WWW.JAMK.FI

# Mobile version of a facility maintenance software

Jonah Ahvonen

Bachelor's Thesis
January 2015

Degree Programme in Software Engineering
School of Technology, Communication and Transport

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

**Description**

| Author | Type of publication | Date |
|---|---|---|
| Ahvonen, Jonah | Bachelor's Thesis | 5.1.2015 |
| | Pages<br>68 | Language<br>English |
| | | Permission for web publication<br>( X ) |

Title
**Mobile version of facility maintenance software**

Degree Programme
Software Engineering

Tutor
Lappalainen-Kajan, Tarja

Assigned by
Elomatic Paper and Mechanical Oy.

Abstract

The purpose of this thesis was to develop a mobile version of a facility maintenance software used by the facility maintenance personnel as a tool to help them with their everyday tasks as well as to help them to keep track of what has and has not been done on a weekly, monthly and yearly basis.

The original desktop version was completed by the software development team at Elomatic. Ever since the desktop version came out, it was suggested by various personnel at Elomatic that a mobile version of this software would benefit the facility maintenance workers even further by allowing them to view, add and edit information about the facility machinery on the spot with their phone or tablet.

What makes this system even more interesting is how exactly the mobile version is developed; web-techniques, such as HTML 5 and JavaScript are used instead of classic mobile development techniques like Java or Objective-C.

As a result of this thesis, a stylish and functional system was developed, although some of the planned features had to be left for further development due to lack of time.

Keywords
Facility, maintenance, software, mobile, web, JavaScript, jQuery, 360, panorama

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

**Kuvailulehti**

| | | |
|---|---|---|
| Tekijä<br>Ahvonen, Jonah | Julkaisun laji<br>Opinnäytetyö | Päivämäärä<br>5.1.2015 |
| | Sivumäärä<br>68 | Julkaisun kieli<br>Englanti |
| | | Verkkojulkaisulupa myönnetty<br>( X ) |
| Työn nimi<br>**Laitosten kunnossapitojärjestelmän mobiiliversio** | | |
| Koulutusohjelma<br>Ohjelmistotekniikka | | |
| Työn ohjaaja<br>Tarja Lappalainen-Kajan | | |
| Toimeksiantaja<br>Elomatic Paper and Mechanical Oy. | | |


Tiivistelmä

Opinnäytetyön tarkoituksena oli tuottaa laitosten kunnossapitojärjestelmän mobiiliversio, jota laitosten huoltohenkilökunta käyttää työkaluna päivittäisissä toiminnoissaan sekä auttaa pitämään heidät ajan tasalla viikottaisissa, kuukausittaisissa ja vuosittaisissa tehtävissä.

Alkuperäisen työpöytäversion tuotti Elomaticin ohjelmistokehitystiimi. Työpöytäversion valmistumisen jälkeen monet henkilöt ehdottivat, että mobiiliversio ohjelmistosta auttaisi laitosten huoltohenkilökuntaa työpöytäversiotakin enemmän, sillä he voisivat selata, lisätä ja muokata tietoja laitosten koneista paikan päältä käyttäen puhelinta tai tablettia.

Erityisen mielenkiintoisen järjestelmästä tekee se miten se tehtiin; klassisten mobiilikehitystekniikoiden, kuten Javan ja Objective-C:n sijasta järjestelmä tuotettiin käyttäen web-kehitystekniikoita, kuten esimerkiksi HTML 5:tta ja JavaScriptiä.

Järjestelmästä saatiin tehtyä tyylikäs ja toimiva kokonaisuus, mutta ajan puutteen vuoksi siitä täytyi jättää osa kaavailluista ominaisuuksista jatkokehitykseen.



Avainsanat
Laitos, huolto, kunnossapito, ohjelmisto, mobiili, web, JavaScript, jQuery, 360

## Content

# FIGURES

## TERMINOLOGY

### HTML 5

HyperText Markup Language 5. The fifth version of the HTML-standard used to make web pages. It defines a standard for how the markup should be written so that a web browser can understand it. (HTML 5 W3C. 2014.)

### XML

Extensible Markup Language is a standard defining how documents should be written so that it is both human- and machine-readable. It is extended by various other markup languages such as XHTML.

### JAVASCRIPT

A programming / scripting language used to interact with the user and to manipulate web pages. Scripts made with JavaScript are commonly run by the web browser, however, lately the usage of JavaScript has expanded a great deal and nowadays it can also be used for other development practices like server-side programming with the use of third-party frameworks or platforms.

### JQUERY

A commonly used JavaScript-library that expands JavaScript with helpful functions and features. One of the more important features is the “jQuery-selector” allowing the developer to easily and intelligently select DOM-elements for interaction or manipulation.

### AJAX

Asynchronous JavaScript and XML. A technique used to send and receive data asynchronously to and from a server. As JavaScript runs only in a single thread, meaning that it can only do one thing at one time – Ajax is an important technique to somewhat counter this. Usage of Ajax can be seen visually when page content is updated without reloading the page. For example, Google’s search suggestions are retrieved with Ajax. (Ajax. 2014.)

### DOM

Document Object Model. A convention for representing and interacting with (X)HTML or XML –objects. In HTML an object is usually something which is opened and closed with HTML-tags. “ Text ” is a paragraph-object with a text node -object in it. (DOM. 2014.)

**SQL**

Structured Query Language. A programming / query language used to interact with databases.

**DATABASE**

A system specifically created for storing data. Data can be added, modified and read with database-queries, using a special programming language such as SQL.

**MYSQL**

A commonly used database management system. Comes with a data storage and a database-motor that interprets SQL-queries. Simply put; a MySQL-database is a place where a computer program saves huge amounts of data for later use.

**PHP**

PHP Hypertext Preprocessor. A server-side scripting language. Often used to generate web pages partially or completely and to interact with databases. For example one can use PHP to read 10 000 stored customers from a database and then present them in a nice table-format on a web page.

**CSS**

Cascading Style Sheet. A style sheet language used to tell a web browser how a web page should look like. There are huge amounts of properties that one can specify with CSS, few of these are: element size, element color, element position and element margin.

CSS is smart in a way that it has a priority-system to handle styling conflicts. This means that it is basically valid to tell an element that its background-color is blue and right after that tell that it is green – CSS ignores the other automatically with its built-in priority-system. (CSS rule priorities. 2014.)

**DEBUGGING**

A process in where a programmer runs a program and inspects its output and behavior, to find out why the program is not working properly or why it is not doing something correctly.

**INSPECTOR**

DOM-inspector. A developer tool used to see what is going on behind a web page. Useful when debugging JavaScript or styling a web page.

**ANDROID**

Operating system used in some smartphones and tablets. Applications for this operating system are programmed in Java.

**IOS**

Operating system used in Apple's smartphones and tablets. Applications for this operating system are programmed in Objective-C.

**JAVA**

Another computer programming language. At first it was used to develop desktop applications and web applets, but nowadays it is also used to program mobile applications for android-devices.

**OBJECTIVE-C**

Yet another computer programming language. It was also used previously to develop desktop applications; however, nowadays it is also used to program mobile applications for Apple's IOS-devices.

**PROTOCOL**

Protocol is a set of rules used when exchanging data between servers. There are different protocols for different data exchanges; for example HTTP transfers hypertext and FTP transfers files. (Protocol. 2014.)

**FTP**

File Transfer Protocol. A protocol used for transferring files between two machines – a client and a server for example.

**HTTP**

HyperText Transfer Protocol. A protocol used to transfer hypertext. In layman's terms HTTP is a courier who brings web pages to users when the browser requests them.

**PANORAMA**

Wide angle view of a place. Normal panorama-image is like a really wide image. 360 spherical panorama is a view in where the viewer is inside and at the center of a sphere looking at the sphere's inner surface. Sphere-like effect is usually achieved by applying a lens-distortion on a box, effectively hiding the corners of a box, making the box look like a sphere.

**UI**

User-interface. A view which the user sees when using a software. The view can be graphical (, physical) or text-based.

**BLACK BOX**

Something goes in and something else comes out, however, nobody knows that happens in between. A programming pattern to avoid especially when it comes to functions so that if something is not working correctly, the person fixing the problem does not have to spend hours trying to figure out what happens in some shady code segment.

**SVG / VECTOR GRAPHICS**

Images made with vector graphics technology are (almost) infinitely scalable without any quality loss. This makes them optimal for web applications, which need to scale in size a great deal to fit different screen sizes.

SVG is the format in which the vector graphics images are saved as. Technically SVGs do not contain any image data, but instead they hold mathematical instructions which, for example, web browsers follow to build the images.

An example of a “mathematical SVG-instruction” would be something like this:

<circle cx="50" cy="50" r="40" stroke="green" fill="yellow" />

(SVG. 2014.)

**API**

Application Programming Interface. Specifies how two software components interact with each other. For example mobile devices might have a camera API available for mobile developers to use for easy access to the camera’s functions from inside a program code.

Something like Camera.TakePicture(); could be a camera API function telling the device camera to take a picture. (API. 2014.)

**CITRIX**

A set of virtualization, networking and mobile workspace products used for remote access of applications and data. (Citrix. 2014.)

**PANO2VR**

Pano2vr is an application that converts a set of images and into a spherical panorama image which can be viewed with a HTML 5 players or a flash player. The images are wrapped into a Pano2vr-skin object with a settings object and after that the skin object is given to the pano2vr player object.

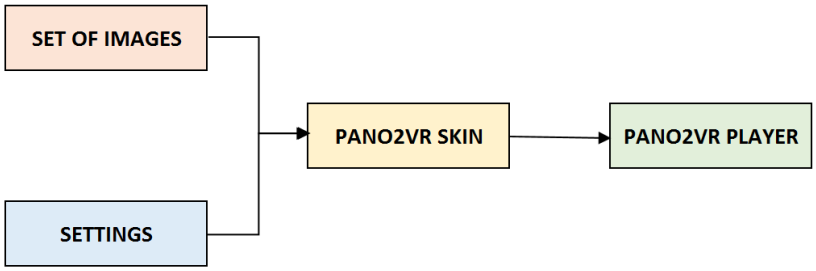


Figure 1. Pano2vr

**BOTTLENECK**

Bottleneck is something in software or hardware that is restricting, for example, an application in performing better than it could. Sometimes it is very important to properly detect and fix bottlenecks, however, there is always something bottlenecking as it is quite impossible for everything to perform exactly the same and even if that was possible, then everything would be the bottleneck at the same time.

# ACKNOWLEDGEMENTS

I would like to thank my coworkers at Elomatic who helped me with planning, developing and testing of this mobile application.

# 1 INTRODUCTION

Elomatic is an engineering design company founded by Ari Elo in the year 1970. Elomatic employs about 750 personnel and has offices in Finland, Poland, India, China, Italy, Serbia, Russia and the UAE. (About Elomatic. 2015.)

Three years ago the developers at Elomatic created their first implementation of a facility maintenance software called ePlant-360°. The potential of this maintenance system was quickly recognized and around a year later in summer 2012 they started planning and developing a second implementation of this software. The second implementation was to be created from scratch and it was to include a better user-interface and many more features.

In the end of 2012 the second implementation was complete and it was received well by the clients. After the second implementation was done – it did not take long until someone had an idea of a mobile version of this software.

A mobile version would allow the facility maintenance staff to do the same operations as they could do with the desktop version, however, they could do that on the spot with their smartphone or a tablet. A mobile system like this would speed up the facility maintenance routines by a great margin, therefore the system would be more than welcome, if introduced to the facility maintenance personnel.

However the developers who had designed the desktop implementation were too busy developing and maintaining other systems and applications, so they really only got the mobile version through the major planning phase in the end of 2013.

It was after a while until Elomatic decided to hire a software engineering student from JAMK to test his abilities in developing the mobile facility management software that had been waited for a long time.

# 2 FACILITIES

Facilities need constant maintenance and upkeep as they are constantly running and producing various things like energy, paper, heat or clean water. It is the responsibility of the facility maintenance personnel to keep all the machines, pumps and such in a good condition.

In order to keep everything in a good condition the facilities assign every machine their scheduled maintenance routines. These routines need to be done ranging from every day to once per two years, therefore, there is a plenty of work involved to keep track of everything.

Even if the schedules work out well, someone or something has to keep track of all the spare parts each machine might or might not need at some point, and on top of that most of these parts need separate instructions on how to change them safely and properly.

That is still not everything though; there are several machines located throughout the facilities and knowing the location of each machine by heart is probably not the best idea, thus for the facilities to be able to keep track of all this occurring each and every day, to multiple machines in different locations, with various spare parts and instructions, there has to be some kind of system in place.

For a long time the system was either pen and paper, Excel spreadsheet or something equally frustrating for the maintenance personnel to work with.

# 3 MOBILE SYSTEMS

## 3.1 Mobile software and development

Mobile software development became popular about seven years ago with the launch of the first iPhone and since then it has been on the rise and constantly discussed in the news and social media. (iPhone. 2014.)

At first the available methods for mobile development were quite scarce – Java and Objective-C mostly, however, nowadays there is a wider variety of options such as Unity, Phonegap and wrapping an application into a web page, which is basically what will be done regarding this system.

## 3.2 Java

Java is used when building mobile applications for Android devices. Many developers find it easier to learn and use than Objective-C, which is used to make applications for iOS-devices. It is arguable whether ever Objective-C performs better than Java, however, hard to compare when Java cannot be used for iOS apps and Objective-C cannot be used for Android apps.

One feature about Java is its native graphics library, which is actually quite powerful and relatively easy to use compared to third-party graphics libraries such as SFML.

## 3.3 Adobe Phonegap

Adobe Phonegap is a mobile development framework started by Nitobi, but purchased by Adobe 3 years ago. Developers can build semi-native mobile applications with the help of the framework by using web-technologies such as HTML5, JavaScript and CSS3.

Phonegap is not used in this project for one because this application is not going to be represented in any smartphone application stores, however, it still is an interesting concept which the writer of this thesis is going to test and find out more about. (Phonegap. 2014.)

## 3.4 Apache Cordova

Apache Cordova offers APIs that allow developers to access native mobile device functions from JavaScript. Apps made with help of Cordova are packaged as native mobile applications, meaning that they can be put in application stores.

Cordova is like Phonegap in a sense that both aim to free developers from using device native languages (Java, Objective-C). To be honest, Phonegap is built on top of Cordova, however, Phonegap wants to be a whole framework for mobile development instead of just a set of APIs like Cordova; therefore Phonegap offers some extra systems like application cloud compilation and packaging so that developers do not have to spend their own processor time to do it.

## 3.5 Conventional approach

Conventional approach is used in this project, which basically means that no complex third-party frameworks will be used – just the traditional web technologies with some helpful libraries. The application will be a mobile-viewable web page accessed with the mobile device's native web browser.

There is nothing too important or useful needed from the device's native functions so that a third-party framework granting access to those would have to be included in the hassle, therefore it is best to leave the external frameworks out. After all, using any framework requires time to learn it first and that is not ideal for what is tried to be achieved here right now.

Either way, these frameworks will be examined and tested separately from this thesis due to interest in mobile development frameworks and web development in general.

# 4 WEB TECHNOLOGIES

## 4.1 HTML5

HTML is the only way to make web pages. For sure there are tools like Adobe Dreamweaver, however, in the end all those will produce HTML in some form.

HTML5 is the fifth revision of HTML-standard and it brings access to many of new APIs such as the Media API or the Drag and Drop API. There are other APIs as well often associated with HTML5, like for example the Local Storage API and Location API, however, they are not commonly recognized as pure HTML5 APIs since they are accessed via JavaScript and therefore not directly via HTML. (HTML5 APIs. 2014.)

APIs are not the only thing HTML5 offers though, as there are many new tags to put in a use as well. The most useful ones are:

- Canvas-tag
    - Allows for drawing graphics with JavaScript
- Video-tag
    - Defines the new HTML5 video-player – no plugins needed!
- Main- and Article-tag
    - Dedicated tags for main content and articles. Formerly these were developer-defined with div-tags, just like everything else.
- Header- and footer-tags
    - Dedicated tags just for header and footer content. These too were just div-tags before, defined by the developer.
- Many more
    - Not too useful from developer's perspective.

Some HTML 4.01 tags exist no more in HTML5. Most of them nobody has never heard of, however, the ones that somebody has probably heard of, are center-tag and font-tag.

## 4.2 JavaScript

JavaScript can be used to manipulate web pages or to make them interactive. Every web page uses JavaScript these days – be that something as simple as outputting a value into the developer-console or using a huge web page analysis-system like Google Analytics, nearly everything is done with JavaScript after the page has loaded.

While JavaScript with third-party libraries like jQuery is powerful, legacy JavaScript without any third party libraries performs a great deal faster. Some developers ignore this performance factor, especially when it comes to smaller projects, because often it does not really make a difference if the application can perform 10 or 30 million operations per second, unless the goal is to make a heavily optimized application. The reason why some developers prefer to use jQuery selectors over vanilla selectors is that jQuery selector syntax is so much simpler to use than vanilla selector syntax and often jQuery's syntax is also more uniform.

Vanilla JavaScript's syntax non-uniformity can be seen in the Figure 2 such as:

- When selecting element by id
    - document's getElementById() –method is used
- When selecting element by class-attribute
    - document's querySelector() –method is used instead

Whereas with jQuery there is always a uniform selector syntax:

- When selecting element by id
    - Syntax is $('#elementid')
- When selecting element by class-attribute
    - Syntax is $('.class')

Other often used JavaScript libraries or frameworks one might come by are, to list a few (it is a common naming convention to name JavaScript-libraries with ".js"-suffix): AngularJS, script.aculo.us, node.js, prototype JavaScript framework, MooTools, kinetic.js, three.js and backbone.js.

| Testing in Chrome 35.0.1916.153 32-bit on Windows Server 2008 R2 / 7 64-bit | | |
|---|---|---|
| **Test** | | **Ops/sec** |
| **jQuery ID Selector** | `var $el = $('#hello');` | 1,245,634 ±0.45% 96% slower |
| **JavaScript ID Selector** | `var $el = document.querySelector('#hello');` | 11,042,698 ±0.24% 66% slower |
| **jQuery Class Selector** | `var $el = $('.bye');` | 528,667 ±0.42% 98% slower |
| **JavaScript Class Selector** | `var $el = document.querySelector('.bye');` | 2,160,424 ±0.21% 93% slower |
| **GetElementById** | `var $el = document.getElementById('hello')` | 32,849,005 ±0.16% fastest |

Figure 2. JavaScript vs. jQuery performance (Test case. jsperf.com)

## 4.3 jQuery

As mentioned before, jQuery is the “must have” for many developers working with JavaScript. It is not nearly as good as legacy JavaScript performance-wise, but the very simple syntax and helper functions even that out pretty well.

There are other options like AngularJS and backbone.js to use instead of jQuery. When it comes to AngularJS, it requires a completely different approach from the developer. Wherein jQuery is used to manipulate already loaded content, AngularJS works before and while content is being loaded. Other main difference between jQuery and AngularJS is that AngularJS tries to extend HTML’s functionality rather than JavaScript’s by enabling the developer to add “directives” on HTML-tags and “angular-expressions” inside HTML-tags.

From the AngularJS and jQuery notations below can be seen that in this example to print out an array of phone information as a HTML-list, the amount of programming / software code is roughly the same. Some developers claim that developing with AngularJS is definitely faster once they get to know it.

```
<ul>
  <li ng-repeat="phone in phones">
    {{phone.name}}
    <p>{{phone.snippet}}</p>
  </li>
</ul>
```

AngularJS HTML-notation

```
var phonecatApp = angular.module('phonecatApp', []);

phonecatApp.controller('PhoneListCtrl', function ($scope) {
  $scope.phones = [
    {'name': 'Nexus S',
     'snippet': 'Fast just got faster with Nexus S.'},
    {'name': 'Motorola XOOM™ with Wi-Fi',
     'snippet': 'The Next, Next Generation tablet.'},
    {'name': 'MOTOROLA XOOM™',
     'snippet': 'The Next, Next Generation tablet.'}
  ];
});
```

AngularJS JavaScript-notation

```
<ul>
  <li id="phonelist">

  </li>
</ul>
```

jQuery HTML-notation

```
$(document).ready(function() {
    var phones = [
        { name: 'Nexus S',
            snippet: 'Fast just got faster with Nexus S.' },
        { name: 'Motorola XOOM™ with Wi-Fi',
            snippet: 'The Next, Next Generation tablet.' },
        { name: 'MOTOROLA XOOM™',
            snippet: 'The Next, Next Generation tablet.' }
    ];
    for(var i = 0; i < phones.length; i++) {
        $('#phonelist').append('<li>' + phones[i].name + '<p>' +
        phones[i].snippet + '</p></li>');
    }
});
```

jQuery JavaScript-notation

## 4.4 PHP

PHP runs on a webserver and preprocesses hypertext before it is even sent to a client requesting it with a web browser. PHP is commonly used to fetch data from a database in to a web page as well as handling sessions and user actions like logins and logouts.

When optimizing a web page (especially for mobile devices) it is generally a good idea to perform as many actions with PHP as possible, because all of it is done by the webserver and not by the client, effectively saving resources (like battery and memory) on the client device.

PHP is often the receiving part of AJAX calls as well, since with PHP it is really uncomplicated, fast and easy to return data to a requesting script.

## 4.5 MySQL

MySQL is an open-source database management system that runs on a webserver like PHP. Data in MySQL database is stored in tables. These tables can be linked to each other, usually with some key-column which is common to both tables. This link is often called “a relation” or “a relation-link”. (Relational model. 2014.)

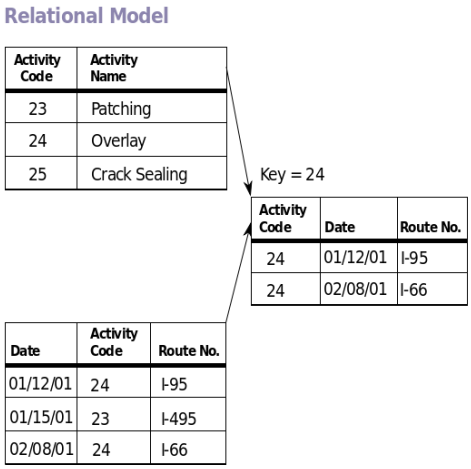


Figure 3. Relation-link between database tables. (Relational model diagram. Wikipedia. Relational model.)

These links make building and querying the database easier and more error-proof. The relations can, for example, block and limit data being inserted and prevent tables that other tables depend on from being removed making the database generally more rigid.

Most of these relation-based checks happen automatically too, given that the database and the relations are built and structured correctly in the first place.

Arguably one could claim that these relations make the database more secure to attacks such as SQL-injections that remove data or drop tables, as the attacker would have to know which tables to clear and drop first because of the dependency-rule.

## 4.6 CSS

Cascading style sheets are a lightweight way to make web pages look better, or actually to look like anything at all. CSS consists of a simple syntax defining simple styling rules to one or a set of elements at the time. These rules are interpreted on the client-side by the browser.

CSS, although simple, can still be built/structured incorrectly, resulting in the browser having to go through the whole web page multiple times to make everything sized and positioned correctly. This is called browser reflow.

A browser reflow is one of the main issues why some web pages take a long time to load. If the page (the HTML markup) is already huge itself – it does not exactly help if the whole page needs to be “flowed” through multiple times just to get some box in to the right position only because the CSS was not done right.

There are some simple ways to minimize the reflow count:

- By reducing DOM-depth
- Removing unused and/or unnecessary CSS-rules (LESS/SASS does this automatically)
- Remove to-be-animated elements from the flow with fixed- or absolute-positioning

(Browser reflow. 2014.)

### 4.6.1 SASS

SASS is an abbreviation from Syntactically Awesome StyleSheets and is a scripting language that results in normal CSS. SASS has two syntaxes one can choose from; other being the older, indentation-based syntax and the other being the newer normal CSS-like syntax with parenthesis and code blocks.

SASS offers extra functionality for making stylesheets, while not actually extending or modifying normal CSS itself. SASS is a standalone scripting language that just happens to be syntactically similar to CSS with some extra features, so including a .sass file on a web page will not do anything. (SASS. 2014.)

This is a snippet of SASS (SCSS):

```
$blue: #3bbfce;
$margin: 16px;

.content-navigation {
  border-color: $blue;
  color:
    darken($blue, 20%);
}

.border {
  padding: $margin / 2;
  margin: $margin / 2;
  border-color: $blue;
}
```

SASS snippet. (Wikipedia. SASS.)

And it compiles to normal CSS like this:

```
.content-navigation {
  border-color: #3bbfce;
  color: #2b9eab;
}

.border {
  padding: 8px;
  margin: 8px;
  border-color: #3bbfce;
}
```

SASS to CSS snippet. (Wikipedia. SASS.)

### 4.6.2 LESS

LESS is a compilable, dynamic stylesheet language inspired by SASS. Both SASS and LESS are in fact very similar and the main difference between these two is that LESS can be compiled real time with LESS.js –JavaScript library. (LESS. 2014.)

# 5 SOFTWARE SPECIFICATION

The software being developed is a mobile version of existing facility maintenance software developed mainly for facility maintenance personnel to use with their daily routines.

The desktop version is currently in use at various (heating) facilities, which are being maintained by more than one maintenance personnel in each facility. Because the software is used by several different clients and in different types of facilities, the software needs to be flexible and customizable enough to suffice the purposes these facilities need it for.

The software will be a mobile software, thus it will be used with different handheld-devices such as mobile phones and tablets of all kinds, which effectively means that the UI needs to be scalable and fit for all mobile screen sizes. One way how this is taken into account is by using vector graphics combined with responsive web design.

The current desktop version has plenty of features and nearly all of them need to be implemented in mobile form, therefore, in order to make this possible the software needs to be systematically developed in order to avoid excess memory and processor usage, messy code and “black boxes”.

The application is built with web technologies, which means that the system is technically a web page. The tricky part is the whole optimization process for mobile devices, as these devices have significantly less processing power and memory available compared to desktop PCs. These optimizations are covered thoroughly later on.

The system will most likely handle confidential data related to the facilities and its personnel, therefore system security and data protection is also something that needs to be properly implemented.

# 6 WORK ENVIRONMENT

## 6.1 Demo

Everything is developed and built in a closed, private environment. The software itself can be accessed anywhere with a web browser, but it is locked behind credentials.

The demo uses its own database filled with arbitrary, although relevant data about the things related to the software such as task logs and item data. This database and the data in it is used for development and marketing purposes as to not give away any real information about clients for example.

The database and all the development files related to the software are backed up and they can be safely interacted and manipulated without the fear of software bugs erasing important information or loss of work (software code).

## 6.2 Tools

For web development in general just a simple text editor is usually enough. Notepad++ is used because of a personal preference. Notepad++ is an advanced text editor when compared to the basic Windows Notepad or Wordpad. It offers code highlighting, plugins and macros to name a few. These all can help a lot and thus Notepad++ got the job.

For image editing the software used is Adobe Photoshop. Developers do not really need to edit anything, but it is still needed for measurements and getting the exact colors from the image so that everything may look like how the UI designer designed it.

# 7 PLANNING AND DEVELOPMENT

## 7.1 User interface and front end

The user interface is a classic style mobile interface consisting of main menu –view, sub views accessed from main menu and slide menus accessed via various UI-components in sub views.

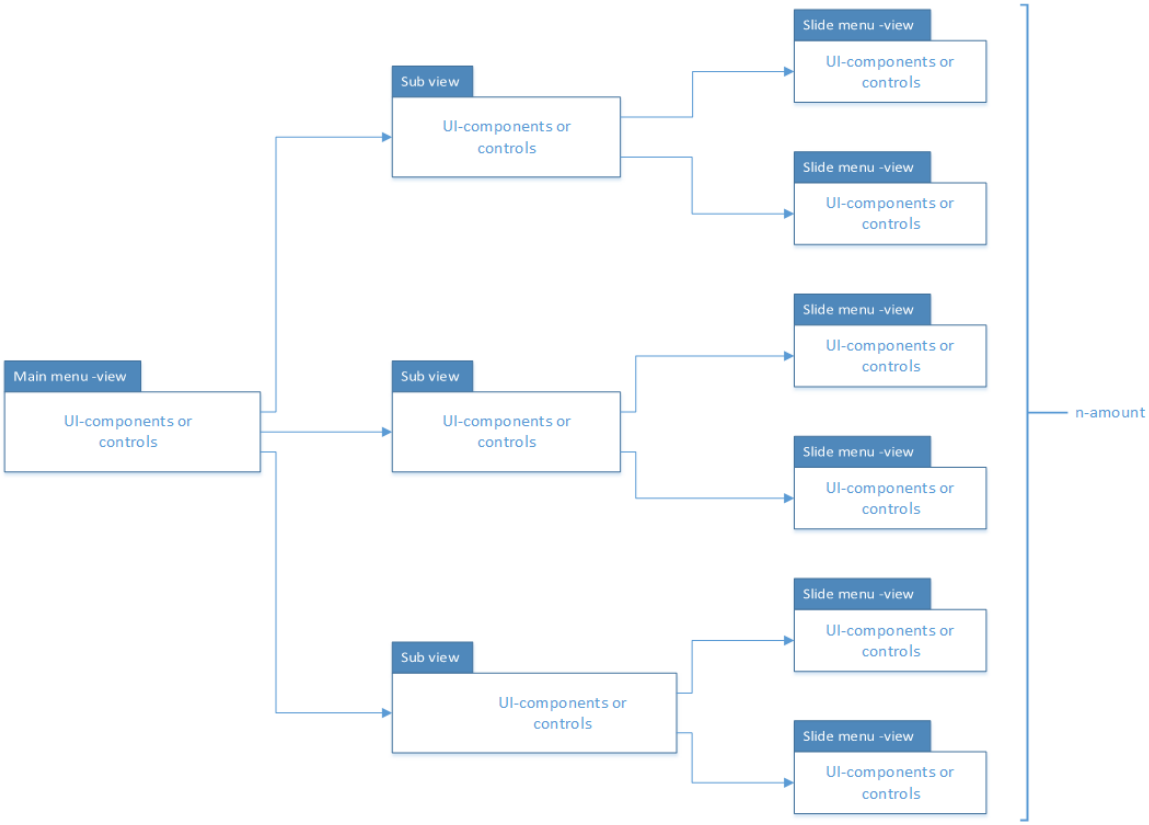


Figure 4. Diagram of views

Everything is primarily designed to be viewed in the portrait mode of a mobile-device, however, using the application in landscape is still a possibility although not advised.

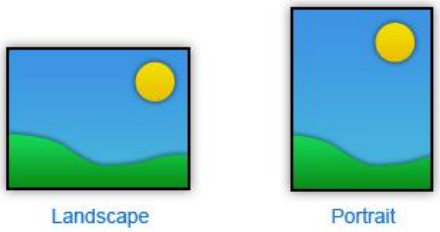


Figure 5. Landscape and portrait. (Portrait and landscape definition. usabilityfirst.com)

### 7.1.1 Front page

The software was originally thought to contain 10 main features / pages:

- Tasks
- Items (units / appliances)
- Panorama view
- Usage diary
- “Kettle diary” (= usage diary, some facilities prefer to use a different name for it)
- Calendar
- Spare parts
- Emap 360°
- Weather
- Feedback


Figure 6. UI - Front page version 1.2 (Design figure. Elomatic.)

After rethinking the features, their usefulness and the work involved in creating each feature, the software's features were reduced from 10 main features to 9 with the removal of Emap 360°. There was simply too much work involved for now. Adding Emap 360° later, when everything else is implemented, is still a possibility.


Figure 7. UI - Front page revisited

Although the layout looks relatively simple, the content in the menu squares was interestingly tough to implement as the icon and text need to be centered, but then again the logo has to be scaled when viewport size changes. After a couple of tries though, a solution was found.

Interestingly enough, the user could initially click the menu squares multiple times in succession, resulting into navigation events firing multiple times and finally crashing the application. This was later fixed with a simple check.

### 7.1.2 Tasks

Tasks page opens up a list of tasks which can be filtered by type or searched by task name. User can add a new task by pressing the plus-icon on top or open up a task-specific menu by clicking on an arrow-icon next to a task.

Tasks are color coded so that the user can see in which state each task is currently in. A task can be untouched, reserved to a person or marked as done for example.

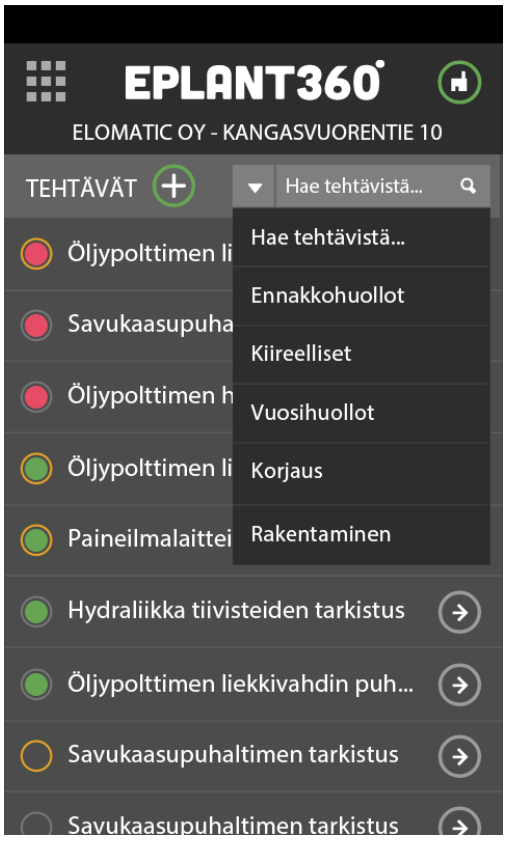


Figure 8. UI - Tasks (Design figure. Elomatic.)

This is the menu that the user will see after pressing the ”Add new task” –button:

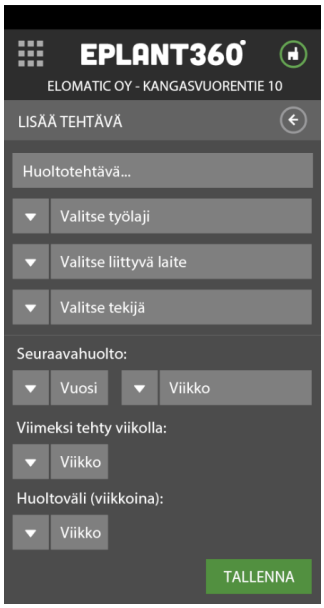


Figure 9. UI - Menu for adding a new task (Design figure. Elomatic.)

Task-specific menu had five options to choose from initially:

- Mark as done
- Show task in panorama view
- Reserve task
- Show task
- Edit task

Two more options, “Show task log” and “Remove reservation”, were added during the development.

This is what the task list and the task-specific menu looks like after being implemented in to the software (initially the menu was essentially the same, with the exception of the two menu items missing):

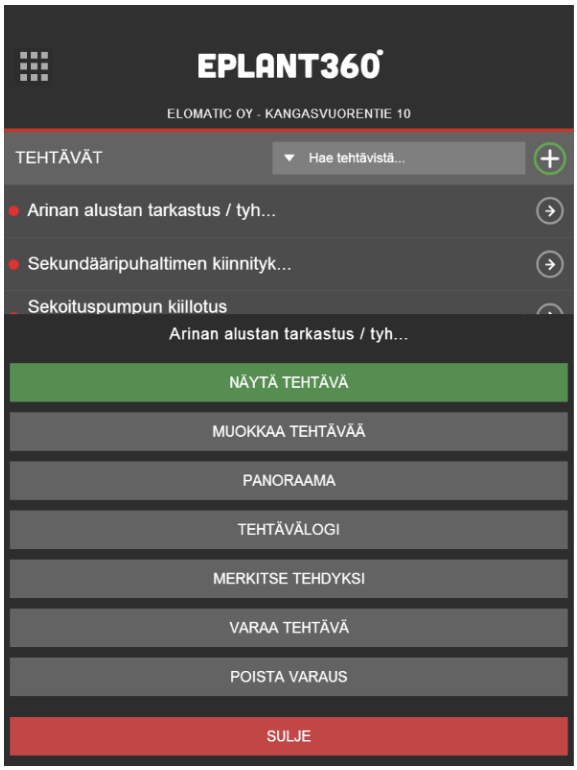


Figure 10. UI – Tasks list with menu open

**Show task** menu shows extended information about the selected task. In the demo this menu is somewhat empty, therefore there really is no need for a figure of it.

**Edit task** menu allows the user to, edit the selected task as well as to delete the selected task. It is just like the “Add new task” menu, however, the inputs are filled with the data the task already has instead of being left blank.

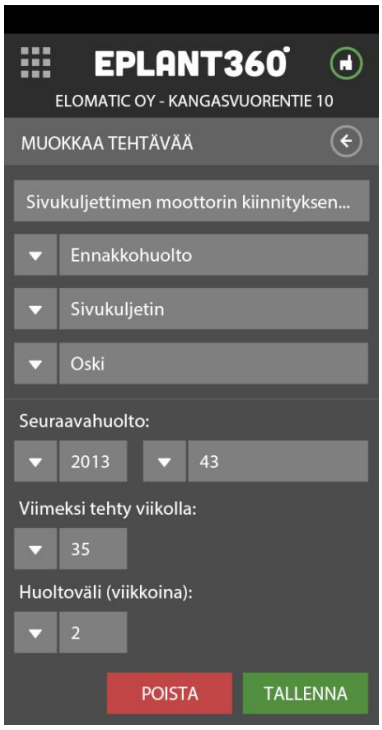


Figure 11. UI - Edit task menu (Design figure. Elomatic.)

**Panorama** option opens up a panorama-view that is focused on a unit in a facility that the selected task is about. This helps the user to better see which unit the task should be performed on.

**Task log** menu shows a log about the selected task, listing all previous times the task was marked as done. This is because there can be repeating tasks and the user needs to be able to see the history of such tasks (as well as every other task).
**Mark as done** option allows the user to mark the selected task as done. The user can select whom did the task from a dropdown-menu and add a descriptive comment too.

**Reserve task** option allows the user to reserve the currently selected task for completion. The user only needs to select a person to reserve the task for and confirm the reservation with a press of a button.

**Delete task reservation** option lets the user remove the current task reservation.

### 7.1.3 Items

Items page lists all the items (or units) facility has. In the item page's main view, shown in the Figure 12, the user can add and view items as well as search for a specific item by text or category.

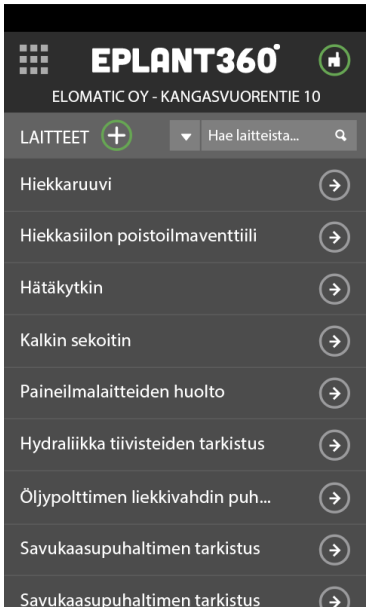


Figure 12. Item page (Design figure. Elomatic.)

When some item is selected by the user, a slide menu will appear from the bottom of the screen (shown in the figures Figure 16 and Figure 17). From this menu the user can perform multiple actions regarding the selected item:

**Item card** option shows extended information about the selected item as well as lists all the documents associated with the item. These documents can be something like a picture of the item, schematics related to the item or maintenance instructions.

**Edit item card** option lets the user to edit the information associated with the selected item. The menu that opens when user selects the edit item card option can be seen in the Figure 13.

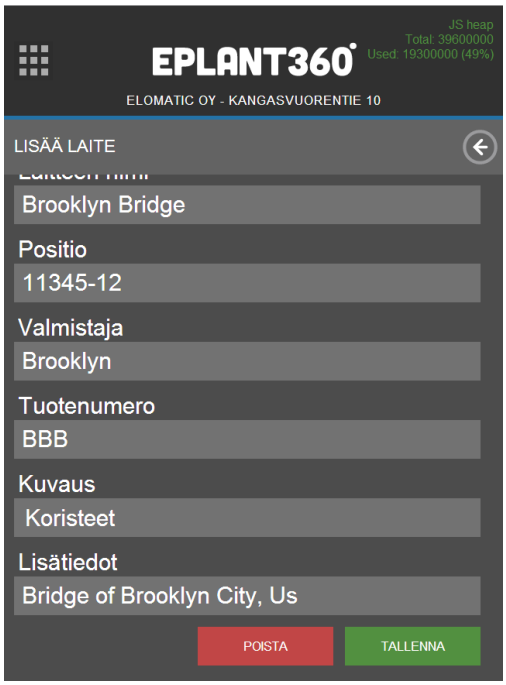


Figure 13. Edit item card

**Attach a document** option allows the user the upload a picture with the device the software is being used on. The picture can either be taken with the device's camera or selected from the device's picture library (Figure 14). Before uploading the document the user is able to preview the document. If the user is not happy with the currently selected document, the user can tap the document preview again in order to select some other document (Figure 15).

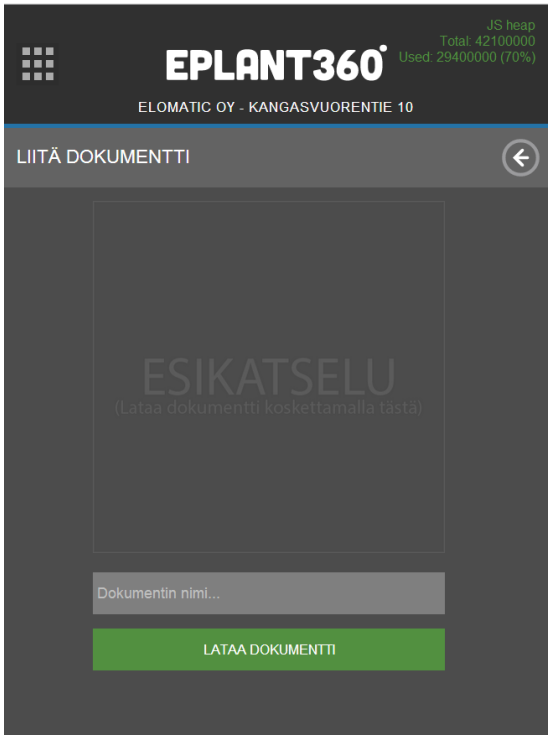


Figure 14. Document upload

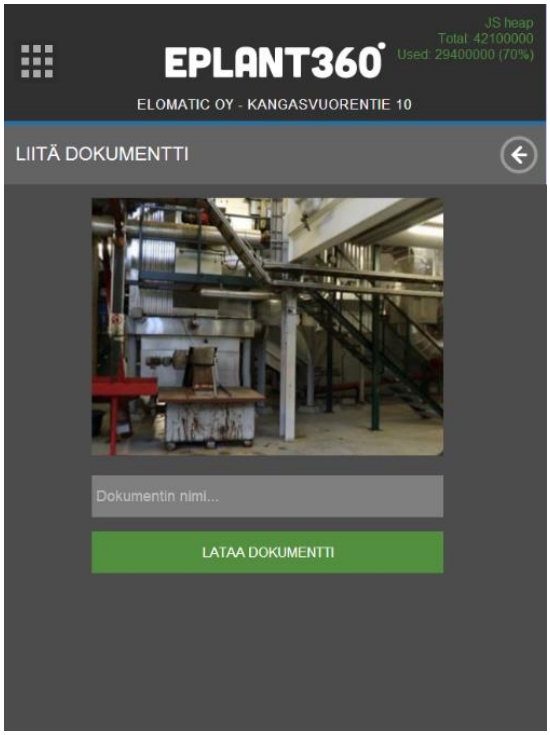


Figure 15. Document upload preview

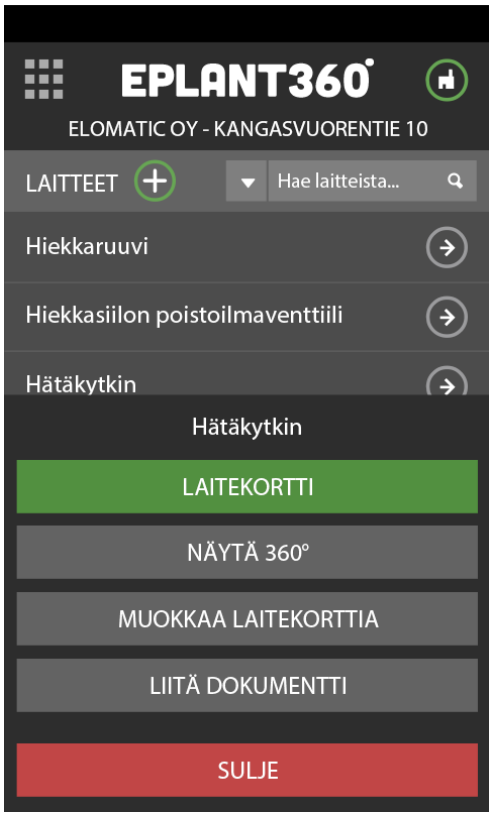


Figure 16. Item list slidemenu (Design figure. Elomatic.)

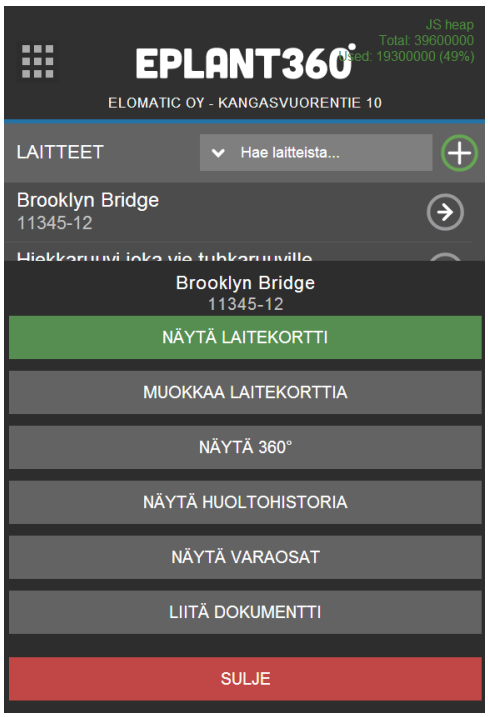


Figure 17. Items slidemenu revisited

**Show 360°** option shows the selected item in a panoramic-view. This makes it easier for the maintenance personnel to know which item they are interacting with.

**Show maintenance history** option shows a list of maintenance operations performed on the selected item. This feature makes it easier for the maintenance personnel to track what has been done to the item, by who and when.


Figure 18. Maintenance history (Design figure. Elomatic.)

**Show spare parts** option lists all available spare parts for the selected item. This menu/option can be seen in the Figure 19.

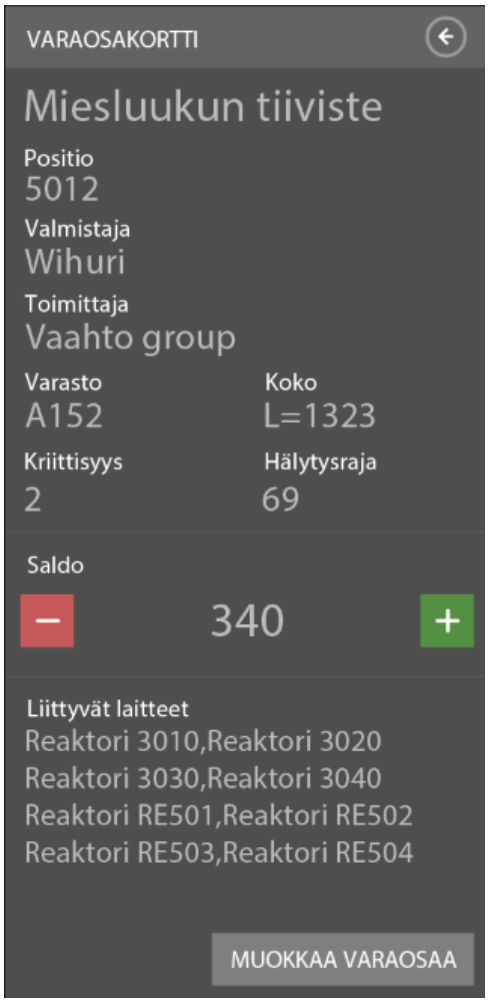


Figure 19. Spare parts menu (Design figure. Elomatic.)

### 7.1.4 Usage diary / boiler diary

Both usage diary and boiler diary are the same, however, some of the customers prefer to use one over another. In the demo both are accessible to make development easier, and in the release version other diary is removed depending on how the customer wants it.

From Figure 20 it can be seen that the diary is actually just a simple maintenance task checklist that shows what has been done, by who and when.

Figure 21 on the other hand, just shows how the user can add a new diary entry.

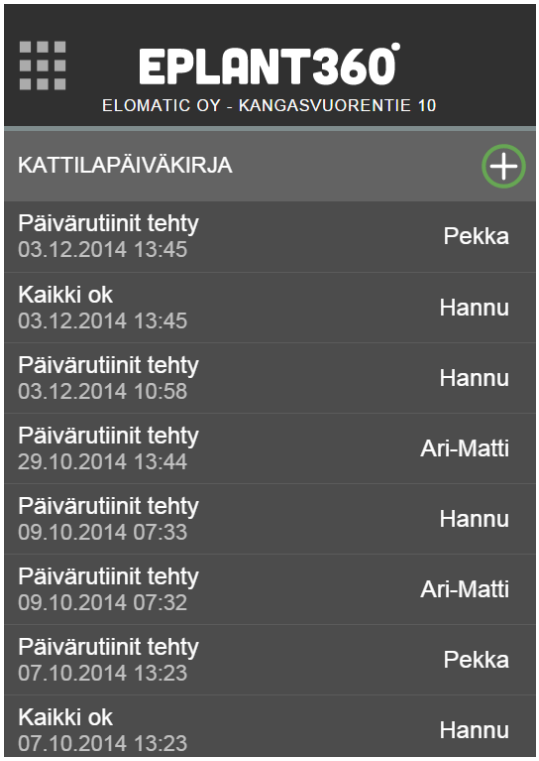


Figure 20. Usage diary / boiler diary

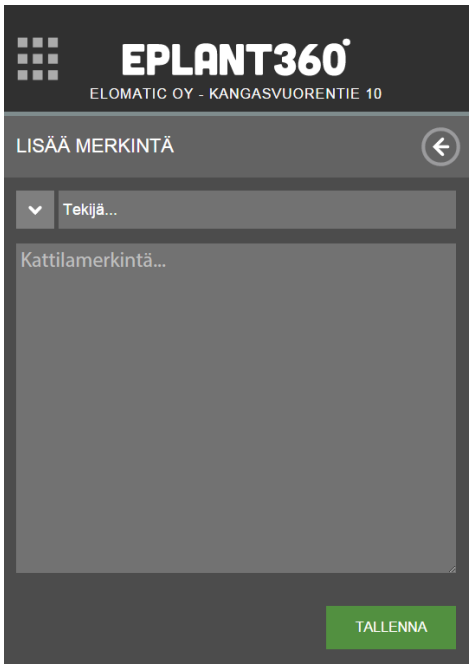


Figure 21. New diary entry

### 7.1.5 Calendar and spare parts

These two features were left unimplemented as there was a limited time period to develop the system and the other features were deemed more important.

### 7.1.6 Weather

Weather view is just a quick and simple way for the user to check the current weather. Weather data is acquired from ilmatieteenlaitos' open data API.

Figure 22 shows what the weather view looks like. The user can utilize the search bar on top to get the current weather broadcast of any place inside Finland.

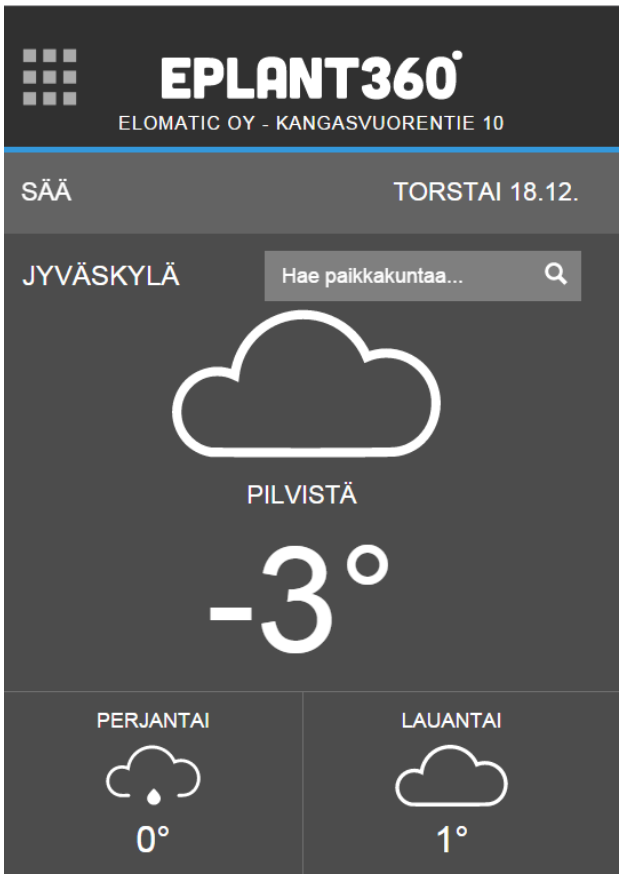


Figure 22. Weather view

### 7.1.7 Feedback

Feedback feature was developed so that the user can send bug reports and enhancement suggestions to the developers at Elomatic. The feature is used a lot in the desktop version, so naturally it would be a good idea to include it in the mobile version as well.

The user can select the priority of the feedback from one these:

- Urgent
- Notice
- Idea
- Question

Figure 23 shows how the feedback form looks like.

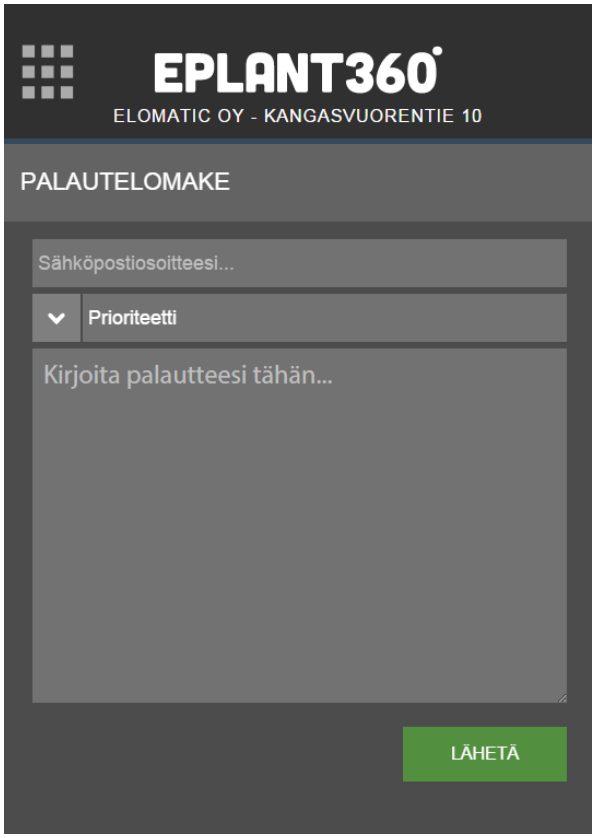


Figure 23. Feedback form

## 7.2 Panorama

Panorama view offers a spherical panoramic view of the facility. Originally the user was supposed to navigate the facility with the help of a mini map view as shown in the Figure 24. The navigation and parts of the panorama got redesigned – mini map was removed all together and the floor and room navigation controls were move from the mini map view to the panorama view. Mini map is still possible to add in, however, it was deemed rather useless compared to the time it would take to implement it for the initial release.

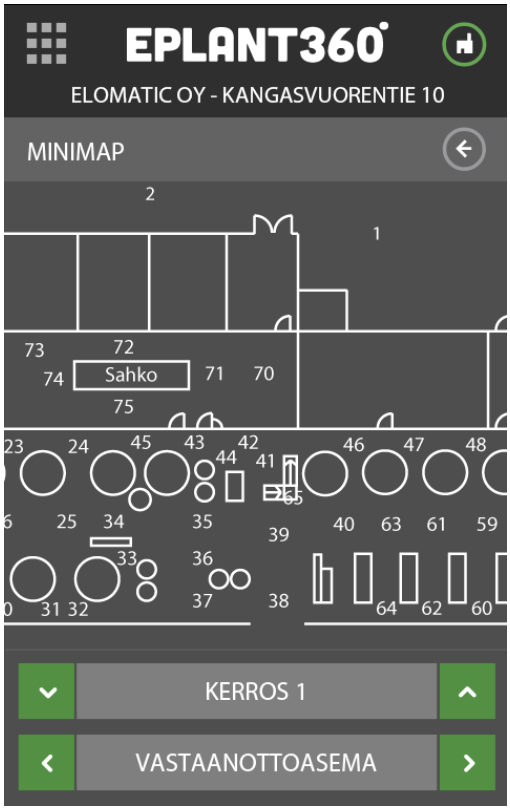


Figure 24. Minimap (Design figure. Elomatic.)

As for the panorama, after a redesign and moving the controls from the mini map to the panorama – it currently looks like as seen in Figure 26, with the exception of the item icons missing from the panoramic view. Looking at Figure 26 it is easy to notice how well

the controls stitch together with the panorama and how the mini map is actually not needed.

When designing the panorama and especially the pano2vr skin for the mobile application; some customer integration was deemed necessary, as in the desktop version every customer has their own skin, which is not ideal, and therefore the skin for this mobile application was made from scratch and this single skin works on all customers with a minimal amount of customer-specific configuration. Later on the generic skin would be used in the desktop version as well.

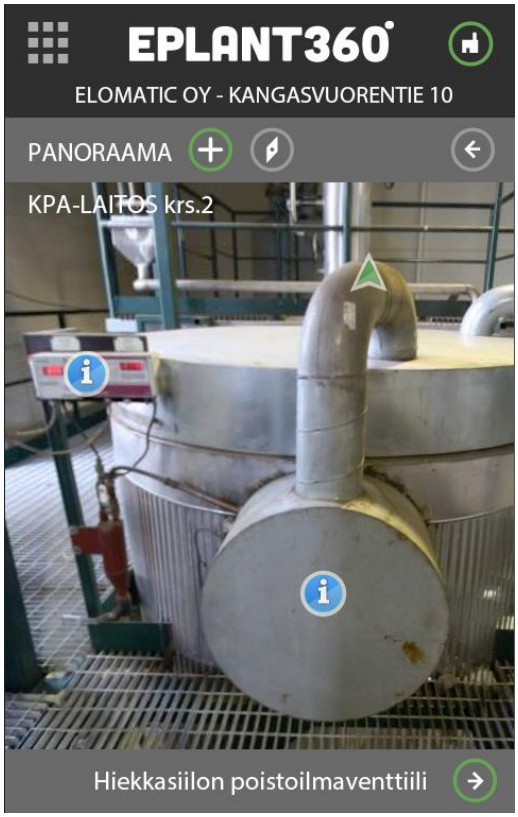


Figure 25. Panorama (Design figure. Elomatic.)

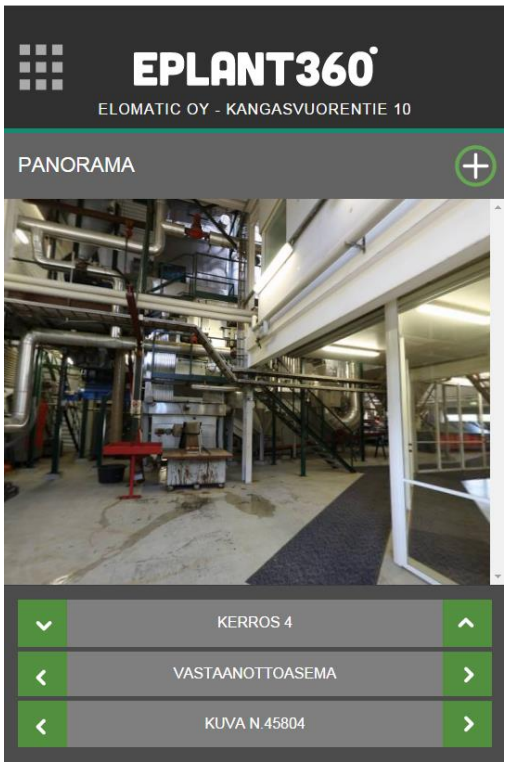


Figure 26. Redesigned panorama

## 7.3 In-depth look of some features

### 7.3.1 Transitions

Since the application has multiple views, menus and pages – there has to be some way to transition between them. Without going into further, technical details and actually focusing more on the visual side of these transitions, there are several different ways the application is able to transition between menus and pages.

For transitioning between pages the application uses jQuery Mobile's mobile page template that offers automatic transitions and transition animations when going from one page to another. All the developer has to do is write something like shown in Figure 27.

```
//Navigating with $(':mobile-pagecontainer').pagecontainer() is important as it uses the
//jQuery Mobile's own transitioning scheme
//<changeHash: false> should also be used to perform some browsing history trickery
function navigateTo(subsite)
{
    isNavigating = true;
    $(':mobile-pagecontainer').pagecontainer('change', subsite, { changeHash: false });
}
```

Figure 27. Page transition code example

jQuery Mobile's built-in transitioning system beautifully and smoothly fades the current page out while fading the requested page in, resulting in a fluid page transition.

Transitioning between menus by sliding the menus on and off the screen is completely custom made behavior aiming to give the application a native and easy to use look. Slide menus are first constructed with HTML and they are completely static at first. When the application loads up, an automatic system makes all the static slide menus into interactive and dynamic. Each dynamic menu is a JavaScript object with methods for opening and closing and attributes for keeping track of the menu states such as if the menu is open or not.

Figure 28 shows a group of collapsed slide menus associated with the tasks page of the application. If one were to expand any collapsed slide menu, it would be possible to see the rather generic structure of these slide menus. Every menu has a "toolbar" or a header section and a content section. What is in the content and toolbar sections differs, but the outcome behaves just the same.

```
<div id = "tehtavalisays_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal" data-discrete = "true">
<div id = "tehtavalogi_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavakortti_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavakortti_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavamuokkaus_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavamerkitse_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavavaraus_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavapoistavaraus_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal">
<div id = "tehtavalogiadd_slidemenu" class = "generic_slidemenu" data-slidemenu = "true" data-direction = "horizontal" data-discrete = "true">
```

Figure 28. Slide menu HTML structure

In Figure 29 the application is crawling through the DOM, seeking for data-attribute matches that it recognizes as slide menus. When it happens to find one – it makes the slide menu into an object and stores it into a list of slide menu objects in order to keep track of all the slide menus. Naturally a DOM element cannot really be “made into an object”, however, storing a reference to a DOM element into a JavaScript object is close enough for all practical purposes.

```
console.log('Automatically found following slidemenus:');

$('div').each(function(i, e)
{
    if($(e).data('slidemenu') != undefined)
        if($(e).data('slidemenu'))
        {
            slidemenuList.push({ element: $(e), id: $(e).attr('id'), isOpen: false, direction: $(e).data("direction") });
            console.log($(e).attr('id'));
        }
});
```

Figure 29. Building dynamic slide menus

Figure 30 simply shows the output if the slide menu builder. From the output it can be seen that the script in fact does what it is supposed to do – to find and build slide menus from static DOM element structure.

Size and general style of the menus is calculated in a JIT (Just In Time) manner in order to achieve precise values as it is very hard, if not impossible to know the exact values for some on the slide menu (style) attributes as those depend on the user (like the screen orientation to name a one).

```
Automagically found following slidemenus:
laitekortti_slidemenu
laitekortti_edit_slidemenu
huoltohistoria_slidemenu
dokumentti_slidemenu
items_slidemenu
kattila_lisays_slidemenu
kaytto_lisays_slidemenu
tehtavalisays_slidemenu
tehtavalogi_slidemenu
tehtavakortti_slidemenu
tehtavamuokkaus_slidemenu
tehtavamerkitse_slidemenu
tehtavavaraus_slidemenu
tehtavapoistavaraus_slidemenu
liittyvalaite_slidemenu
tasks_slidemenu
tehtavalogiadd_slidemenu
```

Figure 30. Slide menu build output

### 7.3.2 Indicators

It is important for the user to know what is happening if it is not obvious. With web development and JavaScript / jQuery, some extra knowledge is needed as there are some pitfalls like the alert-function (used for popup-notifications) which is able to stop code execution while it is shown.

In this application a lot of things that are happening are pretty obvious and most of the time enhanced by animations and transitions. Loading different things, however, is not so obvious usually. To fix this matter there are several loading indicators put in place and shown to user whenever the user is sending data to a server in order to tell the user that the data upload is being executed. Fetching data from the server has so far been fast enough so that a loading indicator has not been required. If this were to change, there is already a system implemented that adds the loading indicators for data being fetching as well.

Transferring between pages and most of the other features the application has are pretty self-indicating so no additional indicators are needed.

### 7.3.3 Seamless design and integrity

It is not easy to build a web-based mobile application and have it behave like a native one. There are lots of browser, operating system and device differences to take into account as well as memory and processor usage, battery drain, mobile data usage and the list goes on and on.

To make this even a bit easier to develop, the application is built with jQuery Mobile user-interface system that helps with building responsive and fast HTML 5 mobile applications. jQuery Mobile merely helps with any of the issues mentioned above, but it does offer couple of useful features like one page template. This template is a combination of HTML, CSS and JavaScript and it is able to "stitch" multiple pages together with some clever internal workings that are partially exposed to the developer in the form of helpful functions and automation. Developer has to only define which pages the application consists of (shown in Figure 31).

```
<!-- jQuery Mobile page is defined with data-role = 'page' -attribute -->
<div data-role = 'page' id = 'items_container'>
    <div role = 'header'>
        <!-- Page header here -->
    </div>
    <div role = 'main'>
        <!-- Main content here -->
    </div>
    <div role = 'footer'>
        <!-- Page footer here -->
    </div>
</div>

<div data-role = 'page2' id = 'items_container'>
    <div role = 'header'>
        <!-- Page 2 header here -->
    </div>
    <div role = 'main'>
        <!-- Page 2 main content here -->
    </div>
    <div role = 'footer'>
        <!-- Page 2 footer here -->
    </div>
</div>
```

Figure 31. jQuery Mobile DOM structure

### 7.3.4 Event handling / optimization

To reduce the amount of JavaScript events bound at any given time, a semi-dynamic event binding was implemented. This system tries to automatically bind only the events that are currently needed, depending on what the user can currently see and therefore interact with – unnecessary events are unbound.

The system mostly comes in effect when the user navigates between views or opens / closes a menu, as navigating usually completely changes what the user can interact with and opening or closing a menu changes only some parts.

Attached, there is an event-flow diagram to further explain what the system does with the events.

### 7.3.5 Loading item- and part lists

When user wants to load some list of data, it is not optimal to load it from a database every time, especially in a system like this where the list contents do not chance that often.

To optimize list loading, the list is only loaded from a database if it is changed or if it is not loaded yet. The application keeps track of the states of the lists and decides whet ever to serve the list from the server or to use the local copy. Figure 32 is a small design diagram of the process.

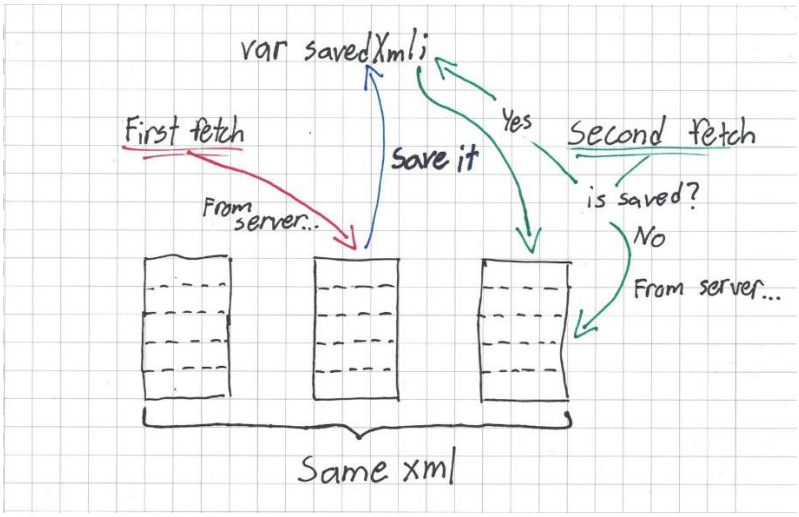


Figure 32. List loading process

## 7.4 Back end

### 7.4.1 Software

All instances of the desktop version (and therefore of this version as well) run on Windows Server with IIS (Internet Information Services). IIS is a web server like Apache, but instead created by Microsoft rather than the Apache Software Foundation.

The database is just a normal MySQL database, which is viable and enough for this type of system.

There is also a PHP mail service running on the background so that the bug reports filled in by customers can be directed to the developer's email for a fast response.

Hardware that the server is using is unknown.

### 7.4.2 Optimizations

Numerous optimization opportunities are taken into account when making the application. Here are the most important / useful ones in a list format:

- Elements are removed from the DOM-flow when modified with JavaScript
    - This reduces the DOM reflow and repaint counts significantly
- Scripts and stylesheets are minimized for release versions
    - As JavaScript is interpreted language, removing excess whitespaces and shortening variable and function names makes it run faster
    - File size is also reduced in the process
- Content is loaded asynchronously
    - UI and interactions will not freeze during content loading
- *(Will be implemented) Heaviest jQuery functions have been switched to their vanilla JavaScript equivalents*
    - jQuery performs some tasks a lot slower than vanilla JavaScript would – these tasks will be implemented with vanilla JavaScript instead of jQuery
- *(Will be implemented) jQuery's slow animation system has been replaced with faster one*
    - jQuery's animation system is really badly optimized and slow. it is replaced by velocity.js instead
- Images are in SVG-format
    - SVGs are technically not even images, but instructions for how to draw one, so it takes a significantly less time to load SVGs than normal images
- UI animations done with CSS3 instead of JavaScript
    - CSS3's animations are hardware accelerated by default, so they function a lot better than the ones made with JavaScript
- Using CDNs
    - Some of the libraries used are loaded from CDN
- Caching
    - Largest files have caching implemented so that they do not have to be downloaded again every time

# 8 TESTS

## 8.1 General

Testing is an important part of software development. Most of it is usually done between revisions and after the software's features are somewhat finalized. Since this project is done basically with a single developer, automated tests come in handy.

Testing software thoroughly is advised because it greatly enhances the user experience and improves the overall stability and integrity of the software. Nobody likes software

errors and they never do anything but harm (except in video games some of them can be quite entertaining, graphical glitches for instance), therefore it is best to test the software well enough so that in best case scenario it contains no errors what so ever.

In short, software testing verifies that the software works as planned and has no major flaws. As software testing can be carried out infinitely, the software developers and the testers usually agree on a certain level of testing success given the time and resources available. (Software Testing. 2014.)

Software testing does not just cover software errors and whet ever or not the software has met the feature requirement. There are various other matters software can be tested for, matters such as:

- Black box testing
    - Software is tested by someone who does not know the internal workings of the program, therefore it is tested purely for features and usability
- White box testing
    - Opposite of black box testing. Tests are performed by someone who knows the inner workings of the software and the tests are more focused on how everything works
- Unit testing
    - If the software is large, it is usually split in smaller parts or modules and these modules are tested separately
- End-to-end testing
    - The software is tested in an environment very close to a real environment in which the program will be run once it is finished
- Stress testing
    - The software is tested to and beyond its limits to see how much it can handle. Stress testable things may include: memory usage, CPU usage, GPU usage and network usage
- Usability testing
    - The user interface  and the “feel” of the software is tested so that it is easy and relatively fun to use

(Types of software testing. 2014.)

All of the above tests are performed in this project to a certain extent, although the majority of the tests performed will be white box, usability and end-to-end –tests.

## 8.2 Automated test

There are numerous frameworks available for testing web software, Jamine and DalekJS to name a few. They apparently perform well and are indeed good frameworks to consider when testing web software, however, due to personal interests and goals, a small automated testing script was made from scratch.

This script crawls through the DOM looking for arbitrary links and form / input elements. Elements that the script should ignore and not interact with are marked with “data-utIgnore = ‘true’” –attribute. When the DOM crawler comes across an element like this, it is simply ignored and the crawler will continue traversing the DOM normally.

Since this is an experimental, self-made testing script – it was deemed easiest to traverse the links at random instead of by some predetermined or heuristic pattern, therefore every time a link is discovered the script will decide at random whether ever or not it should follow that link to somewhere else.

Form / input elements, when detected, will get inspected more closely to find out what kind of input element is in question. Text-based inputs are filled with random debug data, radio buttons and checkboxes are negated (off->on, on->off) and a random item is chosen from lists. Buttons, or anything else that can be interacted with in order to delete or remove data from database are ignored for security and software integrity reasons. That being said, the script is still able to add or modify data in database as it was deemed that this will not result in too much data loss even if something goes wrong.

Both Jasmine and DalekJS perform similar functions as the one written from scratch for this project. Naturally they are more stable, rigid and safer to use as both of them have been developed for a longer period of time. They also offer some rather interesting extra functionality, like screenshots, compared to the self-made testing script. (Jasmine. 2014; DalekJS. 2014.)

One more notable JavaScript unit testing framework is QUnit by jQuery-team. It can even test itself, so apparently there is some kind of singularity achieved right there. (QUnit can test itself. 2014.)

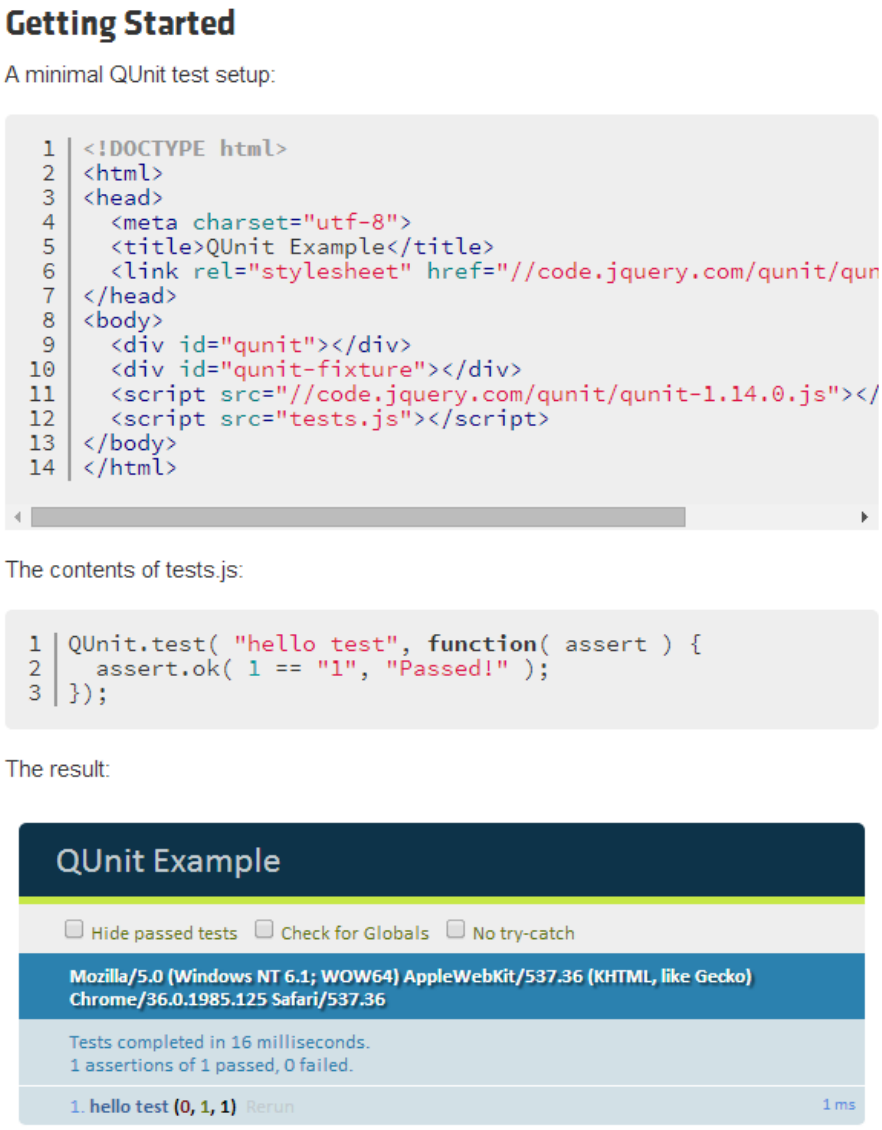


Figure 33. QUnit introduction. (Getting started. qunitjs.com.)

# 9 SECURITY

## 9.1 General

Software and data security are both important matters to keep in mind when developing any system. There are different aspects of software security, all of which are equally important.

## 9.2 Code security

First one is software code security. Software code can be written in very unsafe way if one does not keep up with the latest best practices. One quick example of unsafe code would be using JavaScript's eval-function, which evaluates and executes a string of JavaScript given to it as a parameter. Using eval the wrong way can make the application vulnerable for code injection attacks, slows the application down because the evaluation takes time and makes debugging more challenging for various reasons. (Why using eval is a bad idea. 2014.)

## 9.3 Input validation

Second important aspect about software security is validating all user input properly. Generally this means allowing only numeric, alphabetical or some predefined type of input and limiting length of the input. The input should be checked both server and client side as the user is usually able to disable the client side checks some way. In web development this often comes down to checking the input by using both JavaScript and PHP as JavaScript runs on the client side and PHP on the server side.

## 9.4 Server (software)

It has happened many times before that the software is as secure as it can be, but the server itself is vulnerable. Be that because the software on the server is outdated or exploitable or that the administration has left default or easily breakable passwords somewhere – every mistake like that makes the server vulnerable for attacks. These matters are merely the problem of client side developer, but should still be kept in mind for keeping the general software security up.

## 9.5 Database

When saving data into a database, the input should be validated as mentioned before. But what if the malicious data is getting into the database from somewhere else than the software using the database? Because of this, it is good idea to parse and/or validate the database output as well so that if something malicious has gotten in – it will not be getting out at least.

In order to secure data input and output with a database, using transactions is often a good idea. Database transactions guarantee for one that either everything will come in/out or nothing will. This method completely crosses out data coming out or getting in only partially, which can cause unexpected results and behavior.

Input / output guarantee is not the only reason why transactions should be used though, transactions also prevent two or more applications from accessing the database at the same time. This is good because this way the data cannot get overwritten, changed or otherwise modified for any reason. (Database transactions. 2014.)

# 10 PROBLEMS

## 10.1 Common problems and quirks

It was no surprise that there would be problems present when developing a web application for mobile use. First of all, some workarounds had to be made in order to maintain compatibility between Windows-, Android-, and iOS-devices:

**Overflow / (over) scroll workaround**

iOS devices specifically have this quirk present where the user can "over scroll" an overflowing element, resulting into unwanted positioning and scrolling regarding elements that are positioned with absolute or fixed attribute. A simple fix / workaround for this quirk was found at: https://github.com/pinadesign/overscroll.

**Windows phone viewport meta-tag**

Windows phone's native browser seems to completely ignore viewport meta-tag, which makes the application scale in a wrong way and also does not disable user being able to scale the layout. This quirk was rather easy to fix by defining the same meta attributes in CSS with Internet Explorer specific rules like seen in the Figure 34, however, figuring out that this had to be done took way more time than it should have.

```
@-ms-viewport
{
    width: device-width;
    orientation: portrait;
    zoom: 0.7;
    user-zoom: fixed;
}
```

Figure 34. Internet Explorer viewport CSS rules

**iOS popup prompt disabling all input**

When file upload was in development, the first idea was to prompt the user for the name of the document with the native popup prompt that looks something like in the Figure 35. This however resulted into iOS native browser completely freezing and ignoring all input (the prompt could not even be closed). The whole browser could have been technically considered as crashed, as the only way to continue using the browser was to “task kill” the frozen browser application and launching it again.

No workaround was found for this issue and therefore the file upload and name input was redesigned and now the document name is inputted into a simple textbox.

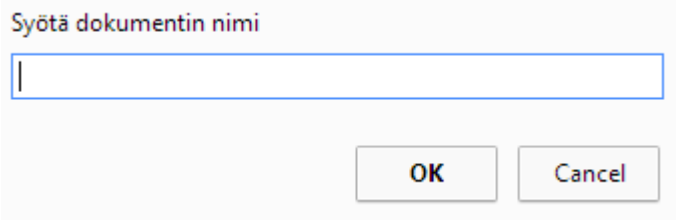


Figure 35. Native input prompt

## 10.2 Known bugs and problems

Overall the application is relatively stable in its current state: everything from animations to interactions has been tested thoroughly and should not cause problems during normal use, however, as volatile as mobile web applications are – some quirks still rarely occur.

There are also some known problems that were deemed minor or too resource consuming (not enough time) to fix, at least for the initial version:

**Using the application via Citrix breaks acquiring of weather data**

This is not really a problem at all as a situation like this will never occur during normal use. This bug was discovered when the application was accessed via Citrix during development and is noted, although prioritized as not important.

The problem itself most likely has something to do with an Ajax call accessing a PHP-script (the script returns the current weather) under the same domain but on a different server.

**Mobile native keyboard pushes layout**

Normally how the native keyboard should work when it slides up from the bottom is that it should push everything in the web application equally up or be on top of the whole application without affecting the layout at all.

There is a problem at least with iOS, where the keyboard affects fixed or absolutely positioned elements the wrong way. This results in the web application layout breaking until the keyboard is closed. There is currently no fix made for this issue, however, it does occur extremely rarely.

# 11 FURTHER DEVELOPMENT

## 11.1 UI improvements

There is little to improve when it comes to the actual graphical design of the application, however, how exactly the UI works could be improved in various ways:

**Slide menus**

Slide menus are generally well structured, simple to make and perform moderately well, however, their opening and closing animations could be improved.

There has also been some discussion about making a different kind of slide menu, a nested one. This nested slide menu differs from the current one by having slide menus inside slide menus. The nested slide menu would be dynamic in a sense that “menu A”

does not necessarily have to open its direct children “menu B” as shown in figures Figure 36 and Figure 37. All this, however, is in demo phase for now.

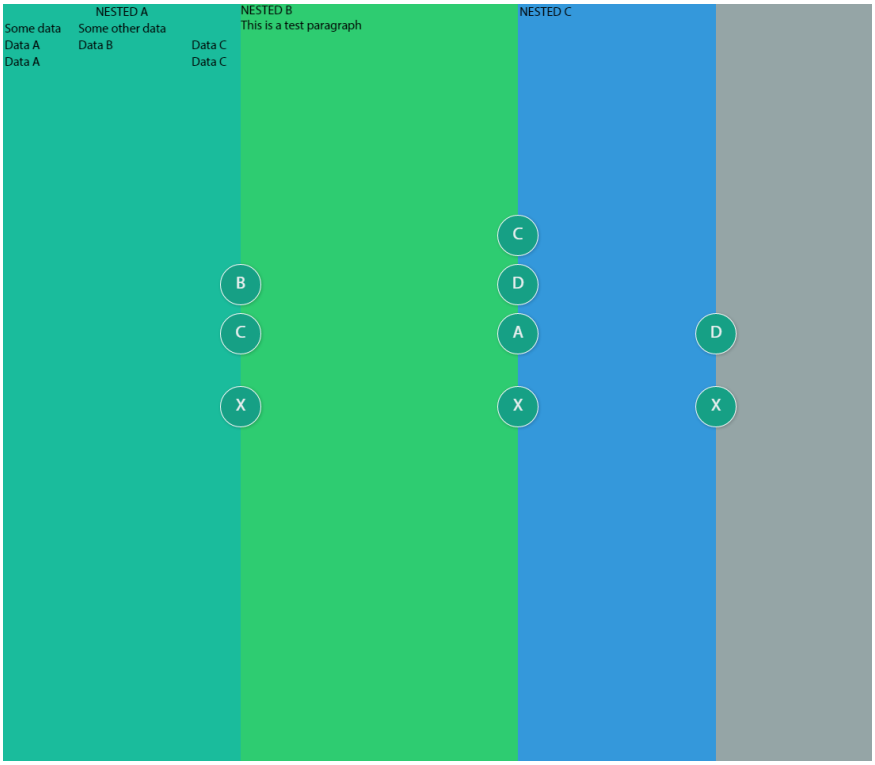


Figure 36. Nested slide menu (normal order)


Figure 37. Nested slide menu (dynamic order)

Attached there is also an early nested slide menu software design document.

As of now the slide menus are animated with the help of a jQuery plugin library called Transit and overall it performs and work well enough. There is another library like Transit and it is called Velocity. Velocity re-implements jQuery's animate-function (as of 28 August, Velocity is now independent and does not require jQuery to work) and supposedly offers major performance and visual improvements over Transit, therefore the slide menu animations could be done with Velocity instead at some point. (Velocity. 2014.)

**jQuery Mobile page transitions**

The way how jQuery mobile transitions from page to page in a single page application is not ideal. It does work and it is fairly fast too, however, there are other options and frameworks that could perform better by a great deal. Switching to those frameworks now that everything is set up to use jQuery Mobile might not be the best choice though.

## 11.2Back end improvements

The back end of the application, as in the server and the database seem to be mostly fine. Database would probably benefit from having views and indexing, however, not having those are certainly not the primary performance bottlenecks

## 11.3 Optimizations

The application is not nearly as optimized as it could be, though most of the major bottlenecks, performance spikes and memory overuse problems have been properly fixed and the application performs quite well overall.

**jQuery**

There is a great deal to optimize when it comes to jQuery and arguably one might suggest not using it all together is the best and most optimal choice. Thinking of the development and the timespan the application needs to be finished in, not using jQuery is not really a choice and therefore only thing that is left is to optimize the usage of it.

Many sources on the internet and jQuery related books advice on saving jQuery selectors (or actually the result) in variables like in the first example in Figure 38. This is a good idea implying that the developers knows what is happening and what is the difference between the examples in Figure 38, otherwise when trying to optimize jQuery selectors like this might result in undefined behavior, because if something changes in the element set which was selected with jQuery and saved into a variable while trying to manipulate it, the changed will not either apply or will partially apply and most likely break the application.

```
//Saving the selector (actually the result of the selector) into a variable (finding and selecting element(s) once)
var selector = $(someSelector);
selector.doSomethingA();
selector.doSomethingB();
selector.doSomethingC();

//Not saving the selector (actually the result of the selector) into a variable (finding and selecting element(s) again every time)
$(someSelector).doSomethingA();
$(someSelector).doSomethingB();
$(someSelector).doSomethingC();
```

Figure 38. jQuery selector use cases

In this application there are very few places where this type of selector optimization can be used and therefore the selectors are not optimized. Not optimizing the selectors might sometimes be a good idea as well because then the selected element set will always be what it should.

# 12 ASSIGNMENT

## 12.1Starting point

The mobile application project started basically from scratch, though there were some planning documentation and UI designs available. Back end, as in database and the means to interact with it, were also in place so the main focus was creating the application itself.

About four months of time was given to come up with some kind of results. Four months was by no means a deadline, but instead sort of a milestone or a progress check.

## 12.2Goal

The goal was to implement as much features as possible into the application during the four months, so that it would be easier to see during and after the four months whet ever some features need to be re-implemented or whet ever the mobile version was even possible to develop like it was first planned.

After four months the application would be in alpha stage, after which it would be planned and developed further when more time and resources were again available.

## 12.3Implementation

Thus, the alpha version was in development for four months, around 7 hours per day. It was created feature by feature in a logical manner, so that when a new feature was implemented, it could be accessed and tested out straight away.

During the development there was some kind of planning talk daily. Features were rethought and changed according to what seemed the best solution for the given time.

Development was carried out the application's performance and further development in mind, so there were some features that were deemed best to implement, for example, as generic as possible so they could be easily reused elsewhere.

## 12.4Results

The goals previously presented were achieved, and as a result, there is now an alpha version of a facility maintenance system, with many features rethought and re-implemented from what the initials plans were. The system performs almost as expected and new features can be added moderately easily.

The current version of the system can be technically already used by end users, as all the most important features have been implemented and they should be working properly. However, the system will still go through some further development in order to weed out most of the known problems and possible bugs.

# 13 CONCLUSION

Developing a mobile system like this was a challenging and interesting work. Many different technologies and techniques were used to complete the system and the outcome was pretty good.

After four months of development, the application is feasible to develop further. The knowledge obtained during the four months helps in improving the system to function even better in future.

All in all the project was interesting to work with and it resulted in a good application and a good idea for a thesis.

# REFERENCES

About Elomatic. 2015. Elomatic in brief article. Accessed on 13.1.2015 http://www.elomatic.com/en/company/elomatic-in-brief.html

Ajax. 2014. Wikipedia article. Accessed on 15 June 2014. Retrieved from http://en.wikipedia.org/wiki/Ajax_(programming)

API. 2014. Wikipedia article. Accessed on 29 July 2014. Retrieved from http://en.wikipedia.org/wiki/Application_programming_interface

Berjon R., Faulkner S., Leithead T., Navara E., O'Connor E., Pfeiffer S., Hickson I. 2014. HTML 5 W3C. W3C article. Accessed on 15 June 2014. Retrieved from http://www.w3.org/TR/html5/

Citrix. 2014. Product about page. Accessed on 8 September 2014. Retrieved from http://www.citrix.com/about.html?posit=foot

Clark R. 2012. HTML5 APIs. Blog post. Accessed on 26 June 2014. Retrieved from http://www.creativebloq.com/html5/developer-s-guide-html5-apis-1122923

Cordova. 2014. Cordova about article. Accessed on 24 June 2014. Retrieved from http://cordova.apache.org/

CSS rule priorities. 2014. Developer manual. Accessed on 29 July 2014. Retrieved from https://developer.tizen.org/dev-guide/2.2.1/org.tizen.web.appprogramming/html/guide/w3c_guide/dom_guide/html_priorities_css.htm

DalekJS. 2014. Framework documentation. Accessed on 28 July 2014. Retrieved from http://dalekjs.com/pages/documentation.html

Database transactions. 2014. Wikipedia article. Accessed on 24 August 2014. Retrieved from http://en.wikipedia.org/wiki/Database_transaction

DOM. 2014. Wikipedia article. Accessed on 15 June 2014. Retrieved from http://en.wikipedia.org/wiki/Document_Object_Model

HTML5 tags. 2014. W3Schools article. Accessed on 26 June 2014. Retrieved from http://www.w3schools.com/html/html5_new_elements.asp

iPhone. 2014. Wikipedia article. Accessed on 24 June 2014. Retrieved from http://fi.wikipedia.org/wiki/IPhone

Jasmine. 2014. Framework documentation. Accessed on 28 July 2014. Retrieved from http://jasmine.github.io/2.0/introduction.html

Julian S. 2014. Velocity. Code library documentation. Accessed on 7 September 2014. Retrieved from http://julian.com/research/velocity/

LESS. 2014. Wikipedia article. Accessed on 20 July 2014. Retrieved from http://en.wikipedia.org/wiki/LESS_(stylesheet_language)

Phonegap. 2014. Phonegap about article. Accessed on 24 June 2014. Retrieved from http://phonegap.com/about/

Protocol. 2014. Wikipedia article. Accessed on 15 June 2014. Retrieved from http://en.wikipedia.org/wiki/Communications_protocol

QUnit can test itself. 2014. Project on github. Accessed on 28 July 2014. Retrieved from https://github.com/jquery/qunit/tree/master/test

Relational model. 2014. Wikipedia article. Accessed on 20 July 2014. Retrieved from http://en.wikipedia.org/wiki/Relational_model

SASS. 2014. Wikipedia article. Accessed on 20 July 2014. Retrieved from http://en.wikipedia.org/wiki/Sass_(stylesheet_language)

Simon L. 2012. Browser reflow. Google developer article. Accessed on 20 July 2014. Retrieved from https://developers.google.com/speed/articles/reflow

Software testing. 2014. Wikipedia article. Accessed on 28 July 2014. Retrieved from http://en.wikipedia.org/wiki/Software_testing

SVG. 2014. W3Schools article. Accessed on 24 June 2014. Retrieved from http://www.w3schools.com/svg/

Types of software testing. 2014. Informative web posting. Accessed on 28 July 2014. Retrieved from http://www.softwaretestinghelp.com/types-of-software-testing/

Why using eval is a bad idea. 2012. Stackoverflow article. Accessed on 24 August 2014. Retrieved from http://stackoverflow.com/questions/86513/why-is-using-the-javascript-eval-function-a-bad-idea