

Aino Leppänen

**AVEX MOTTI**

# **AVEX MOTTI**

Aino Leppänen  
Bachelor's thesis  
Spring 2016  
Information Technology  
Oulu University of Applied Sciences

## ABSTRACT

Oulu University of Applied Sciences  
Degree programme in Information Technology

---

Author: Aino Leppänen  
Title of the bachelor's thesis: AVEX Motti  
Supervisor: Pertti Heikkilä  
Term and year of completion: Spring 2016  
Number of pages: 34 + 1 appendix

---

This application was created to help civilians and Finnish Defence Forces communicate in a crisis situation where all traditional networks would be unavailable.

The software was developed using the MVC (Model View Controller) architectural pattern. By combining HTML5 (The fifth revision of Hyper Text Markup Language), AngularJS, browser's local storage and Apache CouchDB running on server side we created an application that would solve the problem of communication in a crisis situation with minimal costs, no need for installing an application and fast setup.

The results were very positive and with more funding and time the development could be continued further. This application could also be implemented for different usages with slight modifications, wherever communicating is necessary and network connection cannot be guaranteed.

---

Keywords: offline, browser, crisis, NoSQL, AngularJS

## **PREFACE**

This project was done in Oulu and Helsinki during winter of 2015. Remod Oy participated in this project that was organized by Finnish Defence Forces as a voluntary refresher course.

I would like to thank all my co-workers at Remod Oy for giving me this opportunity to work on this interesting topic. Special thanks to Tuukka Kivilahti who worked as the CTO (Chief Technical Officer) and my supervisor, Jesse Hulkko who administrated this project and Tuomas Riihimäki who worked as our backend expert and is currently working as the CEO (Chief Executive Officer) of the company.

Also I would like to thank my instructor from Oulu University of Applied Sciences Pertti Heikkilä for helping me during this thesis and offering guidance for the writing part of this work.

The last thanks go to Finnish Defence Forces and Jaakko Latikka who worked as their organizer for AVEX projects and showed a genuine interest in our project and the issues we were trying to solve.

In Oulu, 23.5.2016

Aino Leppänen

# TABLE OF CONTENTS

ABSTRACT	1
PREFACE	2
TABLE OF CONTENTS	3
VOCABULARY	5
1 INTRODUCTION	8
2 POC	9
2.1 AVEX Motti	9
2.2 Software requirements	11
3 REMOD OY	12
3.1 Company	12
3.2 Finnish Defence Force AVEX refresher team	12
4 PROJECT CONTROL	13
4.1 Jira	13
4.2 Git	14
4.3 GitLab	14
5 DATABASES	15
5.1 Differences between SQL and NoSQL	15
5.2 PouchDB	16
5.3 CouchDB	16
5.4 JSON document structure	16
6 PROGRAMMING	19
6.1 AngularJS	19
6.2 HTML5	21
6.3 UX	22
6.3.1 Technical Implementation	22
6.3.2 UI Design	23
7 BROWSERS	25
7.1 HTML local storage	25
7.2 Application Cache	25

7.3 Modernizr	25
7.4 Browser fingerprint	26
8 LEGAL ISSUES	27
8.1 Finnish Communications Regulatory Authority	27
9 DEMONSTRATION	28
9.1 Scope	28
9.2 Changes	29
9.3 Preparation	29
9.4 Demonstration	30
10 CONCLUSION	31
REFERENCES	32
APPENDICES	34

## VOCABULARY

TERM	DEFINITION
AngularJS	An open source web application framework mainly maintained by Google, written in Javascript.
Apache CouchDB	A document-oriented NoSQL database that uses JSON to store data, JavaScript as its query language using MapReduce and HTTP for an API.
API	An <u>A</u> pplication <u>P</u> rogramming <u>I</u> nterface, a set of routines, protocols and tools for building a software and applications.
CRUD Syntax	<u>C</u> reate, <u>r</u> ead, <u>u</u> ppdate and <u>d</u> elete. Four basic functions of a persistent storage.
DRY	<u>D</u> on't <u>r</u> epeat <u>y</u> ourself, a principle aimed at reducing the repetition of information.

FDF

Finnish Defence Forces

HTML5

The fifth revision of the HyperText Markup Language

HTTP

An Hypertext Transfer Protocol, an application protocol for distributed, collaborative, hypermedia information systems.

Javascript

A programming language developed by Netscape Communications Corporation, Mozilla Foundation and Ecma International.

JSON

A JavaScript Object Notation, an open standard format that uses a human-readable text to transmit data objects consisting of attribute-value pairs.

MVC

A Model-View-Controller, an architectural pattern mostly for implementing user interfaces.

NoSQL

A non-relational database. Providing a mechanism for storage and retrieval of data, which is modelled in means other than the tabular relations used in relational databases.



PoC

A Proof of Conocept, a realization of a certain method or idea to demonstrate its feasibility.

POJO

A Plain Old JavaScript Object, an ordinary object, not bound by any special restriction and not requiring any class path.

PouchBD

An open-source JavaScript database that is designed to run well within the browser.

# 1 INTRODUCTION

The AVEX Motti project started when the Finnish Defense Forces organized a voluntary refresher course, and told the volunteers to think about the given issues and also let them organize how they would solve the issue and present their solution to the Finnish Defense Forces.

Their aim was that by outsourcing problem solving the issues given would be solved more efficiently and they would also update their way of handling them without a gigantic bureaucratic system.

The project group of Remod Oy personnel had to think of a way to solve these issues:

- What should be done if we suddenly lost all information networks?
- How to give civilians instructions about what to do in a crisis situation?
- How to create a working taskforce of these civilians to give information back to the Finnish Defense Forces?

The current society has become very dependent on networks. Considering how much of the information received every day is delivered via the Internet it is good to have a working backup plan in case we would suddenly lose them.

There was a solution on how to skip most of the costs in this kind of situation by using a technology that everyone already carries in their pockets: smartphones. By creating an application that would work in a browser there would be no need to focus on native applications to create a PoC (Proof of Concept).

In a crisis situation civilians have a natural will to restore the situation back to normal.

## 2 POC

In this chapter the idea of this project will be further explained.

### 2.1 AVEX Motti

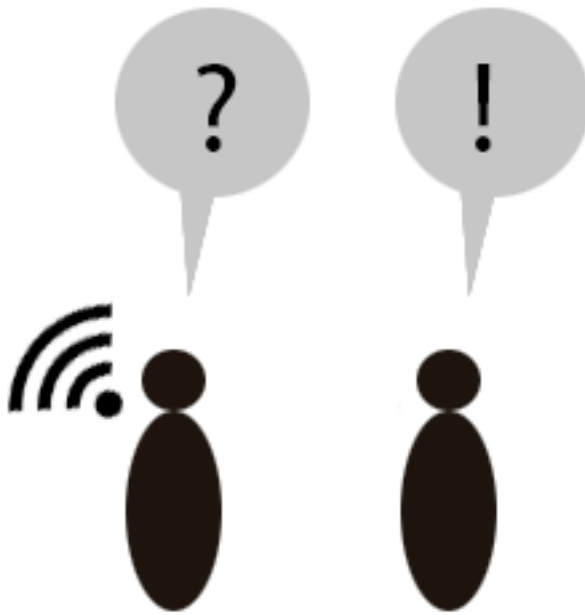
So in a case that networks disappear a temporary Wi-Fi spot could be set up with just IT device (laptop, Raspberry Pi), a power supply and a Wi-Fi-router.



*FIGURE 1. Wi-Fi-spot.*

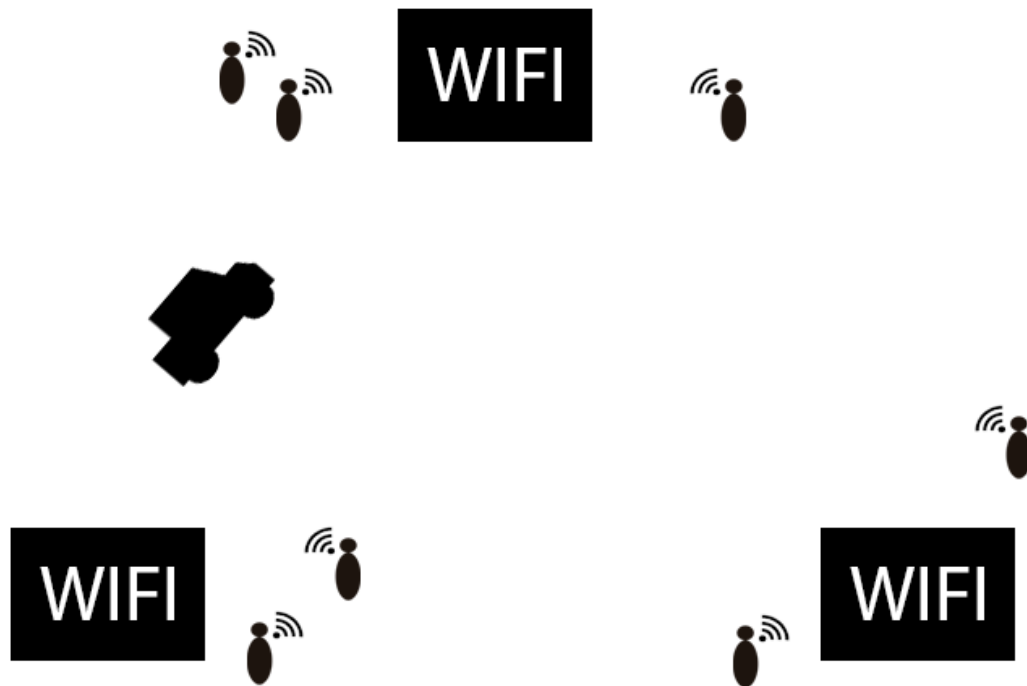
These kinds of spots could be set up to multiple locations in a city and this could be a backup network for a crisis situation. After this civilians could synchronize their phone to an application database offering instructions on what to do and also giving tasks for them to enroll for as operators. These kinds of tasks could

be, for example to check how much water there is in a given apartment complex.



*FIGURE 2. Gathering information.*

After finishing the task and collecting this information the operator can come back to the Wi-Fi spot and synchronize information in their phone with database of the application. While operators are executing these tasks there would be one person moving between different Wi-Fi spots and by carrying their own smartphone with them, synchronizing all the databases of different spots.



*FIGURE 3: Synchronizing databases between different wifi-spots.*

In this way information can be shared in a large area and FDF (Finnish Defense Forces) personnel will have a better knowledge of where help should be prioritized and how to solve the situation with better efficiency.

## **2.2 Software requirements**

To ensure that this application would work on all platforms of the PoC level it was made into a web application instead of a native application. Being a web application it had to have an offline database to carry all the information in the browser instead of saving it to the custom application's memory space.

As few features as possible were desired to be depending on online access which meant that no libraries with links, Google Maps API (Application Programming Interface) or images downloaded from server. This also meant that there was a limited space to store data.

## **3 REMOD OY**

Remod Oy was the company commissioning this project.

### **3.1 Company**

The company is focused on designing, developing, maintaining and fixing ICT (Information and Communications Technology) structures. They also work in software development, virtualization, databases and information networks.

### **3.2 Finnish Defence Force AVEX refresher team**

Remod Oy team consisted of the following persons:

- Juho Juopperi
- Jesse Hulkko
- Tuukka Kivilahti
- Tuomas Riihimäki

Other people involving in this task were:

- Pekka Pussinen
- Pasi Eronen
- Mikko Jakonen
- Aleksi Mustonen
- Marko Nordberg
- Tuomas Reinvuori
- Markus Virtanen

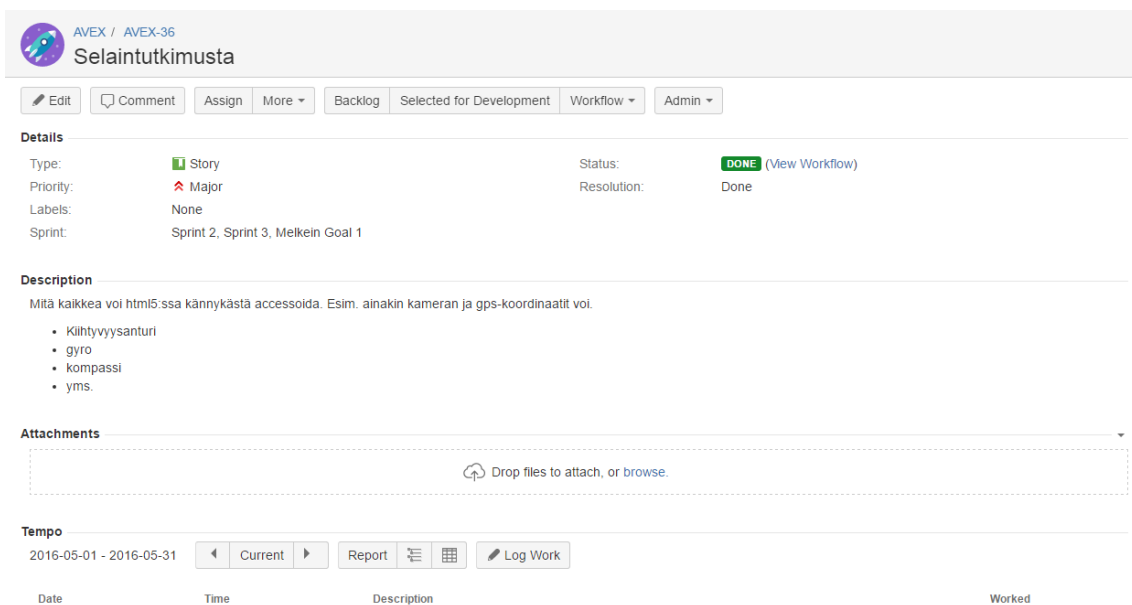
## 4 PROJECT CONTROL

As in any software a project version control was important to AVEX Motti. The project control means tracking your software project in revisions. It keeps your project organized so that all people included in the project know where the project is going next and where it has been at a certain point of time.

If any research has to be done it is good to document findings to some sort of wiki system so that everyone can see your findings and it can also be found later on.

### 4.1 Jira

Jira is a proprietary issue tracking product. (1) It is used to track the progress of a software project by bug tracking, issue tracking and project management functions. After all the desired features of AVEX Motti had been sorted they were listed to Jira and a short description and an estimated time to finish that ticket was written. For each sprint tickets would be nominated for developers according to the time estimate of that ticket.



The screenshot shows a Jira issue page for the project 'AVEX / AVEX-36' and issue 'Selaintutkimusta'. The issue is a 'Story' with 'Major' priority, 'None' labels, and is assigned to 'Sprint 2, Sprint 3, Melkein Goal 1'. The status is 'DONE' with a 'View Workflow' link. The description reads: 'Mitä kaikkea voi html5:ssä kännykstä accessoida. Esim. ainakin kameran ja gps-koordinaatit voi.' followed by a bulleted list: 'Kiihtyvyyssanturi', 'gyro', 'kompassi', and 'yms.'. Below the description is an 'Attachments' section with a dashed box and the text 'Drop files to attach, or browse.'. At the bottom, there is a 'Tempo' section for the period '2016-05-01 - 2016-05-31' with buttons for 'Current', 'Report', and 'Log Work'. A table header is visible with columns: 'Date', 'Time', 'Description', and 'Worked'.

*FIGURE 4. Ticket in Jira.*

## **4.2 Git**

Git is a version control system that is used to store a software project in a branched fashion. (2) By having one master branch and multiple smaller sub-branches a software project can be modified without the risk of breaking the product. It also makes it easier for multiple developers to work on the same project when everyone can easily check who has modified and what in a given part of code.

For each ticket a new branch was created. All work was uploaded to Git after every finished ticket. After that work was checked for quality and changes were suggested if necessary. Then the code would be approved of.

## **4.3 GitLab**

GitLab is a git repository hosting service that offered us a handy tool to visualise data uploaded to Git. (3) It was used to evaluate changes in the code.



## 5 DATABASES

The idea of how to create AVEX Motti was the functionality of PouchDB, which is a NoSQL database in the browser. The next step in this project was to determine if the application should be using an SQL or a NoSQL database with the backend.

### 5.1 Differences between SQL and NoSQL

When determining whether to use an SQL (Structured Query Language) or a NoSQL (No Structured Query Language) database for a project, one has to remember that although being a newer technology, NoSQL is not a replacement for SQL and you should always aim to pick the right tool for the job.

The major difference between these two is the style of storing data. In the SQL database all data is stored as a strict data template, so it is difficult to make mistakes as to where to store and what data.

NoSQL databases use JSON-documents (JavaScript Object Notation) to store data and all data is stored in to field-value pairs. (4) This can be used in a similar way in SQL so that every document can look the same but the actual perk is that they do not have to. Every document could have different field-value pairs and the database itself will not complain, yet this may lead to consistency issues.

SQL queries also offer a JOIN clause which NoSQL has no equivalent for. This means that if you want to search all fields with a specific value, you have to do that manually. (5) This also leads us to a CRUD (Create Read Update Delete) Syntax which is more complicated to do in the NoSQL database because if we have a complex JSON-document we have to be very specific to what we want to do with it. This may lead in the JavaScript side to large queries that have to be repeated multiple times resulting in code that breaks the DRY (Don't Repeat Yourself) principle but it is necessary.

NoSQL databases do not take part in data integrity so you might end up with orphaned records or invalid data if you modify or design the document structure poorly.

NoSQL is commonly thought to be faster than SQL, with a deformed store allowing you to retrieve all information about a specific item in a single request. But the most important thing is to design a database structure well to avoid problems later on.

For AVEX Motti a NoSQL database was decided on. When having tasks that could have different kinds of fields to fill, or that might have images attached to them, it was required a database would not be restrictive about the data structure.

## **5.2 PouchDB**

PouchDB is a JavaScript implementation of CouchDB which runs on browser.

(6) This is the place where a user's information will be stored before synchronizing with the backend.

## **5.3 CouchDB**

Apache CouchDB, which is more commonly known as just CouchDB is a NoSQL database which is designed for a web usage. It is a project created in April 2005 by Damien Katz. (7) This was decided to be used on the server side. Another possibility would have been to use MongoDB but it was discarded due to CouchDB's reliable functioning with PouchDB.

## **5.4 JSON document structure**

Here it can be seen how each object is stored into our database. Each object has an "\_id"-field which is automatically generated by PouchDB. This is the ultimate value to differentiate objects from each other.

A “\_rev”-field is also generated by PouchDB and it has two parts a revision number which tells us how many times the object has been updated, in this case 2 (FIGURE 5), and a long revision id number to track the amount of times an object has been modified and also potential conflicts.

Coordinates hold the values for the location of the task in a latitude and longitude format.

A date means the deadline to complete a task.

In the description, the person adding a task can give instructions and explanations regarding the task.

A documents array holds reports made by users. Each object in the documents array holds information about the user who added it (name and id number), time when the report was submitted and the report text itself.

An enrolled operators array holds id numbers of every user who has enrolled to this task.

A title field has a title of the task.

A type field specifies that this object is a task, another option being a user.

```
pouchdb:api motti +3ms put                                pouchdb-4.0.0.min.js:10
▼Object {title: "Thesis task", description: "Write thesis", date:
Wed Mar 30 2016 12:00:00 GMT+0300 (EEST), operators: 1, type:
"task"...}
  ► _conflicts: Array[0]
    _id: "27267"
    _rev: "2-e3db6a56d8d7985511ed558bf70cd4b2"
  ▼ coordinates: Object
    latitude: "65.00"
    longitude: "25.00"
  ► __proto__: Object
  ► date: Wed Mar 30 2016 12:00:00 GMT+0300 (EEST)
  description: "Write thesis"
  ▼ documents: Array[1]
    ▼ 0: Object
      id: "76281"
      name: "onja"
      text: "I wrote some thesis today!"
      time: 1455715152310
      ► __proto__: Object
      length: 1
    ► __proto__: Array[0]
  ▼ enrolledOperators: Array[1]
    0: "76281"
    length: 1
    ► __proto__: Array[0]
  operators: 1
  title: "Thesis task"
  type: "task"
  ► __proto__: Object
```

FIGURE 5. The JSON-structure for a task object.



## 6 PROGRAMMING

Before this thesis work began working on AVEX Motti, there was already some base for the application. The MVC structure had been created and some simple UI (User Interface) elements had been implemented. There was a way to add task objects to database but not to modify or delete them. These are the technologies used to create this application and there was no need to depart from that baseline.

### 6.1 AngularJS

AngularJS was used for all frontend functionality of the application. Controller and a model were fully executed with AngularJS and in a view, data-binding was used. Another option would have been to use React. One of the features that makes AngularJS so popular and beautiful are data models that are POJO (Plain Old JavaScript Object) classes which do not require getter and setter functions to work so properties can be changed or added and arrays or objects can be looped easily. In comparison to traditional data models, Angular's data models work as a temporary storage area instead of tools for an absolute data perseverance. To differentiate these two, Angular data objects are called "scopes". A scope object does not have any data on its own but relies on the controller to feed it values. All properties in the scope are automatically bound to a view in Angular and they are being constantly checked for changes.

Here is an example of a functionality done using AngularJS. This function checks the amount of operators required to finish a task and it also shows a user how an urgent need there is for more operators by giving a glyphicon with a responding color.

```

$scope.getoperatorsglyphicon = function(task) {

    //quotient = jakoiäännös
    var quotient = task.enrolledOperators.length/ task.operators;

    if(!task.operators){
        return 'userblack';
    }
    if(quotient <= 0.33){
        return 'userred';
    }else if(quotient <= 0.66){
        return 'useryellow';
    }else{
        return 'usergreen';
    }
};

```

*FIGURE 6 Getoperatorsglyphicon*

First, a call is made to a `getoperatorsglyphicon` function, which is given a task as a parameter. Then it is defined a quotient variable is defined where the length of an `enrolledOperators` array is divided by an `operators` value. The `enrolledOperators` array holds an id number of every user who has enrolled to a task and `operators` is the value given to a task showing how many operators are required to complete a task. For example if a task is given to 4 operators and the length of an `enrolledOperators` array would be 2, a variable `quotient` would have a value of 0.5. After this the function has an if-statement. As there is a possibility that the task has no value for the amount of operators the function can return that the glyphicon should be showed black and the function will return 'userblack'. If however the task has the amount of operators next the if-statement will check if the value `quotient` returns a value of 0.33 or less, then 0.66 or less and then it will return responding color as a CSS (Cascading Style Sheet) class name.



FIGURE 7 CSS classes

This would then be displayed in a view where the glyphicon is given a desired color. As yellow glyphicon can be seen in the top right corner.

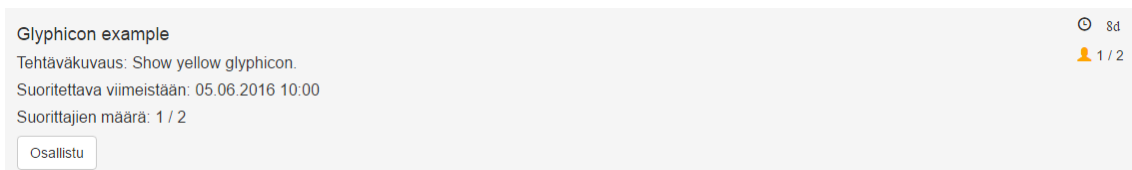


FIGURE 8 Display correct glyphicon.

Closely related to this feature is also displaying infinity icon “∞” if the task has an unlimited amount of required operators. Figure 9 reads: If there are no operators defined for a task, display this icon defined as an HTML symbol, else display the amount of operators.

```
{{task.enrolledOperators.length}} / {{{(!task.operators)?"∞":task.operators}}
```

FIGURE 9 Infinity icon if-statement

## 6.2 HTML5

HTML5 was used to create the basic structure of the view and also to create browser features like a geo location.

```
if (navigator.geolocation) {  
  
    navigator.geolocation.getCurrentPosition(function(pos) {  
        latitude = pos.coords.latitude;  
        longitude = pos.coords.longitude;  
        $scope.$apply();  
    });  
  
} else {  
    console.log('geolocation is not supported by this browser.');
```

*Figure 10 Geo location*

### 6.3 UX

Because the nature of this application is very practicality based it was tried to be shown that in designing the user experience too. It is a single page application to reduce unnecessary loading times and it was made with a Bootstrap library to ensure a good mobile and tablet scalability.

For some browsers the date object caused some issues but to ensure the usability Modernizr was used to detect and create a different input method. More detailed info about this can be found on the browsers section.

All colours are traditional in the UX (User Experience) sense (red indicating danger, yellow noticeability, green positive feedback and so forth).

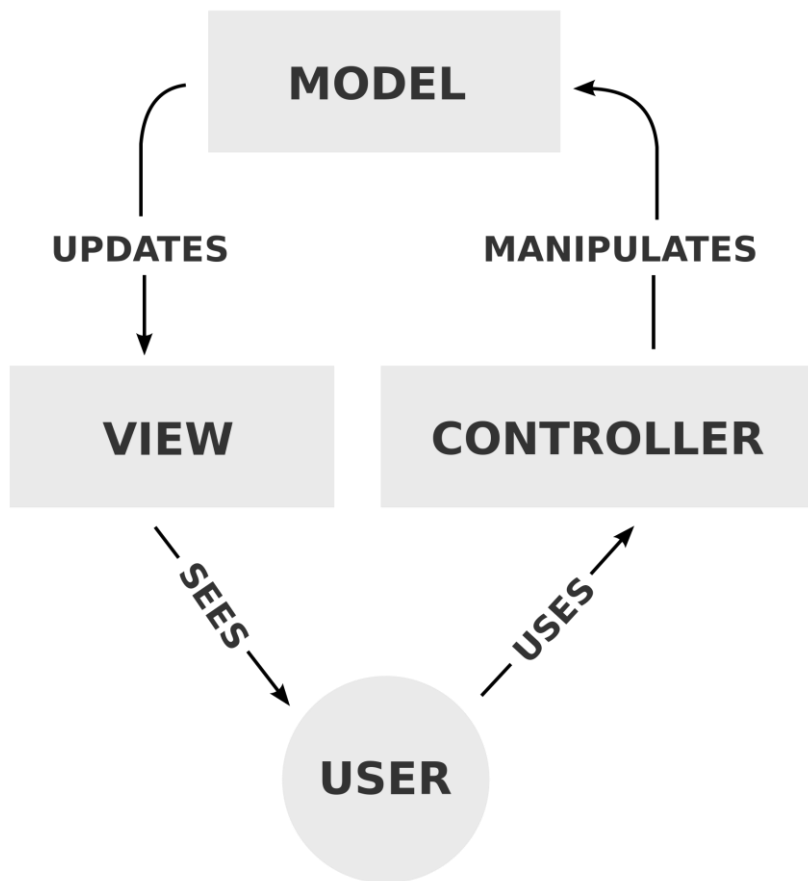
There was also an idea to implement a counter for when the databases last synched but there was not enough time to make it happen. This would show the user that their local database is indeed up to date.

#### 6.3.1 Technical Implementation

The application was created using the MVC architecture. (8) The view was made using HTML5, AngularJS data-binding and Bootstrap elements. The controller would have all the functions and would process the input from the user



and the model (in this application it was named a service) did all the communicating with the backend.



*FIGURE 11. MVC*

### 6.3.2 UI Design

For the UI design, the basic building blocks provided by the Bootstrap library were used. Some ideas for a logo were designed but later discarded.

Figure 12 displays a screenshot of the application as user view. By clicking an “Osallistu” button user can enrol to a task. Below at “Omat tehtävät” it can be seen all tasks for that user.

The screenshot shows a user interface with a dark navigation bar at the top. The navigation bar contains the following items: 'Motti', 'Matti Meikalainen' (with a profile icon), 'Admin', 'Etusivu', 'Joukkoistu', 'Tehtävät', 'Omat Tehtävät', and 'Tiimi'. Below the navigation bar, the main content area is titled 'Tehtävälista'. Under this title, there is a subtitle: 'Valitse itsellesi sopivat tehtävät ja ilmoittaudu niiden suorittajaksi.' Below the subtitle, there are four task cards. Each card contains a task description, an 'Osallistu' button, and a status indicator (a clock icon for time and a person icon for participants). The tasks are: 1. 'Find Mankirk's wife and then return to him at the crossroads.' (78d, 0 / ∞ participants). 2. 'Deliver food to little cabin in the woods. (Car required)' (2d, 0 / 1 participants). 3. 'Check water situation in examplestreet 40.' (474d, 4 / 6 participants). 4. 'Take a photo of every kitten you see in Oulu area.' (505d, 1 / ∞ participants). Below the 'Tehtävälista' section, there is another section titled 'Omat tehtävät' with the subtitle 'Olet ilmoittautunut näiden tehtävien suorittajaksi.' This section contains two task cards, which are the same as the third and fourth tasks in the 'Tehtävälista' section.

Task Description	Time	Participants
Find Mankirk's wife and then return to him at the crossroads.	78d	0 / ∞
Deliver food to little cabin in the woods. (Car required)	2d	0 / 1
Check water situation in examplestreet 40.	474d	4 / 6
Take a photo of every kitten you see in Oulu area.	505d	1 / ∞

Task Description	Time	Participants
Check water situation in examplestreet 40.	474d	4 / 6
Take a photo of every kitten you see in Oulu area.	505d	1 / ∞

FIGURE 12 User view.

## 7 BROWSERS

In this chapter the most important technologies related to browsers will be introduced.

### 7.1 HTML local storage

With a local storage web applications can store data directly to users' browser. (9) Before HTML5, application data had to be stored into cookies but a local storage provides more space for the application information (5MB) and it does not affect the website performance. For AVEX Motti the local storage was used to store PouchDB so that all the synched information would be available for users even at offline mode.

### 7.2 Application Cache

Application Cache is a guide for a browser of which resources it should cache for offline access. (10) By creating a cache manifest and listing all the files to be cached users would be able to access our application also while browsing offline.

### 7.3 Modernizr

Modernizr was used to detect HTML5 features, mainly if a browser could handle a date object or not and make our application respond accordingly. Modernizr is a Javascript library made solely for this purpose. (11)

The problem was that not all browsers support same features and this would break the application when using the date as an input type when "admin" was adding a task with a deadline. If a user was using a browser that could not handle a date object the application would show a little input grid that would take all these fields individually (day, month, and year) and send them to the controller to be converted.

## 7.4 Browser fingerprint

Browser fingerprint is a common technique for identifying users by searching a hash tied to computers characteristics. (12) For AVEX Motti this unique fingerprint was used to identify users without using an authentication system. The reason for not to creating a traditional login system was to keep as much as possible of the application functional offline. The idea was to create a system that did not need to know anything about the user. It only needed to know the tasks the users enrolled in. This of course means that all the users will be viewed as equal by the application and it does not have actual “admin” users. This could be solved by creating admin accounts manually and creating a login system just for them who would work online. For the demonstration, a button to toggle between admin and user views was created. Deleting the browser fingerprint is quite difficult to do so we were able to ensure that no user would delete their “profile” by accident.

## **8 LEGAL ISSUES**

There was also an idea that AVEX Motti could also be used to transport encrypted documents between places, stored in the local database like every other object.

But this brings an interesting question about information security and whether or not it is legal to make a person transport a document without them knowing the contents, or even that they are doing so. Finnish Communications Regulatory Authority was contacted to ask them about the subject.

### **8.1 Finnish Communications Regulatory Authority**

The cyber security centre of Finnish Communications Regulatory Authority sent an answer that it is not actually under their jurisdiction because this is an unusual way for our society to transport information.

The usage of information saved on a terminal is regulated by a personal data law and a working life protection law. Criminal law's violation of secrecy of correspondence could be applied to this situation, for according to transport and communications committee, the messages saved on a terminal still have the security of communications confidentiality. So in this case not informing the user about data on their terminal would be illegal.

However, if the person transporting the document were to be employed by the sender/writer, it could be seen that the sending happened legally. Because employer has the right to dictate about the usage of the machines they own, including holding information. This might apply to this situation.

## **9 DEMONSTRATION**

The aim of this project for us was to demonstrate the possibilities of this system for Finnish Defence Forces. So on 11 January 2016 the project group travelled by train to Helsinki to visit the Finnish Defence Forces headquarters to show what had been created.

### **9.1 Scope**

The most important things regarding the demonstration were that the basic CRUD functions would work properly and when handling NoSQL databases, those functions were required to work offline as well as online which required some research. Reporting findings to a task was on top priority because it was crucial for us to show how the system would actually work in a crisis situation.

During the last weeks of the schedule, the work was focused on few minor details like showing how large percentage of required users had enrolled to a task and displaying right icon according to that, and how many days, hours and minutes there were left to finish a task. They were small details but they still took time to create so that they would look nice and more importantly display the correct information.

One of the tasks was to show a distance between coordinates given to a task and a user. For this research calculating distances on a sphere shaped object was done. Also it was considered how to handle latitude/longitude format when a Finnish way of handling coordinates is usually a handled in different format. In addition it was studied how to enable geo location in a browser and what is that geo location result based on.

## **9.2 Changes**

There were some things that the project group wanted to include in the demo but they had to be cut out because of the time limits such as showing the last time that databases have been synchronized or being too difficult to implement like encrypting a document and loading it to the JSON database. Creating a map that would work in offline mode was such a colossal task that it was not even hoped to achieve it with the given time limit. Identifying users was done by solely relying on the browser fingerprint so that there was really no way of knowing who had admin rights for this software but for the demonstration, a toggle button was created so that it would be easy to switch between the admin and user views. It would have been beneficial to create some actual authentication and the project group had some ideas on how to make it happen but given the time limit it was not possible.

One of the more basic things that had to be left out was sorting the task list according to different filters, like a deadline or an alphabetical order. This was also due to a lack of time.

Overall the most crucial change that had to be made was not to upload images or documents to PouchDB. It would have been useful to add some interesting visual elements to the demonstration and also to showcase different types of data that could be added to a task. A research had also been done about this topic already and it would have been nice to continue by actually applying the feature.

## **9.3 Preparation**

The part of project group that was not working on the technical side of things had prepared a Power Point presentation to explain the idea behind the project to the FDF officers.

The project group travelled to Helsinki to visit the headquarters of Finnish Defence Forces and on 12 January 2016 developers had the last day to prepare for the demonstration. In the dress rehearsal there was an interesting phenomenon when one of the group members had some old database entries on his mobile phone's local storage: when synchronizing with the new database, it completely ran down the application's system because the JSON objects' format was so different from the last refresher course.

This was not positive in a sense that it took developers a while to figure out and fix the problem, but a great success in a sense that this single handily proved that the data stored in browsers local storage will stay there for as long as needed.

There were also some issues with removing data from the database when the whole system was transported to a server but Tuomas Riihimäki managed to solve this issue.

#### **9.4 Demonstration**

The actual demonstration went well. Pekka Pussinen introduced our project using his Power Point presentation as a guide (14) and me and Tuomas Riihimäki showed the application to generals from our tablets. Mostly positive comments were received. One of the officers had to be explained the functionality of PouchDB because there was a misunderstanding that it would cache the information to a smartphone's memory space but that was the only criticism the application got and even that was more of a concern.

The refresher course was deemed successful and the FDF officers were happy with the outcome.



## **10 CONCLUSION**

This project was successful in proving that this kind of application would be helpful in a crisis situation and possible to execute with the technology currently available.

The idea about a browser database that synchronizes to a server when in online mode could be used in a couple of different situations, such as a research team of biologists in a remote location taking notes about wildlife or mine workers studying geological phenomena. This application is more or less working in any location where connection cannot be guaranteed 100% of the time.

## REFERENCES

1. Wikipedia, 2016. Jira (software). Date of retrieval 24.5.2016.  
<https://en.wikipedia.org>
2. Wikipedia, 2016. Git (software). Date of retrieval 24.5.2016.  
<https://en.wikipedia.org>
3. Wikipedia, 2016. GitLab. Date of retrieval 24.5.2016.  
<https://en.wikipedia.org>
4. Wikipedia, 2016. NoSQL. Date of retrieval 24.5.2016.  
<https://en.wikipedia.org>
5. Sitepoint. 2016. SQL vs NOSQL: The Differences. Date of retrieval 24.5.2016.  
<http://www.sitepoint.com/sql-vs-nosql-differences/>
6. PouchDB. 2016. Date of retrieval 24.5.2016.  
<http://pouchdb.com/>
7. Wikipedia. 2016. CouchDB. Date of retrieval 24.5.2016.  
<https://en.wikipedia.org/wiki/CouchDB>
8. Wikipedia. 2016. Model-view-controller. Date of retrieval 24.5.2016.  
<https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

9. W3schools. 2016. HTML5 Local Storage. Date of retrieval 24.5.2016.  
[http://www.w3schools.com/html/html5\\_webstorage.asp](http://www.w3schools.com/html/html5_webstorage.asp)
10. HTML5 rocks. 2016. A Beginner's Guide to Using the Application Cache. Date of retrieval 24.5.2016.  
<http://www.html5rocks.com/en/tutorials/appcache/beginner/>
11. Modernizr. 2016. Docs. Date of retrieval 24.5.2016.  
<https://modernizr.com/>
12. Network world. 2016. Browser fingerprints, and why they are so hard to erase. Date of retrieval 24.5.2016.  
<http://www.networkworld.com/article/2884026/security0/browser-fingerprints-and-why-they-are-so-hard-to-erase.html>
13. Pussinen Pekka, 2016, AVEX Motti loppuraportti [PowerPoint slides].

## **APPENDICES**

Appendix 1: User reporting to task.

Check water situation in examplestreet 40. 🕒 474d

Tehtäväkuvaus: Report back how many days worth of water they have in use. 👤 4 / 6

Suoritettava viimeistään: 15.09.2017 15:00

Suorittajien määrä: 4 / 6

**Raportoi:**

**Tehtävän raportit**

Matti Meikäläinen

There is water to last for 3 days in apartment e 13. 29.05.2016 13:11

---