Second-hand Cars

Computer Science
Business Academy South West

Handed in: 12-06-2015

Cornel Lungeanu
Kristian H. Bock
Bjarke Boisen

Second-hand Cars

Authors:

Cornel Lungeanu
Kristian H. Bock
Bjarke Boisen

Supervisors:

Claus Christensen
Tommy Haugaard
David Lindholm
Karsten Skov

Computer Science, 2nd Semester

Project period:

18-05-2015 to 12-06-2015

Number of pages:

36

Products:

Second-hand Cars report
Source code in pdf
Program files

Preface

Our intention with this report is to give the reader insight into the processes we have used during the project, as well as the development of the system from brief, summarized use-cases and diagram across in-depth analysis of the requirements, ending with a functional program with several possible iterations adding more functionality and complexity.

We have each been involved with the entire development process, but responsibility for the individual areas has been split across the team:

Programming
: Kristian H. Bock

Graphical User Interface Design
: Cornel Lungeanu

Database Analysis and Design
: Bjarke Boisen

Use-case, SSD and SD-diagrams
: Cornel Lungeanu and Kristian H. Bock

Business Analysis
: Bjarke Boisen

Table of Contents

Introduction

The goal of this project to create a system to be used by the customer, Mads Snild, in order to alleviate the problems with managing his assets and transactions that he is experiencing due to acquiring several garages for repairs and tuning. The project will also contain possible solutions for organizational improvements to smooth out the personnel and planning difficulties that the customer has noticed in the different garages. In addition to this, suggestions for tracking sales and progress will be included in the report.

Problem analysis

After Mads Snild (henceforth referred to as the client) acquired 3 garages in addition to his car dealership, several problems have come to light. None of the problems are critical, but they do contribute to a lower profit in the company and being inconvenient.

The first of the problems is a lack of reliable documentation regarding the transactions in the company. The current recording of transactions is being done in a spreadsheet, often containing errors due to the details of specific orders not always being entered during the transaction, but later, resulting in missing or wrong details. This results in the client having a potentially suboptimal adherence to customs- and taxation laws, which causes additional unneeded expenses due to having to pay additional or missing taxes and charges.

The next problem in focus is the lack of an efficient search method for finding specific cars, which causes longer wait times for the customer(s) when they contact the company to inquire about possible buying one of the cars in stock, and also complicates management of the car stock.

The third problem that should be addressed is the low productivity in the different garages that occurs due to lack of communication between the garages, causing less profit due to lower efficiency in the company.

Problem definition

The problem with the cooperation with the customs and taxation authorities involves a lack of coherency between the transactions being performed and the documentation of these transactions. A way to solve this could involve having the transaction being entered into the system while they are happening instead of having them being entered after interaction with the customer has ended. This should result in the transactions being entered into the database through the program, allowing the client to later use the program to fetch them from the database and creating a report of transactions.

The current situation for the car dealer, when he has to look for a specific car, is that he has to manually look through the spreadsheet containing the information about the cars. This could be made much easier through the use a search function and a database that will allow him to search for a car by inputting search criteria. This would also allow the customer in a later version to search the database without going through the car dealer.

Low productivity and time-waste in the garages will be the last problem to be addressed. The client has experienced that the employees at the different garages are standing around drinking coffee and taking breaks instead of assisting their co-workers in the other garages and would like a planning system to be created that can help alleviate this problem. A planning system incorporated in the system is a possible solution and can be supplemented with a number of organizational improvements to improve the coordination between the different garages, as well as possible ways to measure sales numbers and efficiency.

Scope and delimitations

In our project we focus on creating a database and a system to extract data from that database to be shown in a coherent fashion in the system, as well as allowing the user to get various overviews of the available data.
We have limited the project to not contain methods and functionality to manipulate the database, since this were not specifically requested in the assignment text. We have, however, planned for further functionality to the added to the program in later versions and we will be creating use-cases and domain modelling for these versions.
The work planner in the project will be limited to just theory and use-cases, with no code or functionality, but with suggestions of possible organizational improvements.

Inception

Business case:
The client (Mads Snild) has a very ineffective system to handle management of the cars in stock and also has no system to handle the printing of invoices and saving sales reports for later use by the customs- and taxation offices.
In addition to this, he also has problems with a lack of planning regarding his employees, resulting in loss of productivity due to their laziness.
Due to the success of the tire-hotel he has also requested that the system should be capable of managing the tires and the customers.

Vision:
Our vision for the project involved creating a system that would be capable of showing all the information about the cars in stock in the company and also display an overview of the sold cars in the same way. We also planned to include a search function in both of these parts of the system. In addition to these features we later saw a need for a function to create a report of sales and transactions and having the system create this as a file.
Regarding the tire-hotel and the management of this part of the company we chose to focus on the sales and other services and postpone the tire-hotel management for later iterations.

Use-cases:
When we created the use-cases and the goal-actor list, we chose to find as many as possible and then later narrow the list down to the essential cases pertaining to the project. From this list we singled out the use-cases related to the car-stock as priority for the first iteration, and the use-cases for the orders and services to be the focus for the second iteration.


uc
View sold cars *
View sold cars within a period of time *
Create a report of transactions *
Create a report of transactions within a period of time
Manage carstock
Manage TireHotel
Manage users
Create order
View all cars in stock *
Search for specific cars *
User login
User logout
Booking repair
Booking Chip Tuning
Booking Tire
Manage work schedule
View the work schedule
Owner
Employee
Foreman
Customer
powered by Astah

Time estimation and management:
Our starting time estimation allocated 4 workdays to finish up the inception phase and then 5 workdays for each of the first two iterations, reserving the last 4 days for complications and new essential requirements being discovered.
The first iteration we planned to create the basic system and connect the back-end part with the database, after sketching out the ERD diagram pertaining to the iteration, with the the second iteration to contain the management of the assets in the company and the functionality needed to create a report of transactions.
The potential third iteration would contain login and password functionality together with user administration.

Business rules:
During the inception phase we defined a set of rules related to the system and the business area we are creating the system for.

The business consists of four addresses:
- The main office,
- The garage for normal cars, has 6 employees,
- The garage for diesel-powered cars, has 3 employees,
- And the garage for Chip-Tuning, has 2 employees,

One of the employees at each address is the foreman for that garage.

The services offered by the company are:
- Sale of second-hand cars.
- Repair of normal cars.
- Repair of diesel-powered cars.
- Chip-Tuning.
- Tire-fitting.
- Tire-hotel.

Point of contact with the customer is at the main office.

The person responsible for the planning is Mads Snild.
Priority for planning system is:
1. Repair of diesel-powered cars
2. Repair of normal cars,
3. Chip-Tuning,

Employees can be moved between the different garages depending on how much work the garages have.

All employees are capable of repairing normal cars.
Additional training is needed to repair diesel-powered cars.
Specialist training is needed to perform Chip-Tuning. This also qualifies them for doing repairs on diesel-powered cars.

The tyre-hotel and fitting station is located at the garage for normal cars.
The hotel has room for tyres from 150 customers. (max. 600 tyres in total).
The shelves for the tyres are ordered in a matrix of 25x3x2 (25 wide, 3 high and 2 deep).
The created numbering system should be in binary in the format (X,Y,Z), with total length being no more than 8 bit at maximum.
Numbers start at zero.
Example:

10001100

x: 10001 = the 18th down the row, y: 10 = the 3rd above the floor, z: 0 = the row to the left,

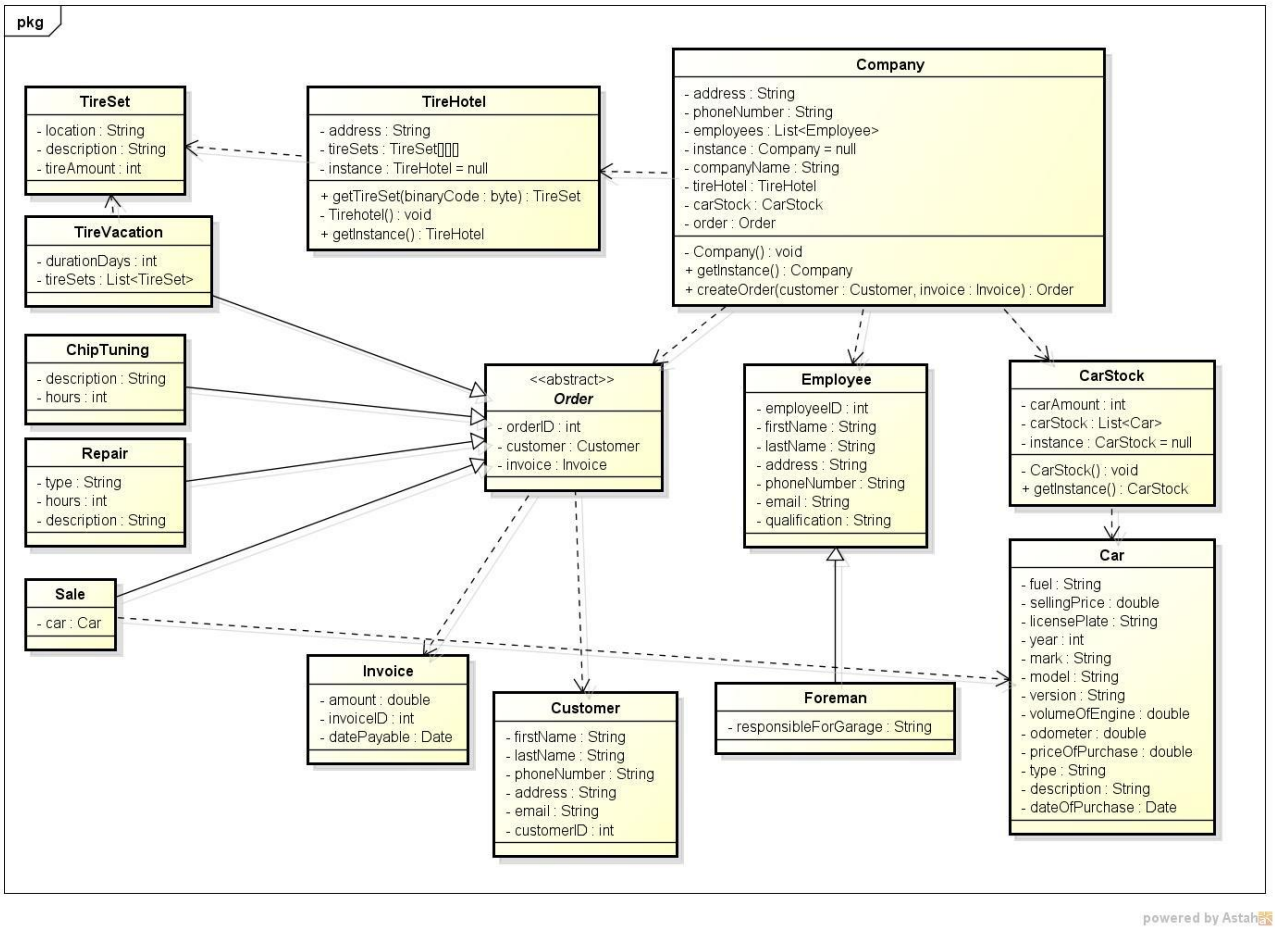
System must be designed in a way that allows full integration with the current webpage in future versions.
Version 1 of the system is to be a stand-alone JAVA-application.
Web functionality is not planned to be incorporated in any of the planned iterations.

Processes:
We accidently started our project using the Waterwall model for mapping requirements and had a complete model of the system before taking a step back and using the UP model and reducing the prototype diagrams of the system to more manageable sizes. It did give us a greater overview of the needed artefacts and use-cases, but later proved to hinder us when we needed to adapt the system to new requirements.

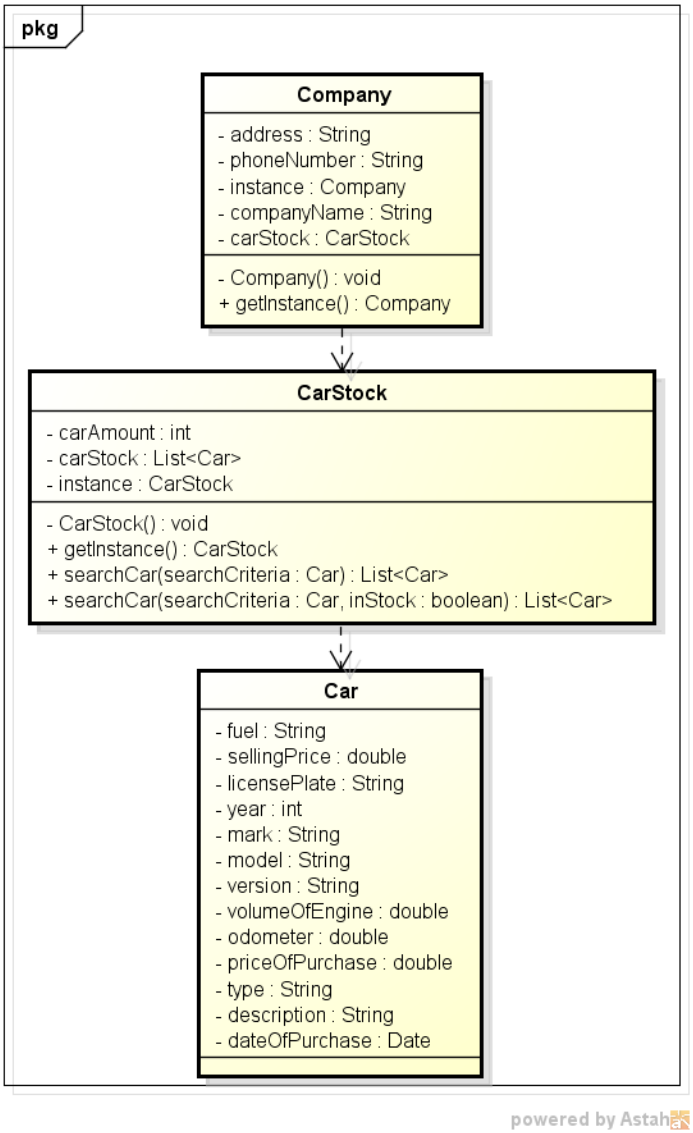
First Iteration

Design and domain modelling:

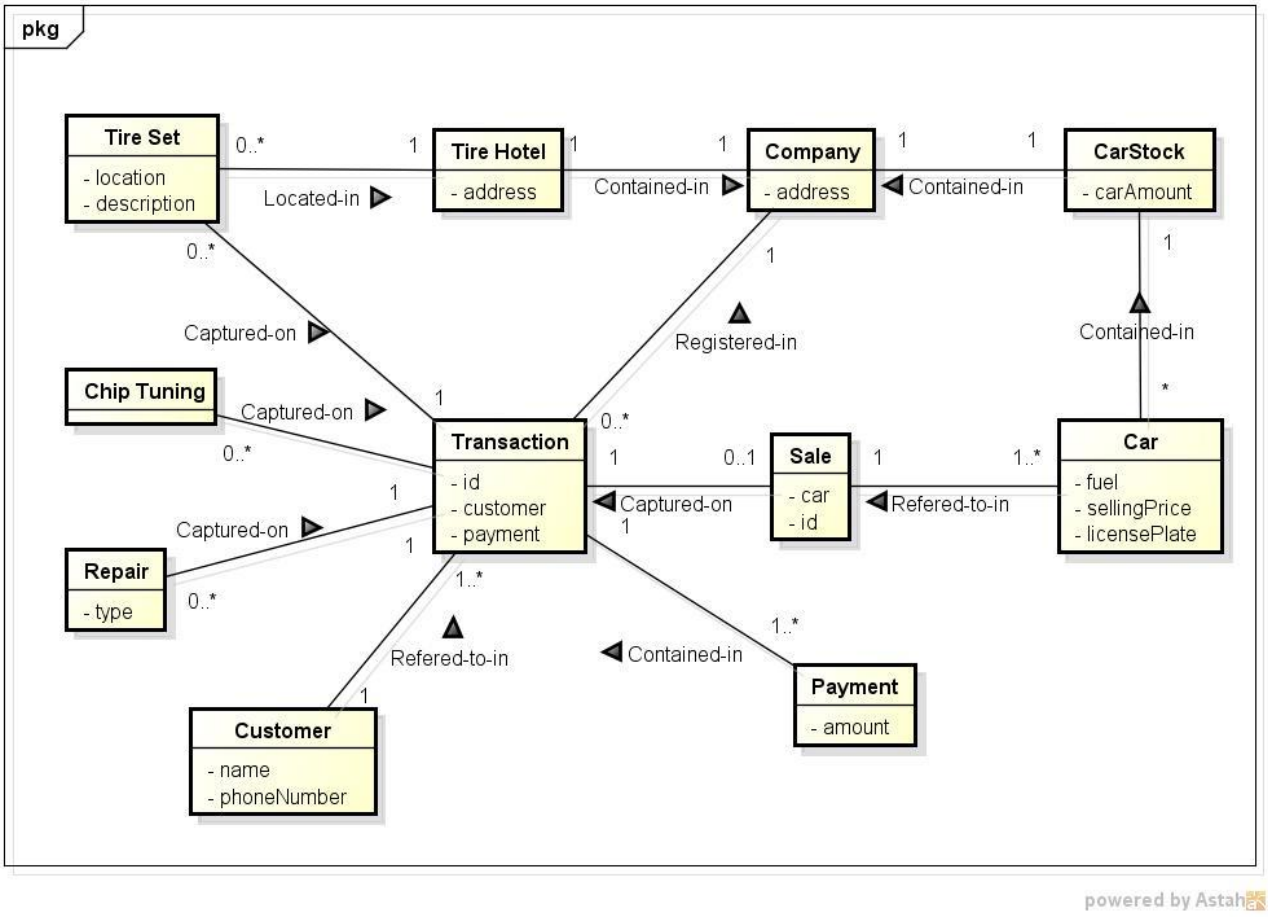
To start the first iteration of our project we mistakenly created the design model for the final form of the system instead of for only the current iteration. This necessitated an almost total revision from the final model:



We then reduced it down to containing only the classes and methods involved in the first iteration and ended up with a much more manageable overview:



Our first domain model ended up being pretty precise, with only the Payment later being incorporated into the Transaction, but again we made the mistake of creating it for the possible final version of the program and not for the first iteration.



Architecture:
The program architecture consists of a Graphical layer as the top-most layer, followed by a Controller layer which manages information flow between the graphical layer, the domain layer and the persistence layer. This follows the Controller pattern and Low Coupling principles from the GRASP.

The graphical layer only has access to the controller, relying on it for all data the GUI needs to display.

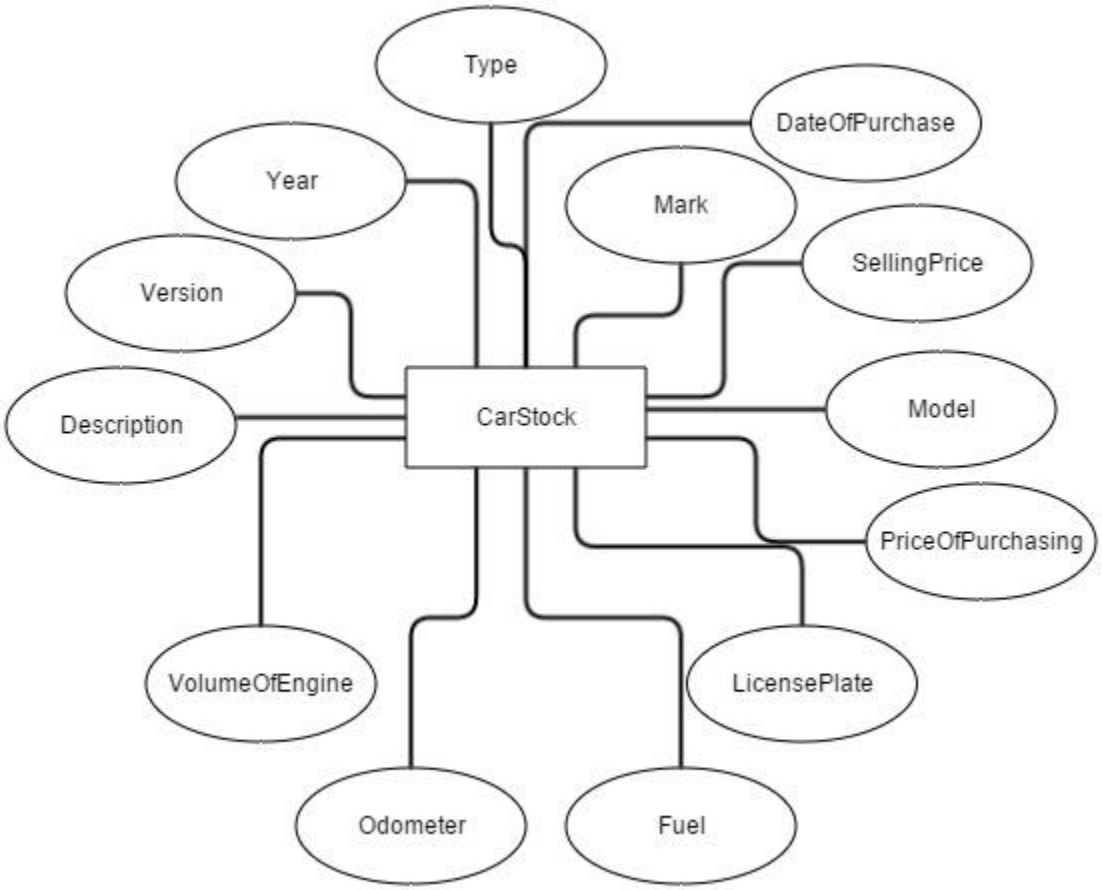
The controller layer contains methods to access the domain layer and the database layer, but does not know about the graphical layer.

The domain layer contains classes that “mimics” the contents of the database, with a Company singleton containing a CarStock singleton to ensure that even with the potential for multiple

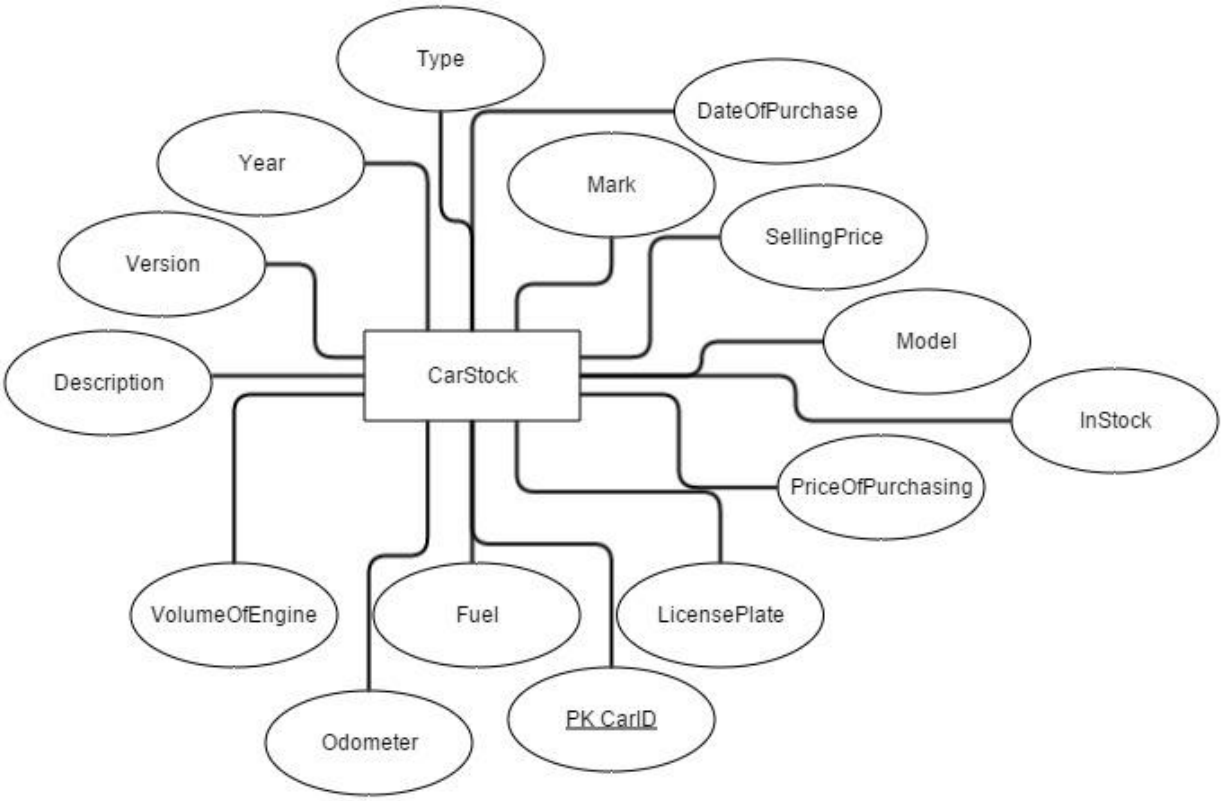
controller instances, all data regarding the cars and other tables remains consistent. The domain layer has no references to any of the other layers.

The persistence layer contains the methods to access and manipulate the database. It also contains a reference to a properties file which has the connection parameters listed. The database name, user name and login is set here before the program is run.

Database and ERD:
For the first iteration we chose to create a simple database with only one table containing the cars the company has in stock. Starting with the analysis of the database needs:



We then proceeded to incorporate an CarID primary field and a variable In-stock to track whether or not the car had been sold:

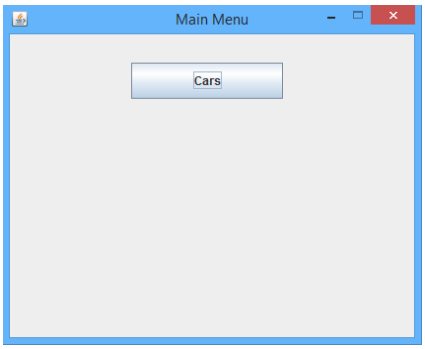


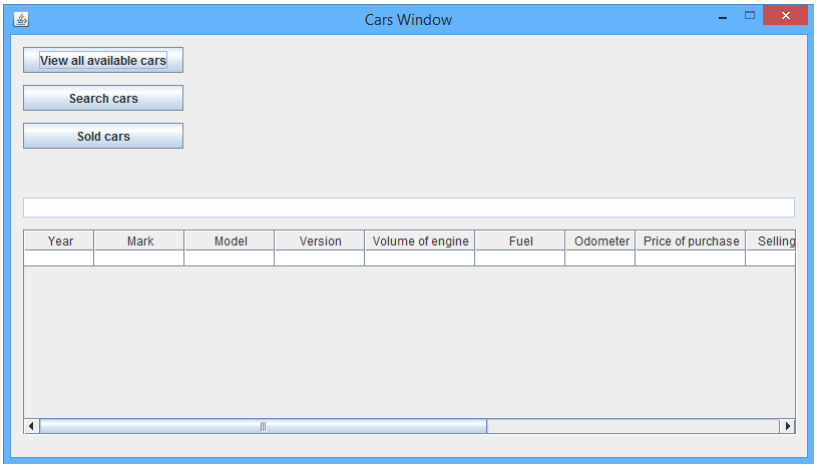
And lastly we mapped the data types needed for each of the variables:

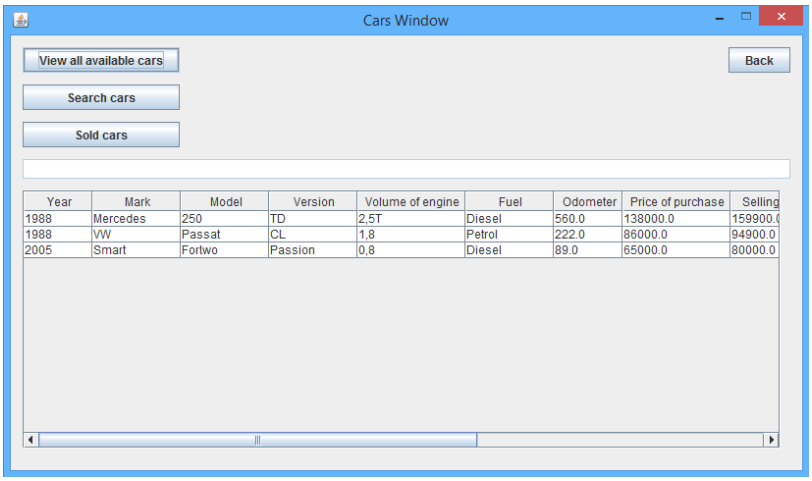
	CarStock	
PK	CarID	INTEGER NOT NULL
	Year	INTEGER NOT NULL
	Mark	VARCHAR NOT NULL
	Model	VARCHAR NOT NULL
	Version	VARCHAR
	VolumeOfEngine	VARCHAR NOT NULL
	Fuel	VARCHAR NOT NULL
	Odometer	INTEGER NOT NULL
	PriceOfPurchase	MONEY NOT NULL
	SellingPrice	MONEY NOT NULL
	Type	VARCHAR NOT NULL
	Description	VARCHAR
	LicensePlate	VARCHAR NOT NULL
	DateOfPurchase	DATE NOT NULL
	InStock	BIT NOT NULL

In the database we set the primary key to be auto-incrementing to ensure no creation of errors due to conflict between identical primary keys. A different choice could have been to have it as VARCHAR and then manually ensuring that none of them were the same.
We created two views in the database during this iteration, that respectively returns a table of cars in stock and a table of sold cars. This also gives back-end part of the system access to resultsets from the database and enables it to create “copies” of the database table content as objects in the memory. Since our goal in this iteration is not to manipulate the database through the program in any way, this ensures that the connection between back-end and database is kept to a minimum.
The naming of the variables in the database were also used as basis for naming the classes and variables in the program to ease the parsing of data from database to program.

Use-cases and data-flow:
The central use-case to this iteration is the Car Stock use-case CUC-001: View all cars in stock.







Use Case: User wants to see all cars in stock

Id: CUC-001

Description: User want to see a table with all cars in stock

Level: High level

Primary Actor: Seller

Supporting Actors: Customer

Pre-Conditions

User need to be logged in (the log in function is not implemented in iteration one)

Post Conditions

User has a list of available cars

Failure end condition:

- user is logged out from the system

- the database connection is lost

- the GUI can’t display the list

Trigger

User choose to view all cars in stock

Main Success Scenario

1. User selects “View all available cars”

2. The system queries the database for a list of all available cars

3. The system receives a list of all available cars

4. User is presented with a list of available cars

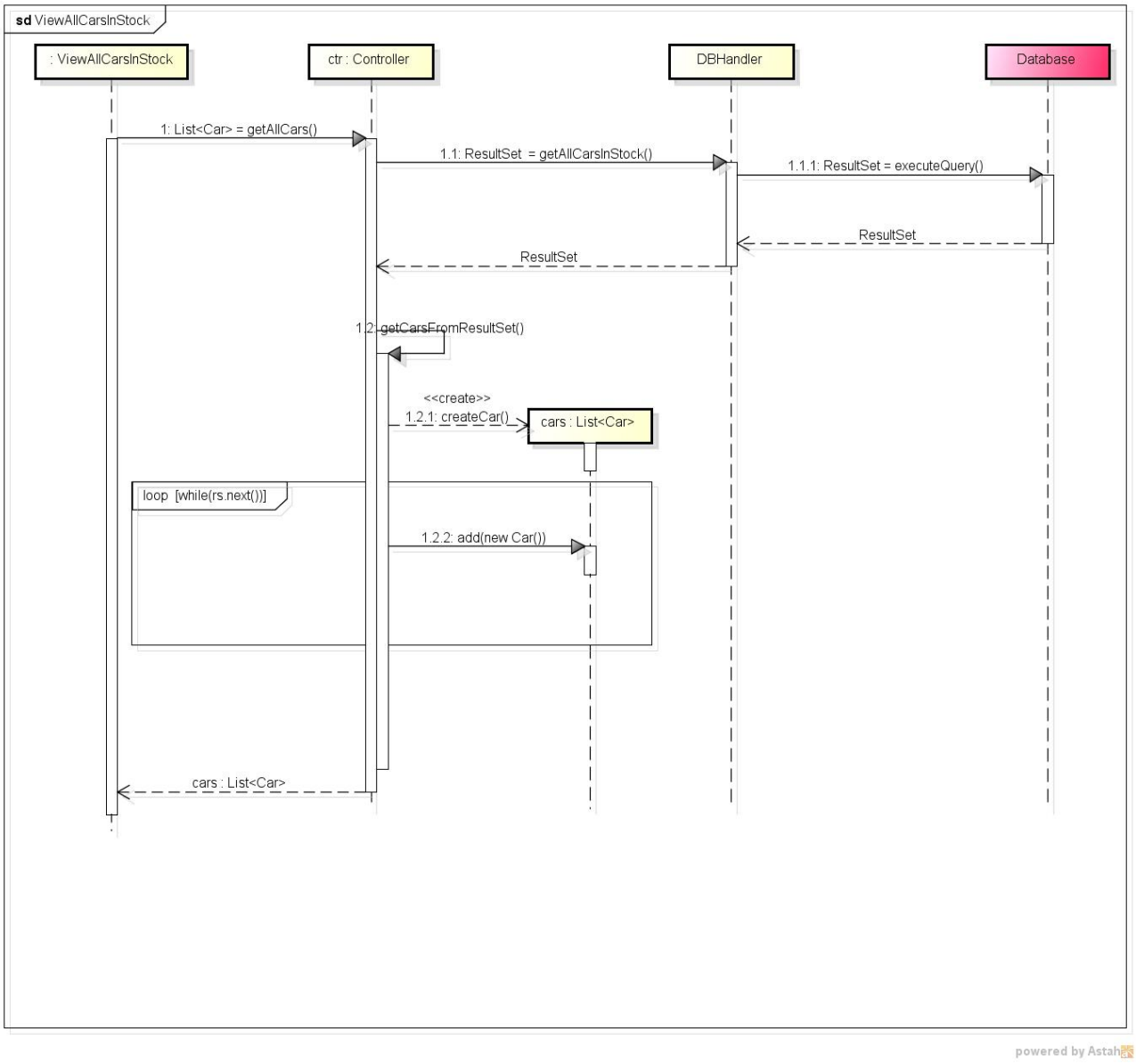
Extensions

3a: No connection to database

1: The system cannot connect to the database

2: The user is returned to the main menu

The dataflow for this use-case is simple:



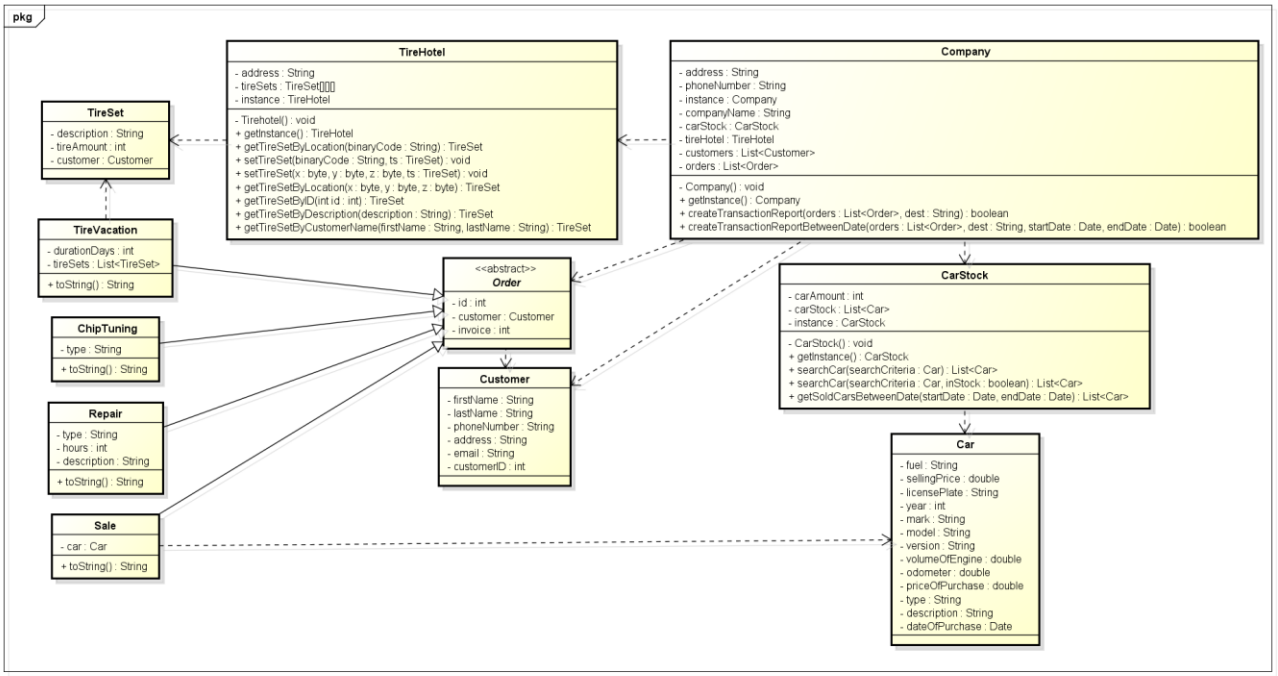
In this iteration we had a minor problem with connecting to the database and getting the needed data. We discovered that due to the methods we used, we closed the connection between the program and the database before we tried to parse the data into objects in the memory. Since resultset objects does not contain any concrete data, but only references to data in the database, this resulted in null resultsets and thus breaking the program. This problem was solved by postponing closing the connection.

Configuration:
As stated earlier, the program contains a properties file with the name of the SQL Server, the database name and the database username and password. The configuration of these files has to be done manually in the properties file, but functionality to change it through the program could be implemented in a future iteration.

Summary:
The goal of the first version of the product was focused on getting the view and search interactions between the back-end functionality and the database to be completed, with more features being added on in later versions.
This version contains the functionality of 3 of the Carstock use-cases, that is, the capability of the program to view all cars and search for a car from specified criteria, and to get an overview of previously sold cars.
The database itself has only one table in this version, containing all the cars and their data. We chose to do this, since having the cars in stock and the previously sold cars in separate tables would result in unneeded work when a car is sold, ie. removing it from one table and inserting it into another.
In the programming part one of the first things that happen is the conversion of data about the cars from the database into data in an array in the program, containing all the same data as the database, thus allowing us to not continually querying the database, but instead using the data in memory, ensuring a faster and more responsive program. This also reduces the need for for additional stored procedures in the database.

Second Iteration

Design and domain modelling:
For the second iteration we refined and expanded the design with all the different order classes, the customer class and the tire-hotel.



Graphical design:
The GUI were refined in this iteration, with each of the windows for the different functions being in their separate JFrames, allowing us to have several windows open at the same time. Coherency is maintained with all the JFrames having the same Controller passed along for each JFrame.

The graphical layer contains six classes.The first shown class is called “MainMenu”.Behind the main we have another five classes.Those are called CarWindow, Sales, SearchASpecificCar,ServiceWindow and SoldCars.

The MainMenu it contains three buttons that allow us to choose between **“Cars”**, **“Service”** or **“Sale”**.

“Cars” button opens “Car Windows” and dispose “Main Menu” frame.
“Car Windows” contains a result table,an error field and four buttons.The buttons are the following:

1. “View all available cars” button that display all the available cars in the result table.
2. “Search cars” button that opens the “Search Cars Menu”.

“Search Cars Menu” contains multiples fields and a “Search” button.Here the user can search cars by: year, mark, model, version, volume of engine, fuel, odometer, price of purchase, selling price, type, description, license plate or date of purchase. By pressing “Search” button, “Search Cars Menu” will be disposed and the results will be displayed in the “Car Windows” result table.If no cars fits the searched criteria the “Search Cars Menu” will be dispose and no result will be displayed.

3. “Sold cars” button that opens the “Sold Cars Menu”.

“Sold Cars Menu” contains two buttons and a hidden panel.The buttons are the following:

- “Get sold cars” button that dispose the “Sold Cars Menu” and display all sold cars in the “Cars Windows” result table.
- “Get cars” button that makes the hidden panel visible.The panel itself contains empty fields where the user can add dates.After filling “From date” and “To date”, the user can see the sold cars in that period of time but not before pressing “Search” button.After pressing “Search” button the result will be displayed on the Car Windows result table.If no cars were sold on that period of time, an error message will be displayed on the Car Windows error field.

4. “Back” button that dispose the “Car window” and open again “Main Menu” window.

“Service” button opens “Service Windows” and dispose “Main Menu” frame.
“Service Windows” contains a panel and four buttons.The buttons are the following:

1. “Repair” button that opens a panel with “From date”, “Description”, “Hours” and “Create” button.

The user can book a repair by filling all the empty fields and then pressing the “Create” button(the function is not yet implemented).

2. “Chip tuning” button that opens a panel with “From date”, “Description”, “Hours” and “Create” button.

The user can book a chip tuning by filling all the empty fields and then pressing the “Create” button(the function is not yet implemented).

3. “Tire” button that opens a panel with “From date”, “To date”, “Description”, “Hours” and “Create” button.

The user can book a tire vacation by filling all the empty fields and then pressing the “Create” button(the function is not yet implemented).

4. “Back” button that dispose the “Service window” and open again “Main Menu” window.

“Sales” button opens “Sales Windows” and dispose “Main Menu” frame.
“Sales Windows” contains a hidden panel three buttons a checkbox named “Date”and a hidden label.The buttons are the following:

1. “Create report” button that create a txt. file containing a sale report.

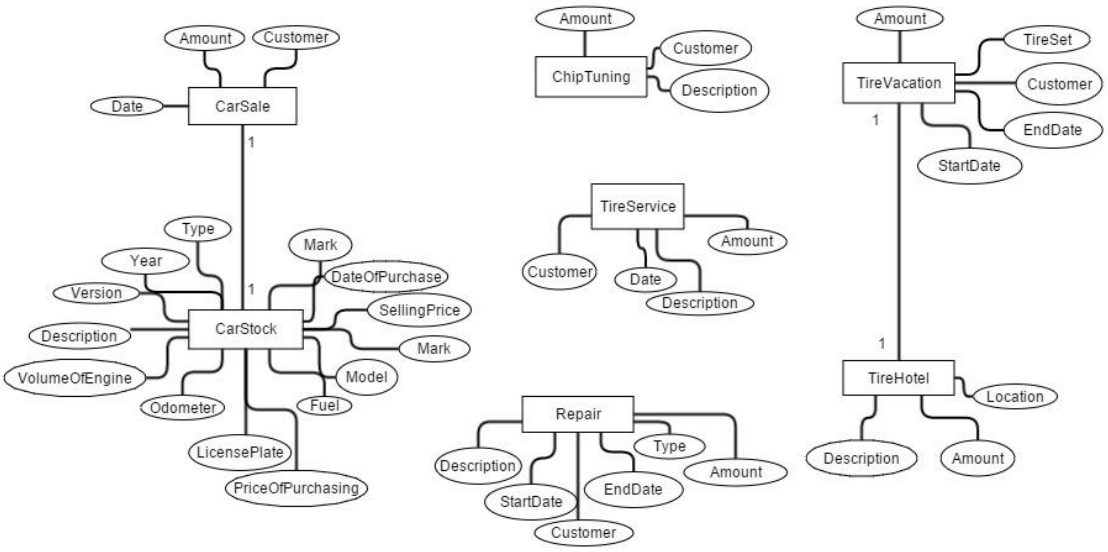
If user check the “Date” box, the txt. file will incorporate the creation date in the filename. If the file was successfully created a “Report was successfully created ”message will be displayed.If the file could not be created a ”Report was not created”message will be displayed.

2. “Use between dates” button opens the hidden panel.the panel contains “From date” text fields, “To date” text fields and a “Create” button.

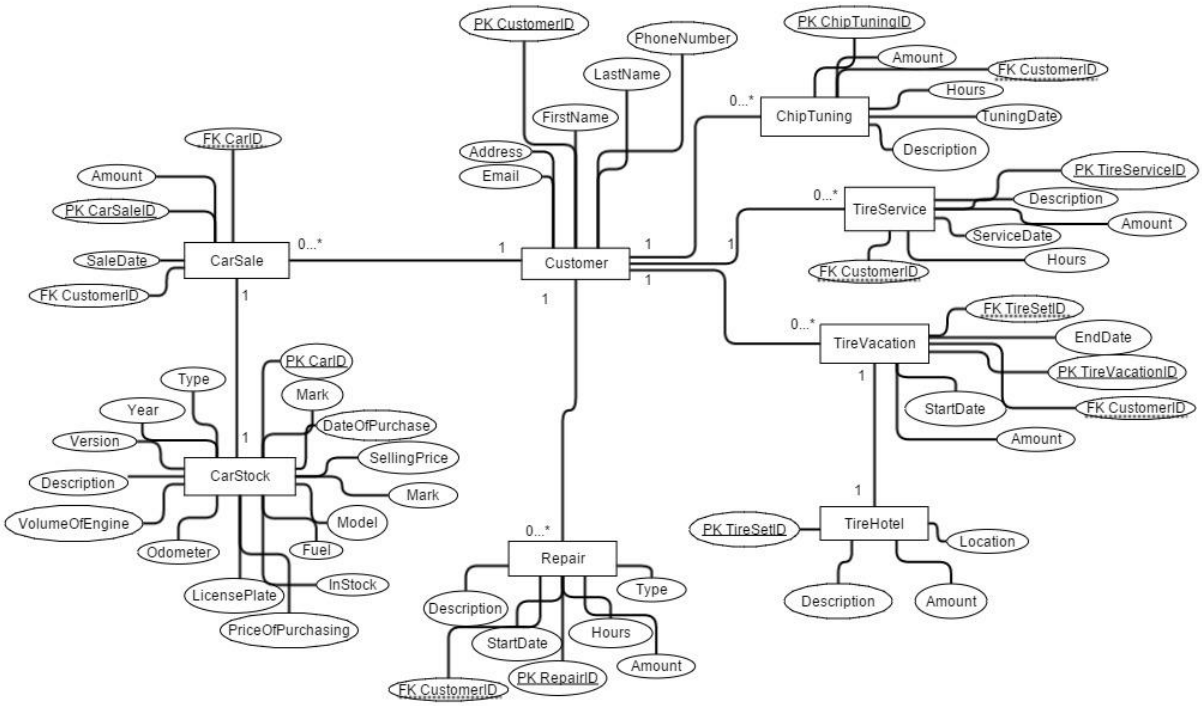
The user can create a txt. file containing a sale report between specific dates by filling the empty field and pressing the “Create” button.If user check the “Date” box, the txt. file will incorporate the creation date in the filename. If the file was successfully created a “Report was successfully created ”message will be displayed.If the file could not be created a ”Report was not created”message will be displayed.

3. “Back” button that dispose the “Sales window” and open again “Main Menu” window.

Database and ERD:
The diagrams for this iteration is quite more extensive, consisting of all the tables needed to hold the different kinds of orders, sales and services:



In the design part we chose to keep the different orders in their separate tables due to the difficulties with keeping to the rules for uniqueness of primary keys. The first attempt had an additional table as supertable for all the orders.



The mapping for this iteration is quite a bit more complex than the previous iteration:

	CarStock	
PK	CarID	INTEGER NOT NULL
	Year	INTEGER NOT NULL
	Mark	VARCHAR NOT NULL
	Model	VARCHAR NOT NULL
	Version	VARCHAR
	VolumeOfEngine	VARCHAR NOT NULL
	Fuel	VARCHAR NOT NULL
	Odometer	INTEGER NOT NULL
	PriceOfPurchase	MONEY NOT NULL
	SellingPrice	MONEY NOT NULL
	Type	VARCHAR NOT NULL
	Description	VARCHAR
	LicensePlate	VARCHAR NOT NULL
	DateOfPurchase	DATE NOT NULL
	InStock	BIT NOT NULL

	Customer	
PK	CustomerID	INTEGER NOT NULL
	FirstName	VARCHAR NOT NULL
	LastName	VARCHAR NOT NULL
	PhoneNumber	VARCHAR NOT NULL
	Address	VARCHAR
	Email	VARCHAR

	CarSale	
PK	CarSaleID	INTEGER NOT NULL
FK	CarID	INTEGER NOT NULL
	SaleDate	DATE
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

	TireService	
PK	TireServiceID	INTEGER NOT NULL
	ServiceDate	DATE NOT NULL
	Description	VARCHAR
	Hours	DECIMAL NOT NULL
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

	ChipTuning	
PK	ChipTuningID	INTEGER NOT NULL
	TuningDate	DATE NOT NULL
	Description	VARCHAR
	Hours	DECIMAL NOT NULL
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

	Tirevacation	
PK	TireVacationID	INTEGER NOT NULL
FK	TireSetID	INTEGER NOT NULL
	StartDate	DATE NOT NULL
	EndDate	DATE NOT NULL
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

	TireHotel	
PK	TireSetID	INTEGER NOT NULL
	Location	VARCHAR NOT NULL
	Description	VARCHAR
	Amount	INTEGER NOT NULL
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

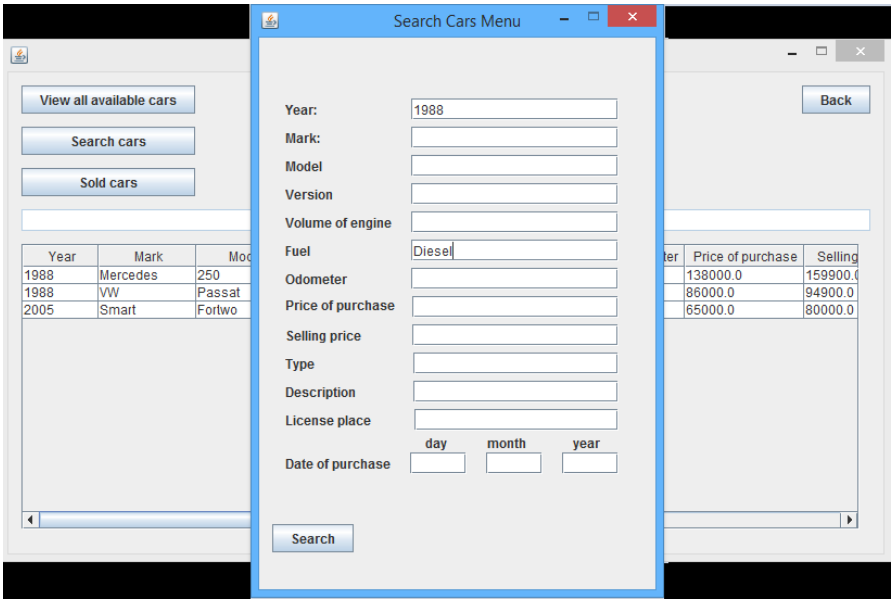
	Repair	
PK	RepairID	INTEGER NOT NULL
	Type	VARCHAR NOT NULL
	RepairDate	DATE NOT NULL
	Hours	DECIMAL NOT NULL
	Description	VARCHAR
FK	CustomerID	INTEGER NOT NULL
	Amount	MONEY NOT NULL

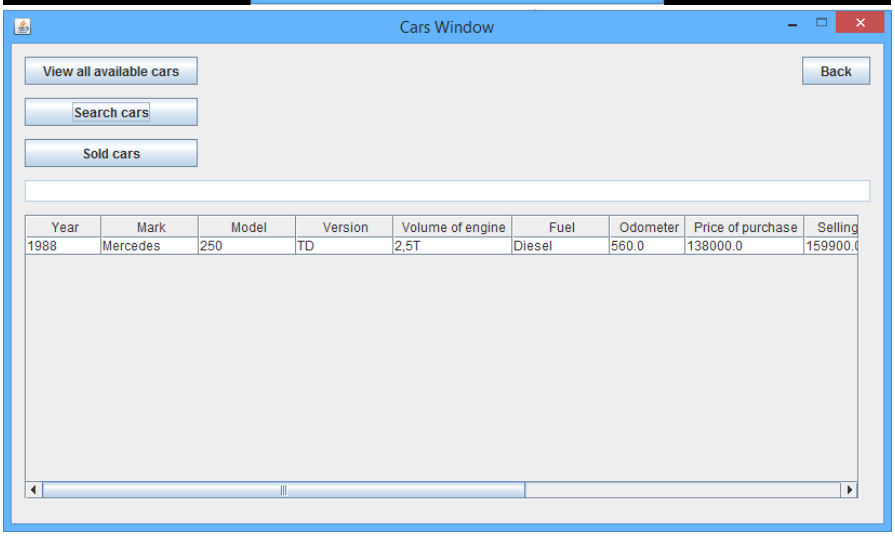
Having all the ID fields being auto-incrementing ensures that like in the previous iteration coherency is maintained.

Several additional views were needed in this part to allow the program to get data from the newly added database tables. Since this iteration does not have any functionality to manipulate the database there is no need for any stored procedures.

Use-cases and data-flow:

The central goal for this iteration was to create the functionality needed to create a report of transactions and sales and create a text file with this data, as well as creating a search function to look through the car stock and the list of sold cars, Car Stock use-case CUC-003: Search for specific cars:





Use Case: Search for specific car

Id: CUC-003

Description: User want to see a table with all cars fits the search criteria

Level: High level

Primary Actor: Seller

Supporting Actors: Customer

Post Conditions

Success end condition

User has a list of specific cars that is displayed

Failure end condition:

- user is logged out from the system

- the database connection is lost

- the GUI can’t display the list of specific cars

Trigger

User choose to view some specific cars

Main Success Scenario

1. User selects “Search cars”

2. User enters search criteria

3. The system queries the database for a list of cars with the specified criteria

4. The system receives a list of cars

5. The user is presented with a list of cars

Extensions

4a: No connection to database

1: The system cannot connect to the database

2: The user is returned to the search criteria menu

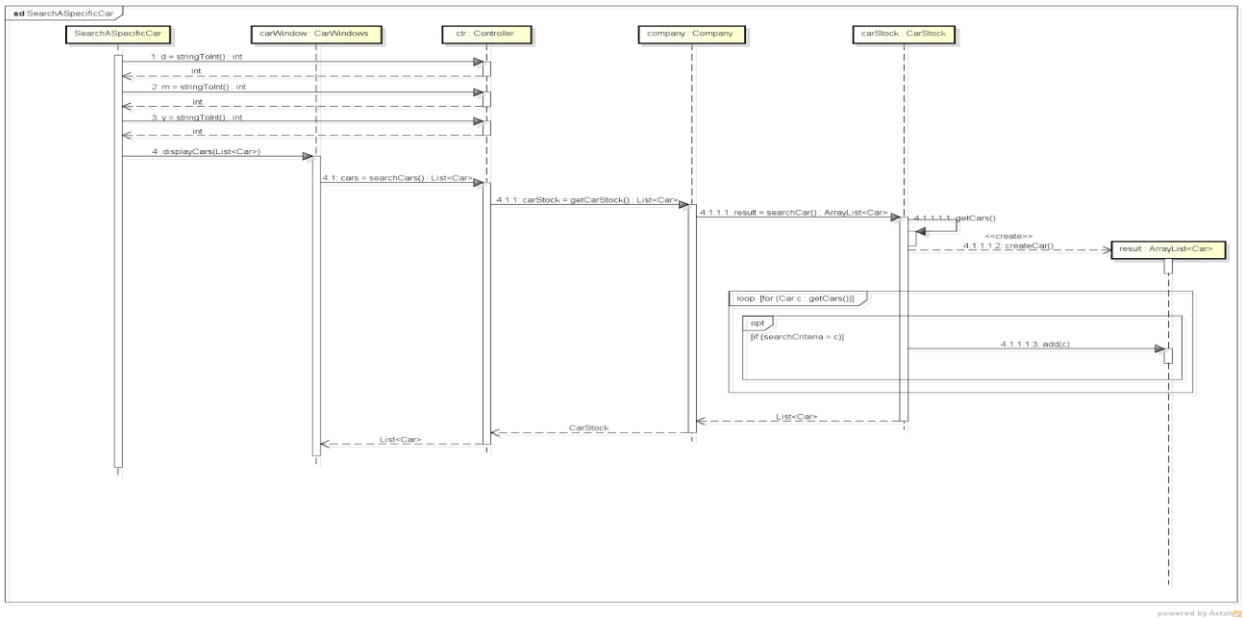
4b: No cars matching the criteria

1: The database has no cars that match the specified criteria

2: The user receives a message that no cars could be found

3: The user is returned to the search criteria menu

The data-flow for this use-case goes from the GUI layer through the controller and into the domain layer and then returns the data to be formatted and displayed:



Summary:
The goal for the second iteration was to incorporate the use-cases for managing the different services in the company and also being able to create a record of transactions to be printed out.

During this iteration we further refined the requirements and realized that several of the use-cases we had created were not required by the demands outlined in the assignment. We then removed the use-cases of management from this iteration and saved them for a possible later one, choosing to focus on the creation of the transaction report and the refinement of existing functionalities.

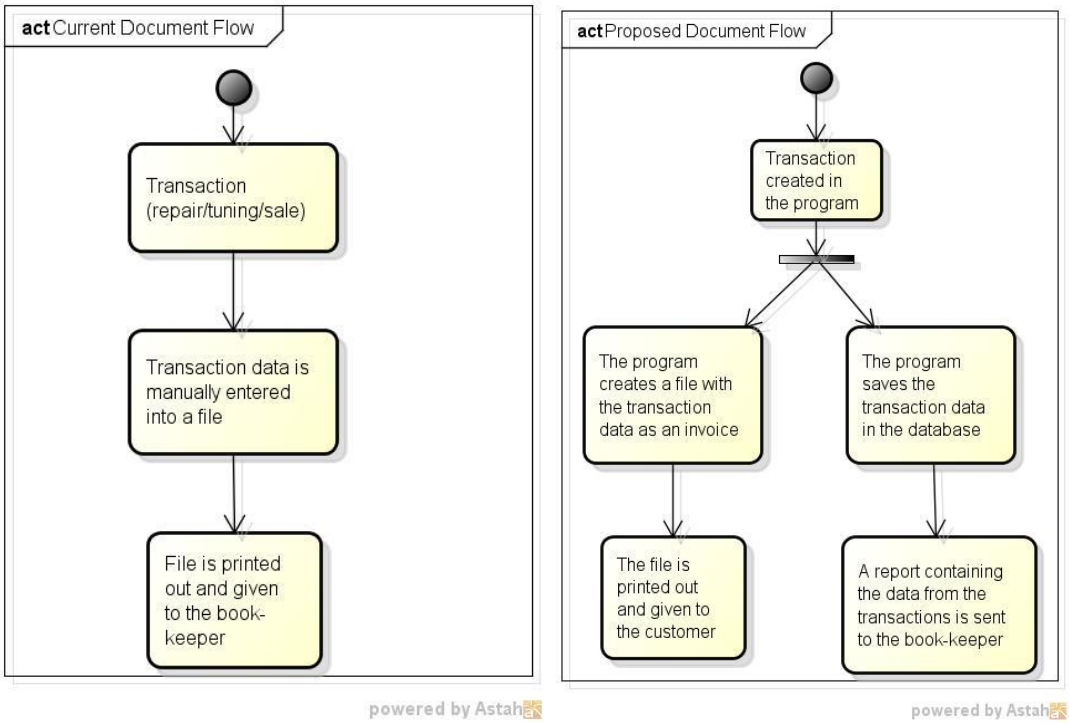
We added several tables in the database to contain the different kinds of transactions, but did not implement any add/remove methods in the program since it was not specified as needed in the assignment text, choosing instead to postpone them for a later iteration if time permits and the client's demands change. Several new views were also created and implemented in the database to allow our program to access data from the tables through those instead of customized Strings in the coding.

Part of the needed GUI elements for the management of the asset have been created, but no back-end functionality has been coded.

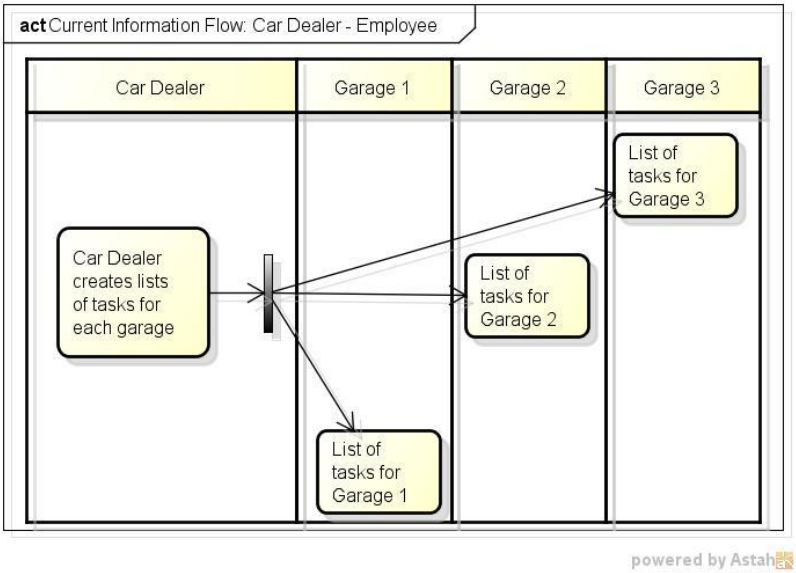
Business problems and improvements

Information and document flow:

Part of the problems in the company is the lack of document and information flow. There is almost no document flow pertaining to the business, only a spreadsheet containing data on the cars and a file containing data pertaining to various purchases and sales, the last of which often contains errors. There is also a lack of automatic creation of invoices for the customers when a transaction is conducted.



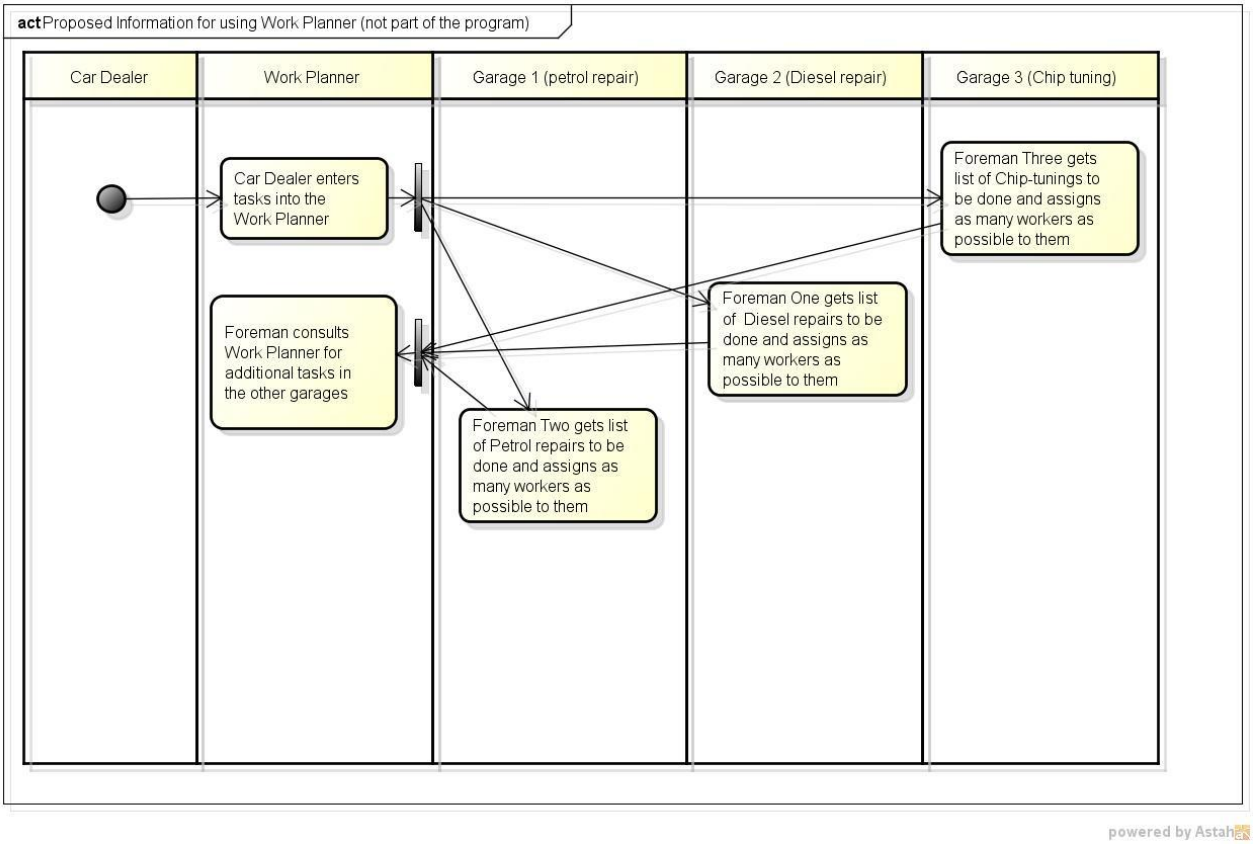
Above is the current information and document flow during a transaction and a proposed flow using the created system during a transaction.
Information about the intra-company information flow regarding the work schedule is lacking, so we are going by the assumption that there is no inter-garage communication regarding tasks and that the foreman of each garage is given a list of tasks by the Car Dealer at the start of the day and then after finishing their tasks, they make no effort to see if any of the other garages need assistance.
This could be in part due to the garages not having had the same owner before and thus having different routines and not being one group of employees, but several distinct groupings. Added to this might be the fact that they are now part of a three-layered organization whereas before they only had to work with two layers.
A way to reduce the separation between the groups might be to move the employees between the garages on a set or randomized schedule. This will not work if there are only the minimum number or less of employees with the needed specializations available to work at the different garages.



The proposed system should help improve the document and information flow by allowing greater access to the information and better communication between the owner and the employees.
No information regarding the supply chain is supplied, so we go by the assumption that the management of items for use in repairs and services are not to be part of the system we are creating.
The last part of the information flow is in regards to the customers and the information about the cars

Planning system and Organization:
The organization and planning in the company is not described in any detail in the given text, only that each garage have a set number of employees, of which one of them is a foreman. No detail is given regarding responsibilities and limitations besides a list of qualifications allowing the employees to work on different tasks.
With the owner's priorities for a planning system being diesel car repairs, then petrol car repairs and finally chip tunings, if an employee is qualified for both diesel repairs and chip-tunings, then that employee should be tasked to first complete the diesel repair and then see if there are any more diesel repairs to do, and only if there are none should he/she check the planning system for petrol repairs to be completed. If there are no waiting repairs, the employee then checks for scheduled chip-tunings to be done and start those, if there are any. This planning system depends on the individual employee having access to the planning system and working independently of each other, with only work-related communication with the other employees being through the system.
A different approach could have the foremen be the one responsible for getting a list of tasks to be completed at their garage and after completion of these to then use the planning system to see the status of the other scheduled orders and allocate tasks among their group according to priority.

This is a more hierarchical way of organizing the work groups with the information flow going from the owner through the planning system to the foreman and then to the rest of the employees.



Measuring success:

A simple way to measure improvements in sales numbers for each year is to look at the revenue versus the costs, this being a simple, but shallow method of tracking efficiency in the company. A better method could be to create several Key Performance Indicators (KPI) for the different processes. This could quantifiable things like number of repairs over time, waste of parts or average time of repairs or qualitative attributes like customer satisfaction, quality of work or employee absences on average per garage. These KPIs do not directly track the potential improvements in sales numbers, but in combination with these they allow the company to identify eventual problem areas and by comparing them to other work areas in their market, or even comparing the garages in the company, they can refine and improve their processes with the goal minimizing the identified problems.

Conclusion

This project ended up being more challenging than anticipated in regards to defining the priority requirements of the system. The current version of the program allows the user to access the database and read the data from it, as well as search with specific criteria. In addition to this the user can through the system create a text file containing a report of all the transactions stored in the database.

Future perspectives

Considering the amount of use-cases we managed to outline in the inception phase of this project, the amount of them we created functionality for in the program might seems rather small, but during the iterations we redefined the requirements outlined in the assignment text and chose to save the remaining use-cases for later iterations since they were not implicitly stated as necessary. Possible future additions to the program could include management of the database, creation and administration of a log-in feature and integrating the program into a website.
Concerning the management of the project, our efficiency might have suffered slightly from having no clear “shot-caller” in the group and no clear time estimation. We made up for this, however, by having everyone present for all parts of the development and design process and also ensuring that we all knew how far along in the process we were at all times.

Appendix

1. Diary.pdf
2. Time Estimation & Overview of iterations.png
3. Second-hand Cars UseCase Diagram.png
4. *Search a specific car Use-Case (Iteration 1)*
   - 4.1. Search a specific car Fully dressed UseCase.pdf
   - 4.2. View cars in stock SD.png
   - 4.3. View cars in stock SSD.png
5. *View cars in stock Use-Case (Iteration 1)*
   - 5.1. View cars in stock Fully dressed UseCase.pdf
   - 5.2. View sold cars SD.png
   - 5.3. View sold cars SSD.png
6. *View sold cars Use-Case (Iteration 1)*
   - 6.1. View sold cars Fully dressed UseCase.pdf
   - 6.2. View sold cars SD.png
   - 6.3. View sold cars SSD.png
7. *View sold cars in a period of time (Iteration 2)*
   - *7.1. View sold cars in a period of time SD.png*
8. *Create report of transactions (Iteration 2)*
   - 8.1. Create report of transactions Fully dressed UseCase.pdf
   - 8.2. Create report of transactions SD.png
9. *Create report of transactions in a period of time (Iteration 2)*
   - 9.1. Create report of transactions in a period of time Fully dressed UseCase.pdf
10. Domain Model.jpg
11. Design Model Iteration 1.png
12. Design Model Iteration 2.png