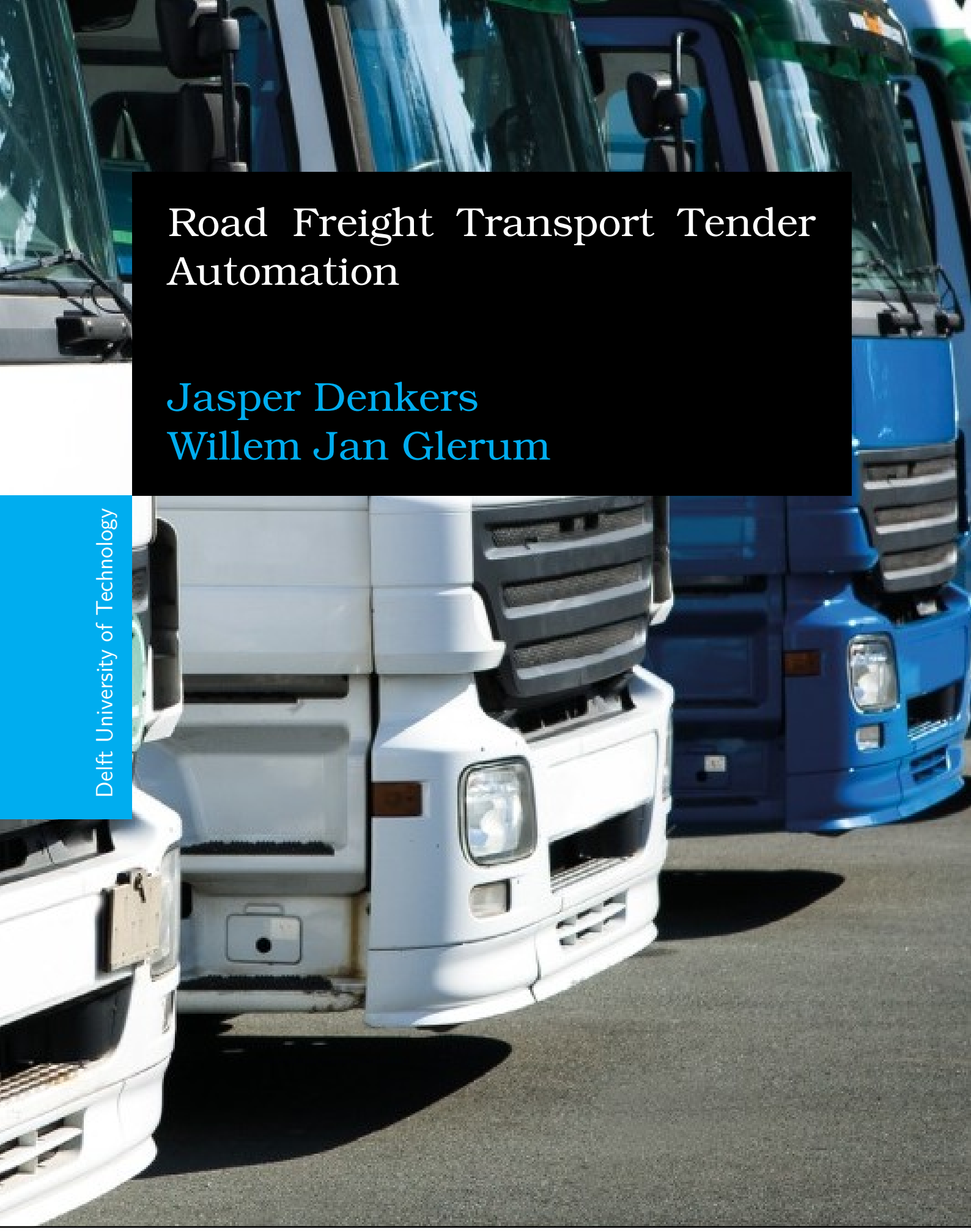

Road Freight Transport Tender Automation

Jasper Denkers
Willem Jan Glerum

Delft University of Technology

TU Delft Delft University of Technology

Road Freight Transport Tender Automation

by

Jasper Denkers
Willem Jan Glerum

Bachelor's Thesis
Computer Science
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
July, 2015

Supervisors:	P. Struiwigh	Transport Tender
	A. Zaidman	TU Delft
Bachelor Project Coordinator:	M. Larson	TU Delft

An electronic version of this thesis is available at
`http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

Transport Tender
Transparantie in Transporttarieven

Preface

This document is the final report of the Bachelor Project as final course of the Bachelor Computer Science at the Delft University of Technology carried out by Jasper Denkers and Willem Jan Glerum (student team). The project is commissioned by Transport Tender, represented by Peter Struiwigh (client) and supervised by Andy Zaidman (supervisor), an associate professor in the Software Engineering Group (SERG) at the Delft University of Technology.

This document gives an overview of the work that has been done, the choices that have been made, the product that has been delivered and the process of twelve weeks in which this happened. Transport Tender is a consultancy company that advices companies by their choice of logistic partners in processes called tenders. During this project an application is developed to automate these tenders.

We would like to thank Peter Struiwigh for his assistance and support during the project. We would also like to thank Andy Zaidman for his supervision and feedback on our work.

Summary

Transport Tender is a consultancy company advising clients how to plan their logistic activities. Its focus is on road freight transport within Europe. In processes called tenders, the actual prices of new potential carriers are calculated for the shipments of the client of e.g. the last one or two years. Based on these prices, different sets of carriers are evaluated to cover all shipments for the client. Shipments are spread over multiple countries and because the potential carriers are mostly profitable in a small set of countries, this analysis is necessary and helpful. By choosing more carriers, chances increase a collectively lower price for the client's transport could be established. However, the client wants to work with as few as possible third parties. This contradiction makes planning hard. Tenders help making this choice easier by making differences between options insightful. In this project, software is created to automate tenders.

During a two week research phase, the client's domain is explored and an understanding of the available data, its formats and the required analysis is developed. Requirements are formulated using the MoSCoW principle. A standardised input format for shipments and rates is designed and a web application is developed that can input these data and return the results of analysing them automatically. The application is developed using the Play Framework and Scala. Using a continuous delivery approach, an up-to-date version of the software was deployed on Heroku. Testing the software is mainly done by using automated unit, integration and functional tests. Code coverage reports are used to be able to focus testing activities. Quality of the software is maintained by analysing the source code regularly with Sonarqube.

The final product is an innovative web application that is both user friendly and that is able to perform analysis quickly. Its innovation is not in a particular algorithm or idea, but in combining multiple sources of data and extracting valuable information. It automates the complete proces of tenders which reduces a lot of manual work for the client. No other similar services are known on the market.

The price of transport is not completely decisive in the choice of carriers. Besides quantitive data like prices, qualitative data could be used in analysis. This could include carrier's reputations or transit times. Besides that, the application's scope could be expended to outside Europe and other types of transport, e.g. transport by ship, train or airplane.

Contents

List of Figures

List of Tables

1 Introduction

Many companies outsource the transportation of their goods. Important for those companies is: to who do you outsource your logistic activities, at what cost and with how many partners? These are big questions and Transport Tender is a consultancy company helping to answer them.

In processes called tenders, Transport Tender looks at different sets of potential carriers to take care of the transport of their clients. This is done by using historical transport data of the client and current rates of the potential carriers to work with. The assumption is made that historical shipment data is representative for the client's logistic activities and therefore the data is useful for performing the analysis.

The outcome of a tender is a report that gives insight in the differences of costs when carrier sets of varying size and composition are chosen. The ideal situation would be to choose one carrier that can execute all the transport of the client for the lowest price: it is cheap and the client has only one company to communicate with. Unfortunately, this scenario is not realistic. E.g. carrier A might be cheap in Germany and expensive in France and carrier B might have opposite pricing. In this case it is obvious to select both carriers and let carrier A and B take care of Germany and France, respectively. This lowers the total costs, which is desirable, but highers the amount of companies to work with, which is not desirable. This contradiction makes the choice of a set of carriers hard. Transport Tender helps companies by making this choice by using the results of tenders.

In this project this tender process will be automated by developing a software application. This application will take the aforementioned data and perform price calculations and accompanying evaluation of sets of carriers. Results are exportable so Transport Tender can use it to advise their clients.

1.1 Structure

This report is roughly separated in three consecutive parts to describe the project. Firstly, it starts with a description of the research of the client wishes and the problem that should be tackled. From this research phase, an overview of requirements is produced. Secondly, a description is given of the architecture (design and implementation) and testing of the product that has been developed based on these requirements. Thirdly, the last sections reflect on the project by looking at the process and its conclusions and it gives recommendations for possible follow ups of the project. Choices made in both design, architecture and process are motivated.

2 Research

2.1 Overview

In this section the initial research phase of this project will be discussed. While doing research was an ongoing activity during the project, the first two weeks were fully dedicated to investigating how the project was going the be approached.

In this phase, a problem definition has been formulated (Section 2.2.1) and analysed (Section 2.2.2). Also, a description of the current situation of the client is given (Section 2.2.3). Building upon the outcomes of this problem definition and analysis, an extensive description of the domain (road freight transport) of the project is given (Section 2.3). The requirements of the project are stated (Section 2.4) using the MoSCoW model. This gives a prioritised overview of properties the final product respectively must, should, could and won't have. To make sure a product of certain quality will be delivered, design goals matching the clients wishes are determined (Section 2.5). Furthermore, with a technological approach, some algorithms are discussed that will potentially support solving the formulated problem (Section 2.6). Lastly, a set of tools and services is listed that will support the development and process during the course of the project (Section 2.7).

2.2 Problem Definition and Analysis

Many companies face the challenge of producing products of the highest quality for the lowest possible price. A substantial part of this price is determined by transport rates. Therefore, the logistics sector is big. In 2011, the Netherlands alone had almost ten thousand enterprises in road freight transport [4]. Together, in the same year, these companies produced twenty billion euros in turnover [4].

Transport Tender, the client of this project, helps companies by choosing the right set of logistic partners in processes called tenders. In such a tender, transport rates of multiple carriers are compared in multiple tender rounds based on the shipment history of the client the tender is carried out for. Transport Tender is a small company which completely executes these tenders for their clients and it offers related consultancy based services. Its clients range from small to big companies that outsource road freight transport within Europe.

2.2.1 Problem Definition

It would be ideal for a company to be able to work with only one carrier which also asks the lowest price for any shipment. However, most carriers cannot provide transport to all the desired locations in Europe profitably. Therefore, a set of multiple carriers is chosen. Choosing more carriers increases the chance of getting lower rates. However, it is desired to work with a small set of carriers to reduce parties the work with and trust. This contradiction, in combination with a big offer of carriers, makes the tender process hard. The aim of this project is to automate tenders and specifically the related analysis of rates.

To get the best result from a tender, a detailed analysis of the rates of potential carriers should be made. To get insight in rates of these carriers, the costs of transport from the client from e.g. the last year are calculated for each carrier and compared. The main results of this analysis are based on the difference in costs from selecting the most profitable set of carriers of size 1 to m, where m is the number of carriers the tender is carried out with. Actually, the main question a tender answers is: how much money can the client save by choosing a specific number of carriers and what is the difference in savings by selecting less or more carriers?

2.2.2 Problem Analysis

To be able to perform these calculations, shipment data from the client should be available. The assumption is made that future logistic activities of the client are about the same as past activities. Therefore, shipment data from the past is used, e.g. from the past one or two years. Also, the rates for these shipments for each of the potential carriers should be available to perform price calculations. The analysis of the sources and formats of these data can be found in Section 2.3.

An application has to be developed that can process and store the previously mentioned types of data and calculate the price of each shipment for each potential carrier. After that, different sets of carriers should be evaluated and the corresponding costs to the sets should be insightfully reported. Automating the processing of these files and generating analysis reports could speed up Transport Tender's workflow a lot. Also, rates of old tenders should be reusable in new tenders. To make this usable, users of the system should be able to browse through old tender information and categorise these data manually.

2.2.3 Description of Current Situation

Transport Tender initially started with the idea to automate the tender process completely and set up a kind of marketplace for tenders. A web application was built which could be used by clients to set up the tender process themselves. This application is currently in use and focuses primarily on the process of the tender, not analytics and data gathering. Since the start of the company, it turned out clients are not likely to perform the tenders but ask Transport Tender to take care of the process. In many cases, a lot of money is involved in the organisation of transport and since the clients pay Transport Tender for the analysis, they do not want to execute the tender process. Therefore a new application is required to focus more on analysing rates and less on being a marketplace.

In the current application clients can register and set up a tender. This includes submitting general tender data, uploading shipments and inviting participating carriers. After registering and accepting an invitation, carriers can view tender information and submit rates accordingly. The application performs basic analysis and Transport Tender manually extends this analysis mainly using spreadsheet software.

2.3 Road Freight Transport

Transport Tender's services are limited to freight transport by trucks within Europe. To be able to perform calculations and analysis of transport prices, we should have an understanding of two important types of data: shipments and rates. These data are used to calculate prices following a generally used system.

In this system, shipments are described by a set of properties. Only the location and amount of load are used to calculate a price. Rates are determined by two dimensional rates tables. These tables consist of columns representing zones of locations and rows representing zones of ascending amounts of load. To determine the price of a shipment, the price is looked up in such a rates table in the cell corresponding to the zones of location and amount of load of the shipment.

2.3.1 Shipments

In tenders, clients use the shipment history of e.g. last year to compare the prices of potential new carriers for those shipments. The key properties to describe those shipments are listed in Table 1.

Property	**Description**	**Example value**
Country code	A code identifying the country of the source or destination of the shipment.	NL
Postal code	Format differs per country. Only the first two digits of the postal code are used.	1234
Date	For optional summing, see Section 2.3.3.	2015-01-01
Quantity	Weight, volume or amount of load units. Differs per load type, see Section 2.3.4.	13.5
Surcharge	Whether a surcharge is applicable when load contains e.g. dangerous goods.	yes
Comment	A description of the shipment to give at a detailed level better insight in what kind of transport is covered.	Delivery before 3 a.m.

Table 1: Properties of a shipment

Most of the clients are able to export this information from Enterprise Resource Planning (ERP) software to an exchangeable file format like Comma Separated Value (CSV) or Microsoft's Excel. These files should contain shipments representable for the logistic activities of the client for the tender, to give the best insight in rates of potential carriers for future logistic activities. An example of such a file can be found in Table 2. The load of these shipments is measured in kilograms.

#	Country	Postal code	Date	Quantity	Surcharge	Comment
1	NL	0123	2015-01-01	50	yes	...
2	NL	1234	2015-01-02	60	no	...
3	NL	1234	2015-01-02	70	no	...
4	NL	2345	2015-01-03	80	yes	...
5	NL	2345	2015-01-04	90	no	...

Table 2: Example list of shipments

2.3.2 Inbound or Outbound

Tenders are inbound or outbound. This means that all the shipments in a tender are from one unique address to other addresses (outbound) or to one unique address from other addresses (inbound). Therefore only one address should be provided per shipment.

2.3.3 Summing Shipments

Shipments provided by the client might contain multiple shipments on the same day to or from the same location. If multiple of these shipments are small, they could be combined and shipped by the same truck. This is called summing and reduces total costs. In some tenders summing is applicable and in some not, depending on the client and load type.

If we look at the example shipments in Table 2, shipments #2 en #3 are the only shipments that could be summed, since they share the same destination or origin on the same date. That would result to a summed single shipment with a load of 130 in total. Shipments #4 and #5 also share the same destination or origin. However, these shipments can not be summed because they where on a different date.

2.3.4 Load Quantity Measuring

Depending on the type of load being transported, a quantity type to measure the amount of load is chosen. One might measure all load in kilograms, but this would lead to problems. For example a truck could be filled with feathers or lead, with the same amount in kilograms. This truck full of feathers would cost the same as a truck with some lead and plenty of space left. This would of course not be profitable.

To cover all types of load, three main load quantities are used, which we describe below. The type of load chosen in a tender is fully dependant on the type of shipments and the client's preferences.

Volume Measured in load meters. The most common way of measuring load quantities. One load meter equals the volume of one meter in depth of the loading space of a truck. If this type is chosen, for each shipment the weight (measured in kilograms) and volume (measured in load meters) should be provided. There is a minimum amount of weight per load meter to prevent problems similar to the example with feathers. This minimum weight per load meter is about 1850 kilograms, but should be variable for each rates table. For example, the rate of a shipment with a volume of 1.5 load meters and a weight of 2000 kilograms is priced at the rate of 1.5 load meters, i.e. 2.775 kilograms.

Weight Measured in kilograms. With this type, volume of the load is not taken into account.

Units Measured in specific units of volume. This is used if the load always consists of units with the same dimensions and the weight should not be taken into account, e.g. pallets.

2.3.5 Surcharges

In some cases, clients might ship goods that e.g. contain dangerous substances or load that should be cooled. Such special shipments should be handled with extra care and are therefore more expensive to ship. The deviation of the prices for these special shipments is calculated using surcharges that could be applied per single shipment.

The amount of surcharges types should be variable, but most clients have only one type. The fee that is added to the price of shipments with a surcharge is a percentage, bounded by a minimal an maximal amount.

2.3.6 Rates and Zones

The possible combinations of addresses and quantities of weight per shipment might require a lot of required rates. To reduce this amount of rates required to price shipments, the two key properties of shipments (location and quantity) are divided into zones. These zones are used as columns (zones of locations) and rows (zones of quantity) to construct a rates table. In logistics, it is common to provide such a table per country. So if a tender applies to shipments in multiple countries, carriers should provide multiple rates tables. An example of such a rates table can be found in Table 3. This table applies for shipments in the Netherlands with load measured in kilograms.

	Zone 1	**Zone 2**	...	**Zone 10**
	00-09	10-19		90-99
100	200	210	...	400
200	400	420	...	400
300	600	630	...	570
500	1000	1100	...	900
750	1500	1650	...	1275
1250	2500	2750	...	2000
FTL	3750	3885	...	2775

Table 3: Example rates table

2.3.7 Location Zones

Most postal codes contain four or five digits. Only the first two digits are used to indicate the location zone of the address of a shipment. This leads to one hundred zones of postal codes per country. If for each of this zone rates should be provided, rates tables would contain one hundred columns. To reduce the amount of columns such table contains, the zones are further distributed into location zones. This distribution is determined per carrier and is therefore different for each rates table.

In Table 3, we see this leads to only ten columns which reduces the amounts of rates that should be provided a lot. In the example the zones are constructed of ten ascending postal code zones, but this distribution could be arbitrary as well. An exception is the United Kingdom, where no numeric postal codes are used, but letters indicating regions. In that case, location zones could be constructed by grouping these regions. The same applies for Ireland, were regions are used to indicate location zones.

2.3.8 Load Zones

To determine prices for shipments of different size, load zones are used. These are zones of ascending quantities that define upper limits of amounts of load, whether these are load meters, kilograms or units. For load quantity measuring in load meters, only upper limits in weight (kilograms) and not in volume should be provided, see Section 2.3.4. Bigger shipments are cheaper relative to load size than smaller shipments. This "discount" is covered by using these load zones.

Full Truck Load (FTL) is the maximum amount of load a truck can ship. If the amount of load of a shipment exceeds FTL, multiple trucks must be used. FTL is always the largest load zone.

2.3.9 Pricing

The price of a shipment is determined by looking it up in the right cell in a rates table. This cell is the intersection of the column and row corresponding to the location zone and load zone, respectively.

The location zone is determined by taking the first two digits of the postal code of the shipment and checking which of the location zones contains this postal code zone. In the case of the United Kingdom and Ireland, the location zone is selected that contains the region or district of the shipment.

The load zone is determined by selecting the load zone with the smallest upper limit higher than or equal to the amount of load the shipment contains.

Lets consider shipment #2 and #3 from Table 2 and the rates table from Table 3. These shipments could be summed, see Section 2.3.3, yielding a combined shipment to postal code 1234 with a load of 130 kilograms. Since the postal code zone of the summed shipment is 12, the rate for this shipment is taken from column two (location zone 2). The combined quantity for this summed shipment is 130 and the lowest upper limit higher than or equal to 130 in the rates table is 200. Therefore, the rate for this shipment is taken from row two. This makes the price for this shipment 420.

2.4 Requirements

In order to fulfil the clients needs in this project correctly, a set of requirements should be formed. However, since requirements could change, this set is not fixed. Since we will be using SCRUM for the development phase of this project, a prioritised list of requirement is managed, called a product backlog.

To prioritise our product backlog we will make use of the MoSCoW principle which describes the importance of each requirement:

MUST A requirement that must be implemented to reach a successful product.

SHOULD A high-priority requirement that should be implemented if possible.

COULD A feature which is desirable and is only included when there is enough time.

WON'T This requirement will not be included in this release, but could be in the future.

An overview of requirements of this project categorised using the MoSCoW principle can be found in Table 4.

2.4.1 Tender Automation

A weighty requirement of the client is the complete automation of the tender process. A lot of time is currently consumed by formatting data with spreadsheet software and transforming data from old tenders to be used in new tenders. Using a predefined format for input data and automatically performing analysis and accompanying results is desired.

2.4.2 Client Reporting

The most important form of reporting of the application is reporting to clients. A client is the originator of a tender and is paying for the process. In this reporting it should in particular become clear what the savings are by selecting a new set of carriers. Besides that, the reporting should give insight in how big this set of carriers should be and which carriers should be selected. Insight should be produced in differences between carriers based on prices categorised per location or shipment sizes.

MoSCoW	Description
Must	An application must be developed that can import and validate shipments and rates data like described in Section 2.3.1 and Section 2.3.6, respectively.
Must	Using imported data, prices for shipments for multiple carriers must be automatically calculated, see Section 2.4.1.
Must	Calculated prices must be exportable.
Must	Based on calculated shipment prices, different sets (see Section 2.6.1) of a variable amount of carriers must be automatically evaluated.
Must	The difference in choosing more, fewer or other carriers must be made visible and exportable to clients.
Must	The application should be web-based, does not require the installation of special software and be accessible via the internet.
Must	The application should be secured and only accessible after authentication.
Should	Carriers should also get feedback on their prices.
Should	Rates data of old tenders should be reusable in new tenders.
Should	Instead of static analysis reporting, an interactive reporting system should be developed to produce reports showing difference in costs between different sets of carriers.
Should	Old tenders should be categorisable to be better searchable for reusal.
Could	Besides the use of quantitative data like prices, qualitative data of transport like lead times could be taken into account in tenders. Since not only prices, but also quality of transport is important, this could be a nice improvement.
Won't	The underlying price calculation software could be used in other applications than tendering. However, this is not in the scope of the project.
Won't	An interactive map of Europe could be generated with information about carriers per region. This is estimated as a lot of work and does not directly produce a big improvement of the final product, but could be a nice feature.

Table 4: Project requirements categorised using the MoSCoW principle

2.4.3 Carrier Reporting

Carriers provide their rates in order to make a chance at getting new clients. They can always get something in return, also when they are in the final set of chosen carriers. Namely, feedback on their prices in comparison to the other carriers in the tender. This feedback can not be exact, since this would reveal market sensitive information between carriers. However, feedback with indications of deviations from prices in comparison to other carriers can be provided. In that way, carriers are more willing to participate in tenders. An example of such an indication could be: "The prices of carrier A are 5% to 10% lower than the prices of carrier B in the Netherlands." It can help carriers by adjusting their prices and lowers the final price a client gets in the end, since it promotes market forces. Lastly, it gives both carriers and the client insight in were carriers are more profitable based on location or shipment sizes.

2.4.4 Data Reuse

A shortcoming in the current process of Transport Tender is the possibility to reuse data. With every tender executed, valuable rate information of multiple carriers is gathered. This information should be indexed and searchable to be able to find rates for new tenders. To make searching through rates history easy, tenders and rates should be able to be categorised.

2.4.5 Platform and Technologies

Since a new application is developed, there are no specific restrictions to a platform or technologies to be used. Besides the restriction of the kind of application, a web application, there are no other systems the application should be coupled with and therefore we are free to choose our technologies.

2.5 Design Goals

This section describes the main design goals for this project. Mainly focusing on performance, quality and maintainability.

2.5.1 Performance

As the input are large amounts of data, the application should be able to cope with this amount of data. The application should stay responsive while analysis is done in the background. This means page loading times should not be affected when a big amount of data has just been uploaded. We can achieve this by applying proper software engineering techniques, developing efficient algorithms and ensuring loose coupling between modules.

The application is of vital importance to the daily workflow of the client. Therefore, it should not crash when handling unexpected inputs. Proper error handling has to be implemented to ensure the application does not stop working during a failure. Furthermore, the errors should contain relevant information to the user, to make it easier to understand what has gone wrong and how to prevent this.

2.5.2 Maintainability

The new application should be easily maintainable, as the client does not have a team of software developers. The code should be easily understood by third party developers. To ensure the maintainability, we use an automated process for deploying new versions of the application.

First a newly implemented feature has to be fully tested locally before deploying it to the master branch on Github. Next, an automated test is run on Travis CI to provide continuous integration. And the last step is to automatically deploy the new version of the application on Heroku after confirmation of the continuous integration service. These services are described in detail in Section 2.7.2.

Furthermore, all code should be analysed regularly with Sonarqube to ensure overall quality. This should be done before sending our code to Software Improvement Group (SIG), so the first report from SIG is positive. Next we use Sonarqube again, to improve the relevant points from the SIG report. Our goal is to continuously look for possible improvements.

2.6 Algorithm

Before we start thinking about the algorithms to implement for solving the problem of getting the best price, we first must have a mathematical representation of the problem. There is a list S of shipments from the client with length n, $S \leftarrow \{s_1, s_2, \ldots, s_n\}$ and m carriers in a set $C \leftarrow \{c_1, c_2, \ldots, c_m\}$. For each pair of shipment and carrier we have a price p_{nm}. This can be represented in a matrix P as can be seen in Table 5. It can also be more generally represented as table of goods and sellers [1].

	c_1	c_2	$\cdots$	c_m
s_1	p_{11}	p_{12}	$\cdots$	p_{1m}
s_2	p_{21}	p_{22}	$\cdots$	p_{2m}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
s_n	p_{n1}	p_{n2}	$\cdots$	p_{nm}

Table 5: Example of a matrix with prices for each shipment and carrier

To get such a matrix of prices, some pre-processing has to be done from the data we get from the client and carriers. For each shipment we have to do a lookup in the shipment table to get its properties and next lookup the price for each carrier in their respective rates tables. When surcharges apply, see Section 2.3.5, we have to compute a different price based on the extra charges. However this does not make the problem harder, as it is just a different price to be filled in the matrix P.

2.6.1 Combinations

When the client only wants one carrier we can just simply sum the price of all the shipments per carrier to calculate the total costs and select the carrier with the lowest total price. If the client does not care about how many carriers are used, we can simply select for each shipment the carrier with the lowest price. Sum the selected prices per carrier and return the total costs for each carrier. These are the easiest computations we can do, however they are not very realistic in the real world.

More realistic examples are when we have to choose x carriers from a total set C of carriers with length m. The order in which we choose x carriers does not matter. So we use the binomial coefficient to compute the number of combination we can make for the carriers:

$$\binom{m}{x} = \frac{m!}{x!(m-x)!}$$

However the client does not know on beforehand how many carriers he wants, therefore we have to make these combinations for every x from 1 to m. This results the following equation:

$$\sum_{x=1}^{m} \binom{m}{x} = \sum_{x=1}^{m} \frac{m!}{x!(m-x)!} = 2^n - 1$$

For every combination we have to do the previous described calculation. Select the x carriers and lookup the lowest price per shipment and sum the selected prices per carrier. This is easily done when we only have two or three carriers, when the total number of combinations is still under ten. However when we have ten carriers this number equals $2^{10} - 1 = 1023$. Fortunately these calculations for selecting the lowest price are easy and the number of carriers per tender mostly does not exceed five or six carriers, with a maximum of ten.

2.6.2 Banker's Sequence

Most algorithms to compute all subsets of a set of items do not enumerate the subsets in monotonically increasing order of size. There is a way to achieve this by using a so called Banker's sequence [2], where first all subsets of size k are computed before subsets of size $k+1$ are considered. Using this construction we can easily compute the cheapest set of carriers for a given number of carriers to use.

2.6.3 Genetic Algorithm

A nice extension would be to store information from previous tenders and use them for future tenders. However the rates we get from a carrier can differ between two tenders. This information can be used to predict which carriers could be cheaper and therefore contacted to give rates for a specific tender.

As can be seen the number of combinations of tenders grows exponentially with the number of carriers we consult for a tender. It is not realistic to compute all sets of combinations if the number of carriers grows beyond ten. If we do want to consider this, we could look at genetic programming [3] using heuristics. This method cannot guarantee the optimal solution, but can come close. This is however outside the scope of this project at the moment.

2.7 Development

The client does not have any preferences for the technologies used for and during the project. The only requirement is that it has to be a web application which should be accessible by multiple persons. In the current situation the application runs on the company website with a custom made WordPress[1] plugin written in PHP. Coupling between the corporate website and the application is not necessary and not needed as customers should not have access to the application. This section discusses which technologies we will use to build the application and which tools will enable us to achieve this.

[1] `https://wordpress.com/`

2.7.1 Technologies

As stated above, there are no requirements regarding the programming language and frameworks to implement the application. During the Bachelor Computer Science we have learned multiple programming languages including Java, Scala, C, Prolog, SQL, PHP, and JavaScript, with the focus on using Java for object oriented programming. Our preference is using Scala as our main programming language, as it is based on Java and we have hands on experience with Scala from the course Concepts of Programming Language. Furthermore, we will be using the Play Framework for building the web application. This is an web framework based on Scala and built by the originators of the Scala language itself. We both have previous and positive experience from using this framework within the course Contextproject, where we used it with Java.

We could have chosen different popular web frameworks to learn something new. Such as Ruby on Rails, Django or Symfony, respectively written in Ruby, Python and PHP. However, we remain with our first chose, the Play Framework, because our previous experience enables us to make a head start. We do want to learn new techniques though, therefore we choose Scala as the main language and not Java. Furthermore, we also want to make use of new technologies for web development for example; AngularJS, CoffeeScript and LESS which can be easily incorporated into Play.

2.7.2 Tools

During the development phase we need a set of tools to build the application. Not all of them are necessary, they do however make it easier. And we use tools to ensure the code quality, measured in different metrics.

Github is an online hosting platform where we can track our code using Git for version control. Git enables to use different branches for building different features and merge them into the main code when finished. Github provides an easy to use interface for all of this. Furthermore, Github provides web-hooks which enables communication with external services, for example continuous integration.

Travis CI provides online continuous integration for software projects using Git. The main advantage of using Travis is that it has easy integration with Github using their web-hooks. Travis can run test on different platforms, e.g. on different versions of a programming language.

Coveralls computes the test coverage in combination with Travis CI and Github using web-hooks. Here we can easily see the code coverage of our tests on different branches of the project. This is a basic overview for us to see how well tested each piece of code actually is.

Sonarqube is a more advanced tool for checking the code quality not only based on test coverage. It performs static code analysis on the source code to see where problems could arise. Checking and reporting for code duplication, code complexity, coding rules, potential bugs and architecture and design.

Heroku provides us with a free hosting platform for web applications. New versions of the application are only deployed after confirmation of the continuous integration service. We use Heroku, because our client does not have a dedicated infrastructure for developing software. For production use of the application we should pay for Heroku or rent a Virtual Private Server.

2.8 Defining Success Criteria

The transport sector is large sector involving a lot of money. Our client Transport Tender helps its clients to save money on transport. To achieve these savings, heavy calculations have to be done based on prices from different carriers. The goal of this application is to find the best set of carriers for a client. And to see whether a client can save substantially more money by selecting more carriers. This involves an extra risk for the client, as the client has to work with more external parties. The current application of Transport Tender focuses on automating the tender process.

This is not needed as customers want Transport Tender to carry out the whole process. However, proper reporting should be provided for the clients to present insightful arguments for choosing a set of carriers.

During the design of the application we must make sure the application is easily maintainable so external developers can easily extend and maintain the application as Transport Tender does not have a development team of their own. The number of combinations of carriers we can make grows quickly as the number of carriers increases in a tender. Therefore we must make sure the performance of the application is adequate. This means processing large inputs should not affect the page load times and responsiveness of the web application.

For the project to be a success, it should provide an application that handles the two major challenges: usability and performance. Big amounts of data should be able to submit easily into the application and afterwards the analysis should be generated quickly.

3 Design

3.1 Overview

The section design discusses the chosen design of our application. First a high level overview of the total application will be given (Section 3.2). Next we will elaborate on the implemented design patterns (Section 3.3) and architecture, learned in the Software Engineering Methods course. A must have is securing the application with an authentication system, to prevent unauthorised access (Section 3.4). Furthermore the overall structure of the web pages will be explained (Section 3.5). An important part of the design is the setup of the data structures and how to parse this data correctly (Section 3.6). And we discuss the main algorithms for executing the tender analysis (Section 3.7).

3.2 High Level Overview

The application is mostly based on the design principles of the used Play Framework. This framework uses the Model-View-Controller (MVC) pattern, which is explained in Section 3.3.1. Therefore the application is divided in those three main packages, see Figure 1. The user interacts with the front-end using the view of the application. And the front-end communicates with the back-end using the controllers.

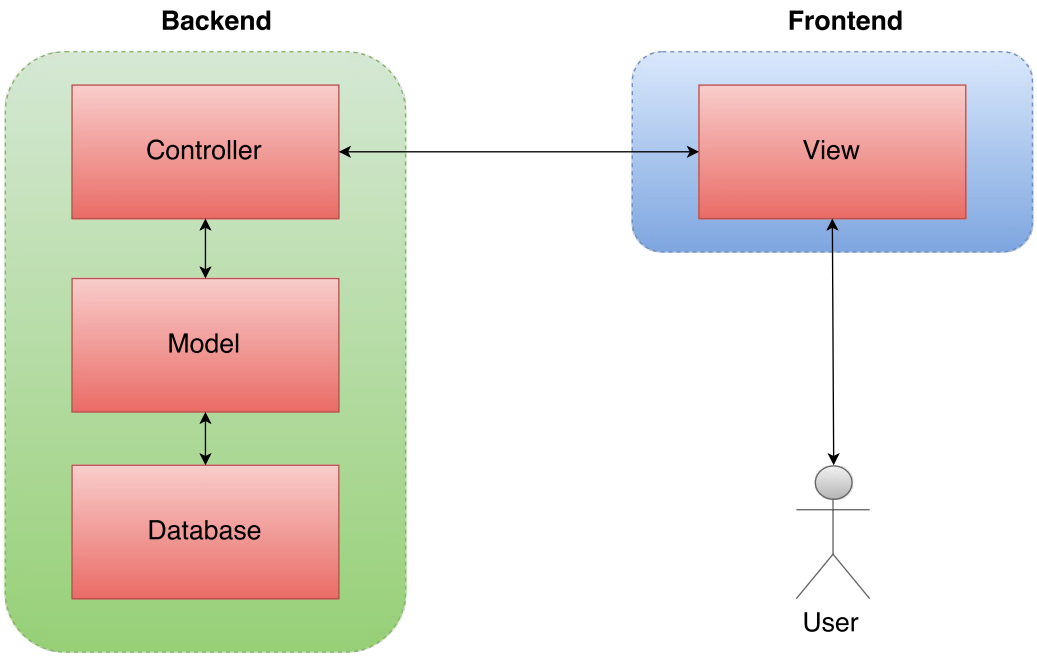

Figure 1: High Level Overview

3.3 Design Patterns and Architecture

For software development it is common to use software design patterns. These patterns are generally a reusable pattern for solving reoccurring problems. We discuss the most significant patterns in the design of the application and the architectural decisions we made.

3.3.1 Model View Controller

One of the most famous design patterns is the MVC pattern, where one splits the application in the three designated parts: model, view and controller. This is also the case in our application

where we will split the business logic, control flow and views, mainly because Play enforces this and it is considered a best practice to use in web applications.

The model is where the business logic happens. Here we have different objects representing real world objects, e.g. tenders, carriers, companies and rates. For all those objects we have classes that take care of saving them in a database, see Section 3.3.3. Next to saving and retrieving the objects, the package model will also take care of processing the data and doing the actual analysis.

Views display the information on the screen of the user. Because we are building a web application, this will be a HTML webpage with all the relevant information per page, see Section 3.5. Also, input from the user in the views is passed on to the controllers to process actions from the users. This can for example be retrieving and creating tenders or setting the status of a tender.

3.3.2 Asynchronous Requests

Our application processes a lot of data and takes up a lot of computational time. To avoid blocking the entire system while processing such data we will use asynchronous I/O. Asynchronous I/O means processing in- and output while allowing other processing to continue before completion of the request. Play uses asynchronous requests for its webserver, instead of tying up HTTP threads doing business logic like in traditional environments using Java Enterprise Edition (JEE). The advantage of this architecture is that the webserver can handle more HTTP requests on a single thread.

We will set up new threads for each computational heavy task, for taking care of processing the data. During the processing the user and other users will still be able to use the system, without blocking the entire system. This will be an improvement from the current situation where the application is not responsive while calculating the analysis for all sets of carriers.

3.3.3 Database Abstraction

To store and retrieve data for the application we will use a database. In our design this can be any database system; H2, PostgreSQL, MySQL, and even an in-memory implementation using e.g. Maps and Lists for storing relational data. This can be achieved by using an abstracting layer with a Data Access Object (DAO). The use of an abstract interface enables us to call functions to store or retrieve objects without caring how they are actually stored in the back-end.

In our initial design we also had a service which communicated between the controllers and DAOs. However, we already had an abstract interface which we could use for each implementation. Therefore, we made the decision to delete this extra abstraction layer, because it gave too much boilerplate code and the interface already did the same, which can be seen in Figure 2.

3.3.4 Dependency Injection

To select the right implementation for the DAO and the authentication environment we make use of Dependency Injection (DI). In the software engineering context DI is a design pattern which implements the inversion of control. Injection mostly means injecting a dependency or service in a dependent object. In our case this means passing on a specific implementation of a DAO into a controller object. This enables the controller to use that class to interact with the database. We will use the implementation from Google, as it allows binding implementations of classes to an interface which we can use in the controller [5]. It is also the first generic DI framework using annotations for Java, which can be used in Scala.

Based on the specified configuration variables in the configuration file of the application we must select the corresponding implementation of the DAO. This enables us to quickly switch between the method of storing data, without adjusting any code. Or using specific implementations for the development, testing or production environment. Not only for the database, but also for the authentication environment, where we can easily switch between different authentication providers based on the client's needs.

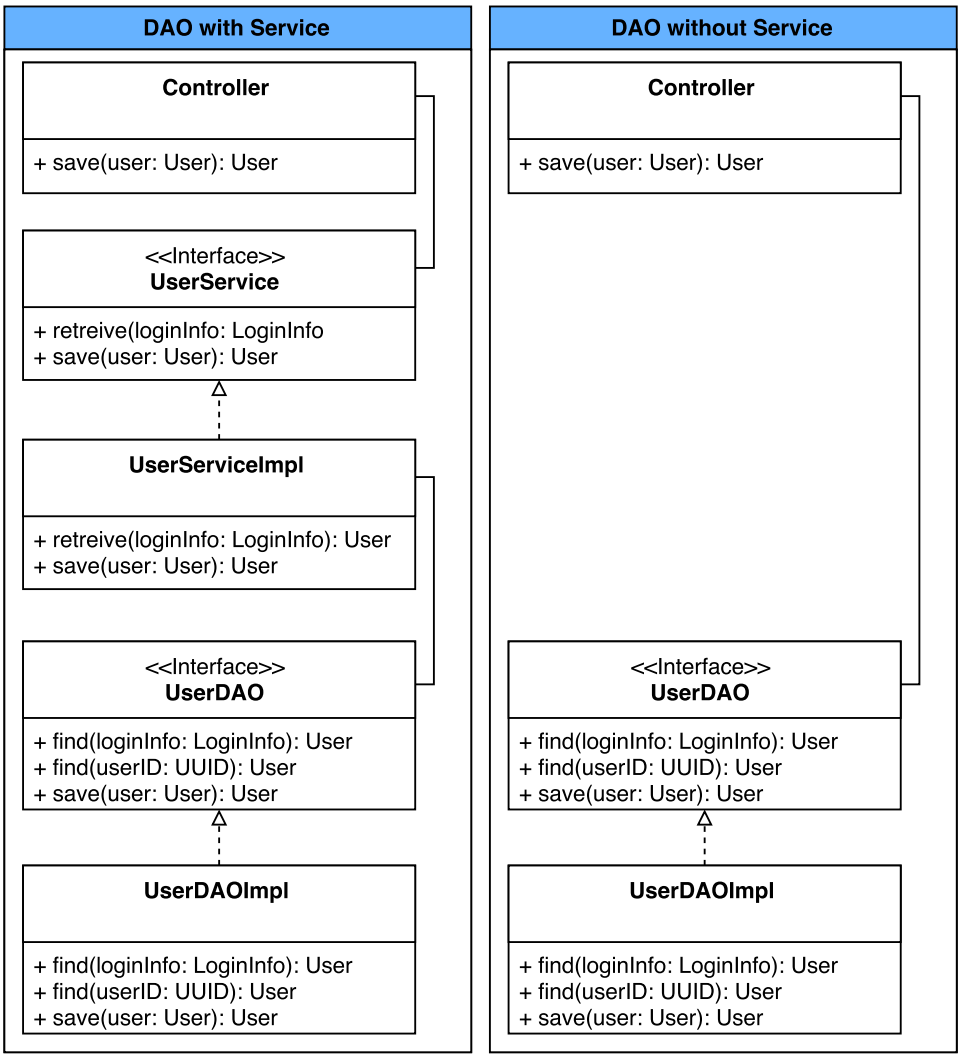

Figure 2: DAO with and without Service

3.4 Authentication

An important requirement is that the application is login protected using credentials to prevent unauthorised access to the system. Play provides a simple to use authentication system using action composition to authenticate actions. However these authenticators are not very extensible. Therefore we have chosen another library which will provide the authentication.

Silhouette[2] is such a library which provides authentication and authorisation with support for several authentication methods, including Credentials, OAuth and OpenID. It makes use of an environment which consists of the identity, authenticator and services implementation. The environment can be set based on the configuration file, using DI. This makes it a flexible and extensible framework for implementing the authentication based on the customers needs and current infrastructure.

3.5 Webapp Page Structure

To build a web application we need a decent page structure to easily navigate between different webpages. To support the authentication, we have four pages:

[2] `http://silhouette.mohiva.com/`

`/login` Here users can supply there credentials to log in. After a successful login the user is taken to the homepage. When the user session exists, the user is automatically redirected to the homepage, without having to log in again.

`/logout` The user session is removed and the user returns back to the login page.

`/signup` Create new users for the application.

`/dashboard` View the information about the current user.

The homepage shows a overview off all tenders in the form of a table. This table can be easily sorted, filtered and searched on the properties of a tender. The user can click a tender to show more information about that tender. On the tender pages there are three sections:

Tender which lists all the properties of a tender and all the given surcharges, comments and carriers.

Shipments shows a button to add shipments and displays all the countries when shipments have been added.

Tender rounds displays the status of each tender round.

Shipments can be added per tender on a special page, with a spreadsheet like table in the browser. When submitting the shipments, the server verifies the format of each shipment. Feedback is returned to the user when mistakes have been made. The tender round page is the most important page for submitting rates for each carrier and country. It display the progress of each carrier on how many rates have been submitted. When all rates are submitted, the user can start the process of analysing all the data. On completion the user can view all calculated sets of carriers to select the best deal for the client of the tender.

The url structure is as follows:

`tenders/:t` display the tender with the id t.

`tenders/:t/shipments` display the shipments for that tender.

`tenders/:t/rounds/:r` display the tender round r for the tender.

`tenders/:t/rounds/:r/rates/:c` an overview of countries per carrier c in that round.

`tenders/:t/rounds/:r/rates/:c/:l` the actual rates table per country l per carrier.

`tenders/:t/rounds/:r/analysis` view the result of the analysis of the round.

As can be seen in the structure above, the url builds up from the basis, a tender. And each time the next part in the url specifies the next element of a tender. This makes it easy and transparent for the user to see where he is in the process.

3.6 Parsing Data

The value of the application is heavily dependant on the ease of uploading data. As described in Section 2.3, two main types of data are importent: shipments and rates. These data are commonly provided by carriers using spreadsheets. However, almost all carriers use a different format in these documents. This makes it hard to import the data into the application and therefore a standardised format is designed. Using this standardised format, the work necessary to set up a tender is kept as low as possible.

Since shipments and rates are supplied in spreadsheet formats, the design of the input and parsing of the data in this application is also based on spreadsheets. Because the application is web-based, this was a challenge. To increase the usability of the application, for both shipments and rates a web page is designed where data could simple be pasted in after copying it from spreadsheet software. This is a big increase in usability in comparison to the clients current application. In that software a template file had to be downloaded, opened with spreadsheet software and filled in and after saving it had to be uploaded again.

After these steps, feedback was presented and if invalid data was provided, the complete procedure had to be repeated to make any changes. This made the process of submitting shipments and rates very intensive, since some tenders contain ten carriers and shipments in more than ten countries. This means over one hundred rates tables should be submitted in those kind of tenders. The formats used in the applications input pages for shipments and rates tables correspond with the table structures in Table 2 and Table 3, respectively.

3.6.1 Shipments

Shipments at least require the following properties to be properly parsed: location, date, load quantity, surcharges and comments. The location is indicated by two parts: a country code and a postal code. The set of countries that is accepted by the application is limited, but configured using a JSON config file and therefore easily extendible. Postal code validation is based on the provided country, since countries have different postal code formats. Two groups of postal code formats are specified yielding a set of in total five different postal code formats to cover the applications range of countries:

Numeric postal code formats apply for most of Europe's countries. These include postal code formats containing four, five or six digits. Some countries even use characters after those digits. These are not stored, since only the first two digits of a numeric postal code format are used to indicatie its location.

Alphabetic postal code formats apply for the United Kingdom and Ireland. Those countries use a different postal system and therefore only regions of alphabetic zones are parsed. In the case of Ireland this means the region name, e.g. *antrim*. The United Kingdom uses one or two letters followed by some more alphanumeric characters. Only the first one or two letters are used to indicate the postal code's location.

An overview of the application's accepted countries and their corresponding postal code formats is given in Table 6.

Type	**Subtype**	**Example**	**Countries**
Numeric	4 digits	1234	BE, BG, DK, HU, LV, LI, LU, NL, NO, AT, PT, SI, CH
	5 digits	12345	AD, BA, DE, EE, FI, FR, GR, IT, HR, LT, UA, PL, RS, SK, ES, CZ, TR, SE
	6 digits	123456	RO, BY
Alphabetic	Ireland	antrim	IE
	United Kingdom	AB	UK

Table 6: Postal code formats per type and country

Based on the shipment type, the load quantity is measured. In the case of kilograms, only an integer value is necessary and parsed. In the case of units, a floating point value is needed. And in the case of load meters, a combination of the two are needed: an integer value indicating weight and a floating point value indication volume.

Dates are only used for the summing of shipments. To indicate if two shipments can be summed, both location and date are compared. Therefor dates are not required to match any format at all, as shipments to the same location and the same date use the same format. It would be very hard to formulate a single format all dates should satisfy and it might increase processing work of shipments since dates might be converted into another format. Since the only processing on these dates is checking for equality, storing them as strings was sufficient.

A variable amount of surcharges and comments are possible per tender. Depending on the amount of surcharges and comments, that amount of extra fields should be submitted per shipment. In the case of surcharges this could be "yes", "1" or "true" or their complements. For comments, every string is accepted.

3.6.2 Rates Tables

The delivery of rates tables is also mainly done by using spreadsheets. As described in Section 2.3.6, rates tables are two dimensional with one dimension for both location and load amount zones. Location zones are submitted via the application by using a text field. In this text field, zones can be specified by putting a sequence of comma separated postal code ranges per line. Each line represents a location zone covering all the postal codes within the location zone's postal code ranges. For numeric postal code formats, the postal code ranges are identified by two digits. For alphabetic postal code formats, the postal code ranges are identified by the regions. When the user is constructing these zones, a list of required zones is provided by the application so the user knows how much work there is left to do.

After creating the zones, a table is constructed like Table 3. This table works the same like the spreadsheet like table for shipments. Rates can be easily copy pasted from an application like Excel. This also increases the usability of the product a lot, because the process of downloading and uploading files is no longer needed. Rates are provided as integer amounts of euros.

Additionally, per rates table, a minimal, maximal en percentage value per surcharge should be provided. Percentages are represented by integer values in the range 0 to 100, minimal and maximal surcharge amounts are represented by an integer amount of euros. If the tender's load measuring type is by load meters, a maximal weight per load meter should be provided per rates table too.

3.7 Algorithm Design

To be certain of the applications accuracy, we have chosen to make the main analysis algorithm as simple as possible. In short it means that based upon the input data, a collection of subsets of the tender's carriers is produced and the relative prices for each shipment are calculated. By doing this, it is easy to see which set of carriers is the cheapest per set size. This means that for each set size, starting from 1, the total turnovers of shipments for these sets are calculated and the prices are compared with other sets with the same size. The cheapest set of a certain size is selected and reported to the client.

An improvement of this calculation is done by checking if the provided sets of carriers cover all the countries of the tender. E.g. in a set of carriers the carriers might not have rates for so many countries, that there is a country that no carrier in the set has rates for. In that case, the carrier set is not evaluated at all, since the complete shipment portfolio can not be tackled by such a set.

This analysis has two subtypes: analysis on country level and shipment level. An analysis on country level means that all of the shipments in a country can be handled by only one carrier. With an analysis on shipment level, per shipment the cheapest carrier is chosen. The results of analysis on shipment level is usually a cheaper distribution of costs. However, some clients want to have a maximum of one carrier per country for their transport and therefore the analysis on country level is used.

4 Implementation

4.1 Overview

Implementation discusses how we actually implemented the application. We will start with a basic overview of the structure (Section 4.2). Next we will elaborate on the libraries we used to include in the project (Section 4.3). A vital part of a web-application is the communication between the front-end and back-end and how we implemented this (Section 4.4). Furthermore, we will explain our deployment strategy (Section 4.5) and which tools we used (Section 4.6).

4.2 Structure

The application is divided in three main parts, specified by the MVC pattern, see Section 3.3.1. The *controllers* package contain the controllers which provide the communication between the front-end and back-end (Section 4.4). Package *views* are divided in different directories based on the url structure as specified in the design (Section 3.5).

A small package is the package *forms*, here reside the definitions of all the used forms in the views. They are in a separate package as they interact between *controllers* and *views* for passing form-data between them. Another small package is the *utils* package which contains the environment configuration for the authentication library (Section 3.4). Also it contains the DI configurations for the DAOs (Section 3.3.4).

The largest part is the *models* package where all the models, data structures and DAOs are defined. The *models* package contains the packages as described in Table 7.

Package	**Description**
analysis	Execute the actual analysis and return the results.
daos	The database definitions and the DAOs reside here.
geo	All the different postal codes and countries are defined here.
parsing	Parses data from a CSV file.
rates	A data structures for saving different rates for different types of shipments. Including load and postal code zones.
shipments	Takes care of all the shipments and its dependent objects, Surcharge and Comment. Furthermore it can parse different types of shipments.
models	Contains all the basic models for the application, e.g. Tender, Carrier, Company.

Table 7: Packages in models

4.3 Libraries

The Play Framework already includes a lot of dependencies which we can use for our application. For example the Jackson JSON Processor library is already included by default. Next to the the built-in dependencies we also use other libraries to prevent us from reinventing the wheel. These are libraries built by external developers, already intensively tested and used in many projects before. To manage all these dependencies we make use of Scala Built Tool (SBT), which is explained in Section 4.6.1

Next we give an overview of all the included libraries in our project, these can be found in Table 8. Furthermore we also make use of plugins for integrating Heroku (Section 4.6.4), scoverage (Section 5.6) and Coveralls (Section 5.8.3) into SBT, to provide us instantly with the right command for running those tools.

4.4 Communication Front-end with Back-end

The front-end communicates with the back-end using the controller package from the MVC pattern. This can be seen in Figure 1. When a user requests a url, the right controller is chosen based on

Library	Description
webjars-play	A library to make use of WebJars in our framework, WebJars are client-side web libraries packaged into Java Archive (JAR) files.
bootstrap	A popular front-end web application framework for building responsive websites.
jquery	The most used JavaScript library to make client-side scripting of HTML easier.
bootstrap-table	An extension to the Bootstrap tables with extra features, such as sorting, searching and pagination of tables.
handsontable	To provide an Excel like spreadsheet in a browser using JavaScript.
play-bootstrap3	A collection of input helpers and field constructors for Play Framework to render Bootstrap 3 forms with error reporting per field.
opencsv	A simple CSV parser for Java.
commons-io	Java library with utilities to assist with development of IO functionality, especially used to process InputStreams of files.
scala-guice	To enable DI in the Play Framework, as described in Section 3.3.4.
play-slick	Makes the Slick module a first citizen of Play for easier integration, which is used for database access, see Section 3.3.3.
postgresql	The JDBC driver for connecting to PostgreSQL databases.
play-silhouette	The authentication library for our application, see Section 3.4.
play-silhouette-testkit	A testkit for Silhouette to test the authentication.

Table 8: Used libraries

the routes file. We can easily change the string of a part of the url to point to a different controller. This makes it a flexible way to generate the navigation. The controller responds with a request, which is returned to the user. Controllers can return different statuses based on the request made and on the state of the application.

We not only make use of standard HTTP request by users from navigating through the application, but also provide a basic API for returning JSON formatted data. Which is mainly used for tenders and shipments to retrieve that data using JavaScript, after the page has loaded. This allows us to first render the basic page and serve that to the user. Next using asynchronous requests we can display the information on the page. The advantage is that the basic page is rendered for the user and the content is loaded after is has been retrieved. Which is especially useful when retrieving a large set of shipments. This prevents long page loads, which might look as if the application is unresponsive.

4.5 Deployment

For deploying our application we first wait for the continuous integration to accept the changes, meaning all tests should pass before deploying. This is done by using the webhooks from Github and Travis (Section 6.8), they trigger the automatic builds on Heroku (Section 4.6.4). We use Heroku as our staging server. A staging server is a mix between a development and production environment. Here we can test our application as it were running in production mode. This enables us to see if our application actually works on a real server, and not just only on a local development machine. This process is visualized in Figure 3.

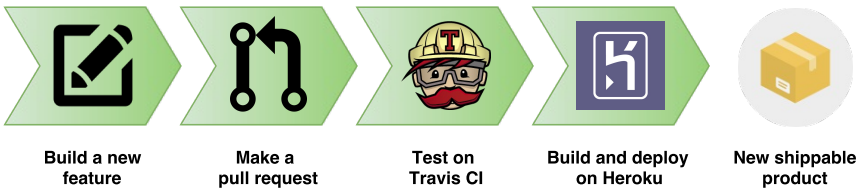

Figure 3: Deployment flow

Furthermore, the client can see and accept a new feature before deploying the feature immediately to production. Preventing us from making mistakes in a production environment. Next we can manually deploy the new version to a production server, ready to use by the client.

The production environment needs three components, see Figure 4:

- **Application server:** The server where the application itself runs on.
- **Database server:** This can be any server running a database, preferably in the same location to provide quick access and the same type used in development and staging.
- **Proxy server:** A server which proxies the incoming web-requests from port 80 (HTTP) and preferably port 443 (HTTPS) to the application server, running on a different port or server.

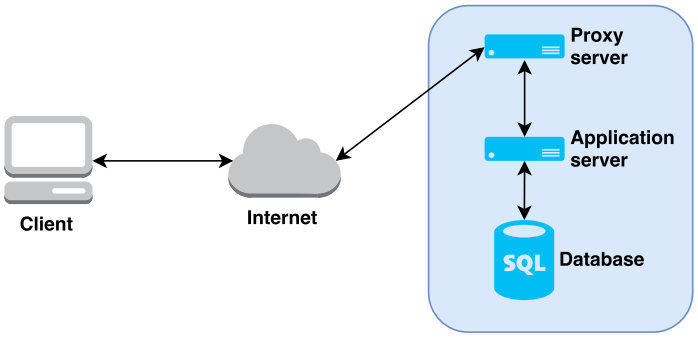

Figure 4: Production environment

It is not needed to have three different physical servers, these can be combined on one server, or installed on virtual machines. Our client does not expect to need a lot of resources, as the company is still small. However we could easily add more application and database servers for extra performance or redundancy.

4.6 Tools

For implementing the application we use a selection of tools to make development easier. We make use of a build tool, version control, an Integrated Development Environment (IDE) and a staging server.

4.6.1 SBT

We use SBT[3] as our build tool for the application. It is an open source build tool especially for Scala and Java projects, with resemblance to Maven or Ant for Java. It supports compiling, building Scala code and even testing using various testing frameworks (Section 5.8.2). Furthermore it does dependency management for our libraries (Section 4.3) using Ivy, which is a transitive dependency manager with support for Maven-format repositories. It supports incremental compiling, meaning it detects source file changes and automatically compiles, tests and deploys our code. This allows for faster development, as we do not have the overhead from starting SBT on every code change.

Play provides us with a wrapper for SBT, called Activator. It adds a quick-start UI where we can easily control SBT using an web application. We use SBT and Activator as they are the de facto tools for managing Scala and Play projects.

[3] `http://www.scala-sbt.org/`

4.6.2 Git

We use Git[4] to manage versions of our source code. Git was chosen as it is the most popular instrument used by developers and the team has extensive experience from previous projects. It also allows us to work on the code simultaneously. We adapted the feature branch workflow, which means we make a new branch for each new feature. This makes it easier for developers to work on their own feature. After a feature is finished, a pull request is made to the master branch. Before merging it with master we can review the changes, to prevent a broken master branch. When the review is finished the new feature is merged with the master branch, ready to deploy.

4.6.3 IntelliJ IDEA

The writing of our code happens in IntelliJ IDEA[5], an IDE for Scala and Java. IDEA has support for a large number of enterprise frameworks, including Play. Furthermore it has numerous built-in developer tools which makes the life of developers easier. It supports build tools, version control systems, and database tools out-of-the-box. Web development tools are included as well, as there is coding assistance for JavaScript, HTML, CSS and many more. We choose IDEA not only for all the built-in tools, but also for the intelligent and instant code completion and refactoring tools. Which enables us to write faster and better code, without having to compile each time to check for syntax errors.

IntelliJ IDEA ships in two versions, community and ultimate edition. The ultimate version includes the built-in tools described above and support for many more, whereas the community edition only includes some of these features. We make use of the ultimate edition, making it easier for us to write good code. Students can obtain a free license for the ultimate edition, other developers should purchase a paid version of the ultimate version.

4.6.4 Heroku

Heroku[6] is a Platform as a Service (PaaS) with support for the most popular programming languages and it also includes native support for Play projects. Using webhooks from Github we can automatically push code from our master branch to Heroku, without any intervention from developers. It retrieves all the needed dependencies for our framework and application using SBT, based on our configuration files.

Applications run in so called Dynos, which are small virtual machines running in the cloud. We use the free development Dynos for our staging area. However, there are some limitations; the Dyno sleeps after 30 minutes, has to sleep for 6 hours per 24 hour period and the PostgreSQL database has a row limit of 10,000 rows. This is enough for the use as a staging server, however not for using in production. The client could sign-up for a paid plan or rent a server elsewhere.

[4] `https://git-scm.com/`
[5] `https://www.jetbrains.com/idea`
[6] `https://www.heroku.com`

5 Testing

5.1 Overview

Testing has been an important part of this project. The final product was developed using a test-driven development approach, see Section 6.3. It was intended to fully test new features and functionalities before they where merged into the master branch.

Due to time considerations, testing has been done focused (Section 5.2). It helped to cope with changes being made to the software's architecture (Section 5.3). In this section, three main types of testing are discussed: unit testing (Section 5.4.1), integration testing (Section 5.4.2) and functional testing (Section 5.4.3). Tests of these types are coded into the software and can be executed automatically. The client provided us with data of an example tender. We describe how we used this data to test the software in a bigger perspective (Section 5.5). Furthermore, test coverage is measured and the results are discussed (Section 5.6). Also, we describe the use of static analysis to get informed about the overall quality of our software and how we used this to prepare and improve the software for an evaluation by SIG (Section 5.7). Lastly, the tools that are used to support testing are described (Section 5.8).

5.2 Determine Focus

In an ideal situation, every part of the source code would be completely tested. However, due to the limited timespan of this project, the choice is made to focus more on some parts than others. The most essential part of the software is to give an accurate calculation of prices and to analyse carrier set compositions. Therefore, most testing effort was put in testing computations and data structures related to those functionalities. These functionalities originate from the *model* part of the application's architecture, see Section 3.2.

The authentication part is less tested to save time. Since authentication is implemented using a third party tested library, leaving it less tested should not compromise the final product's quality too much. The library itself is tested extensively and provides us with an easy to use fake environment to test the basic authentication of the application.

5.3 Preparation for Change

Often during development, class compositions change or the application's structure is refactored. Because unit tests can be easily executed, they can provide feedback on such changes directly. E.g. some tests might fail if a change is not carried out consequently within the whole application. It helped preventing the need of fixes later on in the project and thereby increased overall productivity.

5.4 Main Testing Types

The three main types of testing that are used in this project are unit testing, integration testing and functional testing. These testing types build upon each other.

5.4.1 Unit Testing

With unit testing, individual units of code are tested with multiple test cases. Typically methods, functions or other small pieces of functionality are tested. It is useful to determine whether specifications are implemented correctly and fulfill requirements in diverse scenarios.

5.4.2 Integration Testing

A step above unit testing is integration testing. At the integration testing level, multiple classes are combined and interactions between them are checked. Much integration testing in the software of this project has been done in the data stuctures and database abstraction layer. Testing this was essential for laying a good fundament for performing calculations.

5.4.3 Functional Testing

Functional tests differ from unit tests and integration tests in that they check whether features return correct results without considering side effects. This way of testing was extremely important in this project, because it was the main testing type to check whether price calculations and analysis outcomes were right. Therefore, most effort is put in this type of testing. For functional tests to have a real added value, relevant test cases should be made. These test cases are partly made up to cover as much scenarios as possible. To test the correctness of prices of shipments and outcomes of analysis, data from the client is used. This data is used to perform a complete analysis and compare the outcomes to the results specified by the client.

The final product is a web-based application. Since the application is used from within a browser, specific web testing should be applied to test its functionalities correctly. These tests are executed using the tool Selenium, see Section 5.8.1. It helps executing tests that simulate actions performed by the user and it checks if content is presented correctly to the user.

5.5 Example Data

To be able to execute good functional tests, relevant test data is essential. To make both automated and manual testing efficient, a suite of test data has been set up. This has been done by using a set of generic test objects throughout the automated tests and by using real world tender data provided by the client.

5.5.1 Test Objects

Since many classes require each other to be functional, even some simple integration test might require a lot of objects. Mock objects did not satisfy in this case to execute functional tests properly. To facilitate the functional tests and make the testing process easier and more efficient, a set of generic test objects are made. These generic objects were only used if they were not specific to the test's purpose. Otherwise, the use of generic objects would compromise the test's accuracy. Using these objects, integration and functional tests could remain relatively small and test code duplication was kept as low as possible.

5.5.2 Test Tender

Since the client is experienced in the field of transport tenders, ready to use data was available. This data was provided by the client under the condition that it would be used anonymously. An extension to the current application of Transport Tender is written in PHP to export the data in JSON format. The new software features a JSON parser for importing the test data.

This test data is used in both automated test cases as in a running version of the final application. Firstly, by using it in functional test cases, an extensive test on realistic data could be automatically performed. This made it easy to see variations in both outcomes and performance as the software changed. Secondly, due to the size of the test data, it was a good resource to test the application's usability. By submitting data by hand, like a real end user would do, usability bottlenecks could be found. Also, it gave a good feeling of the application, as it represents a real world use case scenario.

5.6 Code Coverage

As stated in Section 5.2, some parts are more intensively tested than other parts. This is done to increase both test accuracy and overall productivity of the project. To get an idea of which parts are tested and how much, test coverage measurement is used. It helps indicating which parts of the software are being hit by tests, forgotten to test or are not tested yet at all. It also helps to be able to test focused.

We not only performed classic line coverage like in common in languages such as Java. But mainly used statement coverage for Scala, as multiple statements or branches can be included in a single line. Measuring statement coverage means testing every option of a statement. This measure is included in our tools by default.

An overview of test coverage per package can be found in Table 9.

Package	Coverge
app	43.75%
controllers	83.75%
forms	100%
models	71.14%
models.analysis	98.23%
models.daos	99.31%
models.daos.definitions	100.00%
models.geo	100.00%
models.parsing	100.00%
models.rates	100.00%
models.rates.parsing	100.00%
models.shipments	100.00%
models.shipments.parsing	100.00%
utils	61.34%
view.html	88.24%
view.html.analysis	38.07%
view.html.b3	84.00%
view.html.rates	60.23%
view.html.shipments	93.22%
view.html.tenders	63.24%
all packages	84.32%

Table 9: Test coverage per package measured by scoverage

5.7 Code Quality

While testing indicates if code does what is should do, it does not say that much about whether code is of high quality. Producing high quality code increases productivity of the project and maintainability of the software.

We measured code quality using several tools. By optimising or solving problems indicated by the quality measures, we tried to increase code quality. Since SIG uses similar quality measures, it helped us in getting an as good as possible result in the first evaluation by SIG. These measures have been done multiple times a week to tackle arising problems as fast as possible.

5.8 Tools

The tools described below are used in the testing process of the application.

5.8.1 Selenium

It is built in the Play Framework and therefore it was easy to implement functional tests using Selenium, as it did not require additional configurations. With Selenium, it is possible to simulate actions performed by a user in the Document Object Model (DOM) in a real browser. Selenium was chosen both because of its easy integration with the Play Framework and because of its additional test options related to the application's interface.

5.8.2 scoverage

The code coverage tool scoverage[7] is an extension to SBT that produces both statement and branch coverage. The coverage is measured by a percentage of statements and branches that are called by automated tests, respectively. It runs directly in SBT with an additional command and reports

[7] `http://scoverage.org/`

are automatically generated in HTML, so they are easily viewable in the browser. scoverage is used because it was easy to use in combination with SBT and actually was the only mature option available for Scala coverage testing.

5.8.3 Coveralls

As an addition to scoverage, the premium web service Coveralls[8] is used. The main functionality of Coveralls is the same as scoverage: giving an overview of what parts of the codebase are tested. However, Coveralls is integrated with Github and Travis and uses this to extend the basic functionality. It makes it possible to view test coverage per Git branch and a pull request is checked in Coveralls automatically before it is merged. This check looks at the change in test coverage and if the coverage has dropped, a warning is produced. Examples of Coveralls notifications are given in Figure 5. Coveralls is used because it gives a better overview of test coverage per branch of the repository, it was easy to add to the continuous integration pipeline and it warned when test coverage dropped. We used a plugin from scoverage for SBT to push the results automatically to Coveralls.

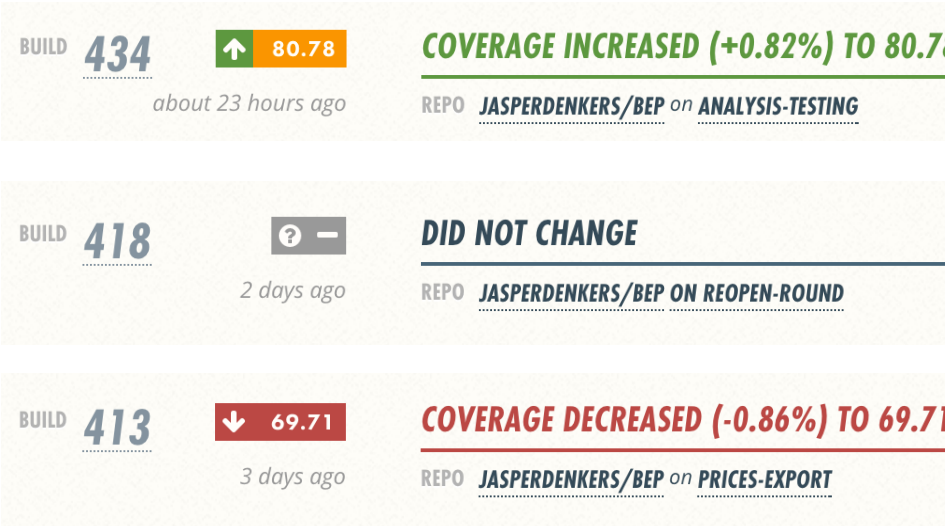

Figure 5: Coveralls coverage change notification examples

5.8.4 Sonarqube

Sonarqube uses the results of scoverage to perform its analysis on the source code. It is used by setting up a Sonarqube server during the project on one of the students private server. While this server was set up, it was easy to perform code analysis from the student's development devices. Sonarqube was chosen because it is free to use, easy to run and it gave detailed analysis for the team to prepare the software for evaluations by SIG.

[8] `https://coveralls.io/`

6 Process

6.1 Overview

In this section, process related subjects of this project are discussed. The main development methodology was Scrum (Section 6.2). To be sure there always was a tested working product for review by the client, we used test-driven development (Section 6.3) and continuous integration (Section 6.4). Some important parts of the code are implemented using pair programming (Section 6.5). Well documented code made sure team members could understand and use each others code and the projects maintainability was increased (Section 6.6). During the project a mid-project meeting has been held with the client and supervisors (Section 6.7). Lastly, the tools we used to support the process are discussed (Section 6.8).

6.2 Scrum

The popular software development methodology Scrum is used to iteratively and not sequentially develop the final product. This methodology is called agile and it helps recognising the change of requirements early. It works well in combination with test-driven development (Section 6.3) and continuous integration (Section 6.4).

The requirements formulated in the research phase of the project were transformed into the product backlog, which was centrally administrated with the tool described in Section 6.8.1. Sprints took one week and per week a planning was made of what work had to be done in the coming week. In the daily scrum meeting the student teamed discussed progress from the previous day and plans for the current day. This was mostly a short informal discussion between the two students in the morning.

6.3 Test-Driven Development

Test-driven development fitted perfectly in the agile development methodology Scrum. It means that before a new feature or change is implemented, its tests are written first. Subsequently, the feature or change is implemented and before it is committed to Git, all tests should pass. This includes both automated tests (see Section 5.4) and manual tests.

Since the software heavily depends on data structures, it was hard to always write tests before a change is being implemented. This is because a data structure is needed before a test could be written at all. However, analysis and computation parts of the software were mostly implemented using the test-driven development approach. Writing tests first made the development team think well about how to implement a feature, a nice advantage of test-driven development. Besides that, only committing tested changes reduced the amount of bugs reaching the master branch. It did still happen however that changes were committed containing bugs. E.g. by not committing all changes in a local repository, or forgetting to run all tests before committing and pushing to Github. The continuous integration setup helped coping with this.

6.4 Continous Integration

Characteristic for Scrum development is to always have a working copy of the software. To support this, continuous integration is used. It means that at a high frequency, changes by all participating developers are merged into a single master branch that constantly also contains changes by other developers. This works well with test-driven development, as merges can only take place when they are successfully tested. Additionally, continuous integration automates the test process on a clean environment. A running copy of the software might work on a developers machine because of some local resources that are not in the repository. In this case, tests will fail on a continuous integration build. The advantage of continuous integration is that it helps catching those issues early and in particular before being pushed to production.

Methodologies related to continuous integration are continuous delivery and continuous deployment. Continuous delivery differs from continuous integration in the sense that it does not focus on merging every change in a master branch. Actually, it focusses on delivering every change to a certain user base. In this project we could say continuous delivery is used with the client as the

user base. It helped keeping an up to date version of the software that could be used and reviewed by the client. This made an iterative approach like Scrum more effective since the frequency of moments where the client could give feedback, including remotely, was increased and requirement misunderstandings were catched early. With continuous deployment, every set of changes classified as ready are immediately pushed to production. Since this project is not taken into production during the timespan this report is about, this was not applicable. However, when the project is going into production and would be used and extended, continuous deployment could be applied.

6.5 Pair Programming

Pair programming means that two persons together implement a piece of code. Because the team for this project consisted of two people, it was inefficient if not impossible to implement the complete software using pair programming, this would lower the average productivity of the team too much. However, important parts of the code have been developed together. These parts included database abstraction, communication between back-end and front-end and main data structures and algorithms. In particular the implementation of data structures are implemented this way. It helped preventing wrong architectural choices as soon as possible and it gave both of the team members a good understanding of their implementations. This was helpful, as data structures are used most by both team members during development of other parts of the software. Some parts that were developed more individually by one of the students are not implemented using pair programming, like front-end design. However, we did frequently discuss different options for the front-end to see what works and looks best.

6.6 Code Documentation

By using pair programming for implementing important parts of the software, both the quality and understanding by the complete development team of these parts increased. However, as described in Section 6.5, it was not realistic to develop the complete software using pair programming. Therefore, most parts are developed individually by one of the team members. To make the implementation and functionality of those parts understandable and usable by others, they are documented. We used clear naming for methods, parameters and objects, and inline comments describing the code's functionality. Also, this documentation increased the maintainability of the project. If in the future the project wil be extended by other developers, undocumented code would be hard te work with. Since maintainability was a wish of the client, time is invested in documenting the code.

6.7 Mid-Project Meeting

Halfway during the project, a mid-project meeting was held. In this meeting, the student team, client and supervisors discussed progress of the project and everyone indicated risks. Also, it was a good moment for the client to meet the supervisors before the final presentation. Risks included whether the project would be finished on time and whether the client's innovative idea would not be revealed too much in the final report's contents.

After the mid-project meeting, there has been more contact with the client and due to adjusting the requirements that were planned to implement, the project has been finished on time. Since the application does not feature a specific algorithm that is innovative, it is hard to not exclude parts of the final report before publishing it. Actually, the innovation in the developed system lies in the smart combination of data and an intuitive interface. The process of tenders is not patented since it is too simple and even described on the client's website. Therefore this risk turned out to be not applicable for this project.

6.8 Tools

The tools described below are used to support this project.

6.8.1 Trello

Trello[9] was used to organise both development and process related tasks. Using an online interface the board was used by both of the team members to plan and divide work. The Scrum product backlog was also administrated in Trello. It proved to be a valuable instrument to support Scrum. Besides the product backlog, an overview of done tasks was tracked per week. Additionally, the supervisor of the project also had access to the Trello board. This gave the supervisor more accurate capabilities in tracking progress and problems. Also, the board gave for both the student team and the supervisor an overview of important dates and deadlines.

6.8.2 Github

Git was used for version control for both code and documents, see Section 4.6.2. Github[10] was chosen to support the development with Git since it is the most popular hosted Git provider and because of the team members extensive previous experience with the platform. With a student license, Github was free to use with a private repository. A private repository was required as the client wants to protect the source code from being available to its possible competitors.

6.8.3 Pull Requests

Pull requests seemed very effective and were used more often while the project advanced. When using pull requests, for each feature that is going to be implemented, a new branch is created. This new branch is used just for committing changes related to the feature, which is usually described by the branch's name. When the feature is successfully implemented and tested, a pull request into the master branch is requested via Github's web interface.

Once again, the continuous integration tools are automatically triggered to check the new feature. These checks are different than checks in the pull requests branch, since they test the application with the new code merged into it. Lastly, when these checks return positive, the feature is merged into the master branch and the pull request is completed. Usually this was not done by the initiator of the pull request. When the other part of the student team merged the pull request, every feature merged into the master branch was evaluated by the whole team. This seemed effective since it reduced bugs from getting into the master branch and therefore from getting into the final product. Besides that both the team members where constantly informed about each others work and the state of the master branch.

6.8.4 Travis

Travis[11] supported continuous integration. It was chosen for continuous integration due to its easy integration with Github and Coveralls, its support for Scala and SBT and its continuous deployment integration with Heroku.

Travis was configured to run its so called builds by executing tests with SBT. The advantage of these tests is that they provide feedback on whether an implemented feature does not only work on the machine of the developer. Checks were executed per Git push and per pull request (see Section 6.8.2).

Usually Travis is free to use for open source projects and paid for private projects. However, with a student license, Travis was free to use during this project in combination with a private Github repository. Travis was chosen over other products because of the popularity of the platform and previous experience of the student team with it in combination with SBT and the Play Framework.

[9] `https://trello.com`
[10] `https://github.com/`
[11] `https://travis-ci.com`

7 Conclusions

The result of this project is an innovative application that is able to automate a big part of the client's workflow. The client had a similar product, but that application lacked two important aspects: usability and performance. Both of these aspects have been greatly improved. Firstly, usability is improved by making the input of data as easy as copy pasting from spreadsheet software. Secondly, the time an analysis takes is decreased significantly. To test the application, an example tender from the old software has been used. The analysis in the new application is executed in about half a minute, whereas the old application took more than half an hour to perform a similar analysis.

We managed to complete all the must haves defined in the requirements. The application can import and validate all shipment and rates data and perform an analysis on this data. Based on this, the application is able to return different sets of carriers with the best price for each set size. The result can be displayed based on different sizes of sets of carriers, which allows the user to select the best set of carriers for the client of the tender. The produced results can be easily exported by the user and reported to the client. Furthermore we produced a web application that can be easily used without installing any software and only when the user is authenticated. Hereby the success criteria, usability and performance, are met.

The client has indicated the project can improve his current workflow. While some extensions should be made to the application to be able to replace the client's workflow completely, it can speed up his process a lot and he is therefore satisfied.

Because the success criteria are met, the must have requirements are completed and the client is satisfied, the project can be seen as a success.

7.1 Reflection

During the two week research phase the team focussed completely on research and did not perform any programming until the research report was finished. Looking backwards, this has not been a good decision. There has been some spare time in this period that could have been used to set up the basic development environment, like constructing a basic project in Git and configuring IntelliJ IDEA. Also, a first version of the final report could have been constructed.

Pair programming turned out to be an effective way of developing some parts of the software. During development, important parts of the software have been programmed by the two team members together. By not performing pair programming the entire project, productivity was maintained and quality of important software was increased, as both developers contributed.

After the research phase and before the mid-project meeting, there has been little contact with the client. In the beginning of the project, a lot of work was put in constructing data structures and algorithms. This was important work, but was not very suitable for feedback by the client. The few contact moments with the client have been addressed in the mid-project meeting, and afterwards it improved during the course of the rest of the project. What have could been differently was to start earlier with some parts of front-end design. Then the client could have been asked earlier for feedback on the application's structure. Also, this would have fitted better in the Scrum software development methodology.

We learned to work full time on a real-world software development project. And for the first time we experienced working with a real client. Additionally, we learned to work together on a project and we both expanded our skills in programming in Scala. Next time, we would begin the project with a bigger team. This would hopefully enable us to be more productive and it will fit better in the Scrum methodology. We enjoyed working together the past twelve weeks.

8 Recommendations

During the project, the student team is convinced of the potential of the project. Therefore, some recommendations for the client related to extending the final product are listed in this section.

8.1 Include Qualitative Data in Analysis

In the current form of the application, only quantitive data is used: rates and prices. The price of transport is very important for Transport Tender's clients and even a decrease of costs of one percent can make a big difference on large volumes. However, the price does not say everything. For the application's results to be more helpful for their clients, qualitative data could be used in the analysis too. E.g. transit times or transporter's reputations.

8.2 Add Custom Analysis Levels

Currently two types of analysis are carried out: on country level and on shipment level. These analysis levels relate to the size of an area that is restricted to one carrier. With an analysis on country level, all shipments in a certain country can be executed by only one carrier. On shipment level, each shipment can be handled by every carrier in the potential set of carriers the analysis is carried out for. However, a client of Transport Tender might want only one carrier for a certain area of logistics, where this area is smaller than a whole country. Extra levels of analysis could be added to the application by adding the ability of defining those areas per tender. We could say these custom analysis levels are between the levels of analysis in shipment level and country level.

An example application of this would be dividing the United States, which is one country, into its states. For a country like the Netherlands, these custom zones could match provinces. In an ideal case, the analysis level is completely definable by the client. These analysis levels could be defined by defining certain zones (instead of countries), identified by groups of postal code zones.

8.3 Interactively Visualise Analysis Results

Currently analysis results are displayed statically, mainly in the form of tables containing prices per carrier (set) and per country. However, quickly looking at the consequences of analysis results by changing tenders's properties is not very easy. E.g. for viewing the difference in outcomes by applying or unapplying shipment summing is only possible by starting a new tender. To cope with this, an interactive analysis results interface could be developed.

In this interface, properties of the tender could be adjusted and results are displayed instantly. These properties could include applying or unapplying summing, adding or removing carriers, including or exluding regions of shipments and switching between analysis types.

Additionally, an interactive map of Europe (or an extended scope) could be visualised. In this map, for a certain carrier his cheap and expensive areas can be visualised using light and dark colors, respectively. For the client, it can indicate where its shipments are heading the most. Also, it could show in which part of the world the prices are the highest and it would be worth the effort to look for new potential cheaper carriers.

8.4 Extend Application Scope

Currently the application's scope is limited to road freight transport within Europe. This scope could be extended on both geographical level and on types of transport. Naturally, the geographical scope could be extended to the whole world. This would require to extend the definition of countries and relative postal code formats and ranges.

Also, besides road freight transport by trucks, transport types by ship, train or airplane could be added. This might require a change in the used formats of shipments and rates tables.

8.5 Advise Based on Rates Database

With every tender that is executed, valuable rates data is stored. Usually when a new tender is started, the potential set of carriers is given by the client. However, based on the client's shipments

and the database of carriers and rates, some matching carriers that participated in old carriers can be added to new tenders. This would require explicit permission by the carriers. Also, it should be taken into account that rates for a tender are based on the client's type of transport and therefore old rates may not be suitable for all new tenders.

8.6 Increase Software's Applications

Because the software is built very modular, it could be easily used for different applications. When a tender is finished, the tender's client might want to quickly find the (cheapest) price for a future shipment. A client interface could be added to the application in which they can easily lookup prices themselves.

Furthermore, the basic data structures and pricing algorithms could be reused in any logistic related application using shipments and rates tables in a similar format.

A SIG Evaluation

A.1 First Submission

De code van het systeem scoort 4 sterren op ons onderhoudbaarheidsmodel, wat betekent dat de code bovengemiddeld onderhoudbaar is. De hoogste score is niet behaald door lagere deelscores voor Unit Size en Unit Interfacing.

Voor Unit Size wordt er gekeken naar het percentage code dat bovengemiddeld lang is. Het opsplitsen van dit soort methodes in kleinere stukken zorgt ervoor dat elk onderdeel makkelijker te begrijpen, te testen en daardoor eenvoudiger te onderhouden wordt. Bij jullie code bestaan de lange methodes uit een aantal blokken functionaliteit, die je ook als aparte methodes zou kunnen implementeren. Voorbeelden hiervan zijn Rates.save en Initial.insertTender.

Voor Unit Interfacing wordt er gekeken naar het percentage code in units met een bovengemiddeld aantal parameters. Doorgaans duidt een bovengemiddeld aantal parameters op een gebrek aan abstractie. Daarnaast leidt een groot aantal parameters nogal eens tot verwarring in het aanroepen van de methode en in de meeste gevallen ook tot langere en complexere methoden.

Grote aantallen parameters komen vaak voor bij talen als C, die weinig mogelijkheden tot abstractie bieden. Bij moderne talen als Scala heb je genoeg mogelijkheden om je datastructuur anders in te delen, dus het zou vrij eenvoudig moeten zijn om het aantal parameters te beperken. Het gaat ons dan vooral om de "echte" methodes, als een case class een groot aantal velden heeft is dat veel minder erg. Een voorbeeld van een kanditaat om te refactoren is SilhouetteModule.provideEnvironment.

Tot slot is de aanwezigheid van testcode erg positief. Zowel de hoeveelheid testcode als de verhouding tussen productiecode en tests zien er goed uit. Hopelijk lukt het om op deze manier te blijven werken tijdens het vervolg van het project.

Over het algemeen scoort de code dus bovengemiddeld, hopelijk lukt het om dit niveau te behouden tijdens de rest van de ontwikkelfase.

A.2 Second Submission

In de tweede upload zien we dat de omvang van het systeem is gestegen, en de score voor onderhoudbaarheid ongeveer gelijk is gebleven.

Bij Unit Size zien we geen verbetering. De genoemde voorbeelden zijn weggewerkt, maar er zijn tegelijkertijd ook weer nieuwe, lange units geintroduceerd. Zie bijvoorbeeld de units in ratetables.js.

Bij Unit Interfacing zien we iets soortgelijks, hierdoor is de score zelfs een klein stukje gedaald. Voor de toekomst is het verstandig om nog eens na te denken over dit soort abstracties.

Wat betreft de testcode is het goed dat jullie naast een stijging van de productiecode ook nieuwe tests hebben geschreven.

Uit deze observaties kunnen we concluderen dat de aanbevelingen van de vorige evaluatie deels zijn meegenomen in het ontwikkeltraject.

B Project Description

De kracht van transport tender ligt in de kennis rondom tarieven. De opdracht voor dit project bestaat uit het stroomlijnen van het tender proces. Ontwikkel een (web)applicatie waarin tarieven en zendingen kunnen worden ingevoerd en waarna een analyse automatisch gegenereerd kan worden. Ook is het gewenst om meerdere locaties samen te analyseren. In het huidige systeem moet voor iedere locatie een aparte tender opgezet worden. Vervolgens wordt met de output in excel een gedetailleerde analyse gemaakt. Momenteel worden analyses dus deels handmatig gedaan. Ook kunnen de tarieven in de database niet ingezet worden voor nieuwe klanten. Wanneer er een nieuwe klant is, wordt handmatig gekeken welke vervoerder bij deze klant zou kunnen passen. Gewenst is een situatie waar zendingen worden ingelezen, en de applicatie een advies zou geven welke vervoerders te benaderen op basis van de tarieftabellen in de database. Een belangrijk deel van de adviesgeving is naast de prijs, ook de kwaliteit van het transport. We zien zien de studenten graag hun creativiteit gebruiken om ook de kwaliteit van het transport in analyses mee te nemen (denk hierbij aan certificeringen, levertijden, etc.).

C Project Plan

C.1 Introduction

This document is a project plan of the Bachelor Project as final course of the Bachelor Computer Science at the Delft University of Technology carried out by Willem Jan Glerum and Jasper Denkers (student team). The project is commissioned by Transport Tender[12], represented by Peter Struiwigh (client) and supervised by Andy Zaidman[13] (supervisor), an associate professor in the Software Engineering Group (SERG) at the Delft University of Technology.

It describes an overview and planning of the process, deliverables and deadlines. During the course of the project, continuous reference back to this plan is made to confirm the progress remains on track. Since this document is prepared at the beginning of the project, not all requirements and information about the scope of the project are already available. Therefore it focusses on the process of the project and not on the requirements. The latter is covered in the research report.

The project is carried out in twelve weeks, starting at 20 April 2015 and finished with a final presentation at 8 July 2015. An overview of important dates can be found in Table 10.

C.2 Assignment

Transport Tender advises companies in the setup process of transport by trucks within Europe. Since there is a big offer of logistic providers and pricing transport is done by using complex tables of rates, it is hard for a company to find the best set of logistic partners. The power of Transport Tender lies in the knowledge about rates and giving their clients insight in those rates. The processing and reporting of these rates and associated data is mostly spreadsheet based. The goal of this project is to automate this process. Besides analytics based on quantitive data, the rates, the project could be extended by using qualitative data like e.g. transit times and quality of transport in the analysis.

Deadline	Title	Description
26 April 2015	Project plan	Overview of activities and deadlines.
3 May 2015	Research report	Summary of results of the research phase.
29 May 2015	Mid-project meeting	Meeting with Bachelor Coordinators, student team and client.
5 June 2015	SIG submission #1	Submission of source code to Software Improvement Group for code quality check.
26 June 2015	SIG submission #2	Second submission to SIG.
1 July 2015	Final report	Official document reflecting on the full scope of the project. Delivered to the Bachelor's Project committee seven days before the final presentation.
8 July 2015	Final presentation	Final presentation including demo.

Table 10: Overview of deadlines

C.3 Deliverables

To support the process, a set of deliverables are presented including three documents and a final presentation. The documents are a project plan, a research report and a final report. The final presentation is held at the end of the project and visited by the Bachelor Project Committee: the client, the supervisor and one of the Bachelor Project coordinators.

[12] http://transporttender.nl

[13] http://www.st.ewi.tudelft.nl/~zaidman/

C.4 Approach

C.4.1 Methods

We will be using SCRUM for this project in the development phase. This enables us to easily prioritise tasks and see the progress made. A sprint will take up a week and we will start on Monday to decide what we will be implemented that week. Furthermore, at the beginning of every day we have a daily SCRUM to discuss what has been done the previous day and what we are going to do that day.

To prioritise our product backlog we will make use of the MoSCoW principle which describes the importance of each requirement:

MUST A requirement that must be implemented to reach a successful product.

SHOULD A high-priority requirement that should be implemented if possible.

COULD A feature which is desirable and is only included when there is enough time.

WON'T This requirement will not be included in this release, but could be in the future.

C.4.2 Techniques

As the project has just started we do not know which techniques we will be using during the development phase, such as programming language, framework, IDE etc. More research will be needed to discover what we should use, this will be presented in the research report.

C.5 Process

C.5.1 Planning

The project is roughly separated in three phases:

Phase 1 - Research In the first two weeks the student team meets with the client and determine an approach to execute the project. This includes gathering requirements and choosing frameworks and technologies.

Phase 2 - Development The biggest part of the project is the development of a solution for the client's problem. This takes place from the third to the tenth week.

Phase 3 - Report and presentation In the last two weeks the student team finishes the final report and concludes the project with the final presentation on 10 July 2015. The final report is delivered to the committee at least seven days before the presentation.

The bachelor project is set to 15 ECTS for the Bachelor Computer Science, this means every member should spent $15 \times 28 = 420$ hours on the project. This project will last 12 weeks which means we should spent 35 hours per week working on the project.

C.5.2 Meetings

The student team has regular meetings with the client and supervisor. These meetings will be held every one or two weeks, and sometimes only with either the client or supervisor. At the end we will present our project to the client, supervisor and coordinators.

C.5.3 Tools

For version control of all documents and code we will use Git[14] and Github[15]. This enables us to collaborate together and merge files easily. Furthermore, using Github we can easily track the history and changes of all files in the repository.

[14] `http://git-scm.com`
[15] `https://github.com`

To manage the product backlog we will use Trello[16], this is an online web-application where multiple users can work together to organise anything, especially projects. This will create an overview for the product backlog and the relevant points for each scrum, with different statuses.

C.5.4 Financing

The tools described in Section C.5.3 can be used free of charge using educational accounts. During the project hosting for the application could also be rented free of charge at for example Heroku[17]. This hosting is sufficient for development purposes, but has some limitations. In order to bring to application in production, some costs arise. These costs will be paid by the client.

C.6 Quality Assurance

In order to ensure the satisfiability of the client with the quality of the final product, during the course of the project some measures should be taken into account. These measures include functionality, usability, efficiency, reliability, modularity and maintainability.

Functionality After the research phase, most of the client's requirements are formulated. During the project these requirements could be either modified or extended. To ensure the quality of the final product, the student team should ensure these requirements are functionally covered by the final product and these functionalities are extensively tested.

Usability For the product to be a valuable extension of the current workflow of Transport Tender, it should be easy to use. The better a tender or analysis could be performed, the better the purpose of the system.

Efficiency The product must cope with big amounts of data. For a responsive application, loading and processing times should be kept as low as possible by proper software engineering and developing efficient algorithms.

Reliability Proper error handling and testing should be implemented in the software to cope with unexpected input or possible failures.

Modularity The analytical part of the software might be of use for Transport Tender in other applications than tender processes. For this software to be applicable in other systems, the system should be built of loosely coupled modules.

Maintainability At the moment the Bachelor project is finished, the client might still have the desire of an improved or extended version of the application. To make improving or extending the software easier for a possible third party, the software should be well structured and documented.

C.6.1 Code Analysis by SIG

Two times during the project the source code of the application is submitted to Software Improvement Group (SIG)[18]. The first submission is before the 75% progress mark of time and the second submission is at the end of the project. The feedback SIG produces will be used to directly improve the quality of the product, for example by restructuring the product or reducing code complexity.

[16] `https://trello.com`
[17] `https://heroku.com`
[18] `https://www.sig.eu`

D Infosheet

Project title: Road Freight Transport Tender Automation
Client: Transport Tender (`http://transporttender.nl`)
Final presentation: 15:30 08-07-2015

Project

Transport Tender helps their clients at choosing carriers to outsource their logistic activities. This choice is made based on an analysis of rates of a set of potential carriers on the shipment history of the client. Such decision processes are called tenders. In this project, software is developed that automates tenders. The software can input data and output results of analysis automatically.

The project faced two big challenging design goals: usability and performance. Usability because large amounts of shipment and rates in different provided formats are required to be submitted via the application. This is addressed by designing standardised data formats and making the submission of data as easy as copy pasting from spreadsheet software. Performance is reached by using proper software engineering and efficient algorithms and data structures.

In a two week research phase the client's domain is explored and requirements are formulated prioritised using the MoSCoW principle.

The iterative agile Scrum software development methodology is chosen in combination with test-driven development to cope with changing requirements and to make sure the minimal requirements were met and a working product was delivered.

The final product is an innovative web application that is able to input the tender related data and that is able to analyse the data and output results automatically. All must have requirements are implemented.

The software's potential will grow in the future as demand for efficient logistics increases. Besides quantitive data like rates, qualitative data like transit times and carriers reputations could be added to the analysis in the future.

Team: Jasper Denkers

Interests: scalable web systems, entrepreneurship.
Contributions: parsing, data structures, algorithms, front-end.

Team: Willem Jan Glerum

Interests: Big Data, Linux.
Contributions: database, deployment, authentication, front-end.

All team members contributed to preparing the reports and the final presentation.

Client

Peter Struiwigh, Transport Tender, p.struiwigh@transporttender.nl.

Coach

Andy Zaidman, Software Engineering Research Group TU Delft, a.e.zaidman@tudelft.nl.

Contact

Jasper Denkers: jasperdenkers@gmail.com.
Willem Jan Glerum: wjglerum@gmail.com.

The final report for this project can be found at: `http://repository.tudelft.nl`.

Acronyms

CSV Comma Separated Value. 4, 18, 19

DAO Data Access Object. 13, 18

DI Dependency Injection. 13, 14, 18, 19

DOM Document Object Model. 24

ERP Enterprise Resource Planning. 4

FTL Full Truck Load. 5

IDE Integrated Development Environment. 20, 21

JAR Java Archive. 19

JEE Java Enterprise Edition. 13

MVC Model-View-Controller. 12, 18

PaaS Platform as a Service. 21

SBT Scala Built Tool. 18, 20, 21, 24, 25, 28

SERG Software Engineering Group. i, 34

SIG Software Improvement Group. 8, 22, 24, 25, 36

References

[1] Linli He and Thomas R Ioerger. An efficient heuristic bundle search algorithm for buyers in electronic markets. In *IC-AI*, pages 729–735. Citeseer, 2004.

[2] J Loughry, J van Hemert, and L Schoofs. Efficiently enumerating the subsets of a set. *applied-math. org/subset. pdf*, 2000.

[3] Michael Negnevitsky. *Artificial intelligence: a guide to intelligent systems*. Pearson Education, 2005.

[4] European Union. *EU Transport in Figures - Statistical Pocketbook 2014*. Publications Office of the European Union, 2014.

[5] Robbie Vanbrabant. *Google Guice: agile lightweight dependency injection framework*. APress, 2008.