

Janne Järvinen

Planning System of Work Shifts for Catering Service Company

Helsinki Metropolia University of Applied Sciences

Master's Degree

Information Technology

Master's Thesis

24 April 2017

Author(s) Title	Janne Järvinen Planning System of Work Shifts for Catering Service Company
Number of Pages Date	58 pages 24 April 2017
Degree	Master of Engineering
Degree Programme	Information Technology
Instructor(s)	Heikki Lauranto, Senior Lecturer
<p>The study was carried out for a catering company called Blockfood. Usually a catering company works on a customer's premises or another place the customer has selected, thus there is no constant work place where the employees would arrive. Additionally, most of the employees work as freelance employees and are called to work on request. This kind of work requires in advance done work shift planning to make sure that there are all the needed employees at every customer event.</p> <p>The objective of the study was to make the work shift planning process more reliable and less time consuming. Previously, finding available employees and placing them to a certain date and event was a time-consuming process for the management of the catering service company because it had to be done manually by phone calls or email.</p> <p>The result of the study is a web application for planning work shifts. This helps to find available employees and place them to a certain event. The application was implemented using modern web application development techniques such as AngularJS, Node.js, Bootstrap and MongoDB.</p> <p>The application testing was carried out together with the catering company using predefined test cases. The result of the study is already a usable and final product which can be used to maintain the catering company's customer event information. Since such an application would act as a central point of the company's work there is a need for further development and maintaining since new ideas and features are requested when the application is used on a daily basis.</p>	
Keywords	catering, angularjs, rest, mongodb, html5, bootstrap, node.js

Contents

Preface

Abstract

Table of Contents

1	Introduction	1
1.1	Objective	2
1.2	Verification	3
2	Background	5
2.1	Resource Planning	5
2.2	Order	5
3	Requirements for Software	8
3.1.1	Manager Role	8
3.1.2	Employee Role	8
3.2	Application Flows	8
4	Technology Used to Implement Software	14
4.1	Frontend	14
4.1.1	Bootstrap	14
4.1.2	AngularJS	16
4.1.3	Bower	18
4.2	Backend	20
4.2.1	REST	20
4.2.2	Node.js	21
4.2.3	MongoDB	21
4.2.4	Mongoose	24
4.3	Other tools	27
4.3.1	Git	27
4.3.2	Yeoman	28
5	Application Implementation	30
5.1	Administrator's Application	30
5.1.1	Application Structure	30
5.1.2	Login Screen	32

5.1.3	Main Menu	32
5.1.4	Orders	33
5.1.5	Adding New Order	34
5.1.6	Editing Order	35
5.1.7	Assigning Employee to Event	36
5.1.8	Removing Order	38
5.1.9	Employees	38
5.1.10	Adding New Employee	39
5.1.11	Editing Employee	40
5.1.12	Removing Employee	41
5.1.13	Logging Out	42
5.2	Employee's Application	42
5.2.1	Login Screen	42
5.2.2	Calendar View	43
5.2.3	Event Details	45
5.2.4	Logging Out	46
5.3	Backend system	46
5.3.1	REST API	46
5.3.2	Database	48
6	Verification	50
7	Discussion and Conclusions	55
7.1	Further Development	56
7.2	Used Technologies	57
8	References	58

1 Introduction

Catering services usually work on customers' premises, so in advance work shift planning is one important part to be able to successfully fulfil the customers' needs.

As said, only a small part of the work is done in catering service's own facilities. Typically, food and service products are preconditioned before going to customer. Usually, the customer finds and arranges a place where to have their events such as weddings, funerals, birthdays, and the catering service does the actual work, finishing the food and doing the serving, there.

In addition to work shift planning, event places create some more challenge because they are selected by the customer so they can be located in a wide area, in this case in Helsinki, Vantaa and Espoo. Quite often, festive places might also have some special features for example contact person, door code, kitchen equipment, and this kind of information would be useful to share to the event employees.

In addition to food products, the catering service also arranges the needed staff for the customers' events. Typically, one event requires one cook and one or two waiters. During the season time from May to October there might be several events on a same day, usually on weekends and this makes the challenge of the work shift planning larger because the catering service must find the right employees to right places. Usually employees have a freelancer contract and they are called to work when needed.

1.1 Objective

The goal of this study was to make the work shift planning process more reliable and less time consuming. Currently, finding available employees and placing them to a certain date and event is a time-consuming process for the management of the catering service company because it must be done manually by phone calls or email.

The present study consists of designing and implementing a service for planning work shifts which helps to find available employees and place them to a certain event. The service is to be usable by a mobile device as well as desktop computer so that the employees can report the days when they are available and based on that the management can assign employees to the event. Further on, when the employee is assigned to work he/she can check the event details such as date, address, special requirements from the service.

Figure 1 shows the overview of the resource planning process.

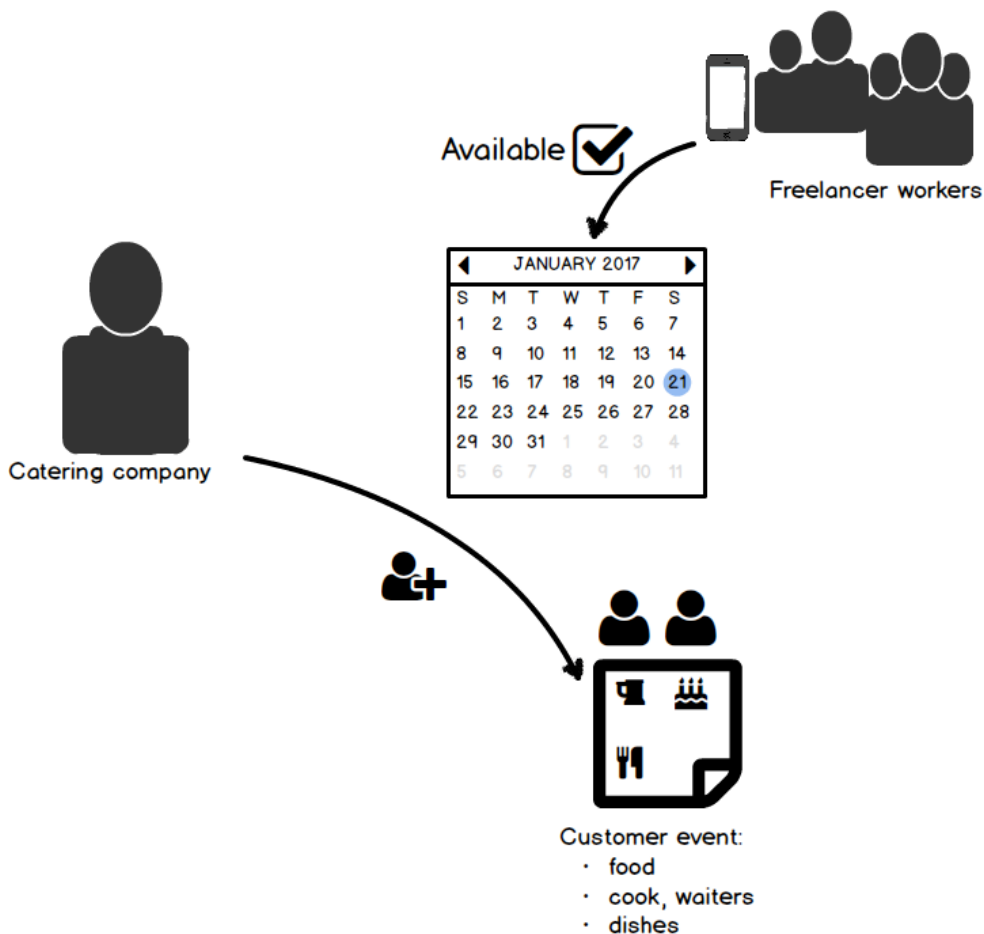


Figure 1. Resource planning of the customer event

The requirements for the solution were collected from the catering service management. The solution should be usable with both mobile and desktop devices. The solution does not need to use any kind of device specific hardware; hence the solution is to be a web application which supports both desktop and mobile displays. Technically, the application consists of a frontend application and a backend application providing a RestAPI (Representational State Transfer) for the frontend. RestAPI is an architectural design pattern for creating platform independent web services. The outcome of the study is a set of several different tools, services and libraries and all of these are explained later in this thesis paper.

To be able to utilize the outcome of the study also in the future it is also required that the produced service would be scalable and easily deployable to other customers and environments.

The aim of the study was, in short, to produce a working service designed using good practices which allows utilizing the created product also in other customers' projects.

1.2 Verification

To explain how the objective was reached this study starts by describing the key technologies and why they were selected. When the technologies had been found, the next phase was to create an application design starting with user roles and use cases continuing to the final solution design which includes the user interface, backend service and database design.

When the application was in a phase where it could be published, a verification was arranged with the catering company. The experiences and findings were used to make possible changes in order to finalize the application to an acceptable level.

This study consists of seven chapters, starting with Chapter 1 which introduces the study and its objectives. Chapter 2 continues with the background information presenting the customer of the study and the actual business problem which was solved in the study. The outcome of this study is software for planning work shifts and Chapter 3 presents the requirements of the software. The created software is a set of several different technologies and they are introduced in Chapter 4. Chapter 5 goes through the implemented software explaining the functionality of the software. The software created in this study

was verified together with the catering company and the results of this verification is presented in Chapter 6. The final chapter lists the results and conclusions of the study together with further development needs of the created software.

2 Background

The study was conducted in the author's own company, Koodipaja. Currently the company is a hobby and the author also has a fulltime work in another company. The motivation for the study became from a Koodipaja customer who asked a possibility to create this kind of service. Another important argument was to create an application which could be sold to other customers and also collecting good practices for this kind of software development.

The customer company Blockfood was founded in 2014 and in addition to the owner itself it has 10-15 freelance employees who are currently invited to work when needed.

2.1 Resource Planning

When a customer event has been agreed on, the catering company starts to plan the event details. The served menu is decided in co-operation with the customer but the resources needed for the event are handled by the catering service.

The challenging part is to find the right number of employees for the events, especially in high season time and the purpose of the present study is to develop this process. Currently the event resource planning requires a lot of paper work and phone calls or SMS message sending.

2.2 Order

The catering company writes an order for the customer event. The order can contain the following information:

- Date and place of the event
- Menu
- The number of guests, adults and children
- Special requirements, like gluten or lactose free guests
- Dishes
- Decorations
- Personnel, cook(s) and waiters
- Time table of the event

- Notes about the event place like door code, contact person, kitchen equipment

An example of a customer order is presented in Figure 2.

<Customer name>	TILAUSVAHVISTUS & KUSTANNUSARVIO <DD.MM.YYYY>
Hääjuhla la DD.MM.YYYY <Event place>	
PATONKIA & PESTOLEVITE ****	
SITRUUNAMARINOITUA LOHTA & PUNASIPULIA ****	
HUNAJA-LIMEKANANPOIKAA ****	
PAAHDETTUA CHILIÄ & MOZZARELLAA & KIRSIKKATOMAATTIA ****	
COUSCOUSSALAATTIA & JUGURTTIKASTIKETTA ****	
VIHERSALAATTIA & VINAGRETTE ****	
TONNIKALAMAJONEESITÄYTTEISIÄ KANANMUNIA ****	
COLESLAW	
NYHTÖPOSSUA & PIPPURIKASTIKE PAAHDETTUA ROSMARIINIPERUNAA WOKKIVIHANNEKSIA	

MANGOJUUSTOKAKKUA PIKKULEIPIÄ KAHVIA & TEETÄ & MEHUA	
	77 aikuista 2 vauvaa 4 bändin jäsentä
Erikoisruokavaliot : Yksi gluteeniton muutama laktoositon muutama ei punaista lihaa & ei äyriäisiä, ei pähkinää	

Figure 2. An example order.

In addition to work shift planning the created application also contains the event information in an electronical format so that the employees can check the needed details with their mobile devices and there is no need to print many copies of the orders. When this order information is provided in a mobile application it is also possible to utilize some online services e.g. showing a map to the event place.

3 Requirements for Software

It is vital for this application to have different user roles because the application contains business data which should only be visible to the management of the company. The following chapters describe the needed user roles and their use cases.

3.1.1 Manager Role

The manager user is the owner of the application and the related business data. The manager publishes the event dates where the employees can register. Based on the information collected from the employees the manager then assigns employees to events.

3.1.2 Employee Role

The employee role is used by the company's employees and they have a limited access to the business data of the application. The application is designed so that employees can use it by their mobile devices. The application is a web application so it is also possible to use it with a normal web browser.

The employees' user interface shows a calendar view where employees can report whether they are available or not. When an employee is assigned to an event he/she can find the event on their calendar view. Employees are not able to see any company sensitive data, such as prices, on their user interface.

3.2 Application Flows

The following tables describe the steps of the different cases of the software to make it easier to understand the requirements. The steps are presented as program flows containing preconditions, user actions and possible postconditions.

Table 1 shows the steps needed to log into the manager's application. When the user navigates to the URL of the manager's application, a log in dialog is shown. When the

user gives a correct username and password he/she is able to enter into the main screen of the application.


Use Case: Log into the manager's application			
Preconditions: Manager has opened the manager's application URL in her browser. The login screen is shown.			
Step #	User action	System state	Details
1	Manager gives her username and password to the login screen		If the username and password are not correct, the login screen is shown again.
Postconditions: When the correct username and password is given, the main screen is shown.			

Table 1. Log into the manager's application

Table 2 shows how the new order is added to the application. First, the user navigates to the Orders page and selects "Uusi keikka". The application shows an empty form where the order information is filled. When all the information is added, the order is saved by pressing "Talleta" button.



Use Case: Add new order			
Preconditions: Manager has logged into the manager's application. The main menu is shown.			
Step #	User action	System state	Details
1	Manager selects Orders from the main menu		
2	Manager selects "Uusi keikka"		
3	Manager inserts the order details	A form for the new order is shown.	Order details includes: event name, place, date and time, contact person and contents of the order.
4	Manager saves the new order		
Postconditions: The systems contain now a date where the employees are needed.			

Table 2. Add new order.

Table 3 presents the steps to assign employees to the order. Employee can only be assigned to an existing order, so the user starts by selecting the order where employees are going to be assigned. If there are employees available they are shown as icons which indicate the role of the employee. To assign the employee to the order, the employee icon is dragged and dropped into the employee area of the order form. Finally, the order is saved by pressing “Talleta” button.



Use Case: Add employee to the order			
Preconditions: Manager has logged into the manager’s application. The system contains a previously created order.			
Step #	User action	System state	Details
1	Manager opens an existing order.	Available employees are shown as employee icons: 	Employee icon indicates the role of the employee. Available roles are: cook, waiter or common employee.
2	Manager drags available employee to the order.	The order view contains an area where employee icon can be dragged to assign an employee to the event.	If the user must be removed from the order, the corresponding icon can be dragged back to available employees’ area.
3	Manager saves the edited order.		
Postconditions: The order contains now the assigned employees and it is ready to be printed.			

Table 3. Add employee to the order.

Table 4 shows the steps to log into the employees’ application. This login functionality is similar to the manager’s application and thus a valid username and password is required to be able to enter to the employees’ application.

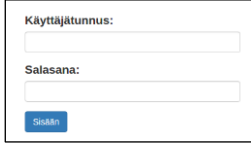
Use Case: Employee logs into the employees' application			
Preconditions: Employee has opened the employees' application's URL in his/her mobile device. The login screen is shown.			
Step #	User action	System state	Details
1	Employee gives his/her username and password to the login screen		If the username and password are not correct, the login screen is shown again.
Postconditions: When the correct username and password is given, the calendar view is shown.			

Table 4. Employee logs into the employees' application.

The employees' application is used for reporting days when the employee is available for the work. These steps are shown in Table 5. The employee selects the dates by touching a day in the shown calendar view. When all the needed dates are selected, the user presses "Lähetä" button which sends the dates into the backend system. Finally, the user logs out from the application by pressing "Uloskirjaus" button.

Use Case: Employee reports the date when he/she is available for the work.			
Preconditions: Employee has logged into the employees' application. The calendar view is shown.			
Step #	User action	System state	Details
1	Employee marks the days when he/she is available to work. The reserved day can be removed by touching the day again.		If there are upcoming events for the current employee, their dates are shown as a different colour.



2	When the dates are selected, the employee sends reported days.		The application shows a message whether sending was successful or not.
3	Employee logs out from the employees' application.		
Postconditions: The system contains now the dates when the employee is available. This employee can be assigned to the selected event.			

Table 5. Employee reports the date when he/she is available for the work.

Table 6 shows how the employee checks if there are any events he/she is assigned to. Such events are shown with a red colour in the calendar view.

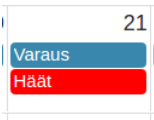

Use Case: Employee checks his/her upcoming events			
Preconditions: Employee has logged into the employees' application of the system.			
Step #	User action	System state	Details
1	Employee checks his/her upcoming events from the calendar view.		If the employee is assigned to the event, the event date is highlighted.
2	Employee logs out from the employee's application.		
Postconditions:			

Table 6. Employee checks his/her upcoming events.

The employee checks the event details by touching the corresponding event in the calendar view, this is presented in Table 7. When the event is selected, the event details are shown in a new dialog.

Use Case: Employee checks the event details			
Preconditions: Employee has logged into the employees' application.			
Step #	User action	System state	Details
1	Employee selects the event date, where she is assigned.		The system shows the order information for the user.
2	Employee reads the order information		
3	Employee logs out from the employee's application.		
Postconditions:			

Table 7. Employee checks the event details.

This chapter described the application flows of the work shift planning system implemented in this study. The application flows shown above were created in the design phase of the system and they were used when the actual applications were implemented. Next the technology used to implement the software is discussed.

4 Technology Used to Implement Software

The outcome of the present study is a combination of several tools and libraries. This chapter introduces these tools and their backgrounds. Common to all of these tools is that they are open source licensed and also platform-independent. Platform-independency is nowadays very convenient since developers and customers might have very different platforms for doing their work.

4.1 Frontend

Frontend in this case means the visible part of the application. Frontend could be for example a web page, mobile application, combination of these or even a desktop application which uses some resources from another system. The application implemented in this study is a web application so the visible part is everything shown in a web browser.

Developing web applications requires several different libraries and frameworks to make the source code maintainable, modular and naturally also usable. The following chapters introduces the used framework which makes the code usable and UI library which makes the application usable and uniform looking.

4.1.1 Bootstrap

Bootstrap homepage gives the following definition "Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web." (getbootstrap.com, 2017). This explains the purpose of the Bootstrap very clearly. It is a widely-used frontend framework for developing modern web applications as well as mobile web applications.

Technically Bootstrap is a collection of CSS-styles and JavaScript functionality which are used to create uniform looking user interfaces. Bootstrap is used in this study together with AngularJS framework but Bootstrap is not limited only to be used with AngularJS. It is also used for example in Asp.Net, Django or PHP applications.

The purpose of the Bootstrap library in this study is to provide a responsive positioning and layout for the user interface. This means that the application is usable with wide

range of screens including mobile devices. Bootstrap is also used to give the application a common and user-friendly interface which feels familiar to the user.

Figure 3 shows a common Bootstrap-style layout which is used in many web applications.

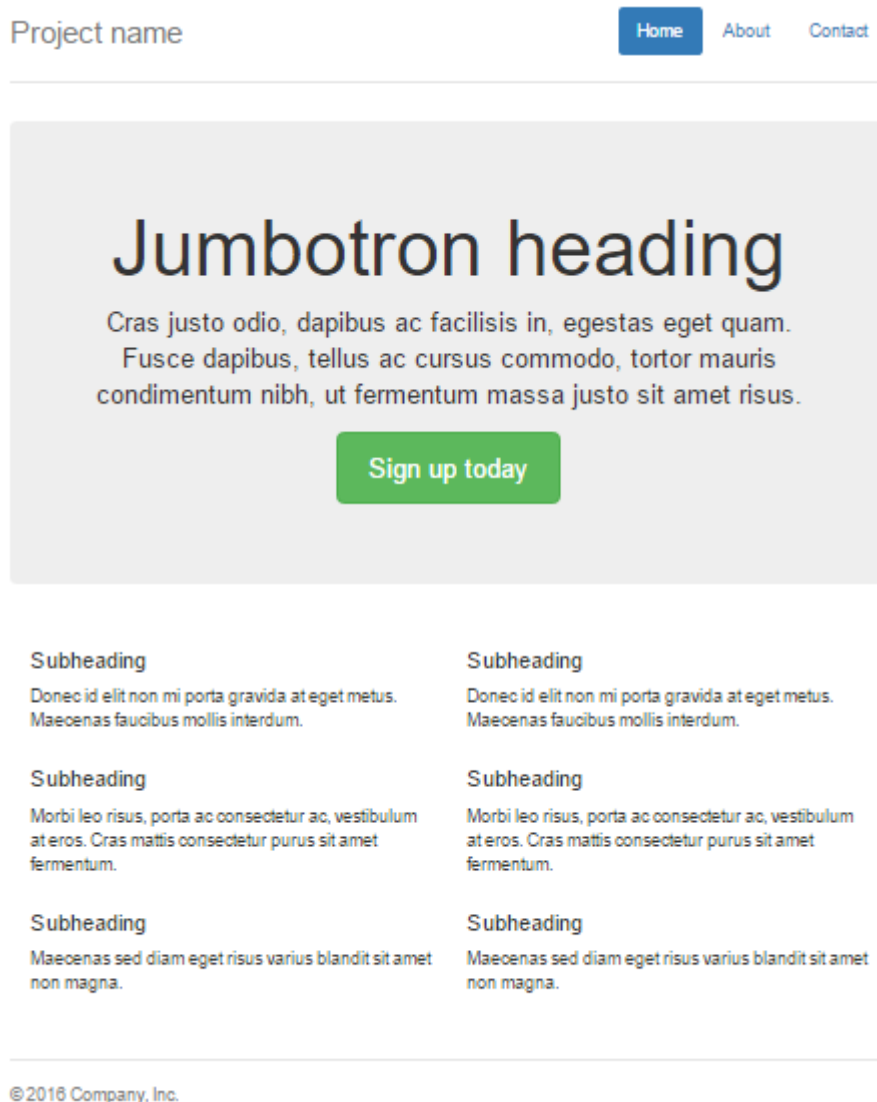


Figure 3. Common Bootstrap layout

The common practice is to start the page with the header section which also contains the name of the company or the application. In the middle of the starting page there can be a summary of the application describing the purpose of the application. This area is called “Jumbotron” in Bootstrap. Bootstrap page is usually ended with a footer section.

In addition to layout helpers, Bootstrap contains also a wide selection of user interface components so that there is no need to use any self-made styles and components. Using Bootstrap's layouts and components make the web-application more user friendly and also look more like real application.

4.1.2 AngularJS

AngularJS is JavaScript framework for creating single page applications. It is maintained and developed by Google.

AngularJS enhances the HTML mark up by its' own directives which makes the resulting web page code readable and maintainable. Applications created with AngularJS follow the MVC (Model-View-Controller) design pattern which is more often used in server side development.

The following example in Figure 4 shows the basic building blocks of the AngularJS application. The example creates a simple shopping list where new items are added by writing their name and pressing "Add new"-button. The list of items is updated by data-binding mechanism of AngularJS and there is no need to refresh the page.

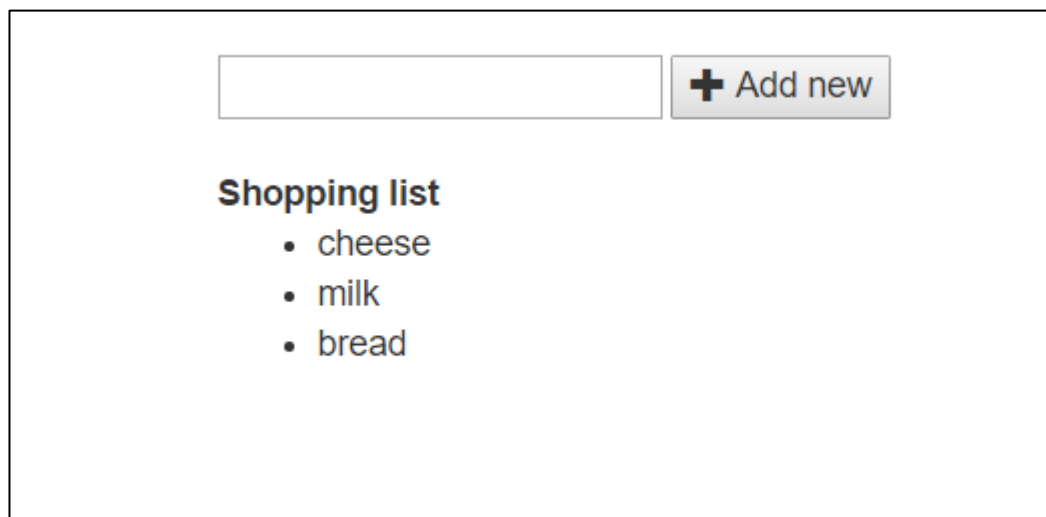


Figure 4. AngularJS example.

The example code presented in Figure 5 contains the following AngularJS elements:

- **ng-app**, Directive marks a web page as an AngularJS application
- **app.controller**, Defines a controller used in a view

- **ng-controller**, Directive tells that an underlying HTML element is using controller functionality defined with directive
- **ng-model**, Defines a databinding to the underlying textbox. The data bound “itemName” is accessed in controller side via AngularJS scope variable
- **ng-click**, Defines a controller function, fired when a connected HTML button is clicked
- **ng-repeat**, Loops an array based data, in this case a list of items added to the shopping list.

```

<!DOCTYPE html>
<html ng-app="myTestBench">
<head>
<!-- Include AngularJS and Bootstrap-->
  <script>
    var app = angular.module('myTestBench', []);

    app.controller('TestController', function($scope) {
      $scope.items = [];

      $scope.addItem = function() {
        $scope.items.push($scope.itemName);
        $scope.itemName = '';
      };
    });
  </script>
</head>
<body ng-controller="TestController">
  <div class="container">
    <div class="row">
      <div class="col-md-12">
        <br/>
        <input type="text" ng-model="itemName">
        <button ng-click="addItem()">
          <span class="glyphicon glyphicon-plus"></span> Add new
        </button>
        <br/><br/>
        <b>Shopping list</b>
        <ul>
          <li ng-repeat="item in items">{{ item }}</li>
        </ul>
      </div>
    </div>
  </body>
</html>

```

Figure 5. Example of AngularJS application

The above example (Figure 5) shows the controller and view combined in a same file but in a real case they would be separated to make the code more maintainable and readable.

The application here is created with AngularJS version 1.5. During the development of this application the AngularJS framework was updated and redesigned to Angular 2.

4.1.3 Bower

As said earlier, the frontend development may require a lot of different kind of JavaScript libraries and other dependencies which speeds up the development work since there is no need to develop all the functionalities by developer herself. These kinds of dependencies can be for example:

- Encryption algorithms
- Date manipulation
- User interface frameworks

The traditional way of including these kinds of dependencies to the application was either to download the local copy of the library or include the source URL of the library to the application's source and use the library from the server of its creator.

Both of these methods require some amount of manual work and when the library is developed further the updated code must be downloaded again to be used in an own application.

To avoid manual work and to handle updates automatically, tools like Bower were developed. Bower is a package manager which handles components and libraries and their dependencies with a more automated way. Bower can handle components such as: HTML, CSS, JavaScript, font and image files. (bower.io, 2017)

The example presented in Listing 1 shows how to install MD5 hash algorithm with Bower:

```
bower install angular-md5 --save
```

Listing 1. Installing MD5 algorithm with Bower

After running the above Bower command the MD5 hash algorithm for AngularJS is downloaded and it is now available to use in AngularJS application.

All the packages downloaded by Bower are saved into the folder “bower_components” which is located in the main directory of the developed application, as shown in Figure 6.

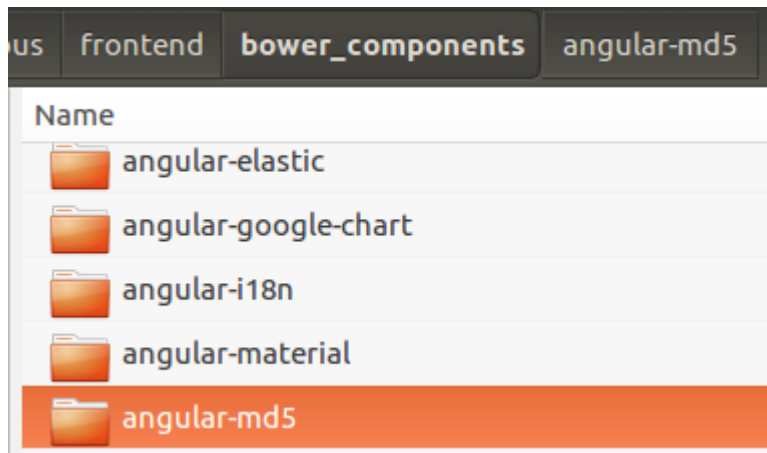


Figure 6. The location of the packages installed by Bower

Since the source codes of the downloaded packages are available for the developer in “bower_components” folder, it is possible to solve errors and other issues by reading the source codes of them.

In the previous example, there was used a parameter “--save”. This parameter makes the Bower to update the local file “bower.json” which contains a list of installed packages and their dependencies. This helps to automate the build process and it is also possible to develop same application with another workstation, since all the needed libraries can be downloaded based on the information contained in “bower.json” file.

4.2 Backend

If the frontend is something which is visible to the application user, the backend is the functionality which occurs in the background, usually in a backend server. This chapter tells how the information is saved and maintained and how it is served to the consumers of the resources.

4.2.1 REST

REST (“Representational State Transfer”) is an architectural design pattern for creating platform independent web services. These services are called application programming interfaces (API) and they are used to expose server resources to be used by external applications. These kinds of external applications can be mobile devices or other web applications, as shown in Figure 7:

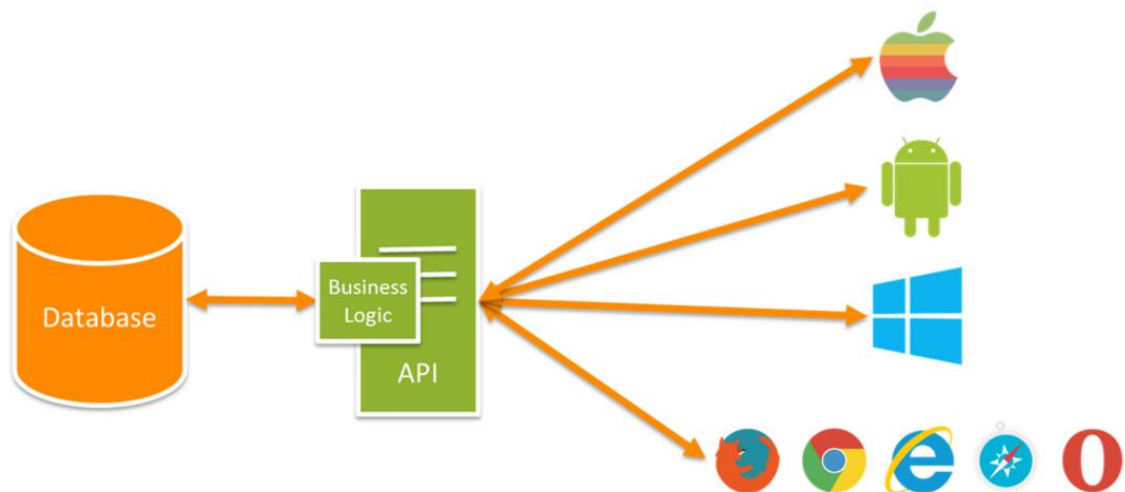


Figure 7. REST API architecture (Kearm, 2015)

Requests made to REST API services are routed based on the used HTTP verb and URL parameters. Figure 8 shows examples on how the employee information is handled by “workers” API:

Resource	Verb	Result
/workers	GET	Returns all the employees.
/workers/1	GET	Returns an employee, whose id is 1.
/workers	POST	Creates a new employee, using the information sent in a HTTP POST request body.
/workers/1	PUT	Updates the employee, whose id is 1. Uses the information sent in HTTP PUT request body.
/workers/1	DELETE	Deletes an employee, whose id is 1.

Figure 8. Different kinds of REST API calls to “workers” resource

With the “workers” API shown on table above, it is possible to get, create, update and delete employee information of the application created in the present study.

4.2.2 Node.js

Traditionally JavaScript has been used only in frontend side of the web application. Previously very common use case where JavaScript was needed was to change the hovered image on the web page. JavaScript was also used to add other, rather simple, functionality to create a real application feeling to the web page.

Node.js is an exception to the traditional JavaScript use case. It is run in a server side instead of a browser. This gives an advantage to the developer because he/she can utilize the same code both in frontend and backend development.

4.2.3 MongoDB

Almost every information system or application has a need to save information and make it available when needed. Very simple way to save information is to save it to the file which could be enough for very simple solution and which is not targeted to multiple users. The more efficient storage for the information is some kind of database where traditional relational databases has usually been used.

One goal of this study was to study modern web application development and thus the selected database system is MongoDB.

MongoDB is a database program which is classified as a NoSQL database. MongoDB differs from the more common SQL database so that it is document oriented and it supports JSON data format. Because of the JSON-style data handling, MongoDB is widely used with JavaScript based backend applications since utilizing the query results is very straightforward.

To illustrate the other differences between MongoDB and regular relational databases, Table 8 compares some of the significant features of very common MySQL database and MongoDB (MongoDB, Inc, 2017).

MySQL	MongoDB
Table	Collection
Row	Document
Column	Field
Joins	Embedded documents, linking

Table 8. Comparison of regular SQL database and MongoDB

MongoDB database does not contain SQL-like query language. Instead queries are formatted as JSON style structures. During the development, some kind of database manager or query editor is indispensable to design and test database queries. The following figures (Figures 9-12) show examples and explanations of MongoDB queries made in the Robomongo database manager.

Find all the documents from the collection

```

1
2 db.getCollection('worker').find()
3

```

worker 0.062 sec.

Key	Value
(1) ObjectId("589f7763d05747e23079db5d")	{ 6 fields }
(2) ObjectId("58addb1373d1139bb0d620d")	{ 2 fields }
(3) ObjectId("5845c9d7e207d8d30afb4842")	{ 6 fields }
(4) ObjectId("587cfa8dba78a1f51474be81")	{ 6 fields }
(5) ObjectId("587cfa52ba78a1f51474be7c")	{ 6 fields }
id	ObjectId("587cfa52ba78a1f51474be7c")
first name	John
last name	Doe
available	[2 elements]
roles	[3 elements]
v	4

Figure 9. Find all the documents from the worker collection

Figure 9 shows a basic example to find all the documents from the MongoDB database. The query does not contain any conditions or field selectors. On the result dialog, there is one employee document opened to show the contents of the returned results.

Find all the documents from the collection, using a condition.

```

1
2 db.getCollection('worker').find({'first_name': 'John'}, {})
3

```

worker 0.062 sec.

Key	Value
(1) ObjectId("587cfa52ba78a1f51474be7c")	{ 6 fields }
id	ObjectId("587cfa52ba78a1f51474be7c")
first name	John
last name	Doe
available	[2 elements]
[0]	{ 3 fields }
[1]	{ 3 fields }
roles	[3 elements]
[0]	{ 4 fields }
rolename	cook
displayname	Kokki
rolevalue	cook
id	ObjectId("587cfa52ba78a1f51474be7f")
[1]	{ 4 fields }
[2]	{ 4 fields }
v	4

Explanation: Finds all the employees, whose first name is “John”.

Figure 10. Find documents, using a condition

As an addition to the previous example, the above query presented in Figure 10 uses now a condition to find all the employees whose first name is “John”. This query is modified on the next example shown in Figure 11 by selecting field which should be returned. Returned fields are listed as a second parameter of the find method.

Find all the documents from the collection, using a condition, but selecting only three fields.

```

1
2 db.getCollection('worker').find({'first_name': 'John'}, {'first_name':1, 'last_name':1, 'available': 1})
3

```

worker 0.063 sec.

Key	Value
(1) ObjectId("587cfa52ba78a1f51474be7c")	{ 4 fields }
id	ObjectId("587cfa52ba78a1f51474be7c")
first name	John
last name	Doe
available	[2 elements]
[0]	{ 3 fields }
day	2017-03-10 00:00:00.000Z
id	ObjectId("58af38df02a585e339b07048")
allocations	[0 elements]
[1]	{ 3 fields }

Figure 11. Find documents, using condition and selecting returned fields.

The last example shown in Figure 12 is a more complicated case where condition is used in a sub-array inside a document.

Find all the documents from the collection, using a condition and aggregate framework.

```

1 var eventDateString = "2017-03-10T00:00:00.000Z";
2 var yesterday = new Date(eventDateString);
3 var tomorrow = new Date(eventDateString);
4 yesterday = new Date(yesterday.setDate(yesterday.getDate() - 1));
5 tomorrow = new Date(tomorrow.setDate(tomorrow.getDate() + 1));
6 db.worker.aggregate(
7   { $match : {} },
8   { $unwind : "$available" },
9   { $match : {
10    "available.day": { $gt: yesterday, $lt: tomorrow }
11  }
12 )

```

0.063 sec.

Key	Value
(1) ObjectId("587cfa52ba78a1f51474be7c")	{ 6 fields }
id	ObjectId("587cfa52ba78a1f51474be7c")
first name	John
last name	Doe
available	{ 3 fields }
day	2017-03-10 00:00:00.000Z
id	ObjectId("58af38df02a585e339b07048")
allocations	[0 elements]
roles	[3 elements]
v	4

Figure 12. Find the documents, using an aggregate framework.

The above query (Figure 12) finds an employee who is available at “2017-03-10”. Since employee document contains an array of days where the employee is available, the MongoDB aggregation must be used. Aggregate operator “\$unwind” loops an array field “available” and uses the last match filter to select event days. To be able to find a certain day, omitting time, it is necessary to compare the selected day to the calculated yesterday and tomorrow.

4.2.4 Mongoose

When database related software is developed, it is usually good practice to use some kind of a database accessing layer between the application and the database. This layer makes it easier to handle regular database calls so that the application developer can concentrate mostly to the application logic. Moreover, this kind of database access layer


abstracts away the database from the application's source level and thus reduces the tight coupling of the source code and database.

Mongoose is such database modelling library designed for MongoDB, the database used in the present study.

Mongoose is a Node.js module, developed for the Node.js backend applications. Like other Node.js modules, Mongoose is installed by npm package manager with the following command:

```
npm install mongoose
```

One benefit of using Mongoose for connecting to MongoDB is that it supports schemas when handling database calls (Mongoose, 2017). Figure 13 shows an example of using Mongoose in the resulting application of this study:



```

Worker.js
1
2 var mongoose = require('mongoose');
3 var Schema = mongoose.Schema;
4
5 var WorkerSchema = Schema({
6   first_name: String,
7   last_name: String,
8   roles: [{rolename: String, displayname: String, rolevalue: String}],
9   available: [{day:Date, allocations: [{order_id:String, rolename: String, displayname: String,rolevalue: String}]}],
10  }, {
11    collection: 'worker'
12  });
13
14 var Worker = mongoose.model("Worker", WorkerSchema);
15

```

Figure 13. Creating the Mongoose schema.

To utilise Mongoose, it is first required to create a schema for the used data model. The above example shows how the schema of the “worker” model is created. “Worker” model has properties such as first name, last name, roles and available days when he/she can work. Roles are used to distinguish whether the employee works as a cook or waiter.

The next step, after defining a schema, is to create an actual data model which is in this case the “worker” model. All the database queries are now made by this “worker” model where the example is shown in Figure 14.

Use a defined data model to create a new document to the database.

```
workerController.js ●
1 |
2 | var mongoose = require('mongoose');
3 | var Worker = mongoose.model("Worker");
4 | var Order = mongoose.model("Order");
5 | var ObjectId = mongoose.Types.ObjectId;
6 |
7 | exports.createWorker = function (req, res, next) {
8 |     var workerModel = new Worker(req.body);
9 |
10 |     workerModel.save(function (err, worker) {
11 |         if (err) {
12 |             res.status(500);
13 |             res.json({
14 |                 type: false,
15 |                 data: "Error occurred: " + err
16 |             });
17 |         } else {
18 |             res.json({
19 |                 type: true,
20 |                 data: worker
21 |             })
22 |         }
23 |     });
24 | }
```

Figure 14. Inserting a new document to the database.

In the above example (Figure 14), the previously created “worker” model is used to create a new document to the database. First the “worker” object is created from the JSON formatted data received from the HTTP request. Then Mongoose method “save” is used to insert this data to the database. In case of error an HTTP error 500 is returned so that the frontend application is able to show the error message to the user. If saving is successful the resulting object containing the created employee information is returned.

4.3 Other tools

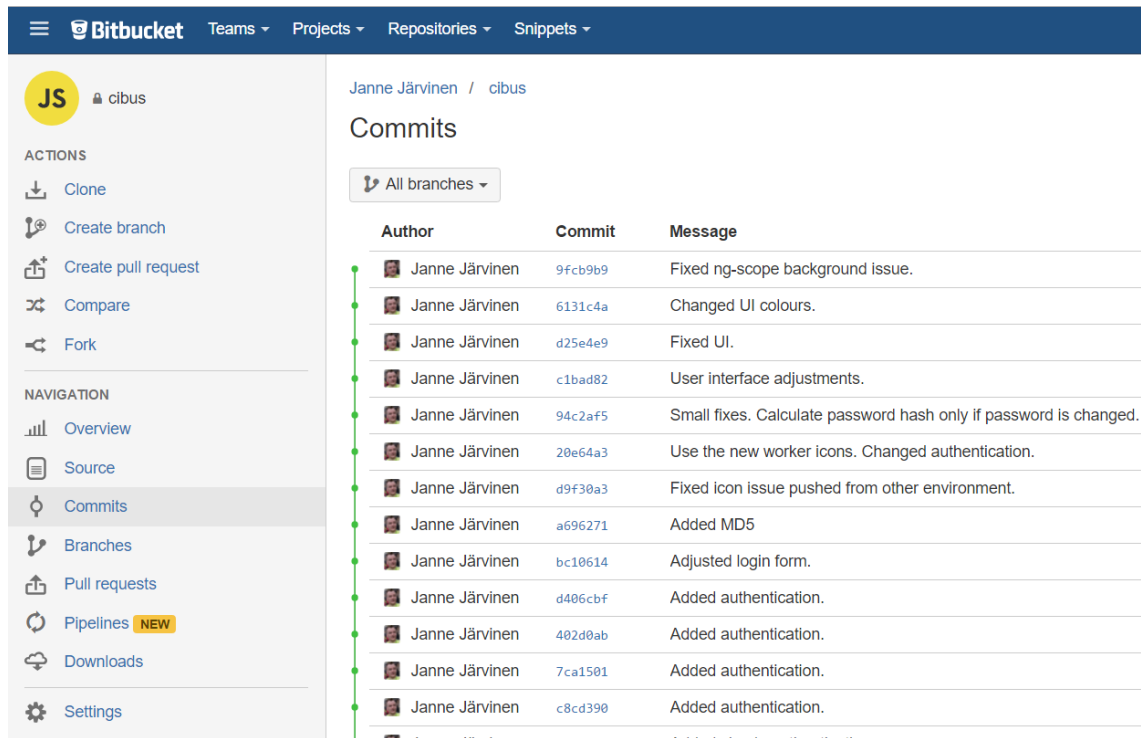
This chapter introduces the other tools used in the implementation of the application created in the present study. The introduced tools were not obligatory for the implementation, but they were used for automating the development process and for saving and maintaining the results of the application development work.

4.3.1 Git

For serious software development, it is necessary to have a system for saving and maintaining the created source code and other files. This system is called a version control system (VCS).

The version control system used in this study was Git which is nowadays widely used and which was familiar because it was used also in the author's daily work. The development of this study was done on both Linux and Windows environments and thus Git was also selected because it works well on multiple platforms.

For hosting a Git repository, there are several cloud services available. For this study, a service called "Bitbucket" was selected because it allows to create private repositories and it is free for small teams (see Figure 15). Like Git as a version control system, Bitbucket has also been used in the author's daily work.



Author	Commit	Message
Janne Järvinen	9fcb9b9	Fixed ng-scope background issue.
Janne Järvinen	6131c4a	Changed UI colours.
Janne Järvinen	d25e4e9	Fixed UI.
Janne Järvinen	c1bad82	User interface adjustments.
Janne Järvinen	94c2af5	Small fixes. Calculate password hash only if password is changed.
Janne Järvinen	20e64a3	Use the new worker icons. Changed authentication.
Janne Järvinen	d9f30a3	Fixed icon issue pushed from other environment.
Janne Järvinen	a696271	Added MD5
Janne Järvinen	bc10614	Adjusted login form.
Janne Järvinen	d406cbf	Added authentication.
Janne Järvinen	402d0ab	Added authentication.
Janne Järvinen	7ca1501	Added authentication.
Janne Järvinen	c8cd390	Added authentication.
Janne Järvinen	...	Added static authentication

Figure 15. The user interface if the Bitbucket service.

Figure 15 shows the example of a Bitbucket's user interface together with the list of the latest commits made to the frontend application of the present study.

4.3.2 Yeoman

To be able to develop software projects efficiently and to avoid repeating same tasks again and again, it is valuable to have some kind of ready-made starting point when the development is started. Usually this kind of starting point means a project template or empty application which contains already some basic blocks of the application.

Depending on the platform where the development is done these kinds of project templates can be created by code editor (IDE) or with some dedicated tool. For instance, when a new project is created with Visual Studio a user selects a project type and the development environment creates an empty application which can already be compiled but which does not yet contain any application logic. In this study, the second option, a dedicated tool is used to reach to the same goal to save time and to avoid starting development without any base.

The tool used in this study is called “Yeoman”. Yeoman is a language independent code generation tool which can be used to create several kinds of projects. Yeoman itself does not contain such a logic but instead it runs external plugins called generators which are used to create a certain type of application project.

Yeoman’s documentation lists possible use cases where the tool can be used (Yeoman, 2017):

- Rapidly create a new project
- Create new sections of a project, like a new controller with unit tests
- Create modules or packages
- Bootstrapping new services
- Enforcing standards, best practices and style guides
- Promote new projects by letting users get started with a sample app

Yeoman contains generators to create for instance the following applications:

- AngularJS
- Ionic
- Chrome extension
- ASP.Net Core
- Django.

From the list above, the AngularJS generator was used in this study. The AngularJS generator created an empty AngularJS application which contained a directory structure, an example controller and view and a basic routing for these. After creating the empty application, the development started by adding the required application logic.

5 Application Implementation

The application designed in the present study is divided into two different parts. The administrator's application is used by the management of the catering company. The employee's application respectively is used by the employees and it is meant to be quite limited and simple to use.

Both of these applications are made with the AngularJS framework.

5.1 Administrator's Application

The previous chapter introduced the different building blocks of the created system. This chapter introduces the different parts of the manager's application and how those parts were developed.

5.1.1 Application Structure

The administrator's application follows the structure shown in Figure 16.

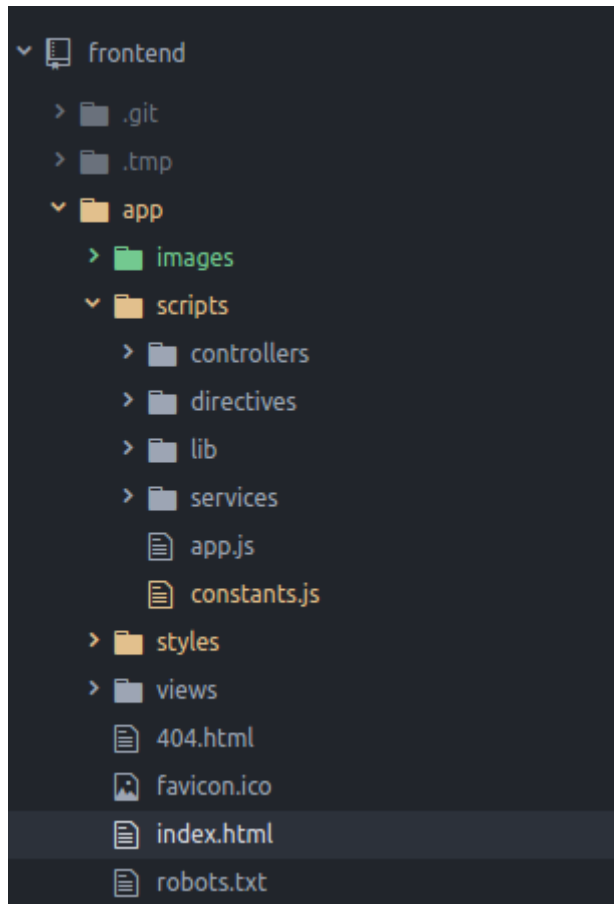


Figure 16. The structure of the administrator's application

The application contains file types such as:

- images
- scripts
- styles
- views

The application structure follows the convention where views and controllers are saved into their own folders. The other option would be to package views and controllers to the folder corresponding to the functionality, like “orders” or “employees”.

Grouping views and controllers into their own folders makes it easier to maintain the similar code and functionalities because files are located based on the technology used on them. In addition to controllers, also other script files like services, are located under the same script folder. When scripts folder contains JavaScript files, styles contains CSS and views HTML files.

5.1.2 Login Screen

To be able to use the application, the user must log into the system first. This is done by giving user name and password into the login screen shown in Figure 17.



The image shows a login form with two text input fields. The first field is labeled 'Käyttäjätunnus:' and the second is labeled 'Salasana:'. Below the second field is a blue button with the text 'Sisään'.

Figure 17. Login screen of the administrator's application

The login functionality is implemented into its own AngularJS service “Authentication-Service” which sends the authentication request to the backend system.

AngularJS services are independent objects which can be used to share information and manage a common functionality inside an application (AngularJS developer guide, 2017). This same approach is used also in other parts of the application when the application communicates to the backend system.

The current implementation uses a basic authentication which is considered to be secure enough during the development phase. When transferring application to the production phase, an SSL should be enabled and configured in a hosting environment.

5.1.3 Main Menu

The starting point of the application is the main menu which is shown after giving a correct user name and password. The main menu contains links for events and employees as presented in Figure 18.

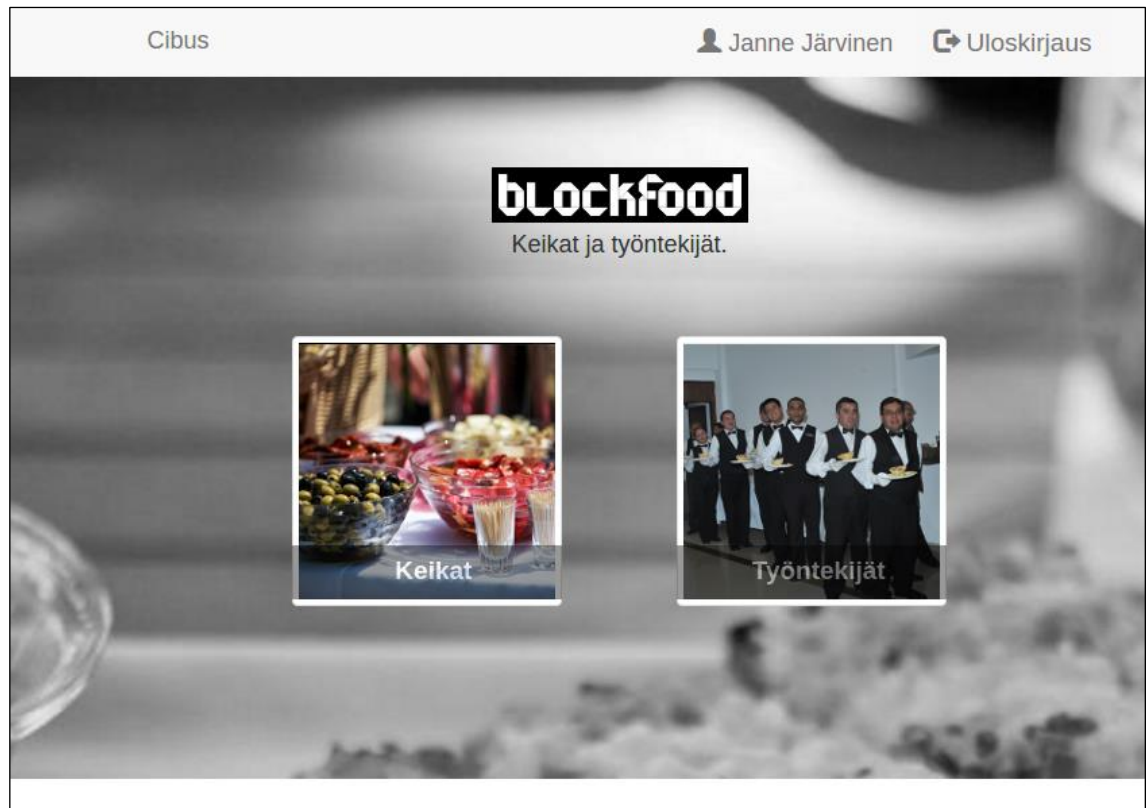


Figure 18. Main menu of the administrator's application

The main menu also shows the current user who is logged in and the link to log out. Depending on the need, the user selects “Keikat” for opening orders or “Työntekijät” for checking employees’ information.

5.1.4 Orders

Figure 19 presents the orders view which shows the orders of the customer events as a list sorted by date. One row contains some basic information e.g. the date and the name of the event so that it is easier to find an order. The order can be opened by clicking the corresponding row. The order line contains also icons for editing and deleting existing orders.

#	Pvm	Kello	Kuvaus	Paikka	Yhteyshenkilö
1	3.7.2017	22.44	Syntymäpäivät	Kisahalli	J Järvinen
2	1.4.2017	11.30	Läksiäiset	Toimisto	Huttunen
3	31.3.2017	15.00	Hääjuhla	Järvenperän VPK, Tallimestarintie 79, 02940 ESPOO	Mäkinen / 040-112233
4	21.3.2017	13.12	Häät	Mestaritali	Komulainen
5	31.1.2017	15.01	Avajaiset	Laaksoalahden vpk-talo	Smith
6	24.12.2016	15.00	Pikkujoulut	Senaatintori	Sopanen

Figure 19. Orders view of the administrator's application

When there is a need to create a new order, it can be done at the orders view by pressing the new order (“Uusi keikka”) button.

5.1.5 Adding New Order

A new order is created by pressing the new order button in orders view. After this an empty order form is shown and the user can insert event details there as illustrated in Figure 20. To be able to distinguish different orders, the user gives name for the order. Usually this is a name of the customer event like “weddings”, “funerals” or “birthday celebration”.

The user also inserts contact person, date and time to the order. Contact person can be for example a caretaker who has a key to the event place or he/she can be one of the customer’s representative.

Event place is inserted to the order and this should be as accurate as possible, so that employees can use their navigators to find the place in case it is not known beforehand.

The actual content of the order is inserted to the "Sisältö" text area. This contains the menu which is ordered by the customer and also other significant information such as the number of guests, possible food allergies and time tables.

The screenshot shows a web form titled "Uusi tilaus" (New Order). The form contains several input fields and a large text area:

- Tapahtuma**: Input field with placeholder text "Keikan nimi".
- Yhteyshenkilö**: Input field with placeholder text "Keikan nimi".
- Paikka**: Input field with placeholder text "Juhlapaikka, jos on".
- Pvm.**: Date input field showing "2017-03-29" and a calendar icon.
- Aika**: Time selection interface with two spinners showing "16" and "40", and up/down arrows.
- Sisältö**: A large, empty text area for entering the order details.

At the bottom right of the form, there are two buttons: a green "Talleta" (Save) button and an orange "Sulje" (Close) button.

Figure 20. Creating new order.

When all the information is entered the order is saved by pressing the save button ("Talleta").

5.1.6 Editing Order

Figure 21 shows how the order is edited. Editing is done in the same way than inserting a new order and when everything is done the edited order is saved by pressing the save button ("Talleta").

Tapahtuma	<input type="text" value="Hääjuhla"/>
Yhteyshenkilö	<input type="text" value="Mäkinen / 040-112233"/>
Paikka	<input type="text" value="Järvenperän VPK, Tallimestarintie 79, 02940 ESPOO"/>
Pvm.	<input type="text" value="2017-03-31"/> 
Aika	<input type="text" value="15"/> : <input type="text" value="00"/> ↑ ↓
Sisältö	<div style="border: 1px solid #ccc; padding: 5px;"> <p> Patonki ja pestolevite Sitruunamarinoitua lohta ja punasipulia Hunaja-limekana Paahdettua chiliä ja mozzarellaa ja kirsikkatomaattia Couscossalaattia ja jugurttikastiketta Vihersalaattia ja vinagrette Tonnikalamajoneesitäytteisiä kananmunia Coleslaw *** Nyhtöpossu ja pippurikastike Paahdettu rosmariiniperuna Wokkivihanneksia *** Mangojuustokakku Pikkuleipiä Kahvi, tee, mehu </p> </div>

Figure 21. Editing an existing order.

After saving the order the application returns to the orders view which shows the orders saved into the system including the recently updated order.

5.1.7 Assigning Employee to Event

The main goal of this study was to create the system for planning work shifts. A very important part of the planning is managing customer orders and assigning employees to them.

When the order is first created to the system it also needs employees. If an employee is registered to be available in the same day as the customer event is arranged an employee's icon is visible in the order editing view which was introduced in a previous chapter.

To assign an available employee to the customer event, the employee icon is dragged from the list of available employees (“Käytettävissä olevat työntekijät”) to the list of assigned employees (“Keikan työntekijät”) as shown in Figure 22. This also works to the opposite direction if the employee must be removed, the corresponding icon can be dragged back to the list of available employees.

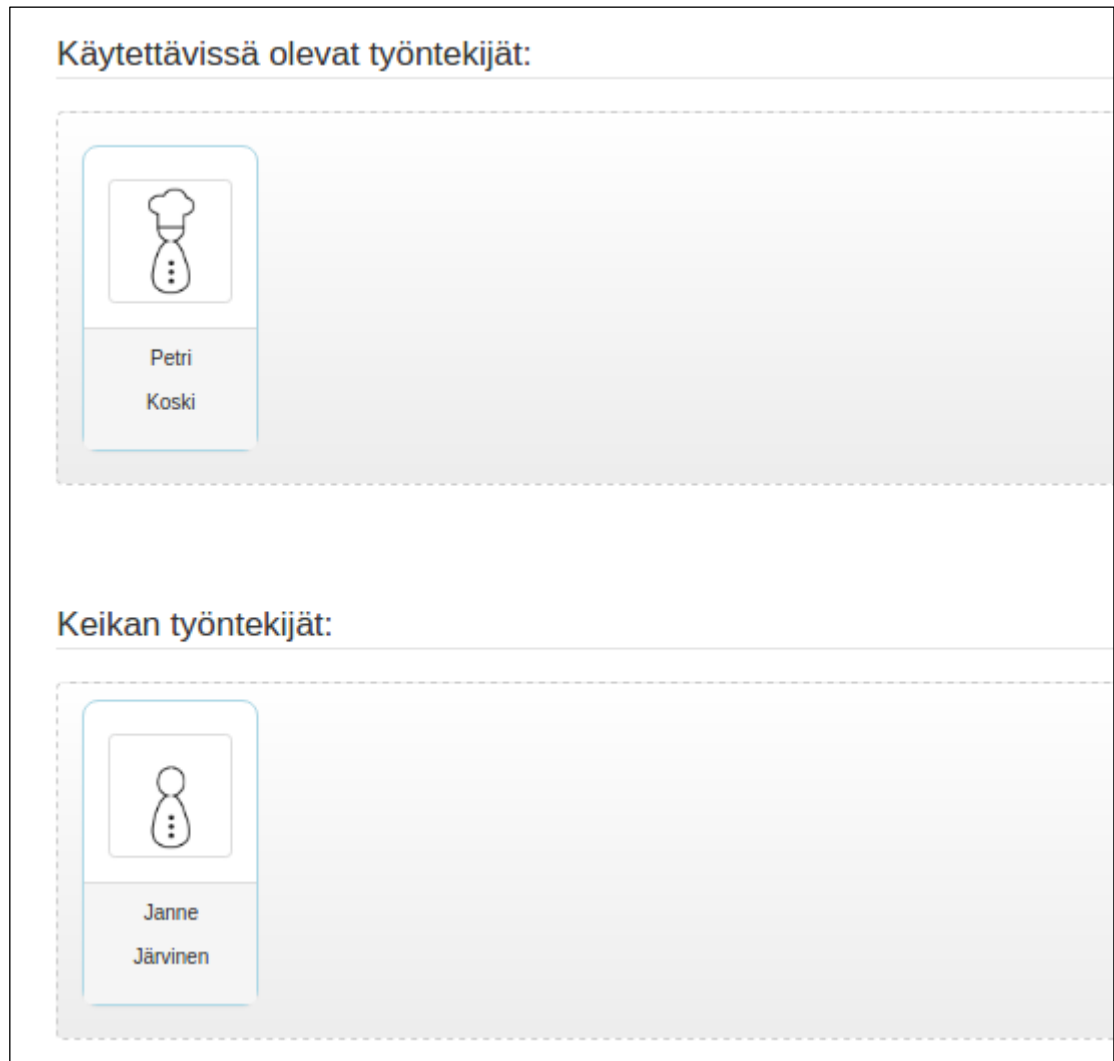


Figure 22. Drag and drop employees to the order.

When the employee is assigned to the customer event the event is shown in highlighted colour on the calendar view of the employee’s application. This way the employee is aware of his/her upcoming work shifts.

Finally, when the needed employees are added into the event, the changes are saved by pressing the save button.

5.1.8 Removing Order

If the order needs to be removed, it can be done from the orders view using the remove order icon. The system does not show any kind of confirmation message box instead a user must press a confirmation button shown next to the original remove icon as shown in Figure 23.

Keikat						
#	Pvm	Kello	Kuvaus	Paikka	Yhteyshenkilö	
1	3.7.2017	22.44	Syntymäpäivät	Kisahalli	J Järvinen	  <input type="button" value="✘Peru"/> <input type="button" value="✔Poista"/>

Figure 23. Confirm deletion of the existing order.

When the order is removed, it will no longer be visible on the calendar view of the employee's application. This way the calendar view is always up to date and employees know exactly when they are assigned to work.

5.1.9 Employees

The list of employees is shown when selecting employees ("Työntekijät") from the main menu. Employees are shown as a list which is ordered on an alphabetical order according to last name as shown in Figure 24.

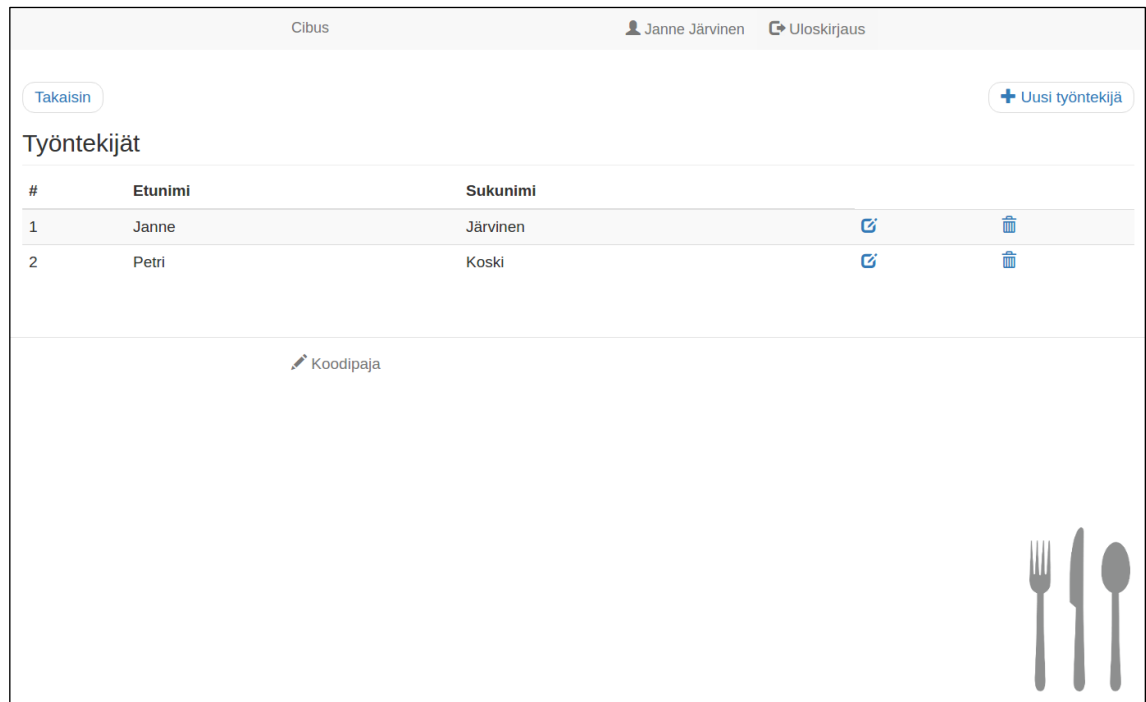


Figure 24. Employees view of the administrator's application.

The employees' view follows the same kind of logic as the previous orders view so that the existing employee is opened by clicking the corresponding row and a new employee is created by clicking the new employee button ("Uusi työntekijä").

The employee can be edited by using the edit icon on the employees view or from the opened employee.

5.1.10 Adding New Employee

A new employee is added at the employees view using the new employee button ("Uusi työntekijä"). The needed information includes first and last name and also the user name and password as shown in Figure 25. The username and password are used when logging into the employee's application.

Cibus Janne Järvinen Uloskirjaus

Takaisin

Uusi työntekijä

Roolit Kokki Tarjoilija Yleis

Etunimi

Sukunimi

Käyttäjätunnus

Salasana

Käytettävissä:

Keikat: **Aika** **Keikka**

Koodipaja

Figure 25. Creating a new employee.

An important property of the employee is the role which tells what kind of work the employee is going to do. Currently the role can be cook, waiter, common employee or all of these. The common employee is usually someone who can for example drive a van which carries food and dishes. These employee roles are then shown together with the employee's name when adding employees to the customer event.

5.1.11 Editing Employee

An employee's information can be edited in the same way as the order by using the edit button in the employee list or in the opened employee view as presented in Figure 26.

Takaisin

Janne Järvinen


Roolit


Etunimi

Sukunimi

Käyttäjätunnus

Salasana


 Kokki


 Tarjoilija



 Yleis

Figure 26. Editing an existing employee.

Employee's roles ("Roolit") can be edited by enabling and disabling roles. When the employee's password ("Salasana") is changed, it is not saved as clear text, instead the system calculates the MD5 hash for the password.

5.1.12 Removing Employee

The employee can be removed by clicking the delete icon on the employees list. The deletion must also be confirmed by pressing a red delete button ("Poista") as shown in Figure 27.



Työntekijät				
#	Etunimi	Sukunimi		
1	Janne	Järvinen		<div style="display: flex; justify-content: space-between; align-items: center;"> <div style="text-align: center;"></div> <div style="text-align: center;"><input type="button" value="Peru"/></div> <div style="text-align: center;"><input type="button" value="Poista"/></div> </div>

Figure 27. Confirm deletion of the existing employee.

If the user does not confirm the deletion of the employee he/she can cancel the deletion by pressing cancel button ("Peru").

5.1.13 Logging Out

When the user has done all the needed work, he/she can log out from the system by pressing log out button (“Uloskirjaus”) on the header line which is always visible.

5.2 Employee’s Application

The second part of this system is the employee’s application which is meant to be used with a mobile device. The employee’s application is used by event employees when they register days when they are available to work. Moreover, employees can check their work shifts and the customer events where they are assigned.

The structure of the employee’s application is similar to the previously introduced administrator’s application. It was also implemented using AngularJS framework.

5.2.1 Login Screen

To be able to use the employee’s application the user must first authenticate. To do this, the system shows the login screen as presented in Figure 28.

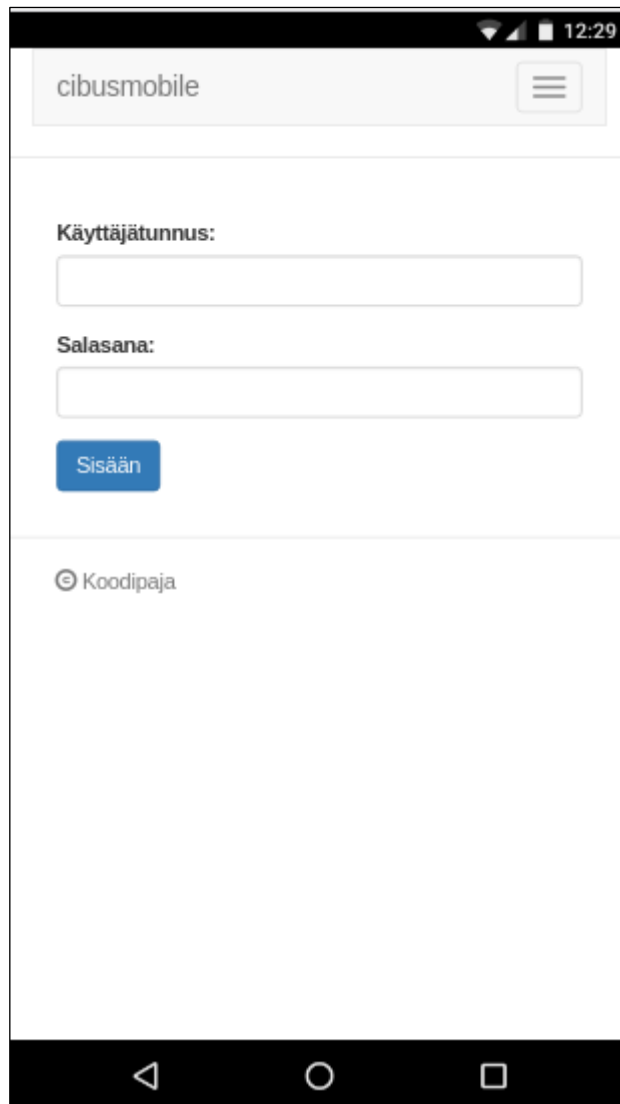


Figure 28. Login view of the employee's application.

The user enters his/her user name and password which were added by administrator in the administrator's application. If the user enters valid user name and password he/she is given access to the application.

5.2.2 Calendar View

After successful login the calendar view, presented in Figure 29, is shown to the user. The user can mark the days when he/she is available to work by touching the corresponding day. The day is then marked as reserved ("Varaus"). The reservation can be removed by touching it again. When all the dates are set in a desired way, they must be

sent to backend system by pressing send button (“Lähetä”). The application shows a message whether the sending was successful or not.

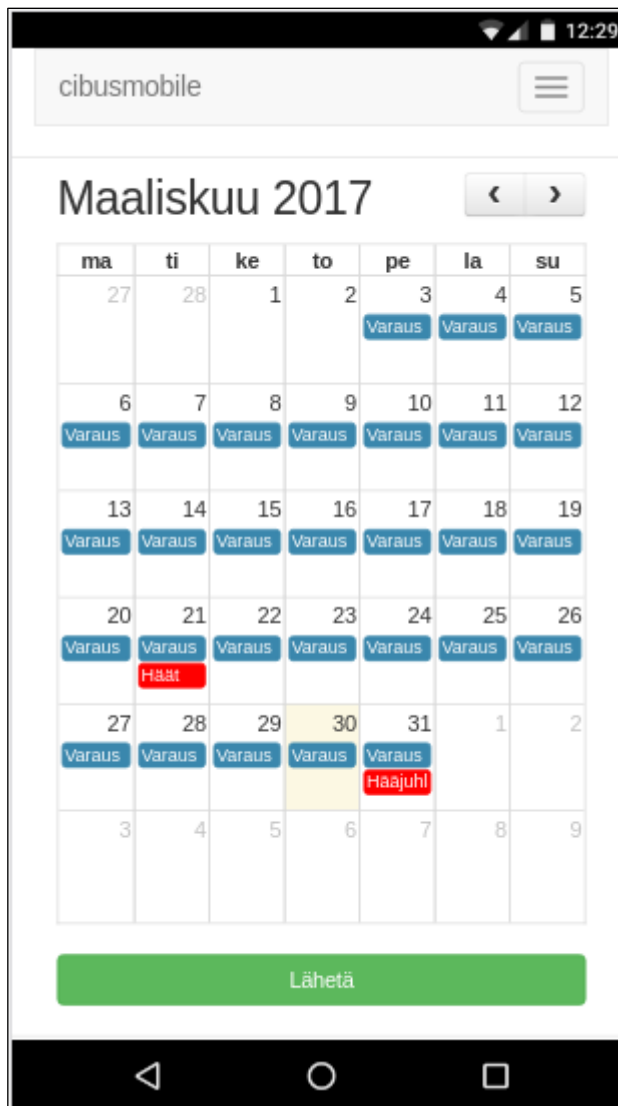


Figure 29. Calendar view of the employee's application.

In case the user is already assigned to some customer event, the event day is shown in the calendar. The event is marked with a red colour and with the event's name. These event days cannot be removed from the employee's application, instead the employee must inform the catering company if he/she has to cancel an already accepted participation.

5.2.3 Event Details

When the user wants to check some upcoming event details he/she can click the highlighted event. Figure 30 illustrates the opened screen which shows the event details where the employee finds the place of the event and the common event information.



Figure 30. Details of the selected event.

When the user has checked the event details the event details screen is closed by pressing the close button on the right upper corner of the screen.

5.2.4 Logging Out

When the user stops using the application, he/she can log out by using logout link (“Uloskirjaus”) which can be found from the header section shown in Figure 31.

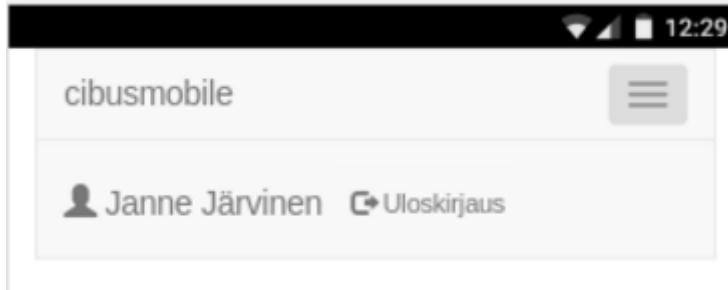


Figure 31. Logout from the employee's application.

Depending on the screen size of the used device the logout link can be always visible or it is located in a dropdown menu in the header section.

5.3 Backend system

As described in the previous chapter, the work shift planning system created in the study consists of two applications: the administrator's application and the mobile application for the employees. Both of these utilise the same information which is provided by the common backend system. This chapter describes the implementation of the backend system.

5.3.1 REST API

Technically the backend system is a REST API service which is implemented as a Node.JS application. The implemented REST API provides the resources for the order and employee views shown in frontend applications.

The following tables (Tables 9-11) show the different resources, provided by the backend system's REST API.

Table 9 presents the authentication resource, which contains only one URL for receiving authentication requests sent as HTTP POST requests.

Authentication		
Resource	Verb	Result
/auth	POST	Receives an authentication request.

Table 9. Authentication API.

Table 10 lists the resources provided by the orders API. Orders API contains verbs for getting, creating, updating and deleting orders.

Orders		
Resource	Verb	Result
/orders	GET	Returns all the open orders.
/orders/id	GET	Returns an order with a given id.
/orders	POST	Creates a new order, using the information sent in HTTP POST request body.
/orders/id	PUT	Updates the order with a given id. Uses the information sent in HTTP PUT request body.
/orders/id	DELETE	Deletes an order with a given id.

Table 10. Orders API.

Employees' information is served from the workers API which is presented in Table 11.

Employees		
Resource	Verb	Result
/workers	GET	Returns all the employees.
/workers /id	GET	Returns the employee with the given id.
/workers /orders/id	GET	Returns employees assigned to the order with the given id.
/workers /available/date	GET	Returns employees who are available on the given date.
/workers	POST	Creates a new employee, using the information sent in HTTP POST request body.
/workers /id	PUT	Updates the employee with the given id. Uses the information sent in HTTP PUT request body.
/workers /id	DELETE	Deletes the employee with the given id.
/workers _dates	POST	Adds dates where the employee is available to work.

Table 11. Workers API used for maintaining employee information.

Similar to the orders API, the workers API contains verbs for getting, creating, updating and deleting employees. In addition, there is a need to make more complicated queries to the employee data. For instance, when the system needs to show the list of the employees of the certain event, it is done using the query “/workers/orders/id”.

5.3.2 Database

The information handled in the system created in the present study is maintained in the MongoDB database. To make database handling more fluent and reliable, the library called Mongoose is used in the backend application. Since MongoDB does not contain any schema handling, Mongoose provides the schema functionality for the application developer.

MongoDB database does not contain tables, like more regular relational database would. Instead of tables MongoDB uses collections to store the data. This backend system contains two collections named “orders” and “workers”. Figure 32 shows the content of one document saved into the “orders” collection:

Key	Value
Objectid("58fa11879de7401400c7164d")	{ 10 fields }
_id	Objectid("58fa11879de7401400c7164d")
event_date	2017-04-20 21:00:00.000Z
event_time	2017-04-21 12:00:37.196Z
event_name	Hääjuhla
contact_person	Mäkinen / 040 123
place	Järvenperän vpk, tallimestarintie 59 02940 Espoo
event_order	Häämenu 1 Juomat asiakkaalta
notes	[0 elements]
assigned_workers	[2 elements]
[0]	{ 7 fields }
rolename	cook
displayname	Kokki
rolevalue	cook
_id	Objectid("58fa10ef9de7401400c7164c")
worker_id	58fa10ef9de7401400c71649
first_name	Maija
last_name	Meikäläinen
[1]	{ 7 fields }
_v	0

Figure 32. Content of the order document.

The order information is saved into the fields matching to the real order document. The relation between the order and the employee assigned to it is made by saving the employee’s id into the “worker_id” field in the “assigned_workers” subarray.

Employee information is saved into the “workers” collection as shown in Figure 33.

(3) ObjectId("58fa10ef9de7401400c71649")	{ 8 fields }
_id	ObjectId("58fa10ef9de7401400c71649")
first_name	Maija
last_name	Meikäläinen
login_id	maiija
login_password	HASH:8d06558afe80cdb1780badd3c211ff53
available	[2 elements]
roles	[3 elements]
[0]	{ 4 fields }
rolename	cook
displayname	Kokki
rolevalue	cook
_id	ObjectId("58fa10ef9de7401400c7164c")
[1]	{ 4 fields }
[2]	{ 4 fields }
_v	1

Figure 33. Content of the employee document.

In addition to regular information fields, “worker” document contains the field “_id” which is created automatically by MongoDB. The “_id” field is used to map the employee to the assigned employee on the order document shown in Figure 32.

6 Verification

The objective of the study was to improve the work shift planning process of the catering company Blockfood. The work shift planning process contains a manual work to contact the company's freelance employees and assign them to the customer events. Previously this was done manually by using phone and email and this required time and effort. To avoid this manual work the work shift planning system was created.

The verification of the implemented system was done with the manager of the catering company since the season time was not yet started. The purpose of this verification was to go through the application and ensure that it fulfils the needs of the catering company and its' work shift planning process.

The verification was accomplished by using the following test cases. The test cases were planned and created beforehand so that all the features of the application were covered. The following test cases contain the needed steps, the data to be inserted and the result of the test case.

The collection of the test cases starts by creating some employees to the system. Table 12 presents the steps required to create new employees. As a result of the test case, there were two test employees created into the system. The role of the first employee is waiter when the second one was created as cook.

Test case 1: Create employees		
Test step	Inserted data	Result
Login to the administrator's application	Käyttäjätunnus: admin Salasana: admin1	Main view is shown.
Select employees ("Työntekijät") from the main menu.		List of employees is shown.
Select new employee ("Uusi työntekijä").	Roolit: tarjoilija Etunimi: Matti Sukunimi: Meikäläinen Käyttäjätunnus: mattim Salasana: matti123	

Save the employee.		New employee "Matti Meikäläinen" is created.
Select new employee ("Uusi työntekijä").	Rooli: kokki Etunimi: Maija Sukunimi: Meikäläinen Käyttäjätunnus: maijam Salasana: maija456	
		New employee "Maija Meikäläinen" is created.

Table 12. Test case 1: Create employees

The next test case shown in Table 13 describes the steps needed to create a new order. As a result of this test case, the new event "Hääjuhla" was created into the system.

Test case 2: Create an order		
Test step	Inserted data	Result
Login to the administrator's application	Käyttäjätunnus: admin Salasana: admin1	Main view is shown.
Select orders ("Keikat") from the main menu.		List of events is shown.
Select new order ("Uusi keikka").	Tapahtuma: Hääjuhla Yhteyshenkilö: Mäkinen/040112233 Paikka: Järvenperän VPK, Tallimestarintie 59, 02940 ESPOO Pvm.: 1.7.2017 Aika: 15:00 Sisältö: Patonki ja pesto-levite. Pulled Pork. Jäte- lövalikoima. Aikuisia 40 Lapsi 5	
Save the order.		Event "Hääjuhla" is created.

Table 13. Test case 2: Create an order

The third test case presented in Table 14 shows how the previously created order document is opened. This test case shows the user how the order document shows only the event information without any assigned employees.

Test case 3: Open event "Hääjuhla 1.7.2017"		
Test step	Inserted data	Result
Login to the administrator's application	Käyttäjätunnus: admin Salasana: admin1	Main view is shown.
Select orders ("Keikat") from the main menu.		List of events is shown.
Select order "Hääjuhla 1.7.2017".		Event "Hääjuhla" is shown. There are not available employees.
Return back to main menu.		

Table 14. Test case 3: Open event "Hääjuhla 1.7.2017"

To be able to add employees to the order created previously, there must be employees who had reported to be available to work on the particular event date. The test case shown in Table 15 presents the steps how the employee reports his/her available dates. As a result of this test case the system now contains the event dates when the previously created employee is available to work. Since the date of the "Hääjuhla" event is included into these dates, the corresponding employee can be assigned to this event.

Test case 4: Set available days for employees "Matti Meikäläinen"		
Test step	Inserted data	Result
Login to the mobile view	Käyttäjätunnus: mattim Salasana: matti123	Calendar view is shown.
Set days to calendar by touching the corresponding day.	1.7.2017 2.7.2017 3.7.2017 4.7.2017 5.7.2017	Selected days are marked on a corresponding day.
Send the selected days by pressing send button.		

Log off from the mobile view.		
-------------------------------	--	--

Table 15. Test case 4: Set available days for the employee "Matti Meikäläinen"

The test case presented in Table 16 repeats the equal steps for the other employee as made in the previous test case.

Test case 5: Set available days for the employee "Maija Meikäläinen"		
Test step	Inserted data	Result
Login to the mobile view	Käyttäjätunnus: maijam Salasana: maija456	Calendar view is shown.
Set days to calendar by touching the corresponding day.	1.7.2017 5.7.2017	Selected days are marked on a corresponding day.
Send the selected days by pressing send button.		
Log off from the mobile view.		

Table 16. Test case 5: Set available days for the employee "Maija Meikäläinen"

The test case shown in Table 17 presents the steps needed to assign employees to the event, which is the main purpose of the system created here. In the present test case the previously created employees are assigned to the event "Hääjuhla". As a result of this test case the event "Hääjuhla" now contains the needed employees.

Test case 6: Assign employees to event "Hääjuhla 1.7.2017"		
Test step	Inserted data	Result
Login to the administrator's application	Käyttäjätunnus: admin Salasana: admin1	Main view is shown.
Select orders ("Keikat") from the main menu.		List of events is shown.
Select order "Hääjuhla 1.7.2017".		Event "Hääjuhla" is shown. Available employees are: Matti Meikäläinen, Maija Meikäläinen

Drag and drop "Matti Meikäläinen" to the event.		Matti Meikäläinen is assigned as a waiter to the event.
Drag and drop "Maija Meikäläinen" to the event.		Maija Meikäläinen is assigned as a waiter to the event.
Save the edited order by pressing save button.		Event "Hääjuhla 1.7.2017" contains now assigned employees.
Log off from the administrator's application.		

Table 17. Test case 6: Assign employees to event "Hääjuhla 1.7.2017".

The following test cases presented in Table 18 and Table 19 show how the employee checks his/her upcoming dates when there is work assigned for them.

Test case 7: Check the event days of the employee "Matti Meikäläinen"		
Test step	Inserted data	Result
Login to the mobile view	Käyttäjätunnus: mattim Salasana: matti123	Calendar view is shown. The event "Hääjuhla 1.7.2017" is shown as a red colour.
Check the event details of the event "Hääjuhla 1.7.2017"		Event details is shown for the selected event.
Log off from the mobile view.		

Table 18. Test case 7: Check the event days of the employee "Matti Meikäläinen"

Test case 8: Check the event days of the employee "Maija Meikäläinen"		
Test step	Inserted data	Result
Login to the mobile view	Käyttäjätunnus: maijam Salasana: maija456	Calendar view is shown. The event "Hääjuhla 1.7.2017" is shown as a red colour.
Check the event details of the event "Hääjuhla 1.7.2017"		Event details is shown for the selected event.

Log off from the mobile view.		
-------------------------------	--	--

Table 19. Test case 8: Check the event days of the employee "Maija Meikäläinen"

As a result of the verification the application was demonstrated to the catering company and the predefined cases were successfully accomplished. During the verification, the new development ideas and needs were found and they were recorded for the further development phase. These new development needs are discussed in Chapter 7.1.

7 Discussion and Conclusions

The whole development process proved that this kind of application which enables the catering company to maintain their customer events and planning work shifts for them would facilitate the catering company's daily work.

Before the development of the application the customer orders were written as Word documents and they were saved locally on the owner's laptop. Employees and their work shifts were maintained manually using phone and email or other messages which required a significant amount of time and effort.

As the result of the study, the work shift planning application was created. The application was deployed into a cloud service provider where it is accessible from the public internet for the catering company and its employees.

The created application is already usable in a daily work of the catering company and thus it fulfils the goal of the study. An advantage of this application is also its platform independence because it is a web based application and can thus be used with a laptop or mobile phone.

During the development and verification, the new ideas and needs were invented and the further development ideas are introduced in the following chapter.

7.1 Further Development

Since this kind of a system would act as a central point of the company's work there is a need for further development and maintaining since the new ideas and features are requested when the system is used in a daily work.

The used technologies make it possible to expand the system with the new features with a reasonable effort. The frontend application follows a model-view-controller design pattern which separates the application logic and the user interface. This eases the development of the new features to the frontend side. Since frontend and backend applications are connected only by the REST interface they can be developed and tested individually.

During the development and verification some new features and ideas were introduced. Most of them were related to the order document which is freely expandable, since MongoDB database does not require schemas. The following list shows some of the new fields and information which would be useful in the order document:

- Billing address of the customer since sometimes the event may be ordered and paid by a different party
- Advance payment which contains the amount if a customer wants to pay beforehand
- Timetable of the event which shows a detailed list of times and steps of the event. This helps to coordinate the catering company's work, like serving and preparing the food on a right time.

Other features which could be implemented as an additional work were:

- To-do list on an administrator's application. This could be used to make notes about upcoming events or other daily tasks
- The event history which shows old events and their orders. Sometimes customers might ask about a menu of some old event if they liked something which they want to order again
- Messaging from the application to employees. The system could send a message to users when there is a new event which needs employees. This way employees could log into their mobile application and report the day if they are available. According to catering company, the email might not be the best choice for this

messaging since the large amount of their employees do not use email very frequently. Instead of email, some kind of instant messaging would be a preferred way. As a further development, there would be a need to investigate if there is any interface available which can be used to send instant messages like WhatsApp messages.

In addition to the further development ideas mentioned by the catering company, the user interface and graphical design could be further developed. The frontend applications were created using Bootstrap styles and components so this will not require a massive effort or refactoring.

7.2 Used Technologies

The secondary objective in the study was to learn and utilise modern web development technologies. Working with technologies such as AngularJS, Node.js, Git and MongoDB was straightforward and efficient. There is also a multiple selection of editors and other tools which support the technologies so the developer can choose the tools which suits best to his/her preference. Atom editor is a good example of an editor which can be expanded by installing modules to add the needed functionalities, such as pretty printing JavaScript source code.

MongoDB on the other hand was a suitable selection for the database since the JSON data format it uses, can be utilized in JavaScript code without any kind of transformations.

Node.js as the backend technology is not yet as common as for example PHP and MySQL. This caused a small challenge when searching a reasonably and clearly priced hosting environment since Node.js applications are not usually hosted on same servers than PHP applications. Finally, the suitable Node.js service provider was found and the application was made available for the catering company.

8 References

AngularJS developer guide, s., 2017. *AngularJS developer guide, services*. [Online]

Available at: <https://docs.angularjs.org/guide/services>

[Accessed 29 03 2017].

bower.io, 2017. *bower.io*. [Online]

Available at: <https://bower.io/>

[Accessed 24 03 2017].

getbootstrap.com, 2017. [Online]

Available at: <http://getbootstrap.com/>

[Accessed 15 01 2017].

Kearn, M., 2015. *Introduction to REST and .net Web API*. [Online]

Available at: <https://blogs.msdn.microsoft.com/martinkearn/2015/01/05/introduction-to-rest-and-net-web-api/>

[Accessed 18 2 2017].

MongoDB, Inc, 2017. *MongoDB and MySQL Compared*. [Online]

Available at: <https://www.mongodb.com/compare/mongodb-mysql>

[Accessed 25 02 2017].

Mongoose, 2017. *Mongoose - Getting started*. [Online]

Available at: <http://mongoosejs.com/docs/guide.html>

[Accessed 26 02 2017].

nodejs.org, 2017. [Online]

Available at: <https://nodejs.org/en/>

[Accessed 15 01 2017].

Scott Chacon, B. S., 2016. *Pro Git*. 2 ed. s.l.:Apress.

Yeoman, 2017. *Getting started with Yeoman*. [Online]

Available at: <http://yeoman.io/learning/>

[Accessed 26 02 2017].

