

# **Well-being reservation system**

Martin Gana

Bachelor's thesis

April 2017

School of Technology, Communication and Transport

Bachelor's Degree Program in Information and Communications

Technology

Author(s) Gaña, Martin	Type of publication Bachelor's thesis	Date March 2017 Language of publication: English
	Number of pages 46	Permission for web publication: yes
<b>Title</b> <b>Well-being reservation system</b>		
Degree programme Information and Communications Technology		
Supervisor(s) Salmikangas, Esa		
Assigned by Solteq Oyj		
<p><b>Abstract</b></p> <p>Solteq is a company offering E-commerce services to its clients. IBM WebSphere Commerce is used in most of the projects. There was a need to use more modern technologies so that Solteq could improve its competitiveness and efficiency.</p> <p>Solteq provides wellbeing services to its employees, there are services such as massages or ergonomics training. There was an existing system used for booking these services; however, it had to be removed because the system was not suitable anymore. The development of the new reservation system helped to gain important knowledge about modern cloud based technologies and best practices for the development of applications running in the cloud.</p> <p>The application consists of two main parts: the backend and the frontend. Both parts were created using JavaScript programming language. The backend part was created using the Node.js framework and the frontend part with the Angular.js framework. The backend and the frontend communicate using the Rest API provided by the backend side. All data were stored in NoSql database Cloudant provided by IBM Bluemix.</p> <p>The new reservation system was created successfully. The system provides features for creating new reservations, setting time schedules for different services and creating exceptions to these schedules. Users of the system are notified by email, which contains information about the reservation and the event that can be added to a calendar.</p> <p>The new system is already used by Solteq's employees in Jyväskylä office. The most important features were implemented. The knowledge gained by creating the system was used to create a small demo feature for a customer's e-shop, which was the base for the new wish-list feature that was successfully sold to the customer and is used in production.</p>		
Keywords/tags Cloud, reservation system, JavaScript, IBM Bluemix		
Miscellaneous		

# Contents

1	Introduction.....	5
1.1	Background.....	5
1.2	Motivation .....	5
2	Current reservation system.....	7
2.1	Removal of old reservation system.....	7
2.2	Message service .....	7
2.3	Ergonomics training.....	8
3	Requirements elicitation .....	8
4	High level architecture .....	9
5	Tools & technologies .....	10
5.1	Node.js.....	10
5.2	Express.js .....	11
5.3	Angular.js.....	11
5.4	Babel.....	13
5.5	JsHint .....	14
5.6	Grunt.....	14
6	Cloud solutions.....	16
6.1	Advantages of cloud solutions .....	16
6.2	Platform as a service (PaaS) .....	17
6.3	Cloud Foundry .....	17
6.4	BOSH.....	18
6.5	IBM Bluemix .....	19
6.6	Docker.....	19
7	Implementation.....	20

	2
7.1 Backend .....	20
7.2 Delivery pipeline.....	21
7.3 Data layer .....	25
7.4 Code Structure.....	26
7.5 Unit testing .....	29
7.6 Running the tests.....	30
7.7 Authentication.....	32
7.8 Frontend .....	33
8 New reservation system.....	36
9 Conclusion .....	40
9.1 Lessons learned .....	42
9.2 New reservation system.....	40
9.3 Proof of concept based on gained knowledge.....	41
9.4 Production feature .....	42
References.....	44

## Figures

Figure 1 Client server architecture.....	10
Figure 2 Two-way data binding in Angular.js .....	12
Figure 3 Example of ES6 map function.....	13
Figure 4 ES6 map function transpiled to ES5 .....	13
Figure 5 Grunt configuration of less task .....	15
Figure 6 Grunt configuration of watch task .....	15
Figure 7 Report from grunt task less.....	16
Figure 8 IBM Bluemix delivery pipeline.....	21
Figure 9 IBM Bluemix stage configuration .....	22
Figure 10 IBM Bluemix jobs configuration .....	23
Figure 11 IBM Bluemix deployment script.....	24
Figure 12 IBM Bluemix build artifact history.....	24
Figure 13 Backend login route .....	27
Figure 14 Backend - login controller .....	28
Figure 15 Backend - extended login controller .....	28
Figure 16 Grunt task for unit tests .....	30
Figure 17 Test report.....	31
Figure 18 Grunt watch unit tests.....	31
Figure 19 JWT Authentication .....	33
Figure 20 Frontend feedback factory.....	35
Figure 21 Solteq ADFS .....	36
Figure 22 App - first login .....	37
Figure 23 App - reservation confirmation .....	38
Figure 24 App - calendar .....	39
Figure 25 App - calendar with user's reservations.....	39
Figure 26 App - setting the schedule.....	40

## List of acronyms

<b>CORS</b>	Cross Origin Resource Sharing
<b>ADFS</b>	Active Directory Federation Services
<b>API</b>	Application programming interface
<b>CSS</b>	Cascading Style Sheets
<b>ES6</b>	ECMAScript 6
<b>HTML</b>	HyperText Markup Language
<b>HTTP</b>	HyperText Transfer Protocol
<b>HTTPS</b>	HyperText Transfer Protocol Secure
<b>IaaS</b>	Infrastructure as a Service
<b>IDE</b>	Integrated Development Environment
<b>JS</b>	JavaScript
<b>JSON</b>	Java Script Object Notation
<b>JWT</b>	JSON web token
<b>MVC</b>	Model-View-Controller
<b>NPM</b>	Node package manager
<b>PaaS</b>	Platform as a Service
<b>URL</b>	Uniform Resource Locator

# 1 Introduction

## 1.1 Background

Solteq is an IT-company offering e-commerce solutions. The current goal of the company is to push their development to modern, cloud based technologies, which can improve the ability to deliver the product faster and make it more maintainable. It is important to keep the products maintainable because maintenance generally lasts much longer than the development phase. Being able to deliver high quality products fast is critical for the business. It is the reason there is a need to improve the processes for development and maintenance.

The research and experiments are needed to implement these new technologies into the process. There is usually not enough time to explore different possibilities and approaches when working on a complex customer project. The best candidate for this purpose was an internal project, which was not critical for the business.

Solteq offers wellbeing services to its employees. Anybody can get a 30-minute long massage session or 15-minute ergonomics training. Ergonomics training is a short session held by a professional who helps to improve workplace quality by adjusting table, chair, monitor and explaining the basic rules about resting eyes and muscles to prevent health difficulties.

Employees could book a massage using the software *IBM notes* which was also used for another purpose. User experience was poor and the design was outdated, therefore it was decided that *IBM notes* would not be used anymore.

This decision was the opportunity for trying the new development processes, technologies and exploring the best practices which could be applied to business cases in the future.

## 1.2 Motivation

The teams in Solteq are usually divided into two groups. The development team and the support team. The development team usually implements new features and fixes

serious bugs. Their responsibility is the system architecture, and they also make major decisions concerning the systems.

The support team takes care of customer requirements that are not concerning new features, their responsibility is to investigate problems reported by customer. Most of problems are usually fixed by the support team, if the problem is serious or the solution is not obvious, then the problem is solved by the development team.

Companies want to maximize their income, they need to consider many factors to achieve this. Software can often be created fast but development is the shortest part of a product lifecycle. The maintenance of software takes generally much longer than the development phase, which means that a product can fail in the long term if there is not enough focus on the code quality. This factor is often underestimated.

## Technical debt

The technical debt is a great metaphor developed by Ward Cunningham, to help to think about this problem. In this metaphor, doing things the quick and dirty way sets us up with a technical debt which is like a financial debt. Like a financial debt, the technical debt incurs interest payments coming in the form of the extra effort that must be done in the future development because of the quick and dirty design choice. It can be chosen to pay the interest, or pay down the principal by refactoring the quick and dirty design into a better design. Although it costs to pay down the principal, there is gain by reduced interest payments in the future. (Technical Debt, 2016)

## Importance of unit testing

Writing tests is good for many reasons. They help to discover potential problems easier. Writing tests improves the code design, and therefore improves the maintainability. Creating the software with unit tests takes more time, however, it pays off in the long run. Programmers feel more confident with a tested codebase. The advantage is that it is much easier to change existing code because developers can immediately see if new changes have any negative effect to existing codebase. There is always a bigger risk that something breaks if the codebase is not covered with



tests. This can cause that developers do not refactor the code when it is needed and maintainability gets more complicated and takes more time.

## 2 Current reservation system

### 2.1 Removal of old reservation system

*IBM notes* was used for making reservations. It was decided that it would not be used in the company any longer. It was not used just for booking massages but also for storing passwords. The reservation system is the last service used in *IBM notes*. Creating the reservation system was the last step in the process of removing the software.

### 2.2 Massage service

To create a reservation, employees needs to be logged in to *IBM notes*. Then they need to navigate to the interface for reservations. Already the first step is complicated and the reason for this is that *IBM notes* is not intended to be used as a reservation system.

Once users get to the interface for reservations, they can see a list of available appointment times set by the technical support, and the service provider has no control over it. It means that the service provider must contact the technical support every time there is a need to change the available massage times.

The timetable for massages is the same every week. Available times are listed, and the user can select the any of those. This is good because the service provider does not have to contact the technical support to create a new timetable for every week, however, the service provider cannot set the time or date as disabled, which is a major limitation. This feature could be very useful in case the service provider is on sick leave or missing for any other reason. The only way to inform other employees is to write a note to a specific date, however, employees are still able to create new reservations.

Another limitation is that the reservations cannot be cancelled, which is a serious problem because the only action an employee can do is to change the reservation time. The reservation must be moved to the future instead of being cancelled. The problem is that it is easy to forget about this kind of a reservation. It causes problems for service providers because they must try to offer the available time to somebody else if the employee does not come to the booked massage session. This is usually done using *skype* or an *email*. The service provider needs to fill the free spot fast because the session lasts 30 minutes.

### 2.3 Ergonomics training

The current reservation system does not allow users to book an ergonomics training. The functionality could be the same as for the massage service. The only difference from the technical point of view is that a massage lasts 30 minutes and an ergonomics training lasts 15 minutes. If the reservation system was general, it would be an easy task to add another service with its own timetable and different duration.

## 3 Requirements elicitation

After the first meeting with the service provider in Jyväskylä office requirements were defined. The most important and the most used functionality is the massage booking feature. The goal is to help the service provider with managing ergonomics trainings as well. The only difference between massage and ergonomics training are the different duration and schedule for these services.

There was the decision to build both massage and ergonomics booking as the first feature because it is the core of the system. There are only two kinds of users in this system, service providers and regular employees.

Regular employees should be able to manage only their reservations, meaning they can create a reservation only in their name and can cancel only their reservations.

There might be a case that one user would create too many reservations and other employees would not be able to get any service. The service providers should have an option to remove any reservation if this kind of situation happens.

There was the need to be able to identify a user to fulfill these requirements. It is necessary to distinguish users depending on their role in the system, which means there is a need for authentication and user authorization.

The project was also meant to be a demonstration of agile development using modern, cloud based technologies to gain a knowledge which can be used in different project. Requirements for each phase were agreed upon and followed by the iteration.

The plan was to start with implementing the most important features and less important features were to be added later if needed. The crucial part of the development process was to get frequent feedback from users; therefore it was needed that the product be delivered as fast as possible so the users could test the most up to date version of the product.

In real business, customers need to know when the specific feature will be implemented so they can make their business decisions depending on this estimation. The customer needs to be provided with this information as accurately as possible, which was the reason the tasks in the project needed to be managed and prioritized.

## **4 High level architecture**

The application was designed to be maintainable and easily extensible so other platform such as mobile application could be added later without serious effect to the existing system.

The backend side deals with the business logic, data validation and data persistence. The client side is mostly presentation layer for end users. The backend and frontend are two separate applications, each of those can be maintained separately. The communication between the backend and the frontend is done using REST API provided by the backend.

There are many advantages connected to this kind of architecture, the client side can be changed without any need to change backend side, changes to client the application do not require the server to be restarted.

There can be multiple client applications and each of those can be maintained by another team so it is much easier to maintain the system if it consists of small pieces. In Figure 1 there is an example of multiple clients using the Rest API provided by the server.

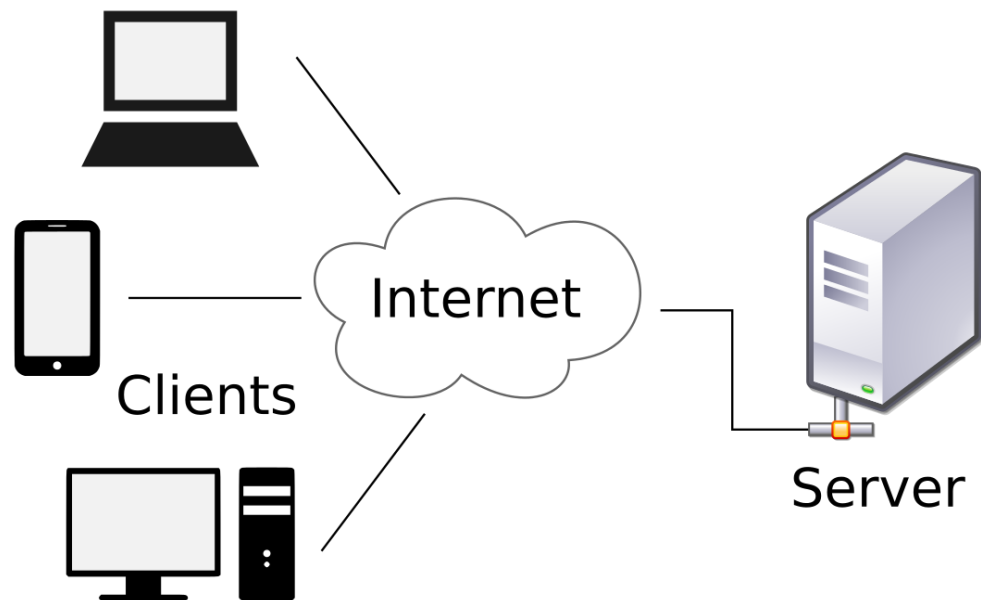


Figure 1 Client server architecture

## 5 Tools & technologies

### 5.1 Node.js

Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.

(Node.js, 2016)

There was a decision to use Node.js for building the backend REST service. It is easy to create prototypes quickly using the JavaScript. This decision also brings the advantage of using the same programming language for both the backend side and the client side application. (Node.js, 2016)

Node.js uses the rich package manager called *npm (node package manager)*. There are many free, open sources packages available, which makes building systems much

easier. Developers do not have to do the same work repeatedly, they just need to install the package they need and configure it. (Node.js, 2016)

For example, the database is needed in almost every application, the developer would just download the package for connecting the specific type of database, configure it with the connection details and that is all the work needed.

This allows developers to focus more on the business logic of the application. JavaScript makes it easy to manipulate with JSON, which is useful when building REST API.

## 5.2 Express.js

Express.js is a popular and so far, the most used Node.js framework for creating a HTTP server, it provides a good quality documentation with variety of extensions created by the open source community. Express.js provides a thin layer of fundamental web application features, without obscuring Node.js features. (Express.js, 2016)

## 5.3 Angular.js

AngularJS is a structural framework for dynamic web applications. Angular's data binding and dependency injection eliminate much of the code the developer would have to write otherwise.

Angular is not a single piece in the overall puzzle of building the client-side of a web application. It handles all the DOM and AJAX glue code a developer once wrote by hand and puts it in a well-defined structure. This makes Angular opinionated about how a CRUD (Create, Read, Update, Delete) application should be built. But while it is opinionated, it also tries to make sure that its opinion is just a starting point the developer can easily change. Angular comes with the following out-of-the-box. (About Angular.js, 2016)

Everything that is needed to build a CRUD app in a cohesive set: Data-binding, basic templating directives, form validation, routing, deep-linking, reusable components and dependency injection.

## Data binding

The data-binding in Angular application is the automatic synchronization of the data between the model and view components. The way Angular implements data-binding lets the developer treat the model as the “single source of truth” in the application. The view is a projection of the model. When the model changes, the view reflects the change, and vice versa.

In Figure 2 there is an illustration of two-way databinding in Angular.js starting from a template.

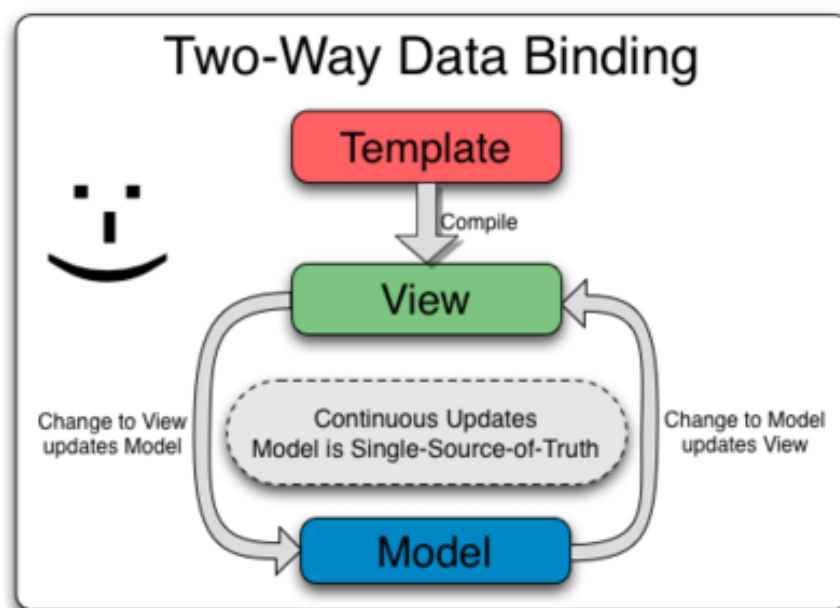


Figure 2 Two-way data binding in Angular.js

The compilation step produces a live view. Any changes to the view are immediately reflected in the model, and any changes in the model are propagated to the view. The developer can think of the view simply as an instant projection of their model.

Because the view is just a projection of the model, the controller is completely separated from the view and unaware of it. This makes testing easy because it is easy to test the controller in isolation without the view and the related DOM/browser dependency. (Databinding, 2016)

## 5.4 Babel

There are many browsers available and their implementation of JavaScript standards is on different levels. It makes the web development more complicated because the developer must do the extra work when checking the compatibility of used language features with different browsers.

Sometimes it occurs that the same piece of code works in the latest versions of modern web browsers such as a Google Chrome but does not work in older browsers such as Internet Explorer 9.

Babel partially solves this problem. It is a very popular tool, which takes the code using the new features and rewrites the same code using the older features most of the browser supports. This is very often used to translate new EcmaScript 6 (ES6) to older EcmaScript 5 (ES5).

There is a snippet of code written using the ES6 in Figure 3 Example of ES6 map function. It is an example of ES6 map function. The arrow function was not available in ES5.

```
1 [1,2,3].map(number => number+2)
```

*Figure 3 Example of ES6 map function*

The developer needs to command Babel to take the code as an input and make the transformation. There is the output of the transformation made by Babel in Figure 4. Code was transformed to ES5 syntax and can be run also in browsers without ES6 support.

```
1 "use strict";
2
3 [1, 2, 3].map(function (number) {
4     return number + 2;
5 });
```

*Figure 4 ES6 map function transpiled to ES5*

The reason the Babel is so popular amongst web developers is that it takes some of their responsibility and they can focus more on creating software than on browser compatibility issues.

## 5.5 JsHint

Developing the JavaScript applications can sometimes be tedious and the developer can face a lot of unnecessary problems, it is mostly because JavaScript is not a strongly typed language.

JsHint is a JavaScript linter. Linters are tools used to improve the code quality. They help the developer to detect possible errors and potential problems. Linters also check the predefined code formatting so every member of a team must follow the same rules and result is a more consistent code base.

## 5.6 Grunt

*Grunt* is a JavaScript task runner used for the automation. This project uses it to run the *Babel* for the code transformation from ES6 to ES5, to run the *JsHint* for checking predefined rules for the code formatting and to run automated tests.

Grunt tasks are configured to run before every commit. If any of specified task fails, commit fails also and the developer must fix code the before trying to make a commit again. The result is the improved code quality because nobody can commit the code if tests are failing or if there are any linter errors.

It is possible to create *watcher tasks* using the Grunt. A watcher task is a task which observes some specified files and waits for a change. The task is started if any of observed files changes. This project uses watcher tasks for transformation the sass syntax to the CSS and for running tests so the developer does not have to start the tasks manually.

There is an example of the watcher for compiling *sass* to CSS files in Figure 5. There is a configuration of the task. The developer needs to provide source and destination files. Source files contain sass source written using sass syntax which are then transformed to CSS and are stored to destination folder.



```

92 ▼      less: {
93 ▼          development: {
94 ▼              files: [{
95                  expand: true,
96                  cwd: 'public/app/assets/less/',
97                  src: ['*.less'],
98                  dest: 'public/app/assets/css/',
99                  ext: '.css'
100             }]
101         }
102     }
103
104     });

```

Figure 5 Grunt configuration of less task

Figure 6 contains the setup for watcher tasks. For example the definition on line 35 means: "Observe all files with extension **.less** in directory **/public/app/assets/less** and if any of those files changes run the task **less**".

```

19 ▼      watch: {
20 ▼          unit: {
21 ▼              files: [
22                  'test/unit/**/*.js',
23                  'src/**/*.js'
24              ],
25              tasks: ['mochaTest:test']
26          },
27 ▼          jshint: {
28              files: ['<%= jshint.files %>'],
29              tasks: ['jshint']
30          },
31 ▼          babel: {
32              files: ['<%= jshint.files %>'],
33              tasks: ['babel']
34          },
35 ▼          less: {
36              files: ['public/app/assets/less/*.less'],
37              tasks: ['less']
38          }
39      },

```

Figure 6 Grunt configuration of watch task

In Figure 7, there is a report after running the task called **less**. The developer is provided the information on which files have changes.

```
~/Documents/wellbeing$ grunt watch:less
Running "watch:less" (watch) task
Waiting...
>> File "public/app/assets/less/calendar.less" changed.
Running "less:development" (less) task
>> 7 stylesheets created.

Done, without errors.
Completed in 0.977s at Mon Aug 22 2016 17:33:09 GMT+0300 (EEST) - Waiting...
□
```

Figure 7 Report from grunt task less

## 6 Cloud solutions

### 6.1 Advantages of cloud solutions

The goal is to be able to deliver the high-quality products fast, therefore there is a need to focus on development, testing and deployment. Usually there are multiple environments needed for the application and all of those need to be configured. An extra effort needs to be put to managing and setting up these environments.

Many tasks connected to configuring the environments can be automated. There are many cloud providers offering services that can be used to minimize the effort used on managing the environments and hardware. These services can do automatic scaling of the resources such as the database. There is no need to take care of the hardware because it is also managed by cloud providers. There are also other useful tools provided such as monitoring, so the developers can see if there are any problems with the application. Error logs can be usually accessed without any problems.

There are many advantages using cloud services; however, there is also a downside to that. Using these services for production application can be expensive so it is important to identify what services are worth to pay for. There were not many people experienced with cloud services who could identify how the company can benefit from using these services, therefore, the goal of this project is also to explore this area.

## 6.2 Platform as a service (PaaS)

Platform as a service is a category of cloud computing services that provides a platform allowing customers to develop, run, and manage applications without the complexity of building and maintaining the infrastructure typically associated with developing and launching an application. PaaS can be delivered in two ways. The first one is a public cloud service from a provider, where the consumer controls software deployment and configuration settings. The provider provides the networks, servers, storage and other services to host the consumer's application. The second way PaaS can be delivered is as a software installed in private data centers or public infrastructure as a service and managed by internal IT departments. PaaS is used to develop web and mobile applications using components that are pre-configured and maintained by the service provider, including programming languages, application servers and databases. (Platform as a service, 2017)

## 6.3 Cloud Foundry

Cloud foundry is an open source cloud computing platform as a service (PaaS) originally developed by VMware and currently owned by Pivotal Software - a joint venture by EMC, VMware and General Electric. Cloud Foundry was designed and developed by a small team from Google led by Derek Collison and was originally called project B29. It is primarily written in Ruby and Go. (Cloud Foundry, 2017)

Cloud Foundry supports the full lifecycle, from initial development, through all testing stages, to deployment. It is therefore well-suited to the continuous delivery strategy. Users have access to one or more *spaces*, which typically correspond to a lifecycle stages. For example, an application ready for QA testing might be *pushed* (deployed) to its project's QA space. Different users can be restricted to different spaces with different access permissions in each of them. (Cloud Foundry, 2017)

When an application is deployed to Cloud Foundry, an image is created for it and stored internally. The image is then deployed to a *Warden* container to run in. For multiple instances, multiple images are started on multiple containers. This is where BOSH comes in - Cloud Foundry's internal Controller uses BOSH to get the underlying

infrastructure to spin up virtual machines to run the Warden containers on. When an application is deleted, all its containers are destroyed and their resources are freed for other applications to use. If the application instance crashes, its container is killed and a new Warden container is started automatically. A container only runs one application ensuring isolation, security and resilience. (Cloud Foundry, 2017)

A load-balancing router sits at the front of Cloud Foundry to route incoming requests to the correct application - essentially to one of the containers where the application is running.

Applications deployed to Cloud Foundry accesses external resources via *Services*. In a PaaS environment, all external dependencies such as databases, messaging systems, files systems and so on are *Services*. When an application is *pushed* to Cloud Foundry, the services it should use also can be specified. Depending on the application language, auto-configuration of services is possible - for example a Java application requiring a MySQL database picks up the MySQL service on Cloud Foundry if it is the only one defined in the current *space*.

## 6.4 BOSH

BOSH is a project that unifies release engineering, deployment, and lifecycle management of small and large-scale cloud software. BOSH can provision and deploy software over hundreds of VMs. It also performs monitoring, failure recovery, and software updates with zero-to-minimal downtime.

While BOSH was developed to deploy Cloud Foundry PaaS, it can also be used to deploy almost any other software (Hadoop, for instance). BOSH is particularly well-suited for large distributed systems. In addition, BOSH supports multiple Infrastructure as a Service (IaaS) providers like VMware vSphere, vCloud Director, Amazon Web Services EC2, and OpenStack. There is a Cloud Provider Interface (CPI) that enables users to extend BOSH to support additional IaaS providers such as Google Compute Engine and Apache CloudStack.

## 6.5 IBM Bluemix

There are several PaaS providers, for example *Amazon Web Services*, *Windows Azure Cloud Services*, *Heroku*, *IBM Bluemix* and many more. There was a decision to use *IBM Bluemix* because Solteq is IBM's partner.

Bluemix supports wide range of programming languages and services, it also integrates DevOps to build, run deploy and manage applications on the cloud. Bluemix is based on cloud foundry, which is good for us since cloud foundry is open source. The advantage is that PaaS provider can be changed to another cloud foundry based one without unnecessary complications.

## 6.6 Docker

There are usually many developers working on the same project. There can be situations when the software works for one developer but does not work for another. Developers usually use different operating systems and different development environments. Their environments are not configured in a same way and it often happens they use different versions of software.

For example, one developer can use Node.js version 4.4.3 while another uses the version 6.7.0 which can cause major problems since version 6.7.0 supports syntax that version 4.4.3 does not. Plenty of time can be wasted trying to figure out what is wrong with software while there is no real problem with software rather than with its dependencies.

Another problem of a team development is the unnecessary work that needs to be done every time a new developer joins the team. The first thing that needs to be done by a new developer is usually to clone the project from git a repository and then setting up all necessary services such as database, mail server or different services. New developer must learn about the project infrastructure which is often not so important to be able to work on small features.

Solving out these two problems can lead to saving some time for developers and reducing the risk that the application will not behave the same way in test, staging or production environment. These problems can be solved using Docker.

Docker is a tool designed to make it easier to create, deploy and run applications by using containers. Containers allow a developer to package an application with all the parts it needs, such as libraries and other dependencies, and ship it all out as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.

In a way, Docker is a bit like a virtual machine. But unlike a virtual machine, rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel as the system that they are running on, and only requires the applications be shipped with dependencies not already running on the host computer. This gives a significant performance boost and reduces the size of the application.

And importantly, Docker is open source, which means that anyone can contribute to Docker and extend it to meet their own needs if they need additional features that are not available out of the box. (What is Docker, 2016)

## 7 Implementation

### 7.1 Backend

The backend consists of *Node.js* application running in IBM Bluemix, it exposes an REST API, which is an endpoint for communication with frontend application. The data is stored in *NoSql* database Cloudant, which is an IBM product based on *CouchDB*. Cloudant is available as a service in IBM Bluemix, the database as a service is used for a simple reason. There is no need to install, maintain or configure the database, the application simply connects the database using provided credentials and everything is ready.

The other benefit of using the database as a service is the automatic scalability. Bluemix provides the DevOps tools Solteq is interested in, therefore it was decided to try it out and build a delivery pipeline using the tools Bluemix provides.

## 7.2 Delivery pipeline

The delivery pipeline takes care of deploying application to two different environments in the cloud, and everything is automatic. Git is used as a version control in this project, and there are three main branches mapped to specific environments in the cloud.

The test branch maps to the test environment and the master branch maps to the production environment. Each environment is a different application in the Cloud Foundry, the advantage of this approach is that we can have different versions of application running in different environments.

Deployment is possible in two different ways, however, both use Cloud Foundry. The first option is to use Cloud foundry command line interface. Another, a more convenient way is to build the delivery pipeline in Bluemix.

The reservation system deployment pipeline is presented in Figure 8. The reader can notice that there is no development environment because development is done locally and there is no need to run the code in the cloud.

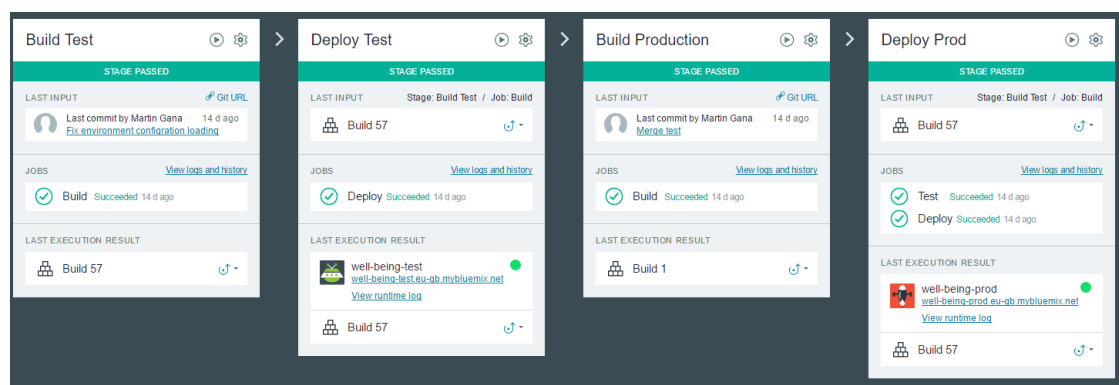


Figure 8 IBM Bluemix delivery pipeline

The first step in the workflow is to create changes to code in the development branch, changes are committed to the development branch after. It is important that the code in the version control is as good quality as possible, therefore there is a grunt task for running the tests and the linter before every commit.

The changes can be pushed to the remote branch so it is available for the rest of the team. The next step of the delivery process is to deploy the changes to the test environment, it is done so the test branch is merged to the test branch.

At that point, there is no manual work needed anymore, IBM Bluemix takes care about the rest and takes the newest code from the remote test branch and builds new version of application, which is then deployed to the test environment.

Figure 9 illustrates the definition of the task called **Build test**. There is the input type field, which is set to SCM Repository, there is also the Git url and Branch specified. The last option is the Stage trigger. These options mean that Bluemix will wait for any change to the test branch in the git repository and start the deployment process if there is any new code.

The screenshot shows the 'Stage Configuration' interface for a 'Build Test' stage. The interface is divided into three tabs: 'INPUT', 'JOBS', and 'ENVIRONMENT PROPERTIES'. The 'INPUT' tab is active. Under 'Input Settings', there are three fields: 'Input Type' (set to 'SCM Repository'), 'Git URL' (set to 'https://hub.jazz.net/git/mgana/well-being-test'), and 'Branch' (set to 'test'). Below these fields is the 'Stage Trigger' section, which has two radio button options: 'Run jobs whenever a change is pushed to Git' (which is selected) and 'Run jobs only when this stage is run manually'. At the bottom right, there are 'SAVE' and 'CANCEL' buttons. A 'DELETE' button is located at the top right of the configuration area.

Figure 9 IBM Bluemix stage configuration

Another tool available in the deployment pipeline is the possibility to set jobs, which are triggered during build phase. The **Build Test** stage installs all dependencies specified in the package.json file, it is done using the `npm install` command. It downloads and installs all the specified dependencies. The `npm test` task runs after, it is used to test the installed application in case there is any problem which occurred during the



installation process. The last command is the *grunt babel*, which translates the syntax of ES6 to ES5 syntax.

There is a definition of jobs shown in Figure 10. The build stage is successful when all specified tasks are successful. Job works like pre-commit hook in development environment. If any job fails, the build fails also.

The screenshot displays the 'Stage Configuration' interface for a 'Build Test' job. The interface is divided into three tabs: 'INPUT', 'JOBS', and 'ENVIRONMENT PROPERTIES'. The 'JOBS' tab is active, showing a 'Build' job configuration. A 'DELETE' button is visible in the top right corner of the job configuration area. Below the job name, there is a 'Build Configuration' section with a 'REMOVE' button. The 'Builder Type' is set to 'npm'. The 'Build Shell Command' is shown in a dark text area with the following content:

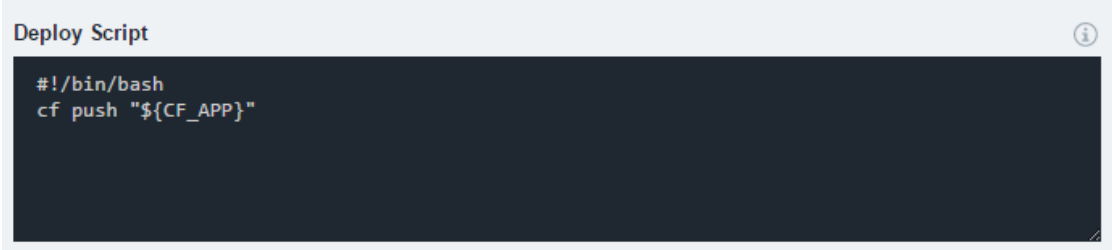
```
#!/bin/bash
# The default Node.js version is 0.10.40
# To use Node.js 0.12.7, uncomment the following line:
#export PATH=/opt/IBM/node-v0.12/bin:$PATH
# To use Node.js 4.2.2, uncomment the following line:
export PATH=/opt/IBM/node-v4.2/bin:$PATH
export NPM_CONFIG_PRODUCTION=false
npm install
npm test
grunt babel
```

At the bottom of the configuration area, there is a link: "Don't have a build script? Create a new one from a template. + ADD".

Figure 10 IBM Bluemix jobs configuration

The result of a successful build is the build artifact, which is also input for the next stage in the deployment process. The name of the next stage is **Deploy test**. The artifact is deployed in this stage, the application is immediately available after the successful deployment. There is the deploy script available in Bluemix console. It can be customized in case there are some special steps that need to be done in the deployment process.

In Figure 11 there is a command needed for the application deployment. The command can be modified and extended. The illustration contains just the bare minimum needed to work.



```

Deploy Script
#!/bin/bash
cf push "${CF_APP}"

```

Figure 11 IBM Bluemix deployment script

There might be the situations that the application does not work properly after the deployment even if the build stage passed without any problems. Usually there is a need for high availability of the application and the downtime should be minimal. Bluemix stores the built artifacts and provides their history. The history of build artifacts is shown in Figure 12. This is a powerful feature allowing the developer to roll back to the last working artifact and minimize downtime of the application.



57	Succeeded	14 d ago
Build Succeeded		
56	Succeeded	14 d ago
55	Succeeded	14 d ago
54	Succeeded	14 d ago
53	Canceled	15 d ago
52	Succeeded	15 d ago
51	Failed	15 d ago
50	Succeeded	15 d ago

Figure 12 IBM Bluemix build artifact history

At this point, the new version of the application is running in the test environment. This environment is used for testing purposes so the testers are notified and developers can continue implementing the new features or fixing bugs found by testers.

The newest version of the application can be deployed to the production environment after the testing is done and all necessary fixes are implemented. The deployment to the production environment is triggered the same way the test deployment is. The test branch is merged to the master branch, Bluemix is then notified that there is a new code in the master branch and the production deployment process is started.

The pipeline for the deployment to the **production environment** is almost the same as the pipeline for the deployment to the **test environment**. The only difference is that there is no need to run tests now because the test and the production environments are the same and the application already works properly in the test environment.

### 7.3 Data layer

The data layer was designed in a way that it is easy to change the database service if needed. There are no direct calls to the database service, framework *Waterline* is used instead. Waterline is an ORM (object-relational mapping) with a support for many relational and non-relational databases, it provides an easy way to change the database type if needed.

The main motivation to abstract the communication to the database was that Cloudant database is offered only in Bluemix, and there might be problems in case of migration to another cloud provider if needed.

Cloudant is a NoSql database, it is schemaless, and therefore allows that it is possible to insert any valid document without a specified schema, the advantage of no schema being that the data do not have to be structured in the same way, and therefore it is much easier to extend the data if needed, even without any change to the backend code, however, in practice it is often needed to define the data structure for stored object so the data is consistent. (Cloudant, 2017)

The reservation system needs to store information about users who create reservations. There is a need to have their emails, which is internally used as a user identifier, also name is important so the service provider knows who has made a booking. The phone number is also stored in the database; however, it is not required. The

database also stores all the necessary information about timetables for reservation. The plan was that the service would be used in multiple offices so it was needed to design the database in a way that there can be different timetables for different offices. There is also a possibility to add exceptions to these timetables, which are also stored in the database.

The term document is used for record stored in the database. Cloudant stores documents in the JSON format so it is easy to use in JavaScript and no extra transformation is needed.

## 7.4 Code Structure

It is important to design a good structure so the code is easily testable, the code should be also easily readable so the maintenance does not require much time and developers can understand it without further knowledge.

It is possible to improve the code structure by implanting unit tests. The unit testing helps to improve the code structure because it forces the developer to write the code with as little dependencies as possible because it is tedious to test dependent code. The code without dependencies is much more reliable because it is possible to make a change to specific unit without affecting the other code.

The main components of the code architecture models, controllers and routes. All of those are custom node modules which are composed to achieve a specific functionality.

Route modules are used to define which controller action should be used for the specific route. The data is extracted from the request, it is validated and passed to the controller. It is also possible to do the extracting and validation in controllers but the unwanted dependency of controller on request is removed this way. Route modules are also responsible for sending the response based on result of the operation.

In the beginning the extracting the data from request and the data validation was done in the controller but it was too complicated to write unit tests for controllers, because developers had to create fake requests but it was extra work which was removed simply by redesigning the code.

Figure 13 contains a snippet of code from a route module. There is a handler for requests to the login url.

```
14     app.post('/login', function(req, res) {
15
16         const email = req.body.email;
17         const password = req.body.password;
18
19         if (!email || !password) {
20             return Response.error({
21                 code: 400,
22                 message: "Please enter email & password."
23             }, res);
24         }
25
26         UsersController.login(email, password)
27
28         .then(function(token) {
29             Response.success(200, {
30                 token: token
31             }, res);
32         })
33
34         .catch(function(customErr) {
35             console.log(customErr);
36             Response.error(customErr, res);
37         });
38
39     });
```

Figure 13 Backend login route

There is a code snippet of the *user-route* in Figure 13. The backend exposes the API endpoint for login. The login data is extracted from the request, then it is checked if the email and password were provided by the user.

The error response is sent if the email or password is missing, in another way it is passed to the controller's login function. The implementation of the login functionality is hidden, the response is handled after *UsersController* is done.

Controllers communicate with libraries and the data layer. Their job is to combine available services and resources to achieve the expected result. As a result of separating the route and the controller there is a much cleaner code. *UsersController* combines *Auth* service, *Users* data layer and returns the authentication token if login was successful or exception if anything went wrong.

A login function is illustrated in Figure 14. It is easy to understand what is happening even without knowing anything about the implementation. The first step is to authenticate against Solteq's AD authentication service, after the successful authentica-

tion it is checked if the user has already a profile created. If there is no profile found the profile is stored to the database, after that the token is generated and returned to the route module which called the login function.

```
12 module.exports.login = function(email, password) {
13
14     return new Promise(function(resolve, reject) {
15
16         Auth.authenticateSolteqAd(email, password)
17
18         .then(User.createIfDoesNotExist.bind(User, email))
19
20         .then(Auth.generateToken.bind(Auth, email))
21
22         .then(resolve)
23
24         .catch(reject);
25
26     });
27 };
```

Figure 14 Backend - login controller

Figure 15 shows an example of extending the login function by logging the information about the user during the login process. Only one line of a code was added to extend the functionality.

```
12 module.exports.login = function(email, password) {
13
14     return new Promise(function(resolve, reject) {
15
16         Auth.authenticateSolteqAd(email, password)
17
18         .then(User.createIfDoesNotExist.bind(User, email))
19
20         .then(Auth.generateToken.bind(Auth, email))
21
22         .then(Logger.logUserLogged(email))
23
24         .then(resolve)
25
26         .catch(reject);
27
28     });
29 };
```

Figure 15 Backend - extended login controller

## 7.5 Unit testing

The unit testing is very important to achieve the goal of clean and maintainable code. It is important to write readable tests, so it can be used like a form of documentation later. New developers do not have to find out how existing code works to use it. They can find understandable examples in tests.

Modules and functions should be small, doing only one thing and doing it well. The benefit of this approach is not just a cleaner code but also a bigger reusability of small components.

The system is fully written in the JavaScript; therefore testing was done using the JavaScript. There are many JavaScript testing frameworks, however, it was decided to use Mocha, mostly because of available extensions and a good documentation.

Mocha is a JavaScript framework running in Node.js as well as in browser. Tests run sequentially, allowing for the flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases. Mocha provides also a way to test the asynchronous code. The expected result is compared to the actual result. This process is called an assertion. (Mocha.js, 2017)

An assertion is a Boolean expression at a specific point in a program which will be true unless there is a bug in the program. A test assertion is defined as an expression, which encapsulates some testable logic specified about a target under test. (Assertion testing, 2016)

There are many popular assertion libraries like *assert.js*, *expect.js*, *should.js*, *Chai* and others.

The *Chai* was used since it has many extensions available as node modules and can be installed using *npm*. Promises are widely used in both, backend and frontend application, therefore there is a need to be able to test them. Chai has many extensions which designed for specified use case. One of those is a module called *chai-as-promised*, it is an extension designed for testing the promises. (Chai as promised, 2017)

Mocha provides hooks. A hook is some logic, typically a function or a few statements, which is executed when the associated event happens. Mocha has hooks that are executed in different parts of test suites—before the whole suite, before each test, and so on. In addition to **before** and **beforeEach** hooks, there are **after**, and **afterEach** hooks. They can be used to clean up the testing setup, such as database data. (Mocha.js, 2016)

## 7.6 Starting tests

The test cases are executed using a grunt task, *grunt-mocha* module is used for this purpose. The advantage of this approach is that the test task can be run by other grunt tasks.

Figure 16 shows a configuration of the grunt task called “mochaTest”. The results of the tests are saved into the *results.txt* file. The key *src* is the source path for the files containing the test cases. Any JavaScript file found in the *test/unit/* directory is executed in this example.

```
35     mochaTest: {
36         test: {
37             options: {
38                 reporter: 'spec',
39                 captureFile: 'results.txt',
40                 quiet: false,
41                 clearRequireCache: false
42             },
43             src: ['test/unit/**/*.js']
44         }
45     },
```

Figure 16 Grunt task for unit tests

The task for running the tests is executed by running the single command. The developer gets results of tests immediately after running the command. Figure 17 contains a report for all the test cases and their results. The execution time is available at the end of the report.



```

Email controller - Sending emails after reservation was created
  ✓ should not send email if user is employee and doesn't own reservation
  ✓ should send email if user is employee and owns of reservation
  ✓ should not send email if user is employee and doesn't own reservation

ICS Generator service
  ✓ should generate ics file for given reservation

Mail Service
  ✓ should compose email from reservation and add attachment

Reservations test cases
  Create new reservation
    ✓ should create reservation
  Validate reservation data
    ✓ should reject reservation - missing user
    ✓ should reject reservation - missing city
    ✓ should reject reservation - missing service
    ✓ should reject reservation - missing time
    ✓ should reject reservation - unknown service
  Reservation time suits schedule
    ✓ day doesn't fit
    ✓ time doesn't fit
    ✓ time and date suit schedule
  Reservations cannot be created in past
    ✓ time in future should be accepted
    ✓ time in past should be rejected
  Check if time is not booked yet
    ✓ time should be already booked
    ✓ time should be free
  Check if user has right to manage reservation
    ✓ employee should not have access to somebody else's reservation
    ✓ employee should have access to his own reservation
    ✓ admin should have access to any reservation

21 passing (317ms)

```

Figure 17 Test report

It is not very comfortable if developers must run the task every time they make any change to the code. Grunt watch task can be used as a solution for this problem. The developer needs to define the files which will be observed.

If any of specified files changes, then the defined task will be executed. It is much more comfortable in comparison to running tests manually. Figure 18 shows a command for starting the task for observing files.

```
$ grunt watch:unit
```

Figure 18 Grunt watch unit tests

## 7.7 Authentication

There are two types of users, the employee and the service provider. These types of users do not have the same permissions in the system. Employee can only create and manage their own reservations; however, service providers can manage any reservation and view some extra information such as notes taken during the massage or ergonomics session. Regular employees are not allowed to view this kind of information; therefore, the system needs to be able to distinguish who the logged in user is.

The backend application does not keep any user specific state, also known as a session. All the communication with the frontend application is done using REST API calls, therefore the information about the user must be sent with every request. This information is stored in an Authorization header and is known as token.

A user logs in using the frontend application which sends an email and a password to the backend application. The backend application then checks if the email is valid and exists in the database, it also validates if the password is correct. If both the email and the password are correct the server generates a token based on the information about the user, such as his email, user id and role. This token is then returned to the frontend application and set to each request as an Authorization header. This way the server can determine which user is making the call and decide if the user has permission to perform the operation.

Figure 19 illustrates the flow of requests in communication which uses JWT authorization tokens.

## Modern Token-Based Auth

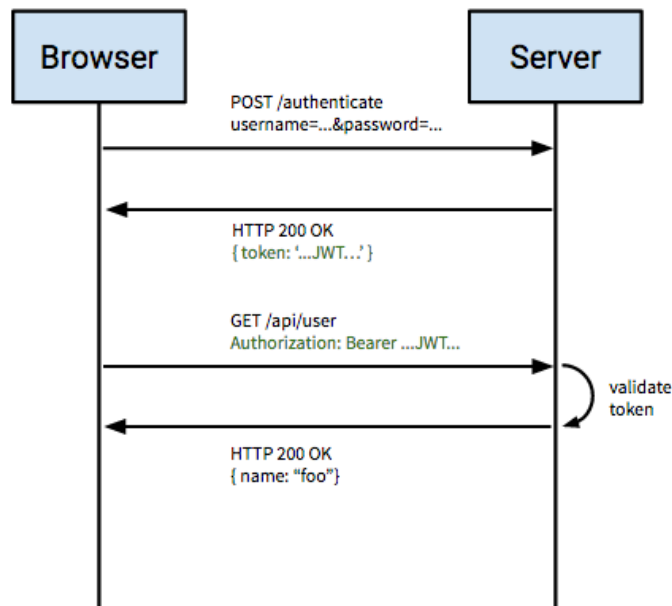


Figure 19 JWT Authentication

It is important to point out that all calls to API need to be done using the HTTPS protocol because of security issues.

HTTPS is a protocol for secure communication over a computer network, which is widely used on the internet. It consists of communication over HTTP (Hypertext Transfer Protocol) within a connection encrypted by TLS (Transport Layer Security). (HTTPS, 2016)

It is easy to steal the token when requests are sent using the HTTP protocol because it is not encrypted. A possible attacker can sniff the network and see all requests sent within the network, the attacker can explore the request and read the token sent in the Authorization header of the request. Then the attacker can use the token and pretend to be a logged user with a full access to the account. The server cannot notice that the token was stolen.

### 7.8 Frontend

It is important to provide simple user interface so the application is easy to use, user experience is also very important. The frontend part was designed by a designer,

which is also the user experience professional. The first step for designing the user interface was to identify who the users are. There was a meeting with service provider, designer and programmers to identify service provider's needs for the system. A part of the discussion was to identify problems using the older system and to propose new ideas to improve the system. The designer then created personas and user stories for them. This technique is helpful for designing the system in a way that it is comfortable to use for people of different ages and with different skills. Some users might prefer using the system via the mobile phone, however, other people might prefer the website. The designer then created the interface with the design using the Photoshop and provided the design for different screens to programmers.

This way the programmers did not have to struggle with creating the design by themselves, they could focus on implementing the functionality and design style the user interface following the screens provided by the designer.

The frontend part was built using the Angular.js framework. The bootstrap CSS framework was used for styling the application. The frontend is responsible for presenting the data available using the API. It communicates with the backend using HTTP requests. The important part of the frontend application is to provide the feedback for user's action, it is important to notify the user if the operation was successful or not. The toastr.js library was used for notifying messages. User is for example notified after the reservation was successfully created.

There are many kinds of notifications in the system so it makes a sense to create a component which takes care of notifications and can be used anywhere. The plugin for notifications can be changed in the future so it would not be a good idea to call the toastr.js library directly, wrapper was created instead. The technology for providing the feedback can be changed in one place without affecting the rest of the code. The wrapper utility provides functions **success**, **info** and **error**, it is shown in Figure 20:

```

3
4  angular.module('myApp')
5
6  .factory('Feedback', [
7
8      function() {
9
10         "use strict";
11
12         return {
13             success, error, info
14         };
15
16         function success(message) {
17             toastr.success(message);
18         }
19
20         function info(message) {
21             toastr.info(message);
22         }
23
24         function error(error) {
25
26             var message = "Error occurred during processing the event";
27
28             if (error.data && error.data.message) {
29                 message = error.data.message;
30                 toastr.error(message);
31             } else {
32                 throw error;
33             }
34         }
35     }
36 ]);
37

```

Figure 20 Frontend feedback factory

The programmer can specify what the notifications look like by using the specific function. Every function creates a window with a specific style. A success notification is green, info notification blue and an error notification red. All styles can be adjusted using CSS.

## Single page applications

Single page applications use client side rendering; they load all templates once and store them in memory, which means less stress for the server. The communication with the backend application is usually done using REST API. The advantage of this approach is smaller traffic (especially useful for mobile internet connections). The application looks more interactive because of fast responses

Single page applications usually provide a rich user experience. Everything looks smooth and the responses are fast. The difference between a single page application and multipage application is that single page applications do not reload sources. Mul-

tipage applications reload sources on every redirect, which means that whenever a user navigates through a site, requests are sent to a server and usually also all HTML content is rendered by a server.

## 8 New reservation system

The new reservation system provides a simple way to create and manage reservations: it is easier to use than IBM Notes and has a more user-friendly interface. The advantage to IBM Notes is that the new reservation is a single purpose application instead of being focused on many different tasks. The new system does not have so many features but that is the advantage because the new system does one thing and does it well.

Users must log in using the company's ADFS authentication. They are redirected to the page where they are asked to enter their company user name and password. If both the email and the password are correct users are redirected back to the reservation system.

Figure 21 shows a screenshot of the site for ADFS login in. Users are asked to select the website they want to be logged in.

adfs.solteq.com

You are logged.

Log on to one of the websites below:

Well-Being 

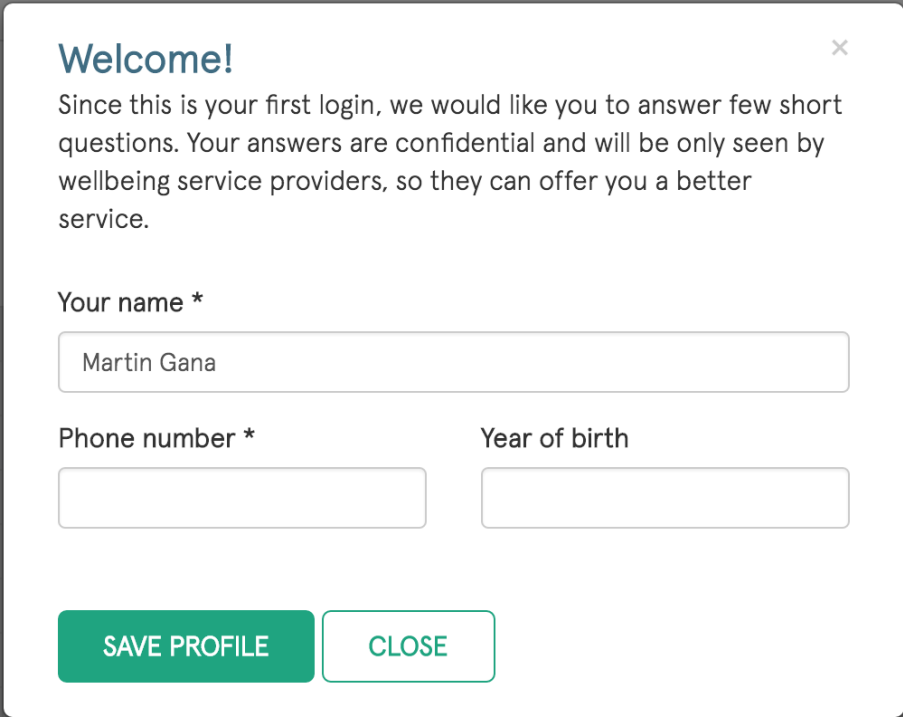
Log out of all the sites that you entered.

Sign out of this site.

Figure 21 Solteq ADFS

Users are redirected to the reservation system after the successful login. The system asks users for basic information after the first login. It is needed by service providers. The first and the last name are prefilled so users can just edit the name if they want to. This is done by splitting the email address of the user. Every Solteq's employee has an email address in a form of *firstname.lastname@solteq.com*, therefore it is easy to provide this kind of feature.

In Figure 22 there is a popup with prefilled name of the user displayed after the first login. The required fields are marked with an asterisk. User cannot create a reservation if any of required information is missing, instead they are asked again to fill it.



**Welcome!** ×

Since this is your first login, we would like you to answer few short questions. Your answers are confidential and will be only seen by wellbeing service providers, so they can offer you a better service.

Your name \*

Phone number \*      Year of birth

[SAVE PROFILE](#)   [CLOSE](#)

Figure 22 App - first login

Figure 23 illustrates the reservation confirmation. There is a link "Fill the profile" which navigates to the same page as the page shown in Figure 22. It is easier to use because users do not have to look for a place where to set their profile information.

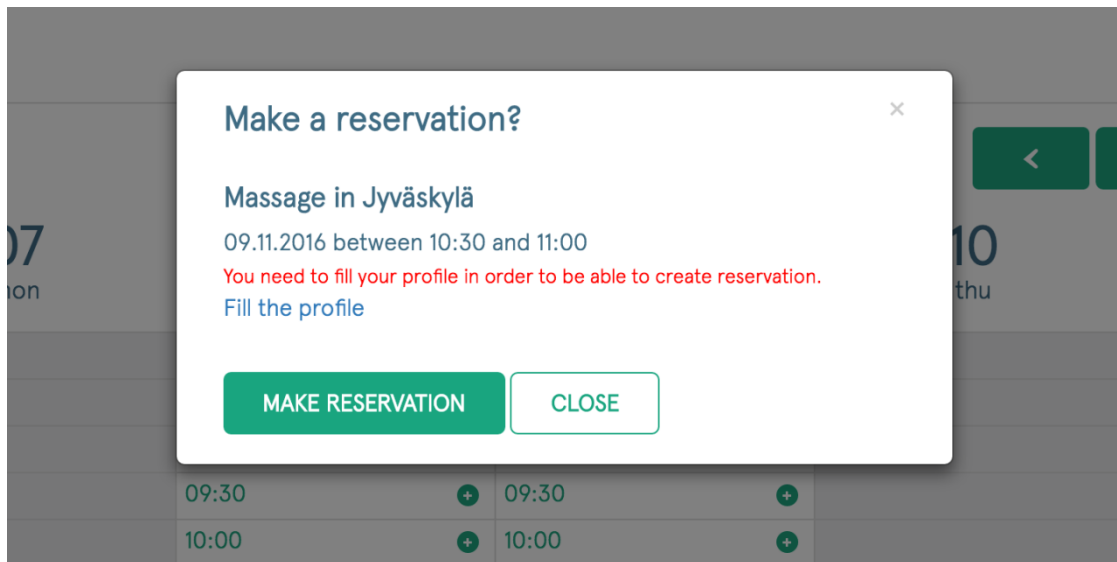


Figure 23 App - reservation confirmation

After providing all the necessary information users are finally able to create a reservation. They can navigate through the calendar using the box containing the button “today” and arrow buttons. Users are not able to navigate to the past. The actual date is distinguished by a different color than the rest of days; it is a small but an important improvement. It is shown in Figure 24 in the Thursday column label.

The white cells with plus sign are used to create new reservations. The occupied times are blue and contain the name of the employee who made the reservation. The gray cells are not available, and the green cells represent reservations made by the logged user. There is a legend under the calendar if any of users does not understand what a specific cell means.

A user can switch between the massage calendar and the ergonomics calendar by clicking the tab with specific service in the left top corner. The content of the calendar is changed based on the selected service. Figure 24 illustrates the most important part of the user interface, the calendar.



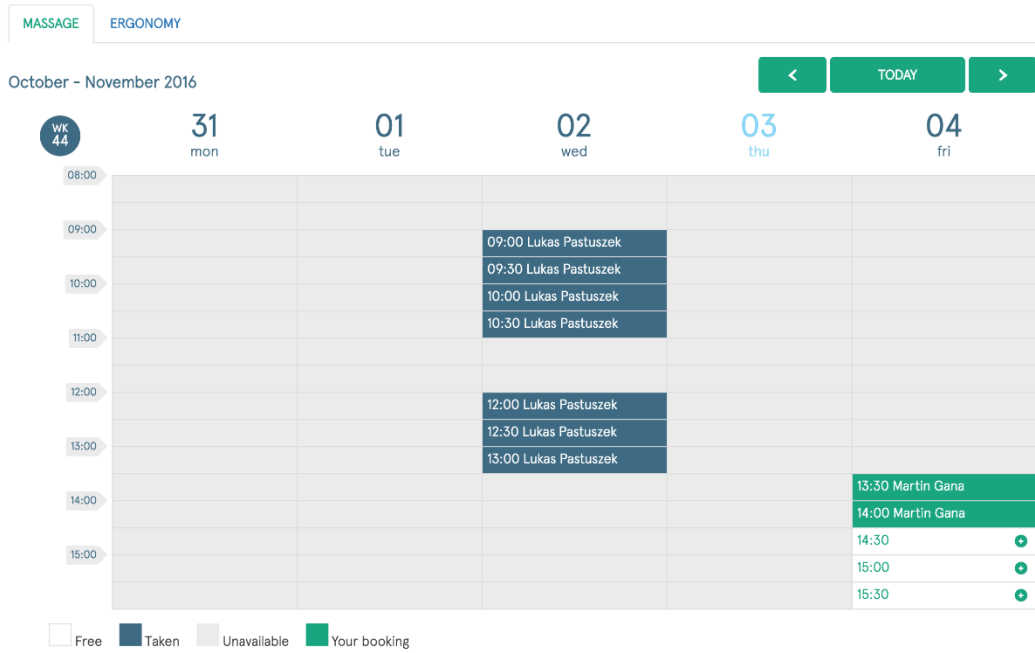


Figure 24 App - calendar

Users can see all their reservations right under the menu, and it is also possible to cancel the reservation there simply by clicking the red trash on the reservation the user wants to remove. It is illustrated in Figure 25.

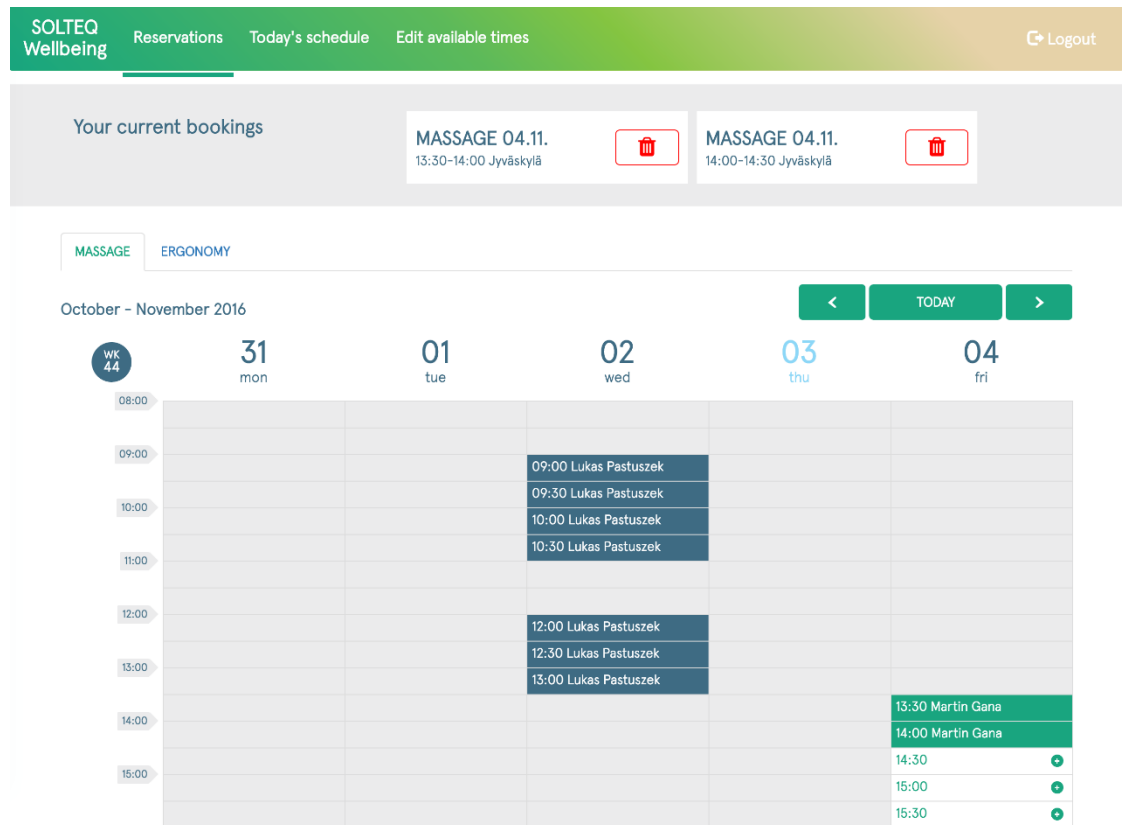


Figure 25 App - calendar with user's reservations

The service provider can set the schedule. The interface for setting the schedule is shown in Figure 26. It looks like a calendar and each cell has a button in it. A user can simply toggle between the enable and the disable options. This feature is available only for service providers, none of regular employees has access to it.

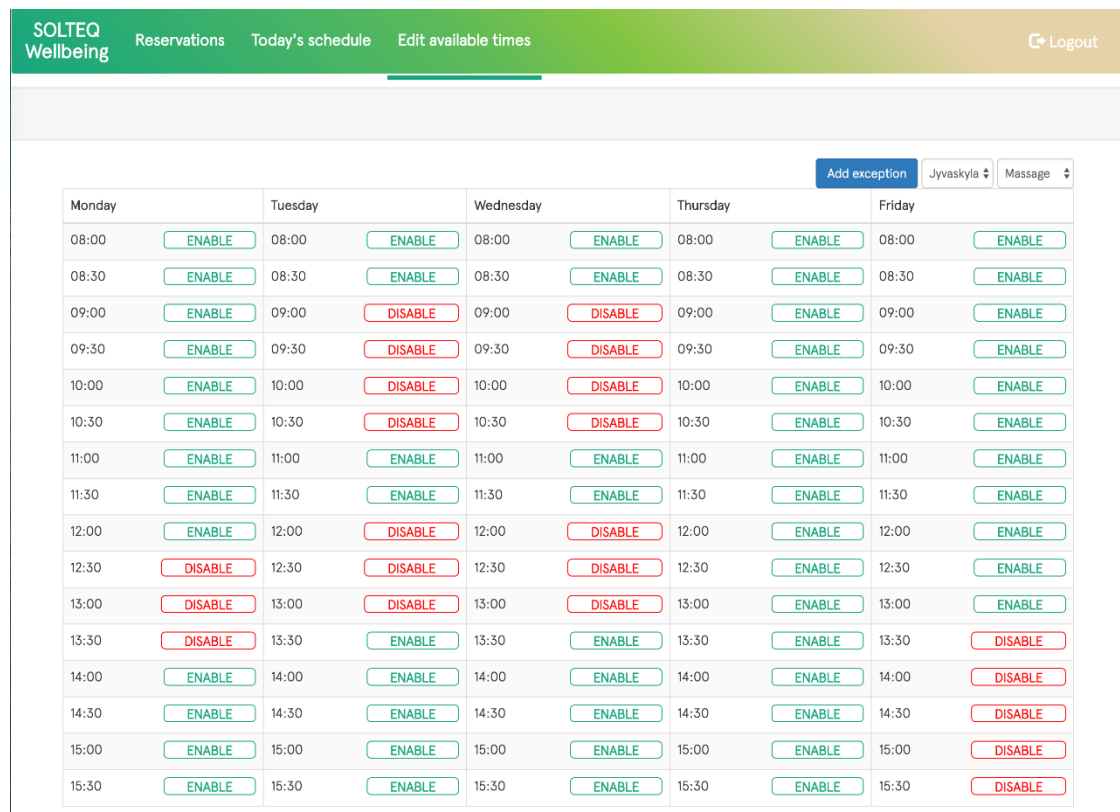


Figure 26 App - setting the schedule

Sometimes there is a need for adding an exception to the regular schedule, for example when the service provider is on a sick leave. The “Add exception” feature was implemented for this purpose. The service provider can select the date and specific service. All times for the date are disabled by default. If the service provider is out of the office just for several hours he can set the available time. It is very flexible and allows service provider to set exception for the specific time on the specific date.

## 9 Conclusion

### 9.1 New reservation system

The application consists of two main user interfaces. The first one is for the service provider. It is possible to set the available times for different services there. It also

allows a service provider to see the list of all reservations and who made these reservations. A service provider can write notes for each session and access these notes later in a form of history. It helps to keep track of an employee's progress when there are some health conditions the service provider helps with. A service provider can also easily set times or dates when it is not possible to have a massage or an ergonomics training, which was not possible in the old system.

The second user interface is used by regular employees interested in booking a massage or an ergonomics training. They have access to the calendar and can see all available times for a specific service. They can reserve a service and after that they receive a confirmation email with an event attached. The event can be easily added to their personal or work calendar.

The new reservation system is already used by Solteq's employees in Jyväskylä office. It has fully replaced the old reservation system. The most important features are implemented and there is a possibility that application will be used also in Tampere, Helsinki offices in Finland and Wroclaw office in Poland.

## 9.2 Proof of concept based on gained knowledge

The knowledge gained by developing the reservation system was used in the development of the proof of concept of a different way to extend the features of bigger systems, specifically a customer's e-shop. There were discussions about a feature which would allow users to select the color of the product they are interested in and the application finds all the products with the similar color.

The light version of this feature was created and added to the e-shop as a demo for a customer. Users can upload an image which is then displayed on the website, after which they select a color from the image and the application will find all similar colors available in the store. The algorithm used for finding the similar color is not important in this case. The important part is how the feature was added to the e-shop. The host website (e-shop) contains a script for loading an external JavaScript application; this application is then executed and loads the HTML content from external resources. The HTML content is then injected to a container which is a part of the e-shop, which allows developers to change the feature functionality without any effect

on the rest of the system. The release of updates is easier because the feature is developed and maintained independently.

### 9.3 Production feature

The customer requested the wish-list feature for the customer's e-shop; the goal was to provide a user with the possibility to save the product on the list so it can be bought later. Another use case for the feature is the list of regularly bought items. A user can create the list and set the quantity for each item. The content of the list can be added to a shopping cart, which is more comfortable than looking for each item every time.

IBM WebSphere Commerce provides the wish list feature; however, there was an assumption that feature would be extended in the future and it would be impossible or at least very complicated to implement other requirements in the wish list feature provided by WebSphere Commerce.

The wish-list feature was not business critical and the estimation was that it would be cheaper than implementing the feature using the WebSphere Commerce so it was decided to build the feature the similar way the proof of concept was. The wish list feature is standalone application which is injected to the e-shop. It has its own code-base and database. It is also easier to maintain and provide an update and fixes because of the size of the application. The wish-list feature is in production use.

### 9.4 Lessons learned

The goal of the thesis was to create a new application which would replace the old reservation system. The application was supposed to be created using new technologies not yet widely used in Solteq. The secondary goal was to explore these technologies and gather some knowledge about the pros and cons of the used technologies which can then be used in different projects.

As a web-developer I gained both technical skills and soft skills. I had an opportunity to gather requirements from the users of the system, some of these users were non-technical people and it was challenging to transform their expectations to software.

The most important skill I learned is software testing. The codebase is widely covered by unit tests. There was also a situation when an automated end to end tests was needed. I learned the importance of refactoring, it is necessary to refactor existing code to keep the codebase maintainable. Refactoring and testing is the essential part of software development although there are many projects with no lack of unit tests which makes refactoring almost impossible which can result in big maintenance problems.

I was expected to write a documentation for the application to make it easy for another developer to get started with developing quickly. The documentation contains also instructions on how to deploy the application to different environments.

Another important skill I improved was the system architecture and Rest API design. At the beginning, I was the only developer working on the project so I made most of the architecture decisions.

## References

About Angular.js. Page on Angular.js Docs. Accessed 04.11.2016. Retrieved from <https://docs.angularjs.org/guide/introduction>

Assertion Testing. Page on Tutorialspoint. Accessed 09.04.2016. Retrieved from [http://www.tutorialspoint.com/software\\_testing\\_dictionary/assertion\\_testing.htm](http://www.tutorialspoint.com/software_testing_dictionary/assertion_testing.htm)

Chai as promised. Page on Github.com. Accessed 24.04.2017. Retrieved from <https://github.com/domenic/chai-as-promised>

Cloudant. Page on Cloudant.com. Accessed 09.02.2017. Retrieved from <https://docs.cloudant.com/>

Cloud Foundry. Page on Wikipedia. Accessed 21.04.2017. Retrieved from [https://en.wikipedia.org/wiki/Cloud\\_Foundry](https://en.wikipedia.org/wiki/Cloud_Foundry)

Databinding. Page on Angular.js Documentation. Accessed 13.04.2016. Retrieved from <https://docs.angularjs.org/guide/databinding>

Express.js. Page on Expressjs.com. Accessed 03.12.2016. Retrieved from <https://expressjs.com/>

HTTPS. Page on Wikipedia. Accessed 03.12.2016. Retrieved from <https://en.wikipedia.org/wiki/HTTPS>

Mocha.js. Page on Mocha official website. Accessed 09.04.2016. Retrieved from <https://mochajs.org/>

Node.js. Page on Nodejs official website. Accessed 15.04.2016. Retrieved from <https://nodejs.org/>

Platform as a service. Page on Wikipedia. Accessed 24.04.2017. Retrieved from [https://en.wikipedia.org/wiki/Platform\\_as\\_a\\_service](https://en.wikipedia.org/wiki/Platform_as_a_service)

Technical Debt. Page on martinowler.com. Accessed 04.11.2016. Retrieved from <https://martinfowler.com/bliki/TechnicalDebt.html>

What is docker. Page on Opensource.com. Accessed 05.09.2016. Retrieved from <https://opensource.com/resources/what-docker>