Mobile order-management system

by the example of K-Con logistics service provider

Moritz Müller

15 credits

Halmstad 2014-06-13
Mobile order-management system
by the example of K-Con logistics service provider

Final project report

Author: Moritz Müller, 871226T539
Supervisor: Essayas Gebrewahid
Examiner: Pererik Andreasson
Abstract

Smartphones and tablets are present in our everyday life. They emerged as an entertaining device but nowadays mobile devices also getting more and more important for business uses. Their big advantage is the light weight and small dimensions. They can be easily used in a train or bus - difficult to imagine with a desktop pc or even a laptop. With business apps employees can do their work in office or out of office.

This thesis deals with the modernization of the order-management system from K-Con. The modernization includes the development of a mobile web application which enables agents to access order-data by smartphone or tablet. The heart of the new order-management-system is a RESTful webservice implemented with the Web API technology. This services provides SSL-encrypted read- and write-access to the order data which are stored in a MSSQL-database. The web-application using this web-service is build with the ASP.NET MVC5 framework and jQueryMobile. An UI-prototype for a desktop application collaborating with the new web-service has been finalized. An agile process approach and prototyping has been used to develop an application that meets K-Con’s requirements.


# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Abstract</strong></td>
<td>I</td>
</tr>
<tr>
<td><strong>Table of contents</strong></td>
<td>II</td>
</tr>
<tr>
<td><strong>1. Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1. Limitation</td>
<td>2</td>
</tr>
<tr>
<td>1.2. Outline</td>
<td>2</td>
</tr>
<tr>
<td><strong>2. Background</strong></td>
<td>3</td>
</tr>
<tr>
<td>2.1. K-Con</td>
<td>3</td>
</tr>
<tr>
<td>2.2. Problem Description and Analysis</td>
<td>3</td>
</tr>
<tr>
<td>2.2.1. Current situation</td>
<td>3</td>
</tr>
<tr>
<td>2.2.2. Planned situation</td>
<td>4</td>
</tr>
<tr>
<td><strong>3. Methods</strong></td>
<td>6</td>
</tr>
<tr>
<td>3.1. Software development methods and tools</td>
<td>6</td>
</tr>
<tr>
<td>3.1.1. Process model</td>
<td>6</td>
</tr>
<tr>
<td>3.1.2. Prototyping</td>
<td>6</td>
</tr>
<tr>
<td>3.1.3. Documentation</td>
<td>7</td>
</tr>
<tr>
<td>3.2. System architecture</td>
<td>7</td>
</tr>
<tr>
<td>3.2.1. Development platform</td>
<td>7</td>
</tr>
<tr>
<td>3.2.2. Programming tools and languages</td>
<td>8</td>
</tr>
<tr>
<td>3.3. Project planning</td>
<td>8</td>
</tr>
<tr>
<td>3.3.1. Milestone planning</td>
<td>8</td>
</tr>
<tr>
<td>3.3.2. Time planning</td>
<td>9</td>
</tr>
<tr>
<td>3.3.3. Sprint planning</td>
<td>9</td>
</tr>
<tr>
<td>3.3.4. Quality planning</td>
<td>9</td>
</tr>
<tr>
<td><strong>4. Design</strong></td>
<td>12</td>
</tr>
<tr>
<td>4.1. Webservice</td>
<td>12</td>
</tr>
<tr>
<td>4.2. Client-server architecture</td>
<td>13</td>
</tr>
<tr>
<td>4.2.1. Server side</td>
<td>13</td>
</tr>
<tr>
<td>4.2.2. Client side</td>
<td>14</td>
</tr>
<tr>
<td>4.3. Storage layer for order-data</td>
<td>16</td>
</tr>
<tr>
<td>4.3.1. Excel-reader proxy</td>
<td>16</td>
</tr>
<tr>
<td>4.3.2. New database system</td>
<td>17</td>
</tr>
<tr>
<td>4.4. Input-mask for desktop-application</td>
<td>18</td>
</tr>
<tr>
<td><strong>5. Implementation</strong></td>
<td>20</td>
</tr>
<tr>
<td>5.1. Webservice</td>
<td>20</td>
</tr>
<tr>
<td>5.2. Responsive web-application</td>
<td>21</td>
</tr>
<tr>
<td>5.3. Database</td>
<td>22</td>
</tr>
</tbody>
</table>
## 6. Testing
6.1. Test driven development ........................................... 24
6.2. Acceptance tests .................................................. 24
6.3. Performance tests .................................................. 25

## 7. Results
7.1. Web application - UI .............................................. 26
7.2. Web application - performance ................................. 27
7.3. Desktop application - UI prototype ........................... 28
7.4. Unit tests - code coverage ....................................... 28
7.5. Code analysis results ............................................. 28
7.6. HTML-Validity ................................................... 29
7.7. Future prospects .................................................. 29

## 8. Conclusions .......................................................... 30

## A. References .......................................................... 32

## B. Appendix ............................................................ 33
B.1. Client contact information ...................................... 33
B.2. Current situation .................................................. 33
B.3. Planned situation .................................................. 34
B.4. Predefined requirements ........................................ 34
B.5. Milestone planning ............................................... 35
B.6. Time planning ..................................................... 36
B.7. Sprint planning ..................................................... 39
B.8. UI-prototypes ....................................................... 39
B.9. Programming tools and languages ............................ 39
B.10. Webservice ........................................................ 40
B.11. Webservice client ................................................ 41
B.12. Responsive webdesign framework ............................ 42
B.13. Hybrid app framework ......................................... 42
B.14. Storage layer for order-data .................................. 43
B.15. UI-prototype desktop application ........................... 43
B.16. Web application - UI ............................................ 44
B.17. Code metric results ............................................. 45
B.18. Acceptance tests ................................................ 45
Chapter 1

Introduction

K-Con is a logistics service provider located in Germany (Kreuzau and Heppenheim) which organises and coordinates the transport of containers by trucks, ships, trains and ferries. K-Con realises container deliveries to Norway, Sweden, Germany, Ireland and Hungary.

As every logistics company, K-Con uses an order-management system to keep track of their business. An order-management system gives information about which container is when and where in use. Furthermore, the system instructs the truck-driver when and where they have to delivery products. The management needs the system to create outgoing invoices as well as for controlling incoming bills. Also, the system serves for optimizing the working time of truck-driver and containers.

This project’s aim is to modernize K-Con’s order-management system by providing centralized access to order-data on desktop and mobile devices like smartphones and tablets. A web-application with responsive views for the devices presents the order-data in an appropriate way. The modernized order-management system should provide a faster and more consistent workflow to handle orders with less faults and misunderstandings between K-Con agents and truck-drivers.

So far all order-data were managed in a single Excel-file by the agents in Kreuzau. The agents working in Heppenheim could download the file from a web-server for read-access. This situation was dissatisfying as agents in Heppenheim could work with stale data. In addition, write-access to the file has to be coordinated by an operator in Kreuzau in order to avoid merging conflicts. In the improved situation, the order-data are stored in a DBMS on a centralized server. Order-data can be retrieved by a RESTful webservice. A web-application with optimized views for mobile devices allows the agents to work outside the office. Simultaneous read- and write access to the order-data is possible.

The ultimate aim in the process of modernizing the order-management-workflow was a centralized order-database available from the web. Operators from Kreuzau and Heppenheim can read/edit the latest data-records as well as adding new ones. As the workflow could not be changed all of a sudden, a stepwise integration was necessary. These steps were considered as the following:

1. Simplify read-access to order-data
2. Store order-data on web-server
3. Provide write-access to new database-system
It was necessary to save the order-data in a centralized data-storage, which can be accessed by different clients (desktop PCs or mobile devices) at the same time. The order-data have to be available online, so that an operator can work independently from office, home or outside. For this reasons, a client server-architecture was chosen.

The server-side is responsible for providing the order-data and managing the concurrent access by multiple clients. Representing the order-data appropriately in a device-specific way is the matter of the different client-sides. The interface between both sides is a web-service offering the order-data in JSON. The web-service can be invoked by a REST-API. For this Web API technology from the ASP.NET MVC5 framework was chosen. In order to exchange the order-data between web-service and clients in a secured way, SSL with a self-signed certificate was used to encrypt the communication on transport-layer. On server-side the order-data are stored and managed with Microsoft technologies like MSSQL and C# as programming language. The web-application uses HTML5, CSS3 and Javascript (jQuery and jQueryMobile) for presenting the order-data appropriately on all devices.

1.1. Limitation

The time and milestone planning for this project considers the steps “Simplify read-access to order-data” and “Store order-data on web-server” of the modernization process to be done until the final project presentation at Högskolan Halmstad. Step 3, providing write-access to the database-system with the new desktop application, is not planned to be realised within this project time. This implementation will take place after the semester outside the context of Halmstad University.

1.2. Outline

The thesis is organized as follows:

- Chapter 2 gives more information about the background of this project, including a detailed description of the customer and the current situation regarding the ordermanagement-system
- Chapter 3 describes the methods which have been used to realize the project. This chapter includes details about the software developing methods and tools, the system architecture and the project planning
- Chapter 4 explains the design decisions and their motivations
- Chapter 5 deals with the implementation of the modernized order-management system based on the previous software design
- Chapter 6 is about how the software was tested and what kind of tests took place
- Chapter 7 presents project results concerning UI- and performance aspects, the code coverage of the unit-tests and findings from the code analysis
- Chapter 8 sums up conclusions and recommendations of used technologies and approaches
Chapter 2

Background

The following chapter gives a brief introduction to the customer and its business. Furthermore, detailed background information about the initial and planned situation will be presented.

2.1. K-Con

The company K-Con is a logistics service provider located in Germany. They organise and coordinate the transport of containers which are carried by trucks, ships, trains and ferries. K-Con realises container deliveries to Norway, Sweden, Germany, Ireland and Hungary.

K-Con has offices located in Kreuzau (near Cologne) and Heppenheim (near Frankfurt) with in total six employees.

In 2013 K-Con realized around 900 orders. The most common target locations were companies in Sweden, France and Norway. K-Con gives service for known companies like M-Real, Stora Enso and Dyckerhoff. Paper bentonite, anhydrite floor and oil-well cement are frequently transported products.

For further information see Client contact information, p. 13.

2.2. Problem Description and Analysis

2.2.1. Current situation

A very common use-case is that a K-Con agent organises a ferry-ticket for a truck-driver who has to deliver the loading from Germany to Sweden. In doing so the truck-driver uses the ferry between Travemünde and Telleborg. The agent is responsible for booking the ferry ticket and entering the booking reference into the order-management system as soon as the ferry-ticket office has sent the booking confirmation. The booking reference also has to be forwarded to the truck-driver, otherwise access to the ferry will be denied.

It can also happen that a truck-driver received the booking number but forgot it or that a forwarded booking reference was invalid. In this case the agents must send the correct booking number immediately. Furthermore, the ship’s boarding time can be late nights or early mornings. That’s why it is advantageous that the agents can access the order-data from mobile device when they are out of office.

In the beginning of the project all order-data were stored in a large Excel-file (xlsx) consisting of multiple worksheets. This file was saved on a computer located in Kreuzau. A scheduled task running on the computer in Kreuzau uploaded the latest version of the Excel-file every two hours onto a web-server. The agents working in Heppenheim could download the file from the web-server for read-access (ftp-transfer).
Changes made on the order-data by agents in Heppenheim had to be synchronized with the order-file in Kreuzau by email or telephone. Figure 2.1 demonstrates the initial situation.

![Initial situation diagram]

Figure 2.1.: Initial situation

The current situation had three major issues:

1. **Excel is required to see order-data**
   The operators at both offices need a PC with Excel installed, so that they can access the order data. If an agent is in urgent need of order-information but out of office, there is no chance for him to access the data. But this can be useful if e.g. a truck-driver is about to check-in to a ferry but forgot the booking-reference (this might occur early mornings or late in the evening). In this case the agent has to get the booking reference as fast as possible and has to send it to the truck-driver by SMS or email.

2. **View of stale data**
   As the Excel-file managed in Kreuzau is uploaded every two hours, it might happen that the operators in Heppenheim are working with an out of date copy of the Excel-file. In worst case, they may give a wrong booking-references or date to truck-drivers.

3. **Merging Excel-file by hand**
   The Excel-file is centrally managed by the operators in Kreuzau. If an agent in Heppenheim needs to make changes to an existing order or want to save a new order, they have to give the information to Kreuzau by email or telephone. This workflow is time-consuming and error-prone as the order-data is entered into the system two times.

As an example, the activity diagram [Current situation](p.33) in the appendix describes the situation when a truck-driver is in need of a booking-reference for a ferry and calls his assigned agent in Heppenheim.

### 2.2.2. Planned situation

The ultimate aim in the process of modernizing the order-management-workflow is a centralized order-database available from the web. This will solve the problem of working with out of date data. Merging new data-records into the database can be done automatically instead of merging data by hand. In addition the problem of inconsistent data will disappear as the database-system handles the data-integrity.

The order-data can be read on a mobile device using a web-application. The dependency to Excel for the read-access will be removed. Agents can use a new desktop application...
to enter or edit order data and send them back to the central database. All in all, concurrent read- and write-access to the latest order-data is possible from both offices. Read-access is also available on mobile devices. Figure 2.2 shows the planned situation.

![Figure 2.2: Planned situation](image)

An activity diagram showing the work flow for forwarding a booking reference to a truck driver in the modernized system can be found in the appendix at Planned situation (p.34).

**Individual software**

All workflows within K-Con are based on self-developed Excel-spreadsheets. Within the spreadsheets, the data are not organized according to any standard. Consequently there can’t be any standard software which is able to process the order-data from K-Con. Standard software for logistic companies like the software suite from LIS (http://www.lis.eu) offers web-based order-management. K-Con decided not to use this software for licensing reasons and missing features. As the self-developed Excel-spreadsheets are serving as data-sources, an individual software has to be created.
Chapter 3

Methods

This chapter describes the development methods that have been used during the project. Concerning the software development, more information about the scrum process model and paper prototyping are given. Furthermore, the chosen programming tools and languages will be explained. The chapter also contains information about the project management like milestone, time and quality planning.

3.1. Software development methods and tools

3.1.1. Process model

The project was realized in an agile manner using the Scrum approach. The aim of the project was to deliver working parts of the final web-application as fast as possible. In the beginning of the project, the requirements of the application were given on a rough level (see Predefined requirements, p. 34). The Scrum approach was chosen because the software could be developed incrementally to what the customer really needs. The fast release cycles allow a quick response to customer feedback.

Scrum is an agile software development framework. While using Scrum the final product will be developed iteratively and incrementally. The work is divided into sprints which will typically last about two weeks. After a certain amount of sprints or sometimes after each sprint a new increment of the software can be released. Thus the customer can use working parts of the software already in an early project stage. On the other side developers retrieve immediate feedback to the software parts that already have been published. That’s how Scrum addresses the challenges of changing customer requirements [1].

Another approach for the process model could have been the waterfall-model. The waterfall-model is marked by a strict, predefined flow of activities within the project. The costs and time-budget are planned at long sight. In order to use the waterfall-model, the requirements have to be available in a detailed level at the beginning of the process. As this was not given in this project, the waterfall-model was not seen as an alternative. In addition, the waterfall-model would not allow the flexibility in the release cycle, as only one ‘big’ release is considered in the end of the project.

3.1.2. Prototyping

Prototyping describes the activity of creating a simple version or parts of the final software product in order to elicit and check the customer requirements and the feasibility of design decisions [2]. Especially paper prototypes help the developers to design user interfaces together with the customer. Paper prototypes are cheap, easy and quickly to produce. They help to avoid misunderstandings in the communication of requirements.

K-Con had no concrete conception and requirements regarding the user interface of the web and desktop application. That’s why the prototyping method was chosen in
order to develop a common idea of the GUI. *Balsamiq mockup* was used as software tool. The results can be found at [UI-prototypes](p.39).

The general idea of prototyping has also been used to evaluate the functionality of the web-application. Each software increment resulted by a sprint provided some kind of prototype to K-Con, which could be tested by them regarding functionality and usability. This kind of prototyping or rather incremental development led e.g. to the implementation of the extended order-search (by start/target-location and container-number) as the usability tests showed up, that the internal order number is not always available for the operator.

### 3.1.3. Documentation

Software is not just a program, it also includes an associated documentation. On the one hand, the documentation helps end-user to understand the software. On the other hand, the documentation is essential for the developers to maintain and extend an existing program [1].

The main tool for documenting this software project is the Unified Modeling Language. Activity diagrams are used to take down the results of the work-flow analysis including the description of the improved situation. The software design is fixed mainly in class diagrams. A package diagram shows the dependencies between the single software parts. Finally an Entity Relation Diagram documents the modelling of the database-layer. *Visual Paradigm* was the program of choice to create the UML-diagrams and ERDs.

Comments within the code were used to describe the method-implementation in more detail than an UML-diagram. The auto-doc functionality from Visual Studio was utilized to created an API-documentation from all classes.

The requirements from K-Con to the web-application are written down as plain enumeration. In addition, the requirements are documented by the test-plan for the acceptance tests. Each acceptance tests describes the steps a user has to execute in the program to come to the defined result.

All documents created during the project are checked in into a local svn-repository. The svn-repository makes it easy to restore previous document states. In addition it is simple to keep track of changes in artefacts. If a revert to an earlier project state e.g. due to changed requirements is necessary, a checkout of a svn-tag is sufficient to continue the project from a specific point in time.

### 3.2. System architecture

#### 3.2.1. Development platform

K-Con owns a virtual server running Windows Server 2008 R2 SP 1. The pre-installed runtime environment is Microsoft .NET-Framework 4.5. Consequently the .NET programming world is chosen as platform for the web-application. Another reason why Microsoft technologies are preferred to Java technologies are my personal skills and knowledge with the Microsoft technologies.

The web-service providing the order-data (hosted on an Internet Information Server) is accessible via REST-API and delivering JSON as response. Consequently any other platforms or technologies (e.g. Java) can make use of it. Although the platform choice on server-side is limited to Microsoft technologies, the client-side is platform independent.
3.2.2. Programming tools and languages

Different programming tools and languages are used to implement the server and client side of the order-management system. As figure 3.1 demonstrates, the server-side is based on Microsoft-technologies while the client-side relies on HTML5, CSS3 and Javascript:

![Programming tools and languages](image)

Furthermore additional tools have been used while the development and project management. **Project 2013** and **scrumdo.com** served as tool for planning and managing the project procedure as well as the scrum approach. The modelling part was supported by **Visual Paradigm** as tool for creating UML-diagrams and database-models. The UI-prototypes were defined with **Balsamiq mockups**.

Detailed information about the used tools and languages can be found at [Programming tools and languages](p.39).

3.3. Project planning

3.3.1. Milestone planning

In order to reach the final goal of the project, the problem was broken into smaller tasks by the top-down approach. In total, three milestones are defined to modernize the order-management:

1. **Simplify read-access to order-data**
   The order-data can be retrieved on a smartphone or tablet using a web-application. The dependency to Excel for the read-access will be removed.

2. **Store order-data on web-server**
   The order-data will be centrally stored on a web-server instead on a local PC in Kreuzau. The problem of working with out of date data will be solved. Merging new data-records into the database can be done automatically instead of merging data by hand. In addition the problem of inconsistent data will disappear as the database-system cares about the data-integrity. All in all, the data-quality will be increased.

3. **Provide write-access to new database-system**
   A new input-mask/form is necessary so that agents can enter or edit order data from mobile devices and a desktop pc.

The checklists with the criteria to reach the milestones can be found at [Milestone planning](p.35).
3.3.2. Time planning

The time planning was done with Project 2013. The time effort for each task was estimated based on experience from the apprenticeship and finished projects within the study. A certain time buffer has been added to each task in order to compensate unpredictable delays, e.g. due to bug-fixes or other implementation-issues. Project 2013 was selected as planning software as it supports the management of dependent work-packages as well as the automatic creation of Gantt-charts for a visual overview of the project process. The project process can be found as task-list and Gantt-chart in [Time planning](p.36).

3.3.3. Sprint planning

The management of sprints and user-stories within the scrum process was realized using the online-tool available at [http://www.scrumdo.com](http://www.scrumdo.com) (see [Sprint planning](p.39)). Scrumdo.com was chosen because it offers user-story and sprint-board management for free. The scrum or sprint progress can be shown in different kind of diagrams. An alternative of using scrumdo.com would be Project 2013, as the time planning is done with it anyway. But Project 2013 has no adequate build-in features for managing user-stories and creating a virtual scrum-board.

The following sprints were defined to reach the first milestone **Simplify read-access to order-data**:

1. **Create web-service for retrieving order-data**
   The web-service has to search for the requested data in the Excel-file. The matching order-data will be transferred back to the requester.

2. **Create web-application using the web-service**
   The web-application is the UI presented to the operators. It will use the web-service to request and display order-data.

3. **Extend web-application to use device-message-functionality**
   Retrieved order-data can be forwarded to another person (e.g. truck-driver) by using the email- or SMS-functionality from the smartphone or tablet. The web-application needs access to the contacts stored on the mobile device.

3.3.4. Quality planning

The quality planning describes the measures which will be conducted to ensure a certain quality level within the project. The quality aspects are divided into two groups - process and product quality.

**Process quality**

**Defining milestones**
By defining milestones, the project management can easily check if the project is on track. A milestone specifies which documents or other artefacts have to be produced and reviewed until a certain point in time. If a milestone shall be reached, all documents have to be approved by the customer within the scope of a set-actual comparison. When a milestone is not reached, the project management can take actions to bring the project on track again.

**Time-planning with buffer**
The project time-planning includes time buffers for each task. Consequently, occurring
problems and delays can be absorbed by this buffer so that delivery at the defined point in time is still possible. If the time buffer is not needed, additional tasks can be included into the milestone or an extended refactoring can be done.

**Product quality**

**HTML5-code validation**
All produced html-webpages have to be in accordance with the current W3C-draft of the HTML5-standard. The webpages will be checked after the last sprint by the using the validation-tool available at http://validator.w3.org. For HTML5-compliance-feedback during the development, the FireFox-plugin “HTML Validator” available at https://addons.mozilla.org/en-US/firefox/addon/html-validator can be used.

**JavaScript validation**
All webpages have to be inspected regarding JavaScript-runtime-errors. JavaScript-errors can be detected with the debugger-console in FireFox, Internet Explorer, Chrome, Opera and Safari. A final check will be made by at the end of the last sprint.

**Cross-browser compatibility**
The web-application has to be compatible with the following five major browsers:

- Firefox (Version 27)
- Internet Explorer (from version 10)
- Chrome (Version 32)
- Opera (Version 19)
- Safari (Windows version 5.1.7)

**Unit-tests**
Unit-tests have to be written within the test driven development approach. For each class-library project a test coverage of 100% has to be aimed.

**UI-tests**
Basic UI-tests will be implemented using Selenium. The concrete test-scenarios will be defined for each milestone.

**Compiler warnings**
All warnings given by the C#-compiler have to be checked and corrected if reasonable. A final check will be done at the end of the last implementation sprint.

**Code analysis check**
The code analysis functionality build in Visual Studio 2012 has to be applied on the whole solution in order to find suspicious code. The code-check will be run with the “Microsoft Managed Recommended Rules” configuration.
**Code metric check**
The code metric functionality build in Visual Studio 2012 has to be applied on the whole solution in order find suspicious code. The following basic conditions are defined to accept the code:

- Maintainability-index: at least 70
- Cyclomatic complexity: at most 10 per method
- Depth of inheritance: at most 5 per class
- Class coupling: at most 20 per class
Chapter 4

Design

This chapter describes the main design decisions that were made during the project. Important decisions on server side concern the web service technology in the context of a client server architecture. Design decisions about the responsive webdesign and hybrid app framework are part of the client side. Other decisions described in this chapter deal with the storage layer and the input mask for the new desktop-application.

4.1. Webservice

The web-service provides the order-data to the outside world in JSON-format. As the web-service is hosted on an Internet Information Server, it is realized with .NET technologies. Microsoft offers two technologies for implementing web-services: Windows Communication Foundation and Web API. The requirements the framework to use had to fulfil were:

- Data-transmission has to be encrypted
- Authentication: Client must login with username/password before retrieving data from the web-service
- Authorisation: A authenticated user has to be authorised to load order-data within the current session

<table>
<thead>
<tr>
<th>Requirement</th>
<th>WCF</th>
<th>ASP.NET WebAPI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Encrypted data-transmission</td>
<td>Offers encryption via HTTPS/Certificate</td>
<td>Offers encryption via HTTPS/Certificate</td>
</tr>
<tr>
<td>Authentication</td>
<td>Authentication possible via Username/Password, Certificate and others</td>
<td>Authentication possible via Username/Password, OAuth, OpenID</td>
</tr>
<tr>
<td>Authorisation</td>
<td>Authorisation possible via Username, Certificate, Windows and others</td>
<td>Authorisation possible via Username/Password, OAuth, OpenID</td>
</tr>
</tbody>
</table>

The comparison in figure 4.1 shows, that both technologies are suitable for implementing the web-service. However, Web API appears to be more lightweight and faster to use than WCF. WCF offers a lot of handy features, but they are not needed in this project. Consequently a spike solution using Web API was created in order to evaluate the feasibility. The spike solution contains a web-service that returns Mock-object. Before data can be retrieved, the client has to provide username and password. In addition the data-transmission is encrypted by a self-signed SSL-certificate.
The spike solution proved that Web API is sufficient to implement the order-data webservice. The WebMatrix/Simple membership-provider from the ASP.NET MVC package can be used to handle the login-logic (authentication, authorisation and timeout-handling). A login-controller will return the so called Authorization-Key to the client after a successful login. This key has to be included by the client in each request. IIS allows to restrict the access to an url on SSL-connections only. Consequently the access to the REST-API can be set up only on SSL-encrypted connections. A self-signed certificate is sufficient in this case as the application will be used inside the company only. There is no need for an official certificate.

4.2. Client-server architecture

K-Con wants a centralized data-storage for their order-data, which can be accessed by different clients at the same time. Examples for different clients are desktop PCs or mobile devices like smartphones or tablets. The order-data have to be available online, so that an operator can work independently from office, home or outside. For this reasons, a client server-architecture was chosen.

The server-side is responsible for providing the order-data and managing the concurrent access by multiple clients. Representing the order-data appropriately in a device-specific way is the matter of the different client-sides.

As figure 4.2. shows, the interface between both sides is a web-service offering the order-data in JSON.

4.2.1. Server side

As soon as a client is logged in at the KCon.Ordermanagement web-application, the user is also automatically authenticated at the KCon.Ordermanagement web-service. A OrdermanagementWebservice-instance (see Webservice client, p.41) represents this connection with the webservice. As the web-application uses plain HTTP-requests to communicate with the webservice and these are by nature stateless, the OrdermanagementWebservice-instance has to be persisted and reloaded during all upcoming http-requests against the webservice (session-management).

If the OrdermanagementWebservice-instance would not be saved, a new login-request is necessary on every request to the KCon.Ordermanagement web-service. This is time consuming and resource-wasting.

![Figure 4.2.: Client server architecture](image-url)
Two approaches were considered for solving this problem:

- Save serialized OrdermanagementWebservice-instance per user/client in HTTP-session on IIS

- Use an own thread-save singleton class, which will store the OrdermanagementWebservice-instances from all clients in a concurrent dictionary. Key is a client-identification, the according value the class instance.

A serialized OrdermanagementWebservice-object is about 5500 bytes large (measuring taken by BinaryFormatter). Assuming ten operators working concurrently with the web-application, this means a storage need of 55,000 bytes (approx. 53.7 kb) in the session-management. This storage need can be served by the IIS without any problems.

An advantage of using the HTTP-session provided by IIS is the automatic recycling of old sessions. As soon as a http-session is timed out or logged out, the used storage for saving the OrdermanagementWebservice-instance will be deallocated automatically. When implementing an own class managing the OrdermanagementWebservice-instances, this clean-up functionality has to be programmed from scratch.

4.2.2. Client side

Responsive webdesign framework

Every page in the web application has to be implemented responsively in order to be usable on different smartphones and tablets. Designing a responsive web-application means working with relative measure units instead of absolute pixels. The main work of implementing a fluent page-design can be done by available web-frameworks.

The main requirement which the framework has to meet is the possibility to create a fluent column design which adapts to the available screen width. The chosen framework also has to support the following UI-elements (according to UI-prototype):

- Button
- Loading panel
- List view
- Sub-list-view or accordion
- Form-controls
- Popup/Notification

The web-pages build with the framework have to be usable in the latest versions of the following browsers/OS:

- Internet Explorer / Windows Phone 8
- Firefox / Android
- Chrome /Android
- Safari / iOS

The last requirement which the framework has to fulfil is the support of theming. The look-and-feel must be easy to customize.

The choice of considered frameworks depended on the research done within the book “Mobile ASP.NET MVC 4” [3]. This book describes the following frameworks as helpful tools while implementing responsive web-design:
• Bootstrap
• Zurb Foundation
• jQuery Mobile
• Sencha Touch
• Kendo UI Mobile

After the first selection round there were still two frameworks to come into consideration (Zurb Foundation and jQuery Mobile). The other frameworks were discarded for the following reasons:

**Bootstrap**: Firefox on Android is not supported ([http://getbootstrap.com/getting-started](http://getbootstrap.com/getting-started) Browser and device support)

**Sencha Touch**: Entirely JavaScript-base, needs Java and Ruby on developer machine, longer training period

**Kendo UI Mobile**: Can only be purchased in Telerik package, additional costs

The comparison between Zurb Foundation and jQuery Mobile can be found here ([Responsive webdesign framework](p.42)). jQuery Mobile has been selected as framework for a spike solution as it offers an integrated AJAX-page-navigation system which is activated by default. With this navigation system page transitions and progress bars can be used out of the box. Furthermore jQuery Mobile is developed by the jQuery-project-team, so there are a lot of similarities to the jQuery-framework. As I’m already familiar with jQuery, jQuery Mobile was the framework to choose for the spike solution.

The spike solution based on jQuery Mobile fulfils all requirements. In addition jQuery Mobile came up with a huge bunch of useful widgets. The API-documentation is sufficient to write working code in a short time. Also jQuery Mobile can be easily integrated in existing HTML-code by using customized ‘data’-attributes in tags.

**Hybrid app framework**

The web-application used so far has to be able to access the email/text messaging program on any mobile device. This is necessary as the operators want to forward reference-numbers or other data from the order-view directly to truck-drivers or other persons. Instead of copying the number into the clipboard, leaving the application and opening an email program on the device, there has to be an option to open the email program with a predefined text-content. That’s why the web-application had to be turned into a hybrid application.

The framework for creating such a hybrid application has to fulfil the following requirements:

• Must support iOS, Android and Windows Phone
• Must provide access to the email-functionality of the device
• Must provide access to the text-messaging-functionality of the device
• Must provide access to the dialing-functionality of the device
• Has to be free of charge
There are several frameworks available, which offer compiling hybrid-apps on a HTML/CSS/Javascript codebase for different platforms:

- Telerik AppBuilder
- Ionic
- PhoneJS
- Appcelerator Titanium
- PhoneGap

The research showed, that only PhoneGap is fulfilling all requirements (see Hybrid app framework, p.42).

A deeper research in the improvements of HTML5 resulted in an alternative solution. HTML5 added some new link types like `sms` and `tel`. These link types were invented for mobile devices to allow interaction from web applications.

- `sms` link type allows to open a sms-dialog with predefined text-content
- `tel` link type allows to start a phone call to given number
- `mailto` link type allows to open an email-dialog with predefined text-content and subject

This three link types are well supported on all OS except Windows Phone. Windows Phone does not interpret the `sms` link type. But there is an Windows Phone App available which mimics the `sms` link type handling (unique costs per Windows Phone device: 0,99 €).

The three link types have been tested with a spike solution on all mobile devices within K-Con. There were no compatibility problems except the sms link on Windows Phone. Consequently the alternative solution using the new HTML5 link types has been applied. The application could stay a web application. That means the use of PhoneGap for creating a Hybrid App was needless.

Another advantage of keeping the application a plain web application: The app distribution can be done by just opening the URL in a browser on the device. In case of creating a Hybrid App, the distribution has to be processed by using the OS dependent stores (Google Play, App Store, Windows Store). Uploading an app to these stores bear the consequences of additional costs (e.g. 400 € for Windows Store per two years). In comparison to this costs, the amount of 0,99 € per Windows Phone device has been accepted by K-Con.

### 4.3. Storage layer for order-data

#### 4.3.1. Excel-reader proxy

A controller in an ASP.NET MVC application will be created on every request by default. Thus a controller is stateless.

Interpreting and reading the order-excel file into memory takes about two seconds. Doing this for every request would slow down the web-service performance dramatically. Hence some kind of cache which keeps the state of the excel-file in memory is necessary.

One thing that has to be considered is, that the excel file will be updated on the server every two hours. Consequently the cache-mechanism has to check, if a newer excel-file is available on the server. If a newer version of the excel-file has been uploaded the cache-mechanism has to renew its content.
The considered solutions to solve this problem were:

- Make the ExcelReader-class a singleton instance. Extend class-functionality with a method to check, if a newer excel-file is available and then reload the excel-content.
- Create a singleton-proxy-class for the ExcelReader-class. This class holds a reference to an ExcelReader-instance for delegation. In addition this class checks, if a newer excel-file is available. The ExcelReader-class doesn’t need to be extended.

Adding the caching-functionality to the ExcelReader-class would lead to a conflict with the SoC-principle. The ExcelReader-class is meant to retrieve order-data from the excel-file only. Caching and lifecycle management is not the responsibility of this class. Thus a proxy-access-class will be used which cares about the caching and the lifecycle management. The ASP.NET MVC controller will work with the ExcelAccessProxy-class instead of the ExcelReader-class. Both classes are defined here (p.40).

4.3.2. New database system

The order-data were stored in an Excel-file so far. The Excel-file consists of multiple worksheets which references each other. The order-sheet contains a link to the address-sheet because each order has a starting and target address. The address-columns in the order-sheet contain short-names of the address. E.g. the entry “DLuV” in the order-sheet refers to the row “Dietsch Logistik und Verkaufsberatung” in the address-sheet. Unfortunately there are no foreign key constraints in the Excel-file. Some users enter “DLuV”, some “dluv” and some “DL und V” to point to the same address. Consequently the data becomes inconsistent over the time.

The Excel-file works perfectly if only one user accesses the data at the same time. Although Excel offers a multi-user-mode, writing to the file concurrently often ends in a data-mess. Especially as the file is transmitted to the office in Heppenheim via FTP.

This are the reasons why a new database-layer was necessary in order to come up with the requirements of the modernized order-management system. The database-layer had to fulfil the following requirements:

- Efficient way of data-storage
- No redundant data in storage
- Consistent data-storage
- Data are queryable
- Concurrent read/write access

The considered solutions for this problem were a relational database and a document-oriented database. A relational database because it is meant to model relations between data-records. A document-oriented database could also solve the problem as it models the data as they were written on a paper document. In this case a paper containing all order-information that is kept in a office-folder.

As the server is already using Microsoft technologies, SQL Express 2008 R2 is the representative for the relational database while CouchDB is the database of choice for the document-oriented approach.

The SQL-database was chosen for this project (see Storage layer for order-data, p.45). Many data are reused in different orders. E.g. the address of a discharging station can occur in a lot of recurring orders. In a document-based database these data would be stored redundantly in every order-document. When the address is changing, all documents containing this discharging station have to be updated. In addition the document-based approach cannot guarantee that the order-documents are pointing
to the same discharging address as foreign keys are not considered in the concept of CouchDB.
All in all a SQL-database is superior for this use case as it offers a consistent and non-redundant data-storage by transactions, referential integrity, relations and foreign keys.
Figure 4.3 shows the entities and its relations that are considered to be stored in the database.

The central entity is the order-entity (‘Auftrag’). In this entity the core-data like order-reference, start- and end-date, state and weight are saved. An order-record has references to other entities in order to avoid redundancy. Thus there are foreign keys to the container, product, start- and target-location as well as recipient and employer.
The single steps to do in an order are modelled in the task-entity (‘Aufgabe’). Per task-entity start- and end-date as well as state and task-type are stored. Again there are references to the start-and target location and the creditor.
The address-entity (‘Adresse’) contains the contact-information of a person, company or location. The type-attribute indicates of what kind of ‘contact’ an address is. This entity hasn’t any references to other entities, but is referenced by the order- and task-entity.
Basic information about the containers used in an order and the products that are delivered can be found in the according entities. Both entities are not dependent on other entities, but referenced by the order-entity.

4.4. Input-mask for desktop-application
At K-Con order-data are mostly entered and edited on a desktop computer. On mobile devices a read-only access is sufficient so far. That’s why the focus of the modernization of the order-management system includes the development of a new desktop-application. This application communicates with the order-management web-service to retrieve and save the data on the server. The last part in this project was to develop a prototype for the user interface of the new desktop application. Realizing the prototype and implementing the desktop application is not part of this project.

Paper prototyping was used to develop a sketch of the new user interface together
with K-Con’s employees. In a first step the main partitioning of the main window could be easily evaluated by creating different sketches on paper. After creating and discussing a raw sketch (see UI-prototype desktop application, p.43), the partitioning of the window was defined as shown in figure 4.4.

Figure 4.4.: Detailed sketch desktop UI

Finally a prototype realized with the Windows Presentation Foundation was created to get a feeling for the look and feel of the later user interface. In addition this last prototype was built to get an idea of the final navigation system in the application. As K-Con’s employees are used to work with the newest Word- and Excel-versions, a ribbon-widget is used for the navigation (see Desktop application - UI prototype, p.28).
Chapter 5

Implementation

This following chapter deals with the implementation of the planned software. In detail, problems and solutions while realizing the web service, the responsive web-application and database tests are explained.

5.1. Webservice

The web-service represents the interface between client- and server-side as the order-data are transmitted by a web-service instance. In order to decouple the web-service and client-side from the domain-classes, data transfer objects have been defined as a data-exchange-contract (see, p. 40). Changes in the domain-layer doesn’t affect the DTO-layer directly. Consequently the client-side is loosely coupled to the domain-layer. The data transfer objects are poco’s (plain old clr objects) which are not dependent on other project-assemblies.

While implementing the Excel-persistence layer, that means programming the logic to find and readout an order which is spread other different worksheets in the Excel-file, the developer’s attention was called to the binding of the web-service and Excel-persistence layer. It was known that the Excel-file will be replaced by a database system in a later project stage. In order to keep a loose coupling between web-service and persistence-layer an interface was defined (‘IPersistenceService’). This interface describes the methods the web-service excepts from any persistence-layer. The ExcelAccessProxy-class implements this interface, thus web-service and Excel-persistence layer can work together. The new database-layer in a later project stage has to implement this interface as well. The whole composition can be found at, p. 41.

As the web-service implementation is at compile-time only familiar with the interface-definition of the persistence-layer, the injection of a concrete implementation is necessary at runtime. The StructureMap-library is used in this project to care about a proper dependency injection. By a xml-file the mapping from interfaces and concrete implementations can be defined. Listing 5.1 shows the set up for the Excel-persistence-layer.

```xml
<DefaultInstance
  Key="ExcelAccessProxyTestEnvServer"
  PluginType="WebService.Contracts/IPersistenceService"
  Scope="Singleton"
/>
```

Listing 5.1: Setup dependency injection

At runtime StructureMap creates an ExcelAccessProxy-instance and injects this object where ever a IPersistenceService-instance is needed. StructureMap also takes care that only
one ExcelAccessProxy-instance will be created. This causes a huge performance improvement as the Excel-file will be read into memory only once. If the ExcelAccessProxy-class wouldn’t be treated as a singleton, the Excel-file would be parsed on every request that is made to the webservice (see Excel-reader proxy p.16).

During the implementation of the web-service, a suggestion for performance improvement rose up. So far the whole content of the excel-file is held in the IIS-memory. Each time a client makes a request, the order-object is assembled by the data from the memory. A performance improvement would be to store the whole loaded order-object in a concurrent dictionary (key is the order-number). If a client is searching for an order which has already been loaded by another client, the webservice can reuse the object kept in the dictionary. The time for building up the order-object is saved.

When an updated excel-file is loaded into memory, the dictionary/cache of course has to be emptied. As the ExcelAccessProxy-class (see ,p.40) is a proxy class for the access to the excel-data, this would be the perfect place to implement an extended caching-logic. The effect of this performance-improvement can be followed at Web application - performance (p.27).

5.2. Responsive web-application

jQueryMobile was used for implementing a responsive web-application those user interface automatically adapts to the viewport of the current mobile device. Working with relative width-specifications is a major aspect while realizing a responsive UI. Figure 5.1 shows how different grid- and column-specifications respectively are used to make the UI adapting to the available width:

![Figure 5.1.: Grid-setting in web-application UI](image)

The different column-widths can be set up by using predefined CSS-classes provided by jQueryMobile.

In section Hybrid app framework (p.15) the solution for accessing the internal message-functionality of the mobile device is described. The according implementation in HTML is shown in figure 5.2.

21
Listing 5.2: Implementation HTML link types

The body-attribute of the mailto- and sms-linktype are used to predefine the text-content of the message. In doing so the web-application is dependent on the mobile operating system interpretation of these link types. As the tests showed that all mobile operating systems process the link type in the desired way (Window Phone needs an addon for appropriate behaviour). The mailto-linktype also accepts a subject-attribute to force the subject of the email.

Subject and message-content are encoded before they are passed to the HTML-output. The encoding ensures that special characters are represented as HTML entities. Characters like '?' or the German umlauts can be inserted into the text-modules without any problems.

5.3. Database

After modelling the database-structure in Visual Paradigm, the SQL-code for creating the database-tables could be generated automatically. The next step was to implement the IPersistenceService-interface with the means of the SQL-database.

As the methods described in the interface are expected to return data transfer objects, a translation from SQL-result-rows to data transfer objects was necessary. In this project the object relational mapping is realized with NHibernate. NHibernate cares about the mapping from attributes coming from a SQL-result-row to properties defined in a DTO. As soon as a DTO has to be persisted a mapping in the opposite direction is required. NHibernate is also responsible for managing this use case.

In order that NHibernate can execute the translation correctly a mapping from SQL-attribute to properties is necessary. Listing 5.3 demonstrates how the attribute 'Beschreibung' from the 'Produkt'-table is mapped to the 'Bezeichnung'-property in the 'ProduktDTO'-class in a xml-file:

```xml
<hibernate-mapping xmlns="urn:hibernate-mapping-2.2"
   assembly="KCon.Ordermanagement.DTO"
   namespace="KCon.Ordermanagement.DTO">
  <class name="ProduktDTO" table="Produkt">
    <id name="Id">
      <generator class="native" />
    </id>
    <property name="Bezeichnung" column="Beschreibung" not-null="true" />
  </class>
</hibernate-mapping>
```

Listing 5.3: NHibernate configuration

A challenge that had to be accepted while implementing the SQL-persistence layer was the realization of proper integration tests. The integration tests shouldn’t be dependent on the rows which are in the database by chance. Each test needs the same prerequisite. In addition no test should influence or manipulate the data for successive tests.
This problem was solved by creating a basic test-class which loads and executes an arbitrary SQL-file before and after a single test has been executed.

```csharp
[TestInitialize]
public virtual void StartUp()
{
    this.executeSQLCommands(this.sqlStartUpCommands);
    this.context = new NHibernateContext(this.config);
}
Listing 5.4: Startup method

[TestCleanup]
public virtual void CleanUp()
{
    if (this.context != null && this.context.IsDisposed)
    {
        this.context.Dispose();
    }
    this.executeSQLCommands(this.sqlTearDownCommands);
}
Listing 5.5: Cleanup method
```

The `StartUp`-method shown in listing 5.4 is called before a test is executed. As first the sql-file specified in the sqlStartUpCommands-variable will be loaded and executed. This sql-file contains `INSERT`-commands to fill the database with the desired test-data. Finally a new NHibernate-context will be initialized. This context represents a connection to the SQL-database.

The `CleanUp`-method presented in listing 5.5 is called after a test is executed, no matter if the test was successful or failed. If there is an existing connection to the database (the NHibernate-context is not disposed), then this connection and context respectively will be closed and disposed. At last step the SQL-file with tear down commands will be executed. This file contains `DELETE`-statements in order to clean the database and delete manipulated data.
Chapter 6

Testing

This chapter is about the different kind of tests that have been applied for software evaluation. It explains how and why the different test methods have been chosen.

6.1. Test driven development

The software in this project was mostly developed in a test driven manner. That means the class-body was created first, but without any method implementation. All methods throw a `NotImplemented`-exception by default. In a next step positive and negative test cases were defined for each method. The implementation of the method-logic didn’t start until all test cases were set up with the unit-test framework integrated in Visual Studio 2012.

The advantages of this approach is that enough time is spend for testing. Testing will not be delayed until the end of the project or even completely omitted. Another advantage of the test driven approach is that the developer bothers himself more about the responsibilities and dependencies of a class or a method. The whole system will be implemented more testable than in a code first approach. Test driven development also guarantees a high test coverage (compare Unit tests - code coverage p.28).

6.2. Acceptance tests

The acceptance tests have been defined during the customer meeting with K-Con. The aim was to define basic work flows which have to be support by the web-application. Each acceptance test consists of steps which the tester has to execute. After that, the results can be compared with the expected results. Table 6.1 shows the plan for testing the order-search by container-number (see more test-plans at Acceptance tests p.45).
### 6.3. Performance tests

The performance- and load-tests respectively are executed with the Web Performance Tests build in Visual Studio. The test-scenario is set up to simulate 25 user accessing the web-application concurrently. Every user logs in into the application and executes either the search by K-Con number or the search by order-data (start/target-location, container-number and state) Finally the user logs off from the application. This procedure is repeated for two minutes.

K-Con’s demand for the page-load of the order-detail-page is up to 2 seconds on a LAN-connection and up to 3 seconds on a 3G-connection. The performance-results can be found at [Web application - performance](p.27).
Chapter 7

Results

The following chapter presents the results of the project. The results include the user interfaces of the web and desktop application, performance analysis of the web application, code coverage of the unit tests as well as code analysis and HTML validity findings. Finally an overview of the future work is given.

7.1. Web application - UI

The user interface was implemented according to the UI-prototypes created in an earlier project stage. Figure 7.1 shows the screen on a Nokia Windows Phone after loading the order-data for ‘14-04-006’:

![Search order](image)

**Figure 7.1.: Retrieved data for order ’14-04-006’**

Certain order-data like the booking-reference or the container-number are endued with a special-link to a popup, which offers the user the possibility to forward the clicked number by email or sms as shown in figure 7.2
More screenshots of the UI can be found here (see Web application - UI, p.44).

### 7.2. Web application - performance

Figure 7.3 shows the average loading time that is needed to display the results by the GetOrders-search.

<table>
<thead>
<tr>
<th>Connection-type</th>
<th>Avg. page load time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>1,75</td>
</tr>
<tr>
<td>LAN</td>
<td>0,92</td>
</tr>
</tbody>
</table>

Figure 7.3.: Results page load time GetOrders-search

Before implementing a cache-logic in the Excel-persistence-layer, times listed in figure 7.4 were necessary to load the data of a single order:

<table>
<thead>
<tr>
<th>Connection-type</th>
<th>Avg. page load time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>1,57</td>
</tr>
<tr>
<td>LAN</td>
<td>1,25</td>
</tr>
</tbody>
</table>

Figure 7.4.: Results page load time without cache

After using a concurrent-dictionary to cache loaded orders, the average page loading changed to the ones shown in figure 7.5:

<table>
<thead>
<tr>
<th>Connection-type</th>
<th>Avg. page load time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>3G</td>
<td>1,40</td>
</tr>
<tr>
<td>LAN</td>
<td>0,74</td>
</tr>
</tbody>
</table>

Figure 7.5.: Results page load with cache

The average time for loading an object by a 3G-connection is reduced by around 11%. On a LAN-connection the loading time is shortened by around 40%.
7.3. Desktop application - UI prototype

The final UI prototype was realized with WPF. It shows the input-mask for an order like in figure 7.6. The left part contains all core order-data. The datagrid under Tasks lists the tasks that are assigned to the current order. By clicking on a task in the grid, the task-details will be loaded into the right part of the window. The navigation is mimicked from the Microsoft Office theme. By different tabs the user can select the different input- and edit-forms.

7.4. Unit tests - code coverage

All unit-tests (54) run successfully. The test-coverage for the class-libraries on server-side is shown in figure 7.1.

<table>
<thead>
<tr>
<th>Assembly</th>
<th>Not covered</th>
<th>Covered</th>
</tr>
</thead>
<tbody>
<tr>
<td>kcon.ordermanagement.di.dll</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>kcon.ordermanagement.dto.dll</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>kcon.ordermanagement.webservice.client.dll</td>
<td>0,63%</td>
<td>99,37%</td>
</tr>
<tr>
<td>kcon.ordermanagement.webservice.persistence.excel.dll</td>
<td>0,74%</td>
<td>99,26%</td>
</tr>
<tr>
<td>kcon.ordermanagement.webservice.persistence.mssql.dll</td>
<td>0%</td>
<td>100%</td>
</tr>
<tr>
<td>Average</td>
<td>0,27%</td>
<td>99,73%</td>
</tr>
</tbody>
</table>

Table 7.1.: Results code coverage

7.5. Code analysis results

The Visual Studio 2012 code analysis with the ‘Microsoft Managed Recommended Settings’ can’t find any issue in the code of the complete solution. Figure 7.7 shows the results.
The analysis of the code metric calculation showed that there are issues with the following classes:

- **KCon.Ordermanagement.Automapper.ObjectConverter**
  Class coupling too high (22)

  Class coupling too high (28), Maintainability index too low (66)

  Class coupling too high (43), Maintainability index too low (62)

- **KCon.Ordermanagement.Persistence.Excel.ExcelReader**
  Class coupling too high (42), Maintainability index too low (51)

- **KCon.Ordermanagement.ViewModel.TaskViewModel (Constructor)**
  Cyclomatic complexity too high (29)

### 7.6. HTML-Validity

All pages created for the web-application are successfully validated against the HTML5-draft. In addition, there can be no Javascript-errors found while using the application. The browser compatibility was manually tested with the browser-versions specified in (p.10). There couldn’t be any anomaly detected in any of the browsers.

### 7.7. Future prospects

The previous sections point out that have been realized in this project so far. However the process of modernizing the K-Con ordermanagement system is not completed yet. The next steps in this process would be the following:

1. **Improving code-quality**
   The classes mentioned in Code analysis results have to be checked according to the code-quality. If possible the code has to be improved so that the code metric analysis can’t find violations any more.

2. **Desktop-application**
   The web-service is completed so far that it can return orders as well as save new or edited orders. The desktop application has to send the order-data to the web-service so that the data will be persisted. Views for the read-access, according to the views for the mobile application, also have to be created.
   The prototype for the UI is already finished.

3. **Importing data from Excel-file**
   The data stored in the excel-file have to be process and transformed so that they can be imported into the SQL-database. This step is necessary because K-Con needs access to the old order-data in the daily business.
Chapter 8

Conclusions

In this chapter, a final set-actual comparison for the project is made. Furthermore, the aspects which have been learned will be outlined.

Set-actual comparison

The project plan allotted to finish the first milestone (‘Simplify read-access to order-data’) until the mid-term report. This milestone could be reached in the planned time (17/03/14). The official acceptance test by the customer took place during the customer meeting on 4th April. K-Con is using the mobile application since the beginning of March. Right now the mobile application supports the following features:

- The user can search for an order by K-Con-number (internal order-number)
- The user can search for an order by extended search (start/target location, container-number)
- The order-view shows the order-core data as well as detailed information about the tasks within an order
- Reference-, booking- and container-number of each order can be forwarded by email or sms (including access to device messaging functionality)
- The mobile application communicates with the server via a SSL-encrypted connection

For the second part of the project, modelling and implementing a database-system for the order-data was planned. In addition the web-service should be extended so that order-data can be send and persisted (write-access). This goals could be reached as well. The database-system is ready to be filled with data send to the web-service.

For the remaining project time creating a UI-prototype for the desktop-application was considered. The prototype could be finalized and serves as a blueprint for the upcoming implementation of the desktop application.

Webservices with Web API

The Web API framework is a framework designed for RESTful applications based on the .NET framework. Although it is a rather new framework, it doesn’t have to fear the competition with the Windows Communication Foundation.

In this project I gained the experience that Web API is very suitable for building REST-APIs with basic CRUD-functionality. The controllers used in Web API are very similar to the ones known from the ASP.NET MVC framework. The default response from Web
API is JSON, which guarantees platform independence. A Web API can be easily set up by deploying the project onto an Internet Information Server. In contrast to WCF, there is no need to define service-contracts or other additional configuration files (also on client-side). Thus, my conclusion is to prefer Web API to WCF. I would only use WCF again if more complex scenarios like callback-channels are necessary.

**jQuery Mobile for mobile development**

In this project I used jQuery Mobile the first time for mobile front-end development. After implementing the front-end and working with the framework I can conclude that jQuery Mobile is a mature tool for mobile development. The framework offers a sophisticated navigation system which automatically handles page transitions and loading panels. This results in a build-in, comfortable user experience. In addition jQuery Mobile provides a lot of helpful and well-designed widgets. This comes in handy as basic work like creating an appealing HTML-form can be done in short time. Also the visual appearance of the application becomes consistent automatically.

**Encrypted data-transmission**

In times of cyber crime and NSA encrypted data-transmission comes more and more into the center of attention. So far my impression was that encrypted data-transmission are difficult to set up and awkward to manage. In this project I learned that implementing a REST-API secured by SSL is straightforward. After retrieving a certificate (in this project a self-signed certificate generated by XCA), the server just has to be configured to accept and use SSL-connections only. The program logic doesn’t need to consider SSL as the encryption takes place on the transport layer.
Appendix A

References

Books:

Internet:
http://jamiekurtz.com/2013/01/14/asp-net-web-api-security-basics (08.02.2014)
http://demos.jquerymobile.com/1.4.1 (18.02.2014)
http://getbootstrap.com/getting-started (18.02.2014)
Appendix B

B.1. Client contact information

K-Con Gesellschaft für Kombinierte Verkehre mit Containern mbH
Michael Krüger
Bleigraben 40a
52372 Kreuzau – Germany
+49 2427 909449
michael.krueger@k-con.eu
http://www.k-con.eu

B.2. Current situation

Current situation - retrieving booking reference

<table>
<thead>
<tr>
<th>Truck-driver</th>
<th>Operator (Heppenheim)</th>
<th>Web-server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requesting booking-reference for ferry</td>
<td>Downloading latest Excel-file</td>
<td>Delivering Excel-file</td>
</tr>
<tr>
<td>Retrieving booking-reference</td>
<td>Opening Excel-file</td>
<td>Searching booking-reference</td>
</tr>
</tbody>
</table>
B.3. Planned situation

<table>
<thead>
<tr>
<th>Truck-driver</th>
<th>Operator (Heppenheim)</th>
<th>Web-application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requesting booking reference for ferry</td>
<td>Send request to web-application</td>
<td>Delivering order-data</td>
</tr>
<tr>
<td>Retrieving booking-reference</td>
<td>Forwarding booking reference</td>
<td></td>
</tr>
</tbody>
</table>

B.4. Predefined requirements

- The Excel-file containing all order-data has to be made available on the web
- The web-application must be able to retrieve order-data from the web-service using a secured/encrypted connection
- The web-application must present the order-data in an appropriate way on mobile devices
- The web-application must be able to send SMS or emails with predefined text modules
- The web-application must be runnable on Android, iOS and Windows Phone. The following devices are used within K-Con:
  - Nokia Lumia 925 (Windows Phone)
  - HTC Desire HD (Android)
  - Samsung Galaxy (Android)
  - HTC Tablet (Android)
B.5. Milestone planning

Milestone 01
Title: Simplify read-access to order-data
Planned date: 17/03/2014
Real date: 17/03/2014

Acceptance criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>State</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>The user can search for an order by KCon-number</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>The user can search for an order by start/target-location and container-number</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>An order-view (like defined in the final prototype) is available</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>The web-service delivers the latest data available in the excel-file</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>The user can forward order-data by SMS and email</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>The application is runnable on all devices specified by K-Con</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>The data-transmission between application and web-service is encrypted by SSL</td>
<td>Accepted</td>
<td></td>
</tr>
</tbody>
</table>

Milestone 02
Title: Store order-data on web-server
Planned date: 25/04/2014
Real date: 25/04/2014

Acceptance criteria

<table>
<thead>
<tr>
<th>Criteria</th>
<th>State</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database models order, task, address, product and containers</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>Object Relational Mapping implemented</td>
<td>Accepted</td>
<td></td>
</tr>
<tr>
<td>Web-service offers write-access for order-objects</td>
<td>Accepted</td>
<td></td>
</tr>
</tbody>
</table>
### B.6. Time planning

<table>
<thead>
<tr>
<th>Task Mode</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Predecessors</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>Project initialisation</td>
<td>1 day?</td>
<td>Fri 07.02.14</td>
<td>Fri 07.02.14</td>
<td></td>
</tr>
<tr>
<td>✔️</td>
<td>Time planning</td>
<td>0.5 hrs</td>
<td>Fri 07.02.14</td>
<td>Fri 07.02.14</td>
<td></td>
</tr>
<tr>
<td>✔️</td>
<td>Milestone planning</td>
<td>0.5 hrs</td>
<td>Fri 07.02.14</td>
<td>Fri 07.02.14</td>
<td>2</td>
</tr>
<tr>
<td>✔️</td>
<td>Quality planning</td>
<td>1 hr</td>
<td>Fri 07.02.14</td>
<td>Fri 07.02.14</td>
<td>3</td>
</tr>
<tr>
<td>✔️</td>
<td>Setup development environment</td>
<td>1 hr</td>
<td>Fri 07.02.14</td>
<td>Fri 07.02.14</td>
<td>4</td>
</tr>
<tr>
<td>✔️</td>
<td>Sprint 01</td>
<td>3 days</td>
<td>Sat 08.02.14</td>
<td>Mon 17.02.14</td>
<td>1</td>
</tr>
<tr>
<td>✔️</td>
<td>Research web-service (WCF or WebAPI?)</td>
<td>1 day</td>
<td>Sat 08.02.14</td>
<td>Sat 08.02.14</td>
<td>1</td>
</tr>
<tr>
<td>✔️</td>
<td>Implementing web-service</td>
<td>2 days</td>
<td>Mon 10.02.14</td>
<td>Mon 17.02.14</td>
<td>8</td>
</tr>
<tr>
<td>✔️</td>
<td>Sprint 02</td>
<td>3 days</td>
<td>Tue 18.02.14</td>
<td>Mon 03.03.14</td>
<td>7</td>
</tr>
<tr>
<td>✔️</td>
<td>Research responsive design framework</td>
<td>2 days</td>
<td>Tue 18.02.14</td>
<td>Wed 19.02.14</td>
<td></td>
</tr>
<tr>
<td>✔️</td>
<td>Implement web-app (Retrieve functionality)</td>
<td>3 days</td>
<td>Fri 21.02.14</td>
<td>Mon 03.03.14</td>
<td>11</td>
</tr>
<tr>
<td>✔️</td>
<td>Sprint 03</td>
<td>3 days</td>
<td>Fri 07.03.14</td>
<td>Mon 17.03.14</td>
<td>12</td>
</tr>
<tr>
<td>✔️</td>
<td>Research Hybrid-App-Framework</td>
<td>1 day</td>
<td>Fri 07.03.14</td>
<td>Fri 07.03.14</td>
<td></td>
</tr>
<tr>
<td>✔️</td>
<td>Extend web-app (accessing device-functionality)</td>
<td>2 days</td>
<td>Mon 10.03.14</td>
<td>Mon 17.03.14</td>
<td>14</td>
</tr>
<tr>
<td>✔️</td>
<td>Prepare mid-term presentation</td>
<td>2 days</td>
<td>Fri 21.03.14</td>
<td>Fri 28.03.14</td>
<td>13</td>
</tr>
<tr>
<td>✔️</td>
<td>Customer meeting</td>
<td>1 day</td>
<td>Fri 04.04.14</td>
<td>Fri 04.04.14</td>
<td>17</td>
</tr>
</tbody>
</table>

Figure B.1.: Time planning until customer meeting

<table>
<thead>
<tr>
<th>Task Mode</th>
<th>Task Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
<th>Predecessors</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>Sprint 04</td>
<td>4.5 days</td>
<td>Mon 07.04.14</td>
<td>Fri 25.04.14</td>
<td>19</td>
</tr>
<tr>
<td>✔️</td>
<td>Analyse data-structure in excel-file</td>
<td>1.5 days</td>
<td>Mon 07.04.14</td>
<td>Fri 11.04.14</td>
<td></td>
</tr>
<tr>
<td>✔️</td>
<td>Develop database-structure SQL/NoSQL</td>
<td>0.5 days</td>
<td>Fri 11.04.14</td>
<td>Fri 11.04.14</td>
<td>22</td>
</tr>
<tr>
<td>✔️</td>
<td>Implement write-access via web-service</td>
<td>4 days</td>
<td>Fri 11.04.14</td>
<td>Fri 25.04.14</td>
<td>23</td>
</tr>
<tr>
<td>✔️</td>
<td>Sprint 05</td>
<td>5 days</td>
<td>Mon 28.04.14</td>
<td>Mon 19.05.14</td>
<td>21</td>
</tr>
<tr>
<td>✔️</td>
<td>Design raw gui-mockups</td>
<td>3 days</td>
<td>Mon 28.04.14</td>
<td>Fri 09.05.14</td>
<td>22</td>
</tr>
<tr>
<td>✔️</td>
<td>Design detailed gui-mockups</td>
<td>2 days</td>
<td>Mon 12.05.14</td>
<td>Mon 19.05.14</td>
<td>27</td>
</tr>
<tr>
<td>✔️</td>
<td>Prepare final report/presentation</td>
<td>3 days</td>
<td>Fri 23.05.14</td>
<td>Mon 02.06.14</td>
<td>26</td>
</tr>
</tbody>
</table>

Figure B.2.: Time planning after customer meeting
Figure B.3.: Time planning as Gantt-chart (until customer meeting)
Figure B.4.: Time planning as Gantt-chart (after customer meeting)
### B.7. Sprint planning


**User:** mormll14  
**Password:** GzSsEw6cv

### B.8. UI-prototypes

![UI-prototypes](image.png)

Figure B.5.: UI-prototypes for portrait and landscape mode

### B.9. Programming tools and languages

#### Server-side

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming environment</td>
<td>.NET Framework 4.5</td>
</tr>
<tr>
<td>Programming language</td>
<td>C# 5.0</td>
</tr>
<tr>
<td>Hosting web-service</td>
<td>IIS 7.5</td>
</tr>
<tr>
<td>Hosting web-application</td>
<td>IIS 7.5</td>
</tr>
<tr>
<td>Framework web-service</td>
<td>Web API 2.1</td>
</tr>
<tr>
<td>Framework web-application</td>
<td>ASP.NET MVC 5.1.1</td>
</tr>
<tr>
<td>Database</td>
<td>MSSQL Express 2008 R2</td>
</tr>
<tr>
<td>Object transformation</td>
<td>AutoMapper 3.1.1</td>
</tr>
<tr>
<td>Dependency injection</td>
<td>StructureMap 2.6.4.0</td>
</tr>
<tr>
<td>Unit-testing</td>
<td>Visual Studio UnitTests 10</td>
</tr>
</tbody>
</table>

#### Client-side

<table>
<thead>
<tr>
<th>Feature</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming environment</td>
<td>HTML5, CSS3</td>
</tr>
<tr>
<td>Programming language</td>
<td>Javascript</td>
</tr>
<tr>
<td>Framework web-application</td>
<td>jQuery 2.1.0, jQueryMobile 1.4.0</td>
</tr>
</tbody>
</table>
Other tools

<table>
<thead>
<tr>
<th>IDE:</th>
<th>Visual Studio Ultimate 2012 Update 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modelling:</td>
<td>Visual Paradigm 10.2</td>
</tr>
<tr>
<td>Project management:</td>
<td>Project 2013, scrumdo.com</td>
</tr>
<tr>
<td>UI-prototyping:</td>
<td>Balsamiq mockups 2.2.19</td>
</tr>
</tbody>
</table>

### B.10. Webservice

**ExcelReader-proxy**

#### ExcelAccessProxy

- `uniqueInstance : ExcelAccessProxy`
- `fileName : string`
- `lastFileModification : DateTime`
- `createReader() : void`

#### ExcelReader

```csharp
public class ExcelReader
{
    public ExcelAccessProxy reader,
    public string fileName,
    public DateTime lastFileModification
}
```

- `loadFile(fileName : string) : void`
- `getPropertyIndex(objectName : string, propertyName : string) : int`
- `getRow(rowIndex : int, columnIndex : int, value : string) : string`
- `getCellValue(rowIndex : int, columnIndex : int) : TargetType`
- `getAddress(rowIndex : int) : AddressDTO`
- `getTask(rowIndex : int) : TaskDTO`
- `getOrder(rowIndex : int) : OrderDTO`

- `+GetOrder(kconNumber : string) : OrderDTO`
- `+GetInstance(fileName : string) : ExcelAccessProxy`

#### ExcelAccessProxy

- `uniqueInstance : ExcelAccessProxy`
- `fileName : string`
- `lastFileModification : DateTime`

#### KCon.Ordermanagement.Webservice.Persistence.Excel

-data transfer objects

- `OrderDTO`
  - `KConNumber : string`
  - `Number1 : string`
  - `Number2 : string`
  - `Weight : decimal`
  - `Product : string`
  - `StartDate : DateTime`
  - `EndDate : DateTime`
  - `Container : string`
  - `ReferenceOut : string`
  - `ReferenceIn : string`

- `AddressDTO`
  - `Position : int`
  - `Creditor : string`
  - `Type : string`
  - `StartDate : DateTime`
  - `EndDate : DateTime`

- `TaskDTO`
  - `Position : int`
  - `Creditor : string`
  - `Type : string`
  - `StartDate : DateTime`
  - `EndDate : DateTime`

- `Data transfer objects`
Excel persistence layer

```csharp
public class ExcelPersistenceService
{
    public ExcelAccessProxy uniqueInstance { get; set; }
    public string fileName { get; set; }
    public DateTime lastFileModification { get; set; }
    public void ExcelAccessProxy()
    {
        // Constructor implementation
    }
    public boolean hasExcelFileChanged()
    {
        // Implementation to check if Excel file has changed
    }
    public void createReader()
    {
        // Implementation to create Excel reader
    }
    public OrderDTO GetOrder(string kconNumber)
    {
        // Implementation to get order DTO
    }
    public void GetInstance(string fileName)
    {
        // Implementation to get instance
    }
}
```

```
public class ExcelReader
{
    // Properties and methods related to Excel reader
    public void loadFile(string fileName)
    {
        // Implementation to load Excel file
    }
    public int getPropertyIndex(string objectName, string propertyName)
    {
        // Implementation to get property index
    }
    public IRow findRow(ISheet sheet, int searchColumnIndex, string searchValue)
    {
        // Implementation to find row
    }
    public TargetType getCellValue(IRow row, int index)
    {
        // Implementation to get cell value
    }
    public AddressDTO getAddress(IRow addressRow)
    {
        // Implementation to get address DTO
    }
    public TaskDTO getTask(IRow taskRow)
    {
        // Implementation to get task DTO
    }
    public OrderDTO getOrder(IRow orderRow)
    {
        // Implementation to get order DTO
    }
    public void GetOrder(string kconNumber)
    {
        // Implementation to get order
    }
    public void GetInstance(string fileName)
    {
        // Implementation to get instance
    }
}
```

B.11. Webservice client

```csharp
public class OrdermanagementWebservice
{
    public string authKey { get; set; }
    public DateTime loginTimestamp { get; set; }
    public int loginTime { get; set; }
    public string host { get; set; }
    public Dictionary<string, string> settings { get; set; }
    public bool IsLoggedIn { get; set; }
    public void makeGetRequest(string command)
    {
        // Implementation to make get request
    }
    public void GetOrder(string kconNumber)
    {
        // Implementation to get order
    }
    public void Login(string username, string password)
    {
        // Implementation to login
    }
    public void Logout()
    {
        // Implementation to logout
    }
}
```

```
public class OrdermanagementWebservice
{
    // Implementation related to orders and services
    public void makeGetRequest(string command)
    {
        // Implementation to make get request
    }
    public OrderDTO GetOrder(string kconNumber)
    {
        // Implementation to get order DTO
    }
    public void Login(string username, string password)
    {
        // Implementation to login
    }
    public void Logout()
    {
        // Implementation to logout
    }
}
```
B.12. Responsive webdesign framework

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Zurb Foundation</th>
<th>jQuery Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fluent column design</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Button widget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Loading panel widget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>List view widget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Sub-list-view/accordion widget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Form-controls</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Popup/Notification widget</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Internet Explorer / WP 8</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Firefox / Android</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Chrome / Android</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Safari / iOS</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Theming</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

B.13. Hybrid app framework

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Telerik</th>
<th>Ionic</th>
<th>PhoneJS</th>
<th>Appcelerator</th>
<th>PhoneGap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform support</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Email</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Text-messaging</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Dialing</td>
<td></td>
<td></td>
<td>Yes</td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td>Free of charge</td>
<td></td>
<td>No</td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>
B.14. Storage layer for order-data

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Relational database (MSSQL)</th>
<th>Document-oriented database (CouchDB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Efficient way of data-storage</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>No redundant data in storage</td>
<td>Relations</td>
<td>No relations, redundant data!!!</td>
</tr>
<tr>
<td>Consistent data-storage</td>
<td>Transactions, referential integrity, foreign keys</td>
<td>Document-based transaction</td>
</tr>
<tr>
<td>Data are queryable</td>
<td>SQL</td>
<td>MapReduce</td>
</tr>
<tr>
<td>Concurrent read/write access</td>
<td>Optimistic concurrency control</td>
<td>Multi-Version Concurrency Control</td>
</tr>
</tbody>
</table>

Decision

The SQL-database is chosen for this project. Many data are reused in different orders. E.g. the address of a discharging station can occur in a lot of recurring orders. In a document-based database these data would be stored redundantly in every order-document. When the address is changing, all documents containing this discharging station have to be updated. In addition the document-based approach cannot guarantee that the order-documents are pointing to the same discharging address as foreign keys are not considered in the concept of CouchDB. All in all a SQL-database is superior for this use case as it offers a consistent and non-redundant data-storage by transactions, referential integrity, relations and foreign keys.

B.15. UI-prototype desktop application

Figure B.6.: Raw sketch desktop UI
B.16. Web application - UI

Figure B.7.: Extended search mask

Figure B.8.: Entries in search result list
B.17. Code metric results

Figure B.9.: Classes violating maintainability index

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Type</th>
<th>Member</th>
<th>Maintainability Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCon.Ordermanagement.Webservice.Client</td>
<td>OrdemanagementWebservice</td>
<td></td>
<td>62</td>
</tr>
</tbody>
</table>

Figure B.10.: Classes violating cyclomatic complexity

<table>
<thead>
<tr>
<th>Namespace</th>
<th>Type</th>
<th>Class Coupling</th>
</tr>
</thead>
<tbody>
<tr>
<td>KCon.Ordermanagement.ViewModel</td>
<td>TaskViewModel</td>
<td>29</td>
</tr>
<tr>
<td>KCon.Ordermanagement.Webservice.Client</td>
<td>OrdemanagementWebservice</td>
<td>43</td>
</tr>
<tr>
<td>KCon.Ordermanagement.Webservice.Persistence.Excel</td>
<td>ExcelReader</td>
<td>42</td>
</tr>
</tbody>
</table>

Figure B.11.: Classes violating class coupling

B.18. Acceptance tests

<table>
<thead>
<tr>
<th>Name: Suche Auftrag anhand Auftragsnummer</th>
<th>Voraussetzung: User ist im System eingeloggt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work-flow:</td>
<td></td>
</tr>
<tr>
<td>1. User trägt ‘14-02-003’ ins Suchfeld ein</td>
<td></td>
</tr>
<tr>
<td>2. User drückt Suchen-Button</td>
<td></td>
</tr>
<tr>
<td>3. User wartet bis Ergebnisseite vollständig geladen ist</td>
<td></td>
</tr>
<tr>
<td>Erwartetes Ergebnis: Die Details zu Auftrag ’14-02-003’ werden angezeigt</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Suche Auftrag anhand Ladeort</th>
<th>Voraussetzung: User ist im System eingeloggt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work-flow:</td>
<td></td>
</tr>
<tr>
<td>1. User drückt den Button ‘Erweiterte Suche’</td>
<td></td>
</tr>
<tr>
<td>2. User trägt ‘Neuss’ ins Ladeort-Feld ein</td>
<td></td>
</tr>
<tr>
<td>3. User drückt Button ‘Erweiterte Suche starten’</td>
<td></td>
</tr>
<tr>
<td>4. User wartet bis Ergebnisseite vollständig geladen ist</td>
<td></td>
</tr>
<tr>
<td>5. User klickt auf Auftrag ‘14-03-023’ (4 Suchergebnisse)</td>
<td></td>
</tr>
<tr>
<td>Erwartetes Ergebnis: Die Details zu Auftrag ’14-03-023’ werden angezeigt</td>
<td></td>
</tr>
<tr>
<td>Name:</td>
<td>Suche Auftrag anhand Entladeort</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>Voraussetzung:</td>
<td>User ist im System eingeloggt</td>
</tr>
</tbody>
</table>
| Work-flow: | 1. User drückt den Button ‘Erweiterte Suche’
2. User trägt ‘Husum’ ins Entladeort-Feld ein
3. User drückt Button ‘Erweiterte Suche starten’
4. User wartet bis Ergebnisseite vollständig geladen ist
5. User klickt auf Auftrag ‘14-03-030’ (2 Suchergebnisse) |
| Erwartetes Ergebnis: | Die Details zu Auftrag ‘14-03-030’ werden angezeigt |

<table>
<thead>
<tr>
<th>Name:</th>
<th>Leite Buchungsnummer per SMS weiter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voraussetzung:</td>
<td>User ist im System eingeloggt, Test ‘Suche Auftrag anhand Auftragsnummer’ wurde ausgeführt</td>
</tr>
</tbody>
</table>
| Work-flow: | 1. User klickt auf Buchungsnummer
2. User klickt im Popup-Fenster auf SMS |
| Erwartetes Ergebnis: | Der Endgerät-typische Dialog zum Verfassen einer SMS wird geöffnet. Der Textinhalt der SMS entspricht der Buchungsnummer |

<table>
<thead>
<tr>
<th>Name:</th>
<th>Leite Buchungsnummer per Email weiter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voraussetzung:</td>
<td>User ist im System eingeloggt, Test ‘Suche Auftrag anhand Auftragsnummer’ wurde ausgeführt</td>
</tr>
</tbody>
</table>
| Work-flow: | 1. User klickt auf Buchungsnummer
2. User klickt im Popup-Fenster auf Email |
| Erwartetes Ergebnis: | Der Endgerät-typische Dialog zum Verfassen einer Email wird geöffnet. Der Textinhalt der Email entspricht der Buchungsnummer |