/General_Data_Protection_Regulation
Accessed: *2018-05-16*

[9] **”What is important for Data Protection in science in the future?”**
by Katrin Schaar,
Humboldt-Universität zu Berlin, Institute for Psychology 2016
pp: 1-4

[10] **MongoDB vs MySql**
https://hackernoon.com/mongodb-vs-mysql-comparison-which-database-is-better-e714b699c38b
Accessed: *2018-04-09*

[11] **Medium: NodeJS vs Ruby on Rails…**
https://medium.com/@TechMagic/nodejs-vs-ruby-on-rails-comparison-2017-which-is-the-best-for-web-development-9aae7a3f08bf
by TechMagic
Accessed: *2018-04-10*

[12] **”Node.js Paradigms and Benchmarks”**
by Robert Ryan McCune,
University of Notre Dame, 2011
pp: 3-4

[13] **“Is Node.js a viable option for building modern web applications? A performance evaluation study”**
by Ioannis K. Chaniotis, Kyriakos-Ioannis D. Kyriakou, Nikolas D. Tselikas, 2013.
pp: 1036

[14] **Developer Apple**
https://developer.apple.com/documentation/uikit/uiapplicationdelegate/
Accessed: *2018-05-18*

A Appendix 1

Example of framework input conversion:

User input:

```
INSERT INTO exampleTable VALUES (Växjö-Kalmar,0,0)
```

Framework output:

```
INSERT INTO exampleTable VALUES (1, Växjö-Kalmar,0,0, 2018-04-29
11:21:01, 2018-04-29 11:21:01)
```