UCN, University College of Northern Denmark
'IT-Programme
Degree in Computer Science
CSC-CSD-S211

First Year Project

Adam Górecki, Andrej Piecka, Aziz Kadem,
Cosmin Raita, Barnabás Selymes

14-03-2022

UCN, University College of Northern Denmark

IT-programme

AP Degree in Computer Science

Class: CSC-CSD-S211

Participants:

Adam Górecki
Andrej Piecka
Aziz Kadem
Cosmin Raita
Barnabás Selymes

Supervisor: Dimitrios Kondylis

Repository path: https://github.com/PieckaAndrej/BilDoctorSystem.git
Repository number: 9081f7d

Normal pages/characters: 37,59[1] pages/90214 characters[2]

[1] characters/2400 = X,X

[2] A normal page is defined as 2.400 characters incl. whitespaces and footnotes. Frontpage, title page, table of contents, literature list and appendices are not included [2]

Table of Contents

Abstract/Resume

This report was written by a group of students, who study at UCN in Aalborg. In the report we describe our results and convey how we approached the task, which was assigned to us by our teachers. The task was to find a company preferable in Denmark and try to give solutions to their business problems with the help of the knowledge that we accumulated in the past one year. The important aspect was to take into consideration the company's technological advancement. Since we are a group of computer science students, therefore we had to choose a company to whom we can develop a system for and this software has to be responsible for providing better alternate solutions for the daily business processes, which conform better to the requirements of the user. That is why we had an interview with the representative of the company so we can measure the unique expectations so we can plan the best solution. For the planning and the implementation we had three whole months to finish. In the report in the coming chapters we present the steps, on which we went through and we give a detailed description of the logic behind our moves.

Introduction and problem statement

Student names

Adam Górecki

Andrej Piecka

Aziz Kadem

Cosmin Raita

Barnabás Selymes

Title

BIL DOCTOR – Car Repair Shop

Subject

We have chosen to collaborate with a car repair shop called "Bil Doctor", located in Sindal. We have already held a first interview aimed at identifying the basic requirements of the system that we will build and decided on a system that will be employer-centred, a useful tool for scheduling, managing employees, managing resources, creating invoices, etc.

Problem/Problem area

Currently, our collaborator mentioned that he is using a subscription-based platform that provides him with the necessary functions for carrying out his tasks. One of his company's problems is the lack of customers and the problem with establishing relations with other car repair shops and we plan to solve his problems by designing and implementing a software which will help his business grow.

Therefore, our system has to address these problems by making a more personalised, easier to use and a more complex (more features added) environment for the employer, one that will also contribute beneficially towards his business by sparing some costs related to the subscription-based platform and resolving some other technical or maintenance issues. The system will fall into the category of an Enterprise Resource Management System.

Problem statement

How can we ensure that the expected quality is met in the short time frame? How can we structure the system in order to maintain the data intact? How can we ensure that multiple users can interact with the system whilst ensuring data integrity?

Method/Procedure

The solutions for this problem statement will mainly be identified through close collaboration with the business owners. We have already identified most of the requirements and are looking forward to a think-aloud test with the company's representative, where we will go over some mock-ups that aim to give a basic idea of what we want to build. After that, we (as a team) will analyse the responses, during a brainstorming session, and will figure out what kinds of solutions we can implement so that the system can live up to its expectations. Then, we will proceed to plan all the UP phases and start working towards achieving what we already conceived.

Interview

Questions

All of our questions can be found in appendix 1.

Background/general questions

During the interview we needed to get to know the background and general information about the company to better understand its structure.

Workflow questions

We asked what the current flow of work looks like. What are the activities the employees have to do to achieve a specific goal and we got to know those goals.

Requirement questions

We found out what are the requirements of the system we're about to make and what parts of work could the system automate. We talked about the difficulties of the current system and discussed the possible solutions.

Ideas

We also proposed a few ideas to our collaborators. The mainly consisted of new features that could be added to the system in order to make it more efficient to work it and to provide a faster workflow.

UP phase plan

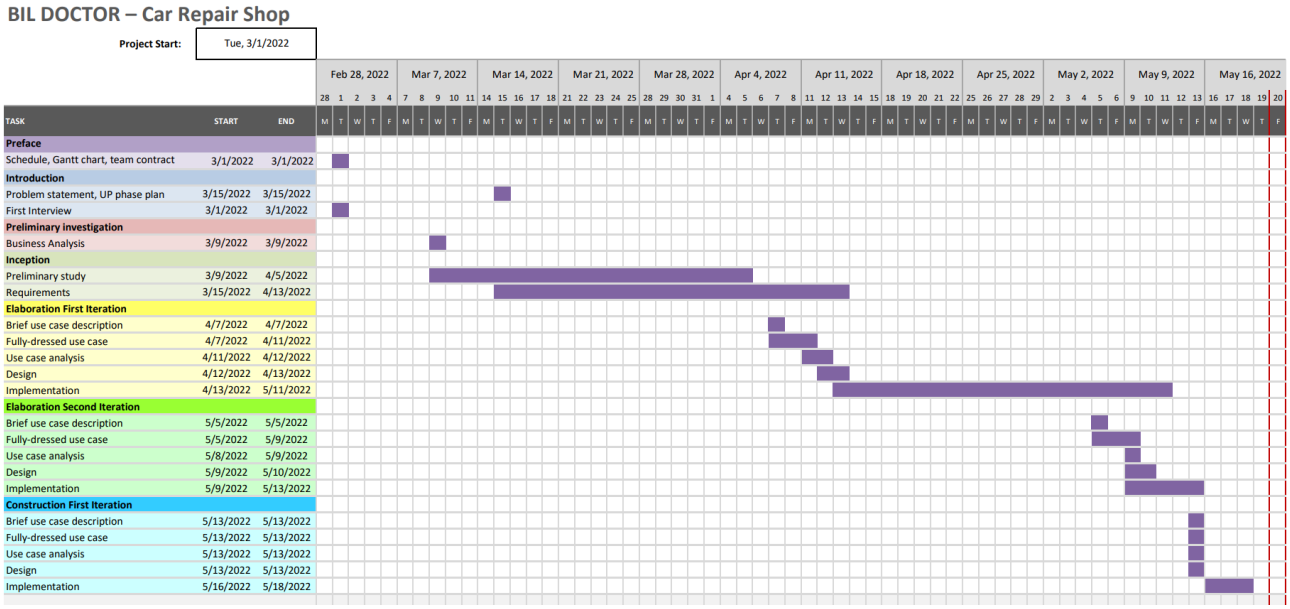
A phase plan is used to help us keep track of our progress. It shows the total number of iterations as well as include milestone dates. These milestones can show us the importance of the implemented part as well as show progress.

Our UP phase plan consists of the "Preliminary investigation" and four other iterations, which each have an allocated time of 5 working days. The "Elaboration" phase has been split into two iterations due to the complexity of the first two use cases that we will design and implement. For simplicity, we have only stated the iterations and the time periods down below, but we have also made a more detailed table that is going to be placed in appendix 2.

- Preliminary investigation - 01.03.2022 - 07.04.2022
- Inception - 07.04.2022 - 20.04.2022
- Elaboration (first iteration) - 25.04.2022 - 11.05.2022
- Elaboration (second iteration) - 12.05.2022 - 18.05.2022
- Construction - 19.05.2022 - 25.05.2022

Schedule

This project took the longest time to prepare and finish in the span of our education yet. Therefore we had to apply our previous experiences from the smaller projects if we wanted to stay on track. In the early stages of the project, we used Trello, because it is easy to access by everyone. We also came up with a Gantt chart, which is less detailed, than Trello, because Trello is more like a checklist that we want to go through unlike Gantt chart, which can be used better to maintain deadlines and is more like a general design for the tasks, so an external project manager or in our a case a teacher can have a quick overview about the state of the project.



Gantt chart

Time tracking

For time tracking we used Trello. We found it the best way in our experience to keep track of our progression from the previous projects. There we list even the smallest tasks that we have to do. Although from a project manager perspective we found the gantt chart better, because we generalise the multiple processes into one general name and it gives a nice overview of our progression. So if we should have to show the state of the project we could show the gantt chart to the stakeholders.

Team Contract

We created the team contract on the first day. We established some of our goals, exceptions, policies and the consequences of non-performance. During our work, we followed these rules and checked others if they were following them as well. The snippet with the team contract and all the other details that we agreed upon can be found in appendix 3.

Software Development process

In order to manage different projects we use the software development process. This process defines and describes the activities that must be performed to transform user requirements into an IT system. It has software development life cycles. There are different types which can be used, such as plan-driven, like waterfall or agile, like Scrum, XP and Kanban. Waterfall has separate process phrases with one sequential pass-through requirements, analysis, design, coding, testing and operations. In this type of development, it is hard to go against the current and usually we must be able to define requirements up-front. On the other hand, the strictly agile methods like Scrum and XP are using iterations, which means that there are small iterations each with requirements

implementation and test. It implements the highest prioritised functionality first and adjusts continuously.

Unified Process

We are using UP, Unified Process, this development process is a transition between the plan-driven and agile. With this, we have to document most of the things we do, but it also uses iterations. The characteristics of UP are risk-driven, use case driven, architecture centric, incremental and iterative.

Use-case driven

This means that the use cases are used to plan and control the progress of the development, it uses the use cases to describe the functionalities of the system. It is used during specifying requirements, analysing and designing so it binds the process.

Architecture centric

The architecture consideration is present from the start and it expresses the common vision of the system. UP also contains a lot of models which we use to get a better understanding of how the system should work, e.g.: the use-case model, domain model, and design model. Software architecture is the process of changing software quality criteria like flexibility, scalability, security etc. into a structured manner that meets previous expectations.

Iterative and incremental

Each iteration increments the functionality and architecture of the system. With each iteration we get an executable system which is incomplete. This leads to early visible progress which makes it possible for us to get in touch with the user and develop a system that meets the needs better, because the user can give us early feedback. It also helps us mitigate risks in the early phases.

The Unified Process has four phases: Inception, Elaboration, Construction and Transition. They all contain elements of business modelling, requirements, analysis and design, implementation, test and deployment, but to different factors.

Advantages

We can have working software early in the life cycle. It is more flexible than the waterfall, so less costly to change and go against the current. It is easy to test, debug and manage risk since we handle risks as we go on. The milestones also make it easy to manage iterations.

Disadvantage

There are of course some disadvantages of this type of development process too. One of them is that iterations are usually rigid and do not overlap. Since not all of the information is gathered at

the start it might be difficult to keep the system architecture. Another problem in a group of developers can be that they divide themselves and people always do the same thing and they receive different roles, like people who always test.

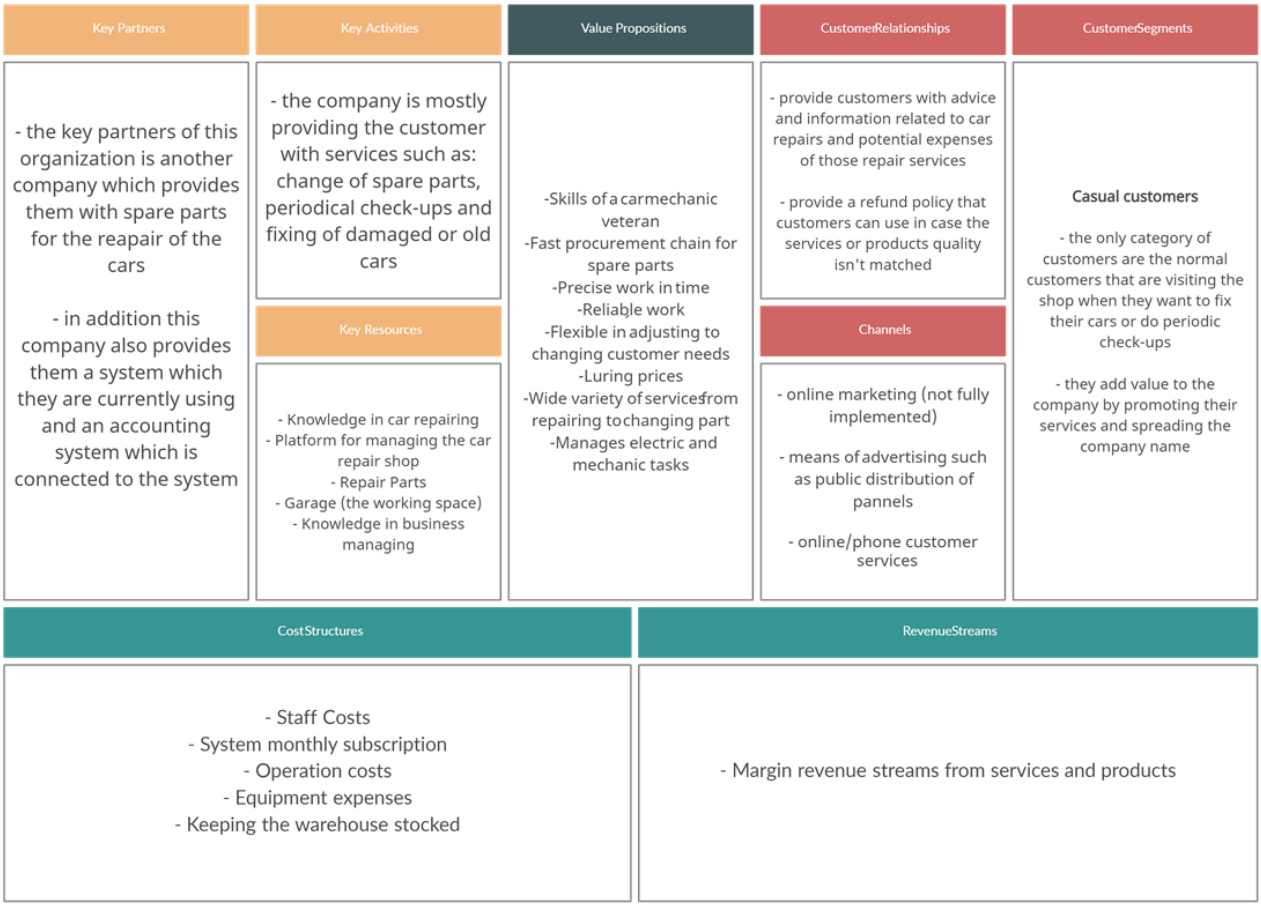
Preliminary investigation

Business analysis

We started the preliminary investigation with the business analysis.

Business model canvas

The business model canvas is a great tool for visualising business models. It represents the logical basis of how the company runs. We divide the canvas into 9 boxes. Each box conveys information about different attributes of the company. We can form two groups with these boxes. One group can be formed from the boxes on the left side. This side describes the inner characteristics of the company, like what the company has to offer, what resources can the company allocate, what are the costs of running the business. On the right side we can see the environment of the company. There we can see information about customers, how the company can segment the customers and how the company earns money. There is a fusion of the left and right side and it is the middle box, which is called the value proposition. Here we list the values that the customers are willing to pay for.



Business model canvas

SWOT analysis

We created a SWOT analysis in order to see how we can achieve our goals with the system. In this table we can easily distinguish the different aspects of our system. Such as strengths, weaknesses, opportunities and threats. Strengths are internal and helpful forces, which we can build up on in case of an expansion, since it is internal it is already present in the company. Weaknesses are internal and harmful. These are forces to which we have to pay attention and make an effort to correct them. Opportunities are external and helpful. These can be used to develop our system, but we have to keep in mind that they are external and not inside the company. The threats are harmful to the company just like weaknesses, but they are external. This makes it harder for us to change and correct, so our goal should be to minimise them. We can later use these to see what our changes should be. It is also very important to touch up on it regularly to get an accurate picture of the system. [1]

SWOT analysis

Stakeholder's grid

We created a stakeholder's grid to list the stakeholders of the company. We added the two owners, customers and the suppliers in general. We didn't find it useful to name them separately so we generalised them as "suppliers". The two owners share every characteristic. They both have the same goals and have the same risks in the company. They both put the same time and resources into the company. Small businesses, like the analysed company, have the opportunity to cultivate a strong understanding of customer needs, therefore it is crucial to design the system according to these needs. The car repair shop is in dire need of new customers, therefore their importance is high.

Stakeholder	Contribution	Interest	Influence / attitude	Importance
Mihai Mihut (CEO, office employe) Adrian Rosca (CEO, mechanic)	Money Knowledge Decisions Support	Organised and well-managed company	High/positive	High
Customers	Money Feedback	Products and car repair services	Medium/neutral	High
Suppliers	Supplying products	Delivering items to the working place	Medium/neutral	Medium

Stakeholder's grid

Risk analysis

We use the structure below for listing and analysing the identified risks for the company. There are inner risks that the company can have an influence on, but there are also outer risks, which come from the outside of the perimeters of the company. Such as competition attracts more customers, but we give a solution for each risk to increase the appeal to the customers.

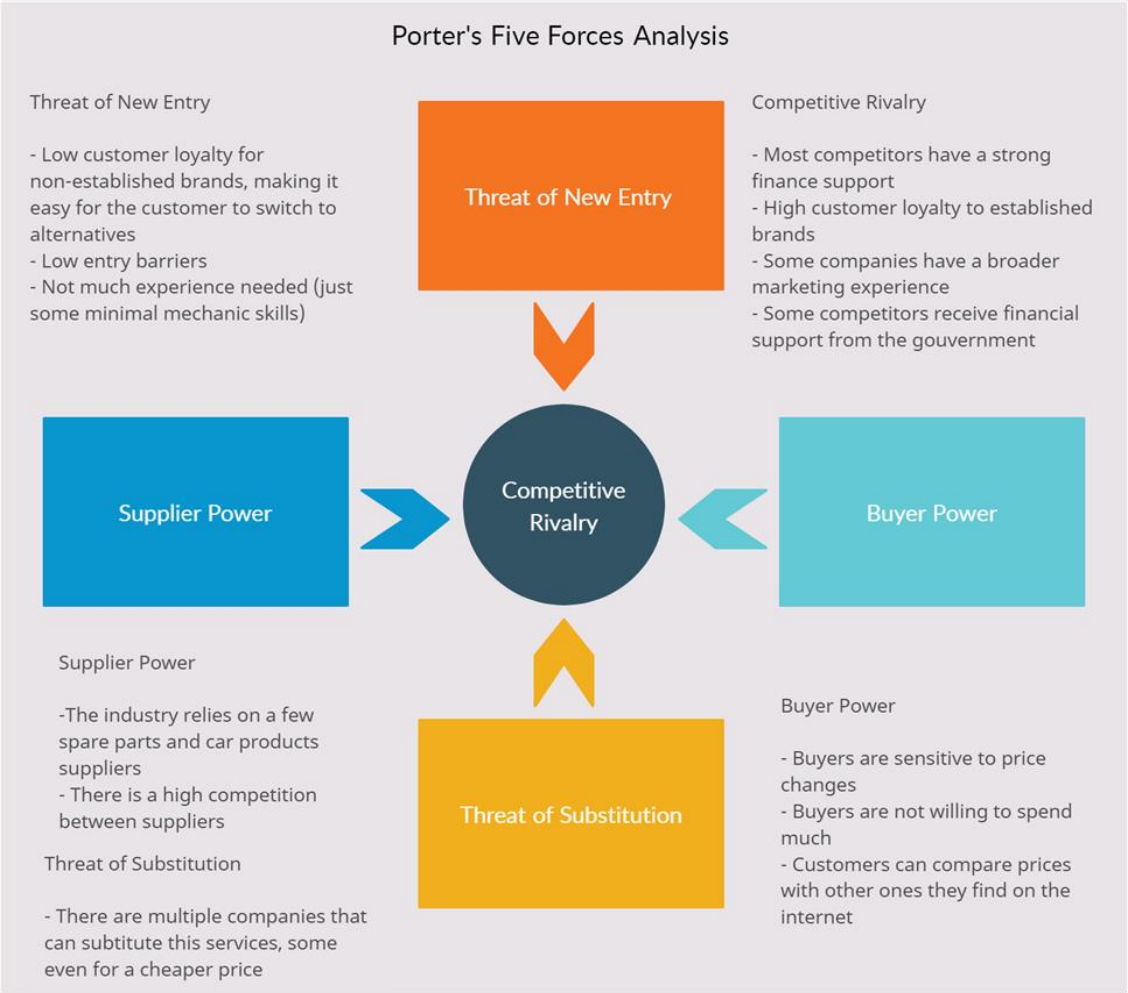
Risk	Probability	Impact	Priority	Solution
Imprecise execution	4	3	12	Ensure that the requirements are clear by frequent contact with client
Staff not confident with current system	4	2	8	Demand precise information about current flaws of the system
System is slow and has errors	4	1	4	Keep the iterative process and test every newly implemented feature

Old and outdated hardware	2	2	4	Analyse current hardware and assess amount for future changes and updates
Customer base regress	2	2	4	Create a system that makes customer service faster and more reliable
Competition attracts more customers	3	1	3	Design a system that is more appealing to the average customer

Risk analysis

Porter's five forces

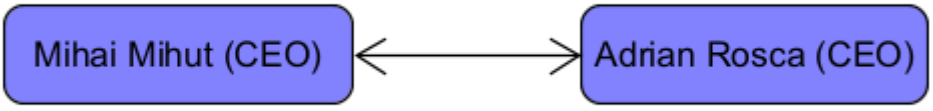
Porter's five forces model is suitable for analysing the microenvironment of the company, more specifically the competitors in the market, who the company competes with. We analyse how easy it is to break into this market, how strong is the bargaining power of the suppliers, and how dangerous are the substitutional services. In general, what is the state of the rivalry-companies and the buyer's power of customers. See this analysis below on the diagram. [2]



Porter's five forces

Organisation type

The organisational structure is quite a basic one. We are facing a startup business that was created only about a year ago and it was conceived by two people that have different responsibilities within it. The first person is Mihai, which is the accountant of the business. He keeps track of invoices and manages the stock of products that they buy from car retailers. On the other hand, Adrian is responsible for the physical job. He is a skilled mechanical engineer that does all the practical work, but he also has a few organisational tasks at hand. All in all, we decided that our organisation fits in the "Entrepreneurial Startup" type.



Organisation type

Business mission and vision

The business mission and vision is a part of the business analysis and preliminary investigation and they are created in order to match the new goals and changes implemented with the new system. Both are used to support the new business case and the system vision that is created. They also show an overall direction for the business, which they can adapt later.

Mission

To maintain a relevant position in the car-repair market and provide customers with high-quality services and products for their cars.

Vision

To create a world environment where people can forget about having car issues.

Conclusion

In the preliminary investigation we analysed the overall state of the company that we are building the system for. We did a thorough business analysis which contained the weak and the strong points of the company, but also solutions of overcoming the issues and thriving in the market. This information will give us a better understanding of how to approach the domain of the project and it will help us in the long term with gathering the essential requirements. Setting the scope from the start is vital mainly because resources like time can be pressing in case realistic expectations are not set. We can, therefore, deduce that the preliminary investigation is an important starting point for the whole project and it resembles a step in the right direction of achieving success.

Inception

Here we outline the scope of the system, we start to consider the architecture in this phase. We have to identify critical risks here to get an understanding of how we can mitigate and address them. This is where we are supposed to decide if the system is worth making or not, what is confirmed by the business case.

Introduction

We spent one iteration in the Inception, here we created the business case, mockups, system vision, captured how the current system works in a workflow, created an employee goal table, which we later used to identify the use cases. After prioritising and creating a domain model, we focused on the database and created a relational model. We also included a glossary, supplementary specifications and business rules.

Preliminary study

We started the preliminary study by making the business case.

Business case

Executive summary

Collaborator is currently using a subscription-based platform providing him with necessary functions for completing the most basic tasks in his company. The customer is not confident with his system and states that it lacks many important features that could “make his life easier”. A personalised system would spare costs of the subscription in the long term and would make his work more efficient.

Reasons

Reasons for the creation of the new system in order to replace the old, subscription-based one. The company (office employee) faces multiple problems with the current system, including

1. Use of outdated hardware that makes the unreliable system work slowly, causing inconsistency at work
2. Ineffective work due to many errors and problems with the current system’s database
3. The customer base is not consistent since there is no feature in the system that would remind customers of regular car check-ups.
4. Company faces significant competition, that provides better services when it comes to customer interaction (CEOs of Bill Doctor don’t know the Danish language)

Business options

1. Do nothing. As it seems like the simplest and least demanding method, in the long term it may cause troubles for CEOs of the company. The customer amount would be stable, costs of the subscription for the current system would not change, but this option doesn’t guarantee any prospects of further development of the company, while competition might grow and take over the clients.
2. Implement a new, personalised system, that would spare subscription-related costs for the company, and make work way easier. This solution would decrease the time of the routine processes and would receive support and attention from the programmers, in order to improve and solve further problems.
3. Invest only in new hardware. As it would make the working process easier and more effective, the costs of subscription will remain the same.

Expected benefits

Faster workflow with smaller chances of mistake. Better customer experience and employee satisfaction. Fewer issues with the system and a stable customer base.

Expected dis-benefits

One time cost of the system will be significantly over the current costs of subscription-based system, although it will bring great benefits in the long term.

Timescale

Project time: 2 months

Costs

New hardware: 15 000 dkk

Software 100 000 dkk

Operational costs: 2 000 dkk (yearly)

Major risks

System may not satisfy the customer's needs, the customer base will start regressing without significant improvement.

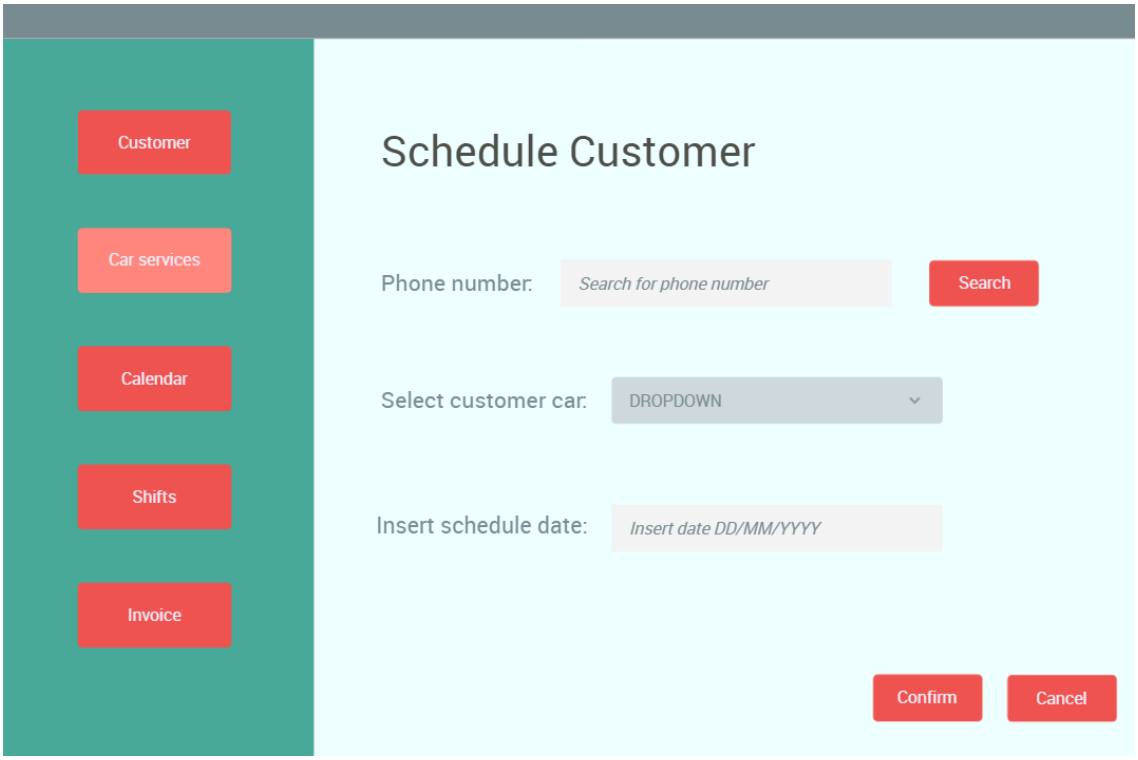
Investment appraisal

	Year 1	Year 2	Year 3
Project costs	115 000	0	0
Operational costs	20 000	20 000	20 000
Benefits	90 000	100 000	125 000
Net benefits	-45 000	+35 000	+140 000

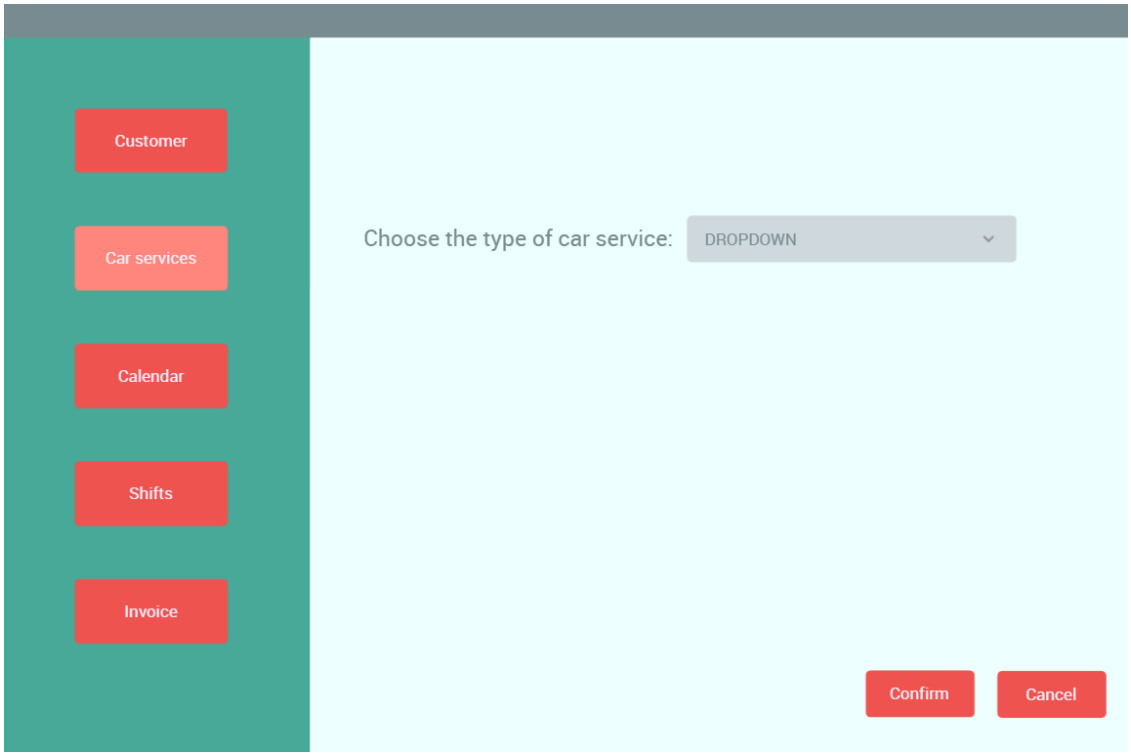
Investment appraisal

Mockups

Before we started to design and implement our system, we had to create a few mockups in order to figure out how our menu, use cases and design elements are going to look like. Below we only showcased the scheduling of an appointment, because we consider it to be a high priority use case. For simplicity of the report, other mockups snippets will be included in appendix 4.



Mockup schedule customer



Mockup car services

System vision

Introduction

The system is being developed for a Car Repair company called Bil Doctor, which is used for registering customers with their vehicles, seeing information about the vehicles. *The purpose of this document is to collect, analyse, and define high-level needs and features of the* Car Repair System*. It focuses on the capabilities needed by the stakeholders and the target users, and* ***why*** *these need to exist. The details of how the* Car Repair System *fulfils these needs are detailed in the use-case and supplementary specifications.*

Purpose and boundary

The purpose of the system is to be able to efficiently take care of all the necessary functionalities that the business requires in order to maintain the stability of operations done by the employees, processes that fall into the category of the identified use cases, such as: creating a sale, scheduling appointments, register check-ups, order products and manage resources and stakeholders.

Problem description

The main problem of the current system is the existence of a less developed system that can be improved by adding some extra features that will greatly benefit the current workflow tasks. This problem can be solved by implementing a system that can be easily maintained and where the users can also find a history of transactions, where they can do the re-stocking for their garage and where they can manage a significantly large database that holds the services that they provide, the available items, their customers and also the customers cars.

References

The template for this Glossary is provided here.

Positioning

Problem Statement

The problem with	establishing relations with other car repair shops
affects	lack of customers
the impact of which is	less money
a successful solution would be	to solve his problems by designing and implementing a software which will help his business grow

System vision - Problem Statement

Product Position Statement

For	the 2 owner of Bil Doctor
Who	needs a better system
The Car Repair System	is a management system
That	can be easily maintained and the users can access the previous sales

System vision - Product Position Statement

Employee-goal table

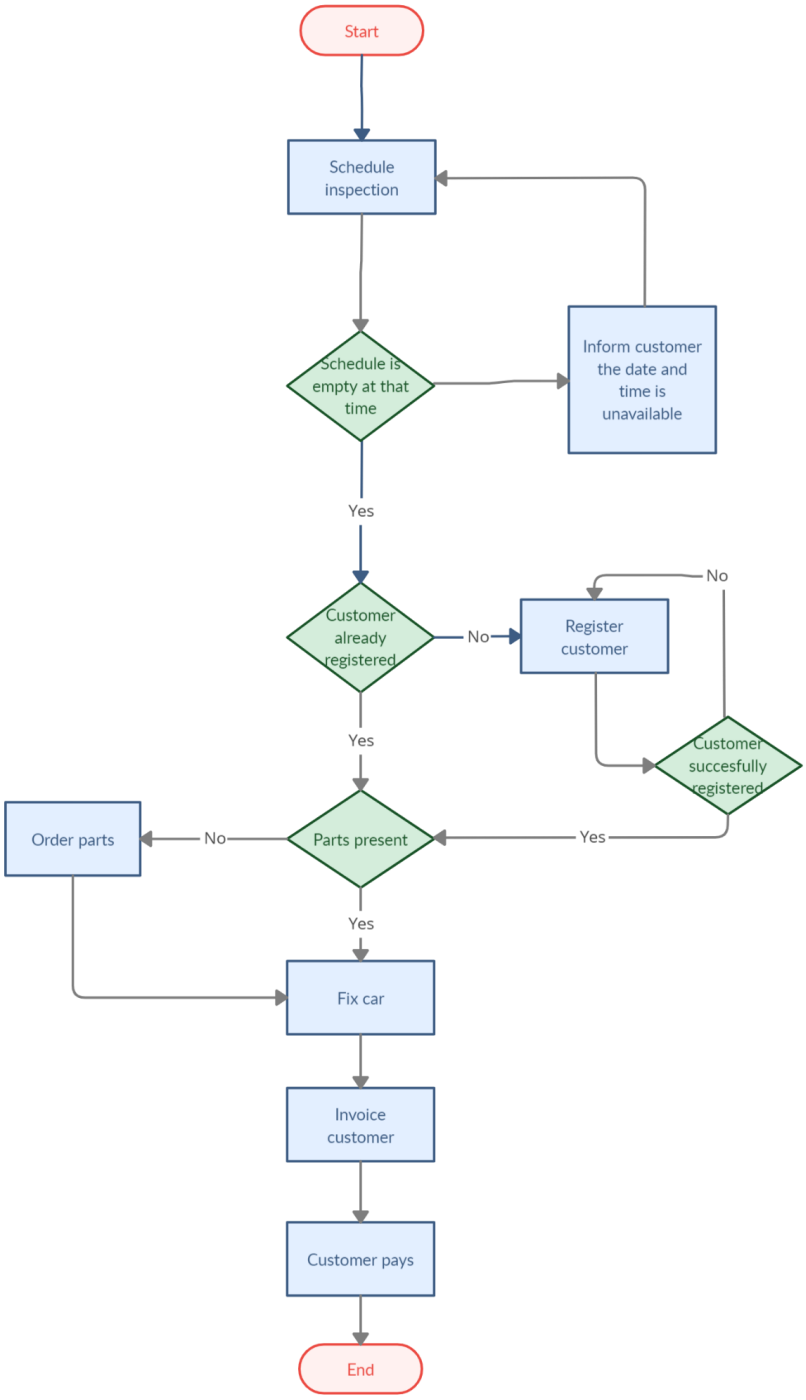
To identify the actors and the use cases we created an actor goal table. We have two actors, who are interacting with the system. On the right column there are the listed activities. A Mechanic/ CEO has access to the system to make him independent from the Office employee/ CEO. So the Mechanic can order the necessary parts without waiting on his colleague. Every other activity is the responsibility of the Office employee/ CEO.

Employee	**Goal**
Mechanic/ CEO	See available parts Order parts
Office employee/ CEO	Order parts Register customers Register vehicles Register invoice Managing employees See available parts

Employee-goal table

Workflow

Workflow is a representation of the current order of actions done in the company by the actors. In the diagram below we listed the main activities that are performed on a daily basis by the employees. This diagram will give us a better understanding of the tasks that have to be done in a normal day and will eventually give us ideas for use cases and conceptual classes that we will use later in designing the system. [3]

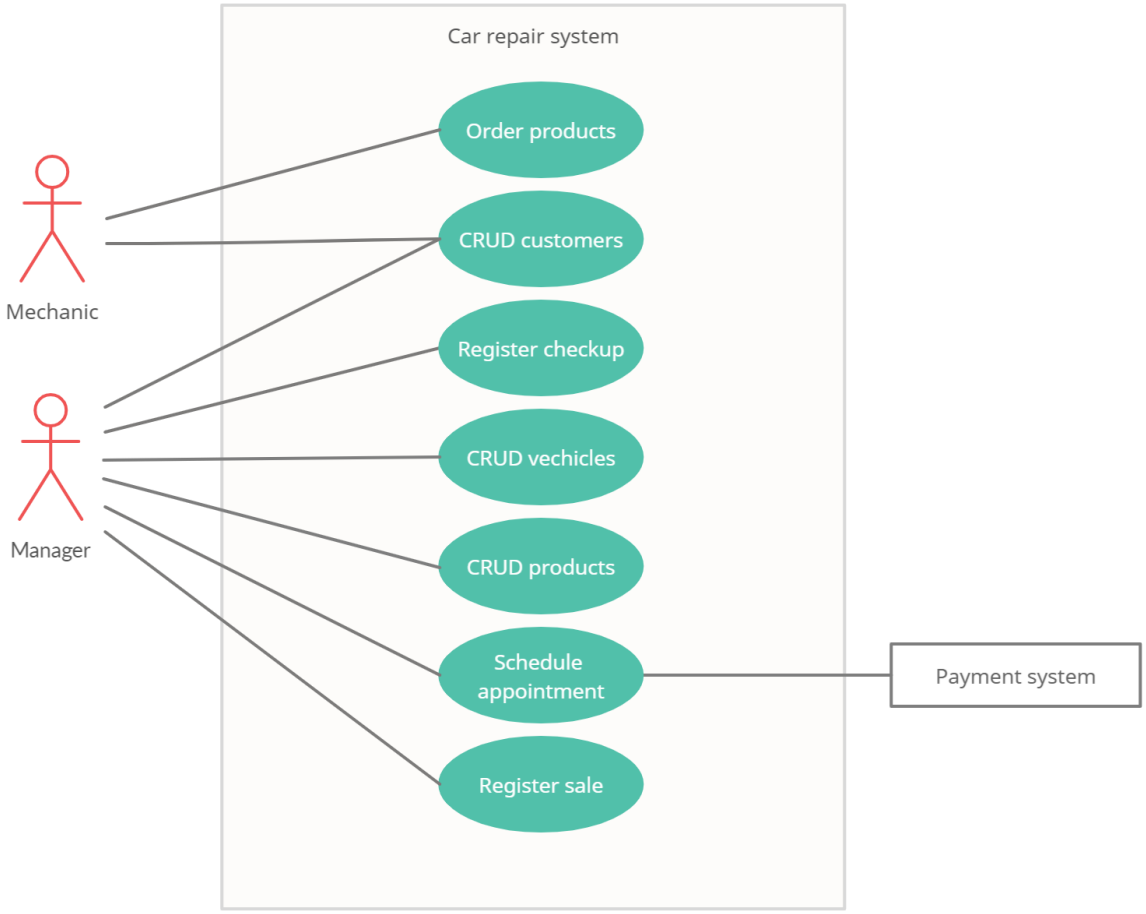

Start
Schedule inspection
Schedule is empty at that time
Inform customer the date and time is unavailable
Yes
Customer already registered
No
Register customer
No
Yes
Customer succesfully registered
Order parts
No
Parts present
Yes
Yes
Fix car
Invoice customer
Customer pays
End

Workflow

Requirements

Use case diagram

In software engineering, we have this commonly used term called use case. A use case defines an activity, which the user carries out in order to finish a task. [2] We identified the following use cases:



Use case diagram

Use case priority

We prioritise our use cases in order to make a sequence from them. We follow this sequence so the important use cases are going to be finished earlier. As the diagram below shows, we are ranking our gathered use cases by assigning them two indexes that are measured on a scale from 1 to 5, the business importance and the complexity of the use case. These values are assigned based on the business value that they can provide to the company, meaning how much revenue they can generate by themselves (in the case of the business importance) and their designing and implementation complexity, referring to the time that it takes to map and code the specific use case

(in the case of the complexity). We then multiply the two indexes and based on the value that we are getting, a ranking position is assigned to each use case accordingly. In our case the “Register sale” use case is the highest priority one, because it can provide the highest revenue for the company and it will include the majority of classes from the domain model, also making it the most complex use case.

Use case	Actor	Business importance	Complexity	Ranking
Register sale	Office employee	5	4	1
Schedule appointment	Office employee	4	3	2
Register checkup	Office employee	5	2	3
Order products	Office employee	4	2	4
CRUD customers	Office employee	3	2	5
CRUD products	Office employee	3	2	6
CRUD vehicles	Office employee	2	2	7

Use case priority

Brief use case descriptions

Register sale - The office employee, after finishing the service, chooses a customer for whom the service was done. Then, he can input the services that were provided and choose the products that were used from the database. The sale is created and the system asks for payment.

Schedule appointment - When a customer calls to schedule an appointment, the office employee opens the calendar and chooses a date that fits the customer. Then the date is registered and added to the system.

Register checkup - The office employee will be reminded when a customer's checkup period is going to come close. Then, the employee can notify the customer and ask for permission to make an appointment for them.

Order products - The request for new products is sent to the providers when the stock amount is under chosen amount or when the employee decides to order.

CRUD customers- create, read, update and delete customers.

CRUD products- create, read, update and delete products.

CRUD vehicles- create, read, update and delete vehicles.

Domain model

Class candidates

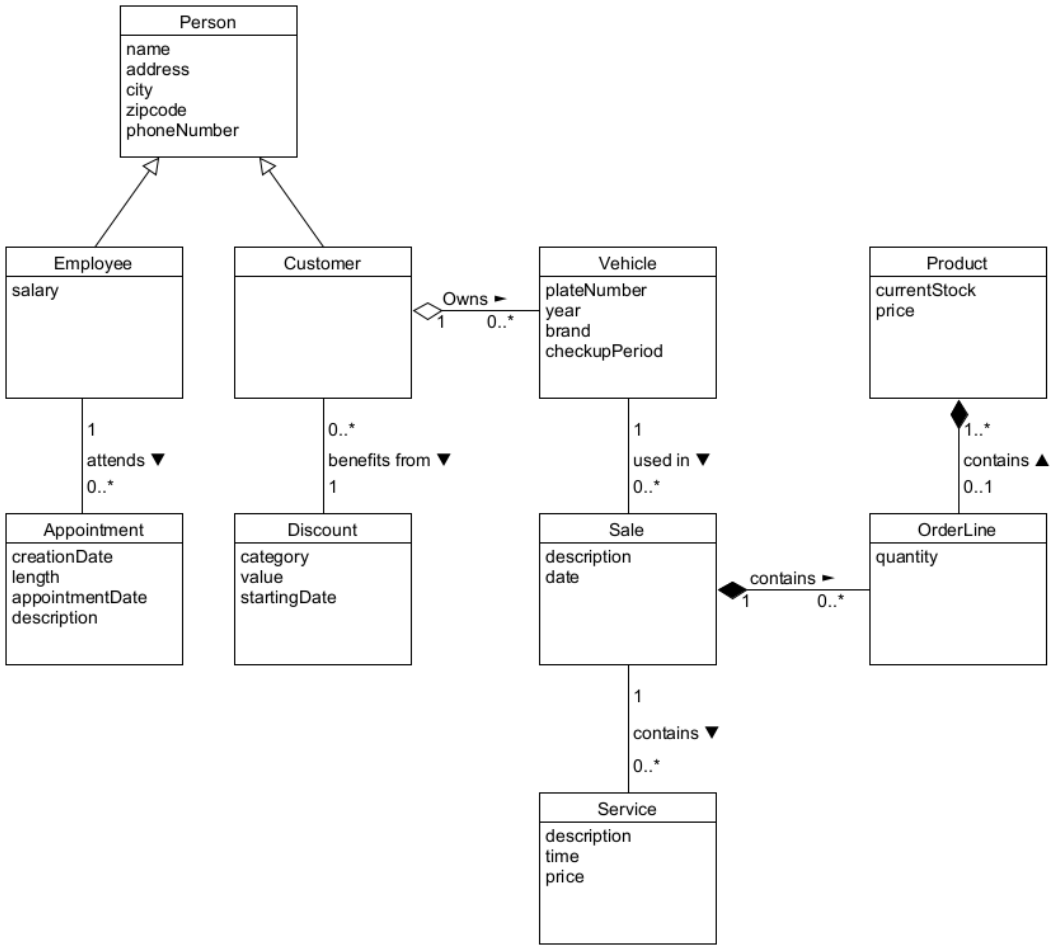
Before creating the domain model we took into consideration what the different classes could be. Then we evaluated each of them on why they should be involved or not involved. After that, we decided whether they should be included in the domain model or not.

Candidates	Evaluation	In/out
Sale	The sale of services and products	In
OrderLine	A part of Order	In
Customer	The customer who owns a car that is being worked on	In
Employee	An employee working in the car repair	In
Payment		Out
Product	Items used whilst providing the services	In
Vehicle	Vehicle of a customer	In
Invoice	Invoice saved after the sale, can be part of the sale	Out
Appointment	Scheduled appointment	In
Service	Service done to the vehicle	In
Day	Every day in the calendar	Out

Class candidates

Diagram

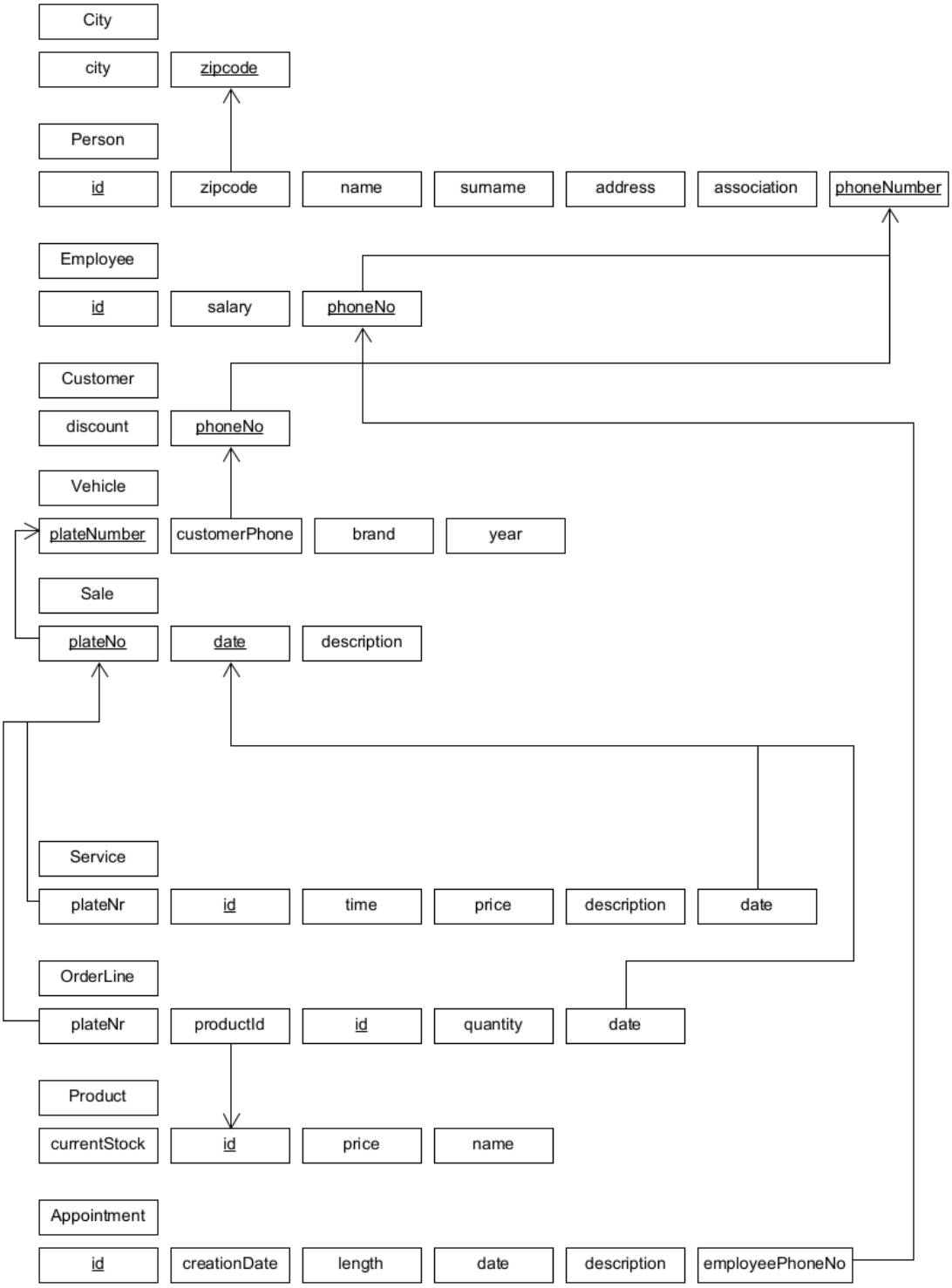
Using the selected candidates a domain model is created. In the domain model we aim to give the user perspective of the system domain by mapping the conceptual classes and assigning the relevant attributes to them. The classes are interconnected in a way that we achieve our system's goals by allowing different components to communicate with each other, maintaining different relations. For example, in this domain model we are showcasing the Person class which is a superclass for the subclasses Employee and Customer, creating a generalization relation between these 3 classes. In the domain model we can also find aggregation and composition relations. The Vehicle class uses an aggregation link with the Customer class, whilst the OrderLine class has composition like with the Sale and the Product, which means that in case a product or a sale is deleted any orderline that is associated with it is going to be deleted as well. Other than the special relations we also have the normal associations between the classes, which are represented without the rhombs. The connection lines between the conceptual classes also have a multiplication count and an arrow that indicates the way the relation takes place, like in the case of the Vehicle and the Customer where we can deduce that a customer in the system can have 0 to multiple vehicles and the customer is owning the vehicle/s.



Domain model

Relational model

This model is created according to the domain model while trying to follow the BCNF. It is a map of the database that shows the relations between the tables, by assigning them primary and foreign keys, whilst also respecting the normalisation forms' rules. The underlined attributes resemble the primary keys, whilst the column pointing at the primary keys are foreign keys for that particular table. In order for the relational model to respect the 3.5 normalisation for, it must in the first place respect the third one and "each attribute must represent a fact about the key, the whole key, and nothing but the key". As shown in the figure below, we are including all the conceptual classes and their attributes, whilst adding some other 'ID-s' and tables that can sustain the relations of the domain model. [4]



Relational model

Supplementary specifications

Introduction

The supplementary specifications are non-functional requirements that are referring to qualities of the system that are different from the use cases. In this sense, the supplementary specifications are divided into the URPS scheme, which basically appeals to Usability, Reliability, Performance and Supportability. To all these concepts we are also adding additional requirements such as resource limitations, hardware, interfaces, business regulations and legalisation, that are all fitting for the supplementary specifications.

Purpose

The purpose of this Supplementary Specification is to grant the reader knowledge of non-functional requirements we discovered are necessary for creation for the project.

Scope

The scope of the project is very wide. Bill car rental requires a system that will help with day-to-day tasks of the office employees. Customers require the system to be reliable, quick, and that it can be accessed from multiple devices.

References

The template for this Supplementary Specifications is provided here.

Overview

The rest of the document will include the overview of non-functional requirements with descriptions and introduction to the work we did during the project period.

Functionality

Within our scope, we have identified a handful of functional requirements. The requirements are mainly aimed at providing the basic functions that can bring profits to the company. Use cases such as “register sale” or “schedule appointment” are essential for carrying out the basic activities of the business.

Functional Requirement

The highest priority requirement is registering a sale. This use case ensures that the system can support the creation of the sale and ultimately create revenue for the company. That is the reason for ranking the functionality so high.

Usability

The required training time for a normal user is not that significant, since the interface itself is user-friendly and the system follows the workflow, with which the employees are already used to. The

tasks carried out with the help of the system are similar to those done on paper, physically, but the time of the completion is greatly reduced.

Usability Requirement

- Intuitiveness - the interface is easy to navigate through. Buttons are easy to understand and the system is intuitive. Time needed to learn the system from 0 for the user is not more than one working day.
- Responsiveness – system reacts quickly, pop-up windows are not showing up slowly and the user can smoothly navigate.

Reliability

Availability – We can ensure that our system can have a 99% availability rate and it will mainly be used during the working period of the garage shop which is every working day from 8 to 17. The system has to usually be allowed between 5-10 minutes in which you cannot perform operations whilst using it, since it has been failed. The bugs and defect rates will be minimalized with the help of test classes and with the usage of design patterns.

Reliability Requirement

Availability – It is really important to ensure that the system can be up and running for the maximum possible time.

Performance

Our system is designed as a tool for the employees of the business, therefore the issue with the number of customers accessing it at the same time is not relevant enough. The system will only be used by a fixed number of employees so the outcome of the maximum capacity utilisation is predictable.

Performance Requirement

Response time – It is vital that the system can have a quick response time so there are no delays in the workflow and the physical tasks can be carried out with success in a short time after the order is placed.

Supportability

Whilst implementing the system, we aim at using coding standards and keeping all the conventions that are known amongst the developer's team. This ensures that the system can be understood easily and modified in case of need, without causing too much trouble.

Design Constraints

We have decided to implement our system on a three-layered architecture. The language that we are using is Java and the IDE is Eclipse. The code is shared through a github repository and the design

is made using patterns such as the “Singleton” pattern, “Order-order line” pattern, “DAO” pattern and “Composite” pattern.

Glossary

Introduction

The glossary is a document that is meant to define terms of the report that are hard to be defined or that are unknown to the reader. This document will contain the following sections: the purpose of it, the scope, a reference to the template, the overview and some definitions about terms and UML stereotypes.

Purpose

The purpose of the glossary is to clarify terms mentioned throughout the report for the reader. By doing this it can eliminate any misunderstanding regarding technicalities or complex concepts and make the document more readable and more concise overall.

Scope

The scope of this project covers the technical terms used within the report made for our system. It includes explanations of the expressions and words used by developers and designers and aims at making them more understandable for the broad audience/readers.

References

The template for this Glossary is provided here.

Overview

This document contains two main parts, which are essential for its structure and those are the definitions and the UML stereotypes. In the definition section we will explain technical terms, whereas in the other one we are describing the UML models that we are using.

Definitions

In this section we will describe a few terms as it follows:

Terms

- UP - Unified Process is an iterative incremental process, structured on interactions, that contributes to a better planning of the project
- Mockups - Is a full-scaled model, it is created to help us represent parts of system which we can later use for describing the functionalities
- Workflow - A diagram that describes the entire set of automatized tasks performed by an employee.

UML Stereotypes

In this section we will list a few design patterns and diagrams that are used within our project as it follows:

- Domain model - A domain is a conceptual diagram that holds the whole domain of the project, showcasing the links between the system classes
- Use case diagram - A use case is, in short, a functionality of the system. The use case diagram holds all the requirements identified in the preliminary study.
- Singleton pattern - The singleton pattern is a design pattern that ensures that only one instance of the class that is build on can be created
- DAO pattern - The DAO pattern is a structural pattern that allows the developers to isolate the architecture layers

Business rules

In this part we cover what a business rules document would normally cover. The purpose of the business rules are to form a basis for automation. We state the business tools, business architecture and business style in this section. This part of the report will also help to give a better understanding over how the business operates and we can define each process - that we want to highlight - exactly how that specific task should be executed, so you can refer later back to this document if someone deviates from the path.

Scope

The scope of the document associates with the scope of the project, which covers the car repair system.

Overview

We use this format to explain our business rules and sort them into a group of business rules. A business rule is basically a process and we define how it should be executed.

A business rule

Category	Name of the process	Approved proceeding
Override with explanation	Set Discount	If the customer has key importance for the business, then the employee is allowed to make exceptions and give extraordinary discounts
Override by pre-authorized actor	Set Salary	The salary cannot be changed without the permission of the owner.

Override with explanation	Put two appointments into the same time period	The employee is allowed to put two appointments into the same time period only if he can guarantee the satisfying service of multiple customers.
Guideline	Put one appointment into the calendar	The employee is allowed to put appointments into the calendar, when the customer makes contact in any way.
Guideline	Set description and price of the sale	It is the employee's responsibility to write meaningful descriptions and set a price, which correlates with the invested resources
Strictly enforced	Delete vehicle	The employee is not allowed to delete a plate number from the database.

Business rule

A business group rule

Our system is small to categorise business rules, because the scope is too small for that, therefore we use 4 levels of enforcement. Any categorization would be unuseful, when only one person interacts with the system, but if the business starts to expand and they hire new employees, then the categorization comes in handy. See it below:

Enforcement Level	**Description**
Strictly enforced	Violations are disallowed in all cases - achieving some new state successfully is always prevented
override by pre-authorized actor	The behavioural rule is enforced, but an actor with proper before-the-fact authorization may override it .
Override with explanation	The behavioural rule may be overridden simply by providing an explanation
Guideline	Suggested, but not enforced

Business group rule

Conclusion

The inception phase represents the first iteration of the project and it focuses on handling the gathered requirements from the preliminary investigation. In the inception we are focusing on settling on some system functionalities that are important for the system. These functionalities are going to be placed within use cases, which are then ranked by their priority. After choosing the sequence of the use cases that we will work on, we made the domain model, which is the most important diagram for the system mainly because it showcases all the conceptual classes and the relation between them. The inception phase will, therefore, give us a head start for designing and, ultimately, implementing and testing our software.

Elaboration

In this phase we have to find the main body of the functional requirements and start to implement them. We have to establish the architecture which we already considered in the Inception. We also have to keep attention to risk and address them as they come up. We spend most of time designing and analysing but we already start implementing a fair amount and the business is still heavily present as well.

Introduction

In the elaboration we included two iterations that will address the two most important use cases. We chose to separate the elaboration in two iterations because the use cases that we included for this phase are complex enough to be a part of the elaboration. The main purpose of these two iterations will be to design and implement the use cases according to the diagrams and information that we already have from the inception phase.

First iteration - Register sale

Brief use case description

Register sale - The office employee, after finishing the service, chooses a customer that the service was done for. Then, he can input the services that were provided and choose the products that were used from the database. The sale is created and the system asks for payment.

Fully dressed use case description

This description is one of the most important parts of this iteration, because every part of the iteration follows this fully dressed description.

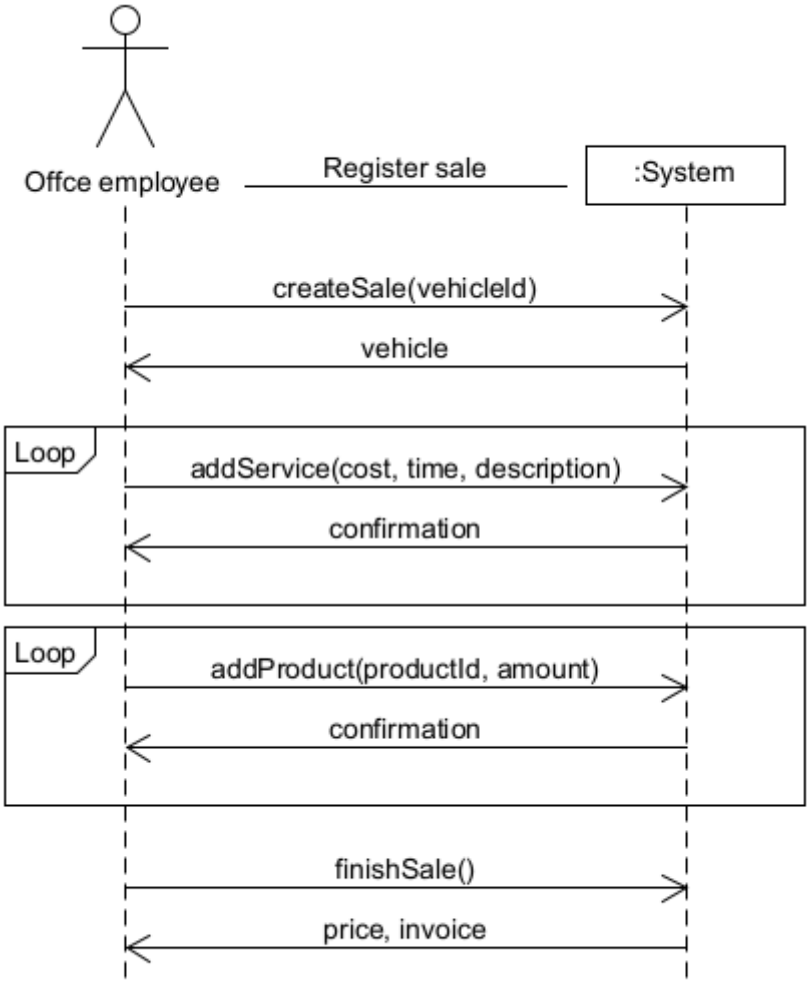
Use case name	**Register sale**	
Actors	**Office employee**	
Pre-conditions	**Product, Customer and Vehicle exist**	
Post-conditions	**Sale is registered and invoice is sent**	
Frequency	**Every time the service is provided to the customer**	
Main Success Scenario	**Actor (Action)**	**System (Response)**
	1. Finds a vehicle that the service was done for	**2. Returns the desired vehicle**
	3. Inputs performed services on the vehicle	**4. Registers the services to the sale**
	5. Inputs products that were used	**6. Finds products in the database and registers them to the sale**
	7. Confirms sale	**8. Creates the sale and makes the invoice**
Alternative flows	***2a. Vehicle not found*** ***Employee creates a new vehicle*** ***6a. Products not found*** ***Employee creates a new product*** ***6b. Not enough products in the database (stock too small)*** ***System informs customer of the size of the stock of the product*** ***7a. Employee removes product from the sale*** ***System deletes the product or service form sale*** ***7b. Employee cancels the sale*** ***System returns to the main screen and cancels all the progress***	

Fully dressed use case - Register sale

Use case analysis

System sequence diagram

System sequence diagram is constructed according to the fully dressed use case description. It portrays interaction between the user and the system, where we don't know how the system works - black box analogy. The calls that the user makes resemble methods that are called on the UI. [5]



SSD - Register sale

Operation contracts

The operation contracts define the precondition in order to run the operation. If these preconditions are not met then the operation won't start at all. We use this tool to indicate the outcome of the operation in our system. [5]

Operation	createSale(vehicleId)
Use case	Register sale
Precondition	vehicle is in the database
Postcondition	Object sale is created and a vehicle is assigned to the sale

Operation contract 1 - Register sale

Operation	addService(cost, time, description)
Use case	Register sale
Precondition	Object sale exists
Postcondition	Service is added to the sale object

Operation contract 2 - Register sale

Operation	addProduct(productId, name, amount)
Use case	Register sale
Precondition	Object sale exists and the product is present in the database
Postcondition	Product is added to the sale

Operation contract 3 - Register sale

Operation	finishSale()
Use case	Register sale
Precondition	Object sale exists with at least one service or product
Postcondition	The sale, services and order lines are registered in the database and the invoice is made. The product has decreased current stock.

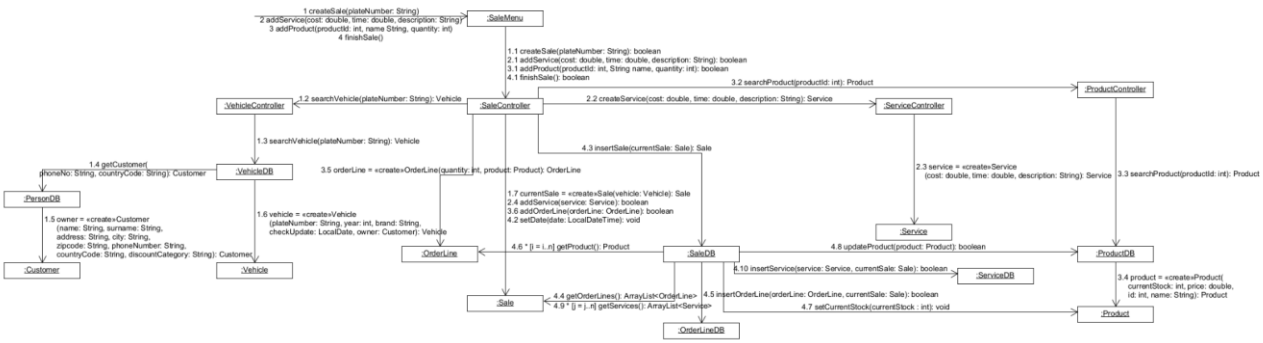
Operation contract 4 - Register sale

Design

Communication diagram

We designed the interaction diagram to illustrate the interactions between the classes. The classes are inside lifelines and we connect these lifelines with lines. We place our messages on these lines. The messages are the methods that are being called between the lifelines. The reason why we create an interaction diagram is to get a better understanding of the behaviour of the system. We place the sequence number before the methods. The parameters and the datatypes are also illustrated on the diagram. [5]

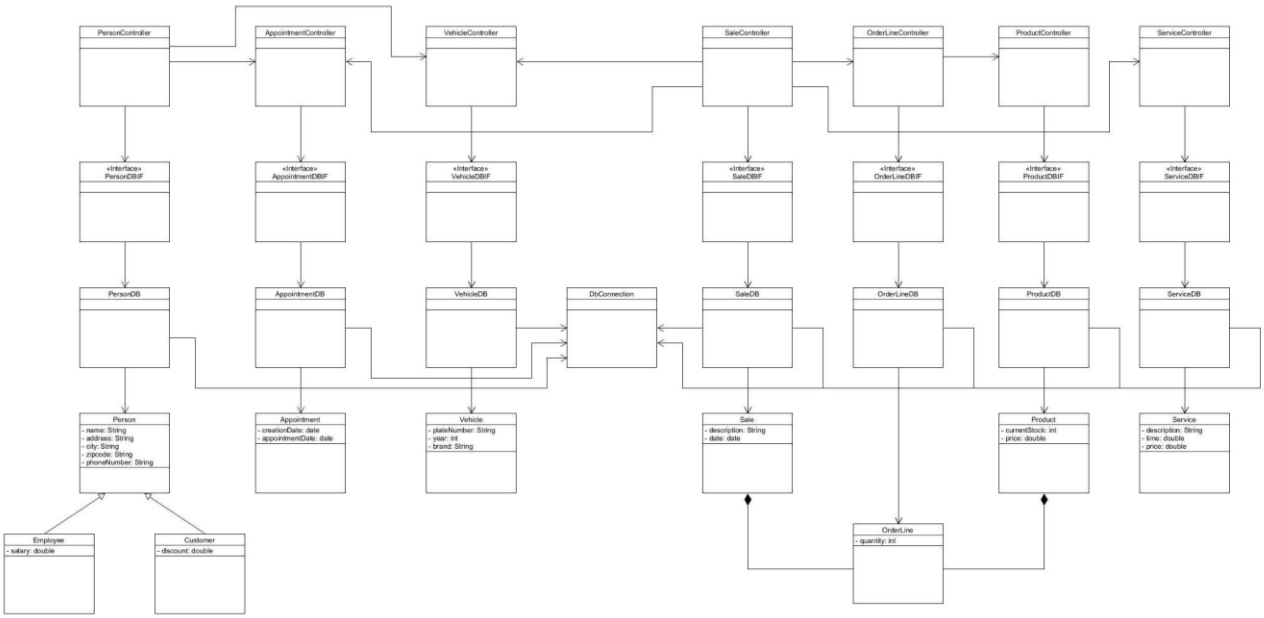
In this communication diagram we are creating a sale by passing the plate number of the car that we will use for the sale to it. Then, we are adding products and services to the sale by using loops, until everything that was included in the sale is added. Then, the user can choose to finish the sale, which will ultimately print out the information and the total price of the sale. All the steps of the process are shown in this diagram, with the help of the arrows that highlight which methods and attributes are passed between the classes, respecting the main system operations from the SSD.



Communication diagram - Register sale

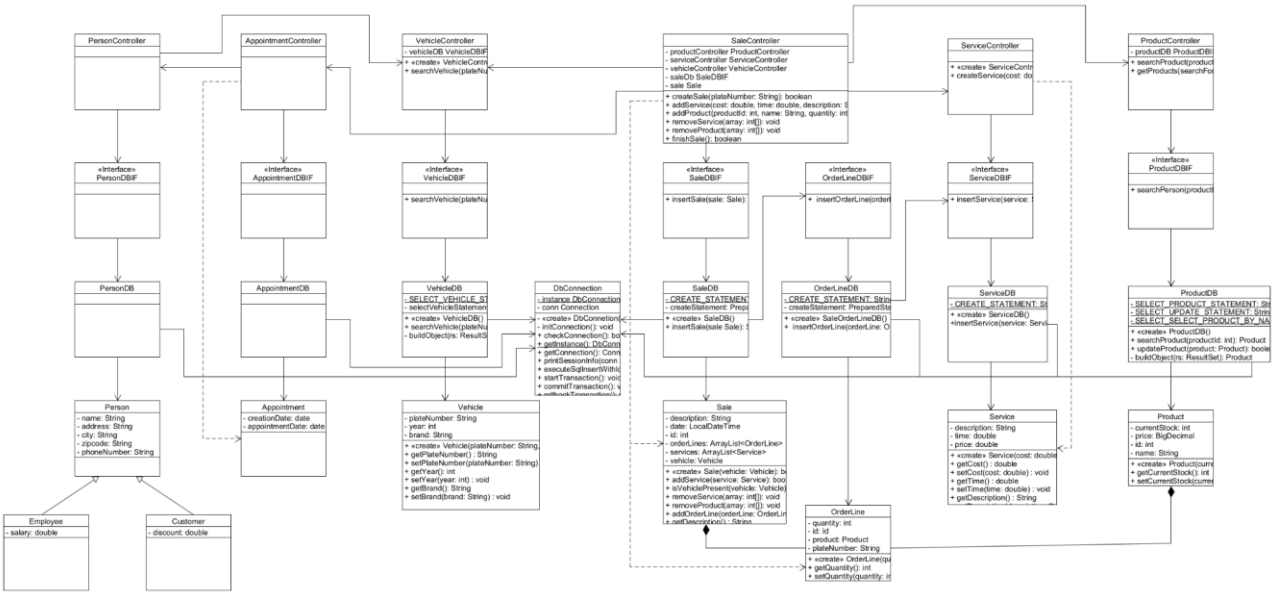
Design class diagram

A design class diagram is the diagrammatic illustration of the specifications of classes and interfaces in the application of a software program. It includes classes with attributes and associations typically that gives the outlook of how a program is supposed to operate with the interconnections. It can be usually the base for writing the code in a program or conversely can give an overview of different classes, attributes, and associations as well as methods within the program. [5]



Design class diagram 1 - Register sale

First we only added the classes with the fields and after we have created the communication diagram we added the methods to the design class diagram.

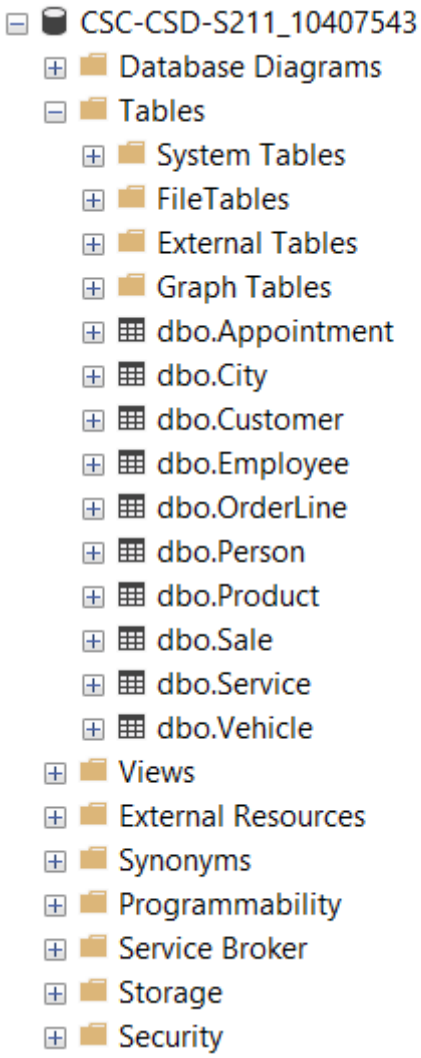


Design class diagram 2 - Register sale

Implementation

SQL Scripts

The image below shows an overview of how the database looked after the creation of the tables. We only used one database, from a person from the group, for testing our sql scripts. [1]



SQL database and tables

Every time something didn't work out, or when we simply inserted the data into the tables we had to drop them first and then recreate them from scratch. The code down below shows how we are first altering the table to get rid of the constraint and then, if the table exists, dropping it. This way we are ensuring that there are no errors while trying to drop the tables, because there is no foreign key connected between them anymore.

```
USE [CSC-CSD-S211_10407543]


ALTER TABLE dbo.Person
DROP CONSTRAINT if exists PersonCityFK;
ALTER TABLE dbo.Employee
DROP CONSTRAINT if exists EmployeePersonFK;
ALTER TABLE dbo.Customer
DROP CONSTRAINT if exists CustomerPersonFK;
ALTER TABLE dbo.Vehicle
DROP CONSTRAINT if exists VehicleCustomerFK;
ALTER TABLE dbo.Sale
DROP CONSTRAINT if exists SaleVehicleFK;
ALTER TABLE dbo.[Service]
DROP CONSTRAINT if exists ServiceSaleFK;
ALTER TABLE dbo.OrderLine
DROP CONSTRAINT if exists OrderLineSaleFK, OrderLineProductFK;
ALTER TABLE dbo.Appointment
DROP CONSTRAINT if exists AppointmentEmployeeFK;

GO

drop table if exists dbo.City;
drop table if exists dbo.Person;
drop table if exists dbo.Employee;
drop table if exists dbo.Customer;
drop table if exists dbo.Vehicle;
drop table if exists dbo.Sale;
drop table if exists dbo.[Service];
drop table if exists dbo.Product;
drop table if exists dbo.OrderLine;
drop table if exists dbo.Appointment;
```

SQL tables cleanup

We created the tables based on the relational model. As the image below shows, we tried to be consistent by using the same amount of columns with the same names and with the right foreign key according to the relational model that was designed.

```
CREATE TABLE dbo.City (
    zipcode VARCHAR(5) PRIMARY KEY NOT NULL,
    city VARCHAR(25) NOT NULL,
    )
GO


CREATE TABLE dbo.Person (
    [name] VARCHAR(25) NOT NULL,
    surname VARCHAR(25) NOT NULL,
    zipcode VARCHAR(5),
    [address] VARCHAR(50) NOT NULL,
    association VARCHAR(1) NOT NULL,
    phoneNumber VARCHAR(20) PRIMARY KEY,
    CONSTRAINT PersonCityFK
        FOREIGN KEY (zipcode) REFERENCES City(zipcode)
        ON DELETE SET NULL,
    )

GO


CREATE TABLE dbo.Employee (
   salary MONEY NOT NULL,
   phoneNo VARCHAR(20) PRIMARY KEY,
   CONSTRAINT EmployeePersonFK
        FOREIGN KEY (phoneNo) REFERENCES Person(phoneNumber)
        ON DELETE CASCADE,
    )

GO
```

SQL creation of tables

Code

When we want to declare a class, constructor, field or a method before it we need to choose what type of access modifier we want. This tells the language what can access it. There are 4 types of access modifiers in Java, public, private, protected and default. If it is private it can only be accessed within the class, and cannot be accessed by anything outside the class. If it is default then it can be accessed within the package. In case of protected access modifiers it can be accessed within the package and outside the package through child classes. The public access modifier can be accessed anywhere.

We created many classes and to create objects of a class we use the constructor which looks the same as a class or method declaration would look except without a data type. In a class, we can also declare fields which are variables inside a class. These are almost always private because we do not want anyone to be able to access them because we want to make these that can only be modified under our control. And when we want to change or access it we use setter and getter methods. When we want to call a method or constructor we can also pass parameters with them, we use these parameters to pass data into the method or constructor. The parameters are declared after the method in the parentheses.

In some cases, we want to have a list of objects, in this case, we use a collection, which is an object that groups multiple elements into a single unit. Examples of this are ArrayList, Array, LinkedList etc. If we want to perform some action on all elements of the list we need to iterate through it, for this we use loops. Java has several types of loops, such as while loop, for loop, and loops with iterators.

Exceptions

We used exceptions to define actions that will happen whenever something out of ordinary will happen in the system's flow. It lets us handle errors that we can later stack together if the process is more complex. Using exceptions we can define the error message and what will happen in the process and whenever we need it, so we reduce the use of multiple if/else statements.

```
try {
    while (!DbConnection.getInstance().checkConnection()) {
        time = times[thisTimeIndex];

        for (int i = 0; i < times[thisTimeIndex]; i++) {
            lblConnection.setText("Lost connection - reseting in " + time);
            time--;
            Thread.sleep(1000);
        }

        if (thisTimeIndex < (times.length - 1)) {
            thisTimeIndex++;
        }

        lblConnection.setText("Lost connection - establishing connection...");
    }
} catch (SQLException | InterruptedException e) {
    e.printStackTrace();
}
```

Exceptions

Transactions

Whenever we are inserting data into the database we are risking facing a serious issue that may cause errors with the database. Whenever there is a problem in the inserted information, the connection is lost or we need to cancel insertion in the middle of the process, the database might throw an error or data might be partly inputted. In order to avoid such issues we use transactions. By using them whenever we insert into, remove from or update multiple tables at once we reduce the risk of errors. Firstly, we start the transaction setting autoCommit to false, then whenever everything is prepared, we call commitTransaction() method that's commiting the changes and in case of catching an error, we can safely rollback the transaction cancelling all the changes.

Lambda and Stream

Sometimes, to simplify some actions that need to be performed we use lambdas. Lambda allows us to use a nameless method inside a different method's body. It requires us to input a parameter and then action that needs to be performed. In this example of action listener, we use lambda expression in creation of the new thread, that calls cancelAppointment() method.

```
JButton cancelButton = new JButton("Back");
cancelButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        Thread cancelSale = new Thread(() → {
            cancelAppointment();
        });

        cancelSale.start();
    }
});
```

Lambda

We used streams when we needed to loop through an Array and process the whole list. With streams we could filter the array, map it or have a count of any variable we needed to. We also find lambdas very useful while using the streams, because we could define the action of a lambda inside the stream whenever we needed it.

```
// Check if object is null, because in that case we cannot call the toString() on it

if (o ≠ null) {

    // Refills the list with the items which contains the written text

    ((DefaultComboBoxModel<Product>) box.getModel()).addAll(saleCtrl.getAllProducts()
            .stream().filter(p → p.getName().toLowerCase().contains(o.toString()
            .toLowerCase())).toList());
}

// Checks if you press enter, if you do than it selects item from list

if (e.getKeyCode() == KeyEvent.VK_ENTER) {

    // Checks if there is a product in the list, so it can select item from there

    if (saleCtrl.getAllProducts()
            .stream().filter(p → p.getName().toLowerCase().contains(o.toString()
            .toLowerCase())).count() > 0 ) {

        // Checks if the item in the editor is Product, if not it selects the first one
        // in the list which matches is the most

        if (!(box.getEditor().getItem() instanceof Product)) {
            box.getEditor().setItem(saleCtrl.getAllProducts()
                    .stream().filter(p → p.getName().toLowerCase().contains(o.toString()
                    .toLowerCase())).toList().get(0));
        }

        // Changes the id field automatically to the id of the selected product

        inputPanel.getFields()[0].setText(Integer.
                toString(((Product)box.getEditor().getItem()).getId()));
    }
```

Stream

Coding patterns

Singleton pattern

```
public class DbConnection {

    private static DbConnection instance;
```

Singleton 1

This pattern involves a single class which is responsible for creating an object while making sure that only a single object gets created. This class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. [6]

```
 * Constructor for the DbConnection
 */
private DbConnection() {
    initConnection();
```

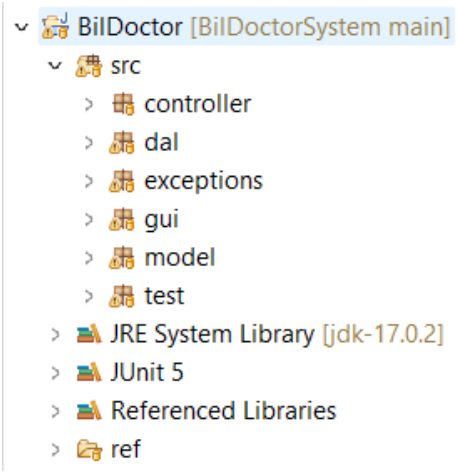
Singleton 2

```
 * @return the instance of DbConnection
 */
public static DbConnection getInstance() {
    if (instance == null) {
        instance = new DbConnection();
    }
    return instance;
}
```

Singleton 3

DAO(Data access object) pattern

This structural pattern helps the programmers to isolate the business layer from the persistence layer using an abstract API. In the picture below you can see our packages and how we separated our layers. We differentiate the controller layer, model layer, persistence layer (this was previously called container layer). We don't store business information anymore in the application, but in our separated SQL database. In the persistence layer the program handles the commands, which we want to run to execute on the SQL database. See below the architecture of the application. [7]



System's packages

Tests

Use case scenarios

After having the fully dressed use-case description we derive the use case scenarios from them and we put them in a table for easier visibility. This leaves us with 5 possible scenarios

Scenario 1	Sale registered	Basic flow	
Scenario 2	Vehicle not found	Alternate flow	A1
Scenario 3	Product not found	Alternate flow	A2
Scenario 4	Invalid quantity	Alternate flow	A3
Scenario 5	Employee removes product	Alternate flow	A4
Scenario 6	Employee cancels sale	Alternate flow	A5

Use case scenarios - Register sale

The basic flow of events:

1. Find the vehicle that the service was done for
2. Register the services to the sale
3. Find products and register them to the sale
4. Finish the sale and make the invoice

The alternate flows:

1. Vehicle not found
2. Product not found
3. Invalid quantity
4. Employee removes product
5. Employee cancels order

Test cases

Test cases containing valid/invalid values

After choosing the conditions to test for we put them in a table and indicated that in each scenario what result are we expecting. In the tables the V means valid, I means invalid and n/a means that it is not important for the test. The success scenario has valid information everywhere while the others have invalid somewhere. The table also includes the expected results of the scenarios.

Test case ID	Scenario	Vehicle	Product	Sale	Expected results
RC 1	Scenario 1 - Sale registered	V	V	V	The sale is created
RC 2	Scenario 2 - Vehicle not found	I	n/a	n/a	System asks for a different vehicle
RC 3	Scenario 3 - Product not found	V	I	n/a	System asks for a different product
RC 4	Scenario 4 - Invalid quantity	V	I	n/a	System asks for a different amount of the product

RC 5	Scenario 5 - Employee removes product	V	V	n/a	Product is removed from the sale
RC 6	Scenario 6 - Employee cancels sale	V	V	I	The sale is cancelled

Valid/Invalid test cases - Register sale

Test cases with real values

We chose some data values to test the scenarios.

Test case ID	Scenario	Vehicle	Product	Sale	Expected results
RC 1	Scenario 1 - Sale registered	AAA	1	registered	The sale is created
RC 2	Scenario 2 - Vehicle not found	AAB	n/a	n/a	System asks for a different vehicle
RC 3	Scenario 3 - Product not found	AAA	20	n/a	System asks for a different vehicle
RC 4	Scenario 4 - Invalid quantity	AAA	-1	n/a	System asks for a different amount of the product
RC 5	Scenario 5 - Employee removes product	AAA	1	n/a	Product is removed from the sale
RC 6	Scenario 6 - Employee cancels sale	AAA	1	cancelled	The sale is cancelled

Real values test cases - Register sale

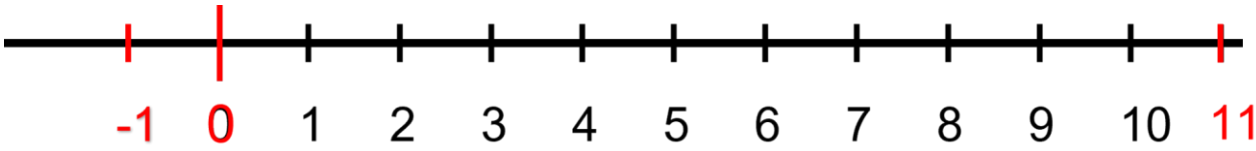
addProduct()

This method adds product and checks if there is enough of it in the stock. It covers the allowed quantity of the products and informs user of the available stock whenever it is too low. Meaning this method basically has to pay attention to the values of two things, the stock and the inserted quantity and based on these it should decide what it should return.

Equivalence Class Partitioning

This technique is used to divide the data into different intervals. We do this in a way that the data in the same partition behave the same way and should return the same value. This way it is enough for us to only test one value on each interval since all of them should return the same.

In the case of the stock, it cannot be negative so we only need to check if the stock is 0 or positive, so we only have 2 partitions.



Equivalence class partitioning - Register sale

In our case for the inserted quantity we looked at the range of numbers which we have to deal with and realised that we should for sure split it between positive and negative numbers, since every number in the negative interval should return an exception. In the positive range of the numbers it should add the product to the sale unless the inserted quantity is more than the current stock. This way we have split it into 3 intervals 0 and negative numbers, from 1 to the stock and numbers above the stock. After that we could use this to know where we should test.

Boundary Value Testing

After we have done the equivalence class partitioning we can use boundary value testing. This is used to test at the extreme ends of the interval, meaning that we can only test at the minimal end where the return value changes and one before and after that, since in those intervals behave the same then everything on that interval should also return the same value. It is usually more likely to have errors on the boundaries so by testing these values we maximise the chances of finding errors.

For the stock if it is 0 it should always return an exception and if it is another positive number then it should take into consideration the inserted value and decide according to that value.

In case of the inserted quantity one of the boundaries was at 0 and the other at the stock, which can be 0 or a positive number, but we only need the inserted quantity if it is positive, so we specified as 10 for the tests, where we want to test the positive stock. As such we needed to test at -1, 0 and 1 at the 0 boundary and 9, 10 and 11 at the 10 boundary.

In this table we summarise the different test cases we have to create and what is the expected return value. All in all we have 9 tests with the invalid quantity scenario.

Test case no.	Stock	Inserted quantity (subtracted)	Return
1.	10	-1	QuantityUnderrunException
2.	10	0	QuantityUnderrunException
3.	10	1	true
4.	10	9	true
5.	10	10	true
6.	10	11	false
7.	0	0	false
8.	0	1	false
9.	0	-1	false

Boundary value testing - Register sale

Unit tests

1. searchProductTest()

   In the “searchProductTest()” we are aiming to search for a product by its “id” by making use of the database layer class “ProductDB” to return the built object and compare it to the one that is stored in the database. By this we can make sure that the product was added correctly to the database.

2. searchVehicleTest()

   Similarly with the “searchProductTest()”, in the “searchVehicleTest()” we want to match the data taken from the building object method in the “VehicleDB” and compare it to the one in the database. By this we can make sure that the vehicle was added correctly to the database.

Integration tests

1. testRegisterSaleShouldReturnTrue()

In the testRegisterSaleShouldReturnTrue() we are checking if the whole process of registering a sale is correct and if the components work consistently and vacuously between each other.

Preparing database

Before the tests can run, we need to set up the database, so the tests can run with known variables. To do that we retrieve all the rows from a table that is being tested on. After retrieving them they are stored as a ResultSet and then we delete all of the retrieved rows from the table. Then we inject the database with testing data. After the testing process happens, we delete the testing rows from the table and retrieve all of the original rows that are stored in the ResultSet.

Advantages of using this approach are that the testing values are always the same.

Disadvantages are that tests run for longer, because of the retrieval and writing to the database. Also there occurs an SQL exception during the process, the original values are most likely not going to be retrieved.

Conclusion

In the first elaboration iteration we designed and implemented the registering of the sale. We did this by creating and using the necessary artefacts and coding the iteration accordingly. The iteration also includes the designing of the GUI, the plans for testing and the creation and implementation of the database that mirrors the relation model. The first iteration allowed us to create most of the classes that will be needed for the project and give us a starting point for the implementation of the use cases that will follow up.

Second iteration - Schedule appointment

Brief use case description

Schedule appointment - When a customer calls to schedule an appointment, the office employee opens the calendar and chooses a date that fits the customer. Then the date is registered and added to the system.

Fully dressed use case description

Use case name	**Schedule appointment**
Actors	**Office employee**
Pre-conditions	**Employee and Customer exists**
Post-conditions	**Appointment is registered and notification is sent**

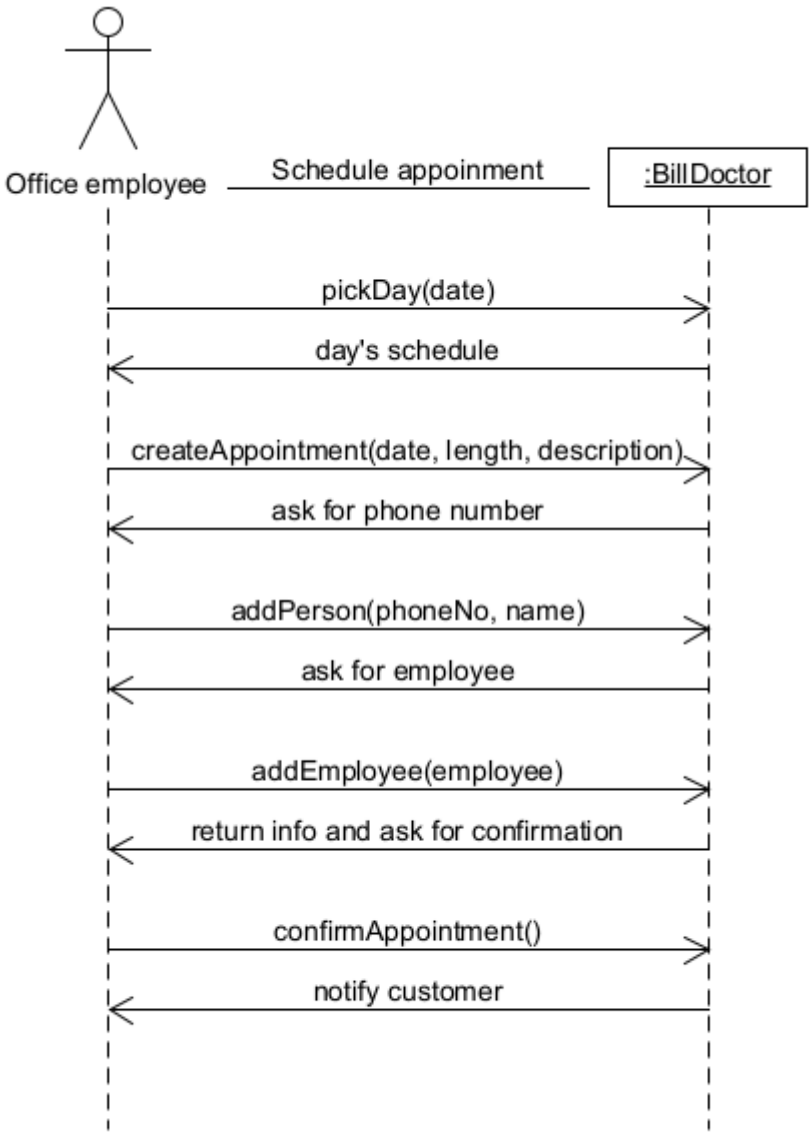
Frequency	**Every time customer wants to set an appointment**	
Main Success Scenario	**Actor (Action)**	**System (Response)**
	1. Chooses a specific date, length and description for the appointment	**2. Asks for the phone number and the name of the customer**
	3. Inputs the phone number and the name of the customer	**4. Asks for the employees that will attend for it**
	5. Selects the employee that will attend the appointment	**6. Returns all the information of the appointment**
	7. Confirms the appointment	**8. Finishes appointment and inputs it into database, later sends a notification**
Alternative flows	***1a. Length has an incorrect value*** ***System asks for a different length*** ***1b. Appointment overlaps with another*** ***System asks for a different day or length*** ***5a. Database is down and employees cannot be assigned*** ***Check the connection and try again*** ***5b. Incorrect employee assigned to the appointment*** ***System asks for another employee*** ***7a. Employee cancels the appointment*** ***System cancels the appointment***	

Fully dressed use case - Schedule appointment

Use case analysis

System sequence diagram

We constructed the SSD based on the fully dressed use case, accounting for the main success scenario. Therefore we have showcased the system operations and the system's responses accordingly and follow the flow structure, designing the methods and their respective attributes that are passed. In the diagram we are also using a loop box that goes through the list of employees until all the necessary ones have been selected to attend for the specific appointment.



SSD - Schedule appointment

Operation contracts

In the operation contracts we aim at giving a better understanding of the vital conditions for the system operations that are changing the state of the domain model. We chose to define all the above system operations as they are all influencing our system in a major way.

Operation	createAppointment(date, length, description)
Use case	Schedule Appointment
Precondition	Date is available and length is correct
Postcondition	Appointment is scheduled with date, length and description

Operation contract 1 - Schedule appointment

Operation	addCustomerInfo(phoneNo, name)
Use case	Schedule Appointment
Precondition	Object appointment exists
Postcondition	Customer's credentials are assigned to the appointment

Operation contract 2 - Schedule appointment

Operation	addEmployee(employee)
Use case	Schedule Appointment
Precondition	Object appointment exists and the employee is present in the database
Postcondition	Employee is assigned to the appointment

Operation contract 3 - Schedule appointment

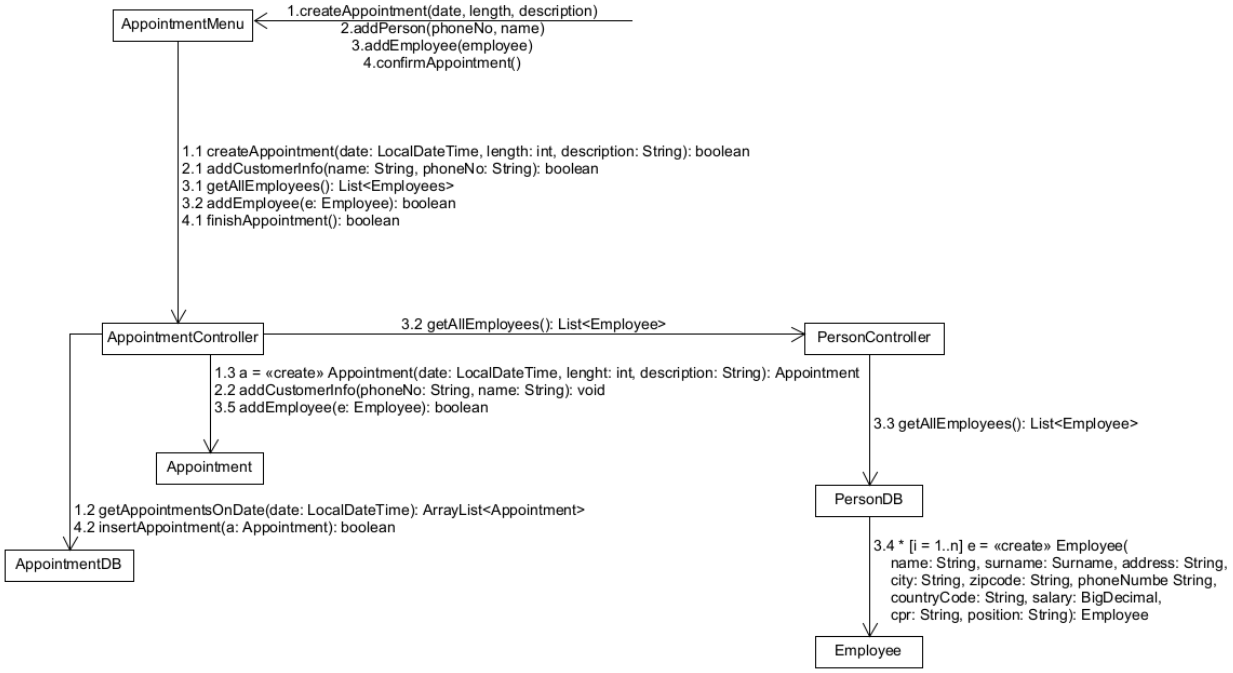
Operation	confirmAppointment()
Use case	Schedule Appointment
Precondition	Object appointment exists with at least one employee and with customer's credentials
Postcondition	Appointment is registered in the database and the notification is sent

Operation contract 4 - Schedule appointment

Design

Communication diagram

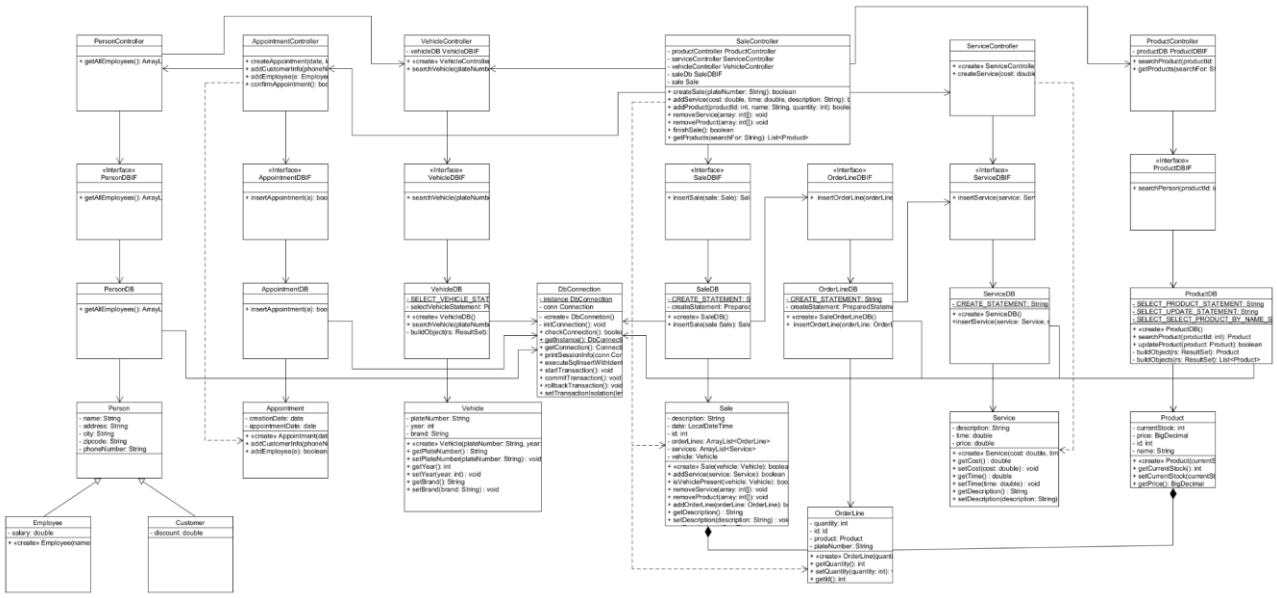
The communication diagram is a design (sequence) diagram created with the purpose of showing the relation between the system classes that are used in the specific use case. For the "Schedule Appointment" use case we chose to pass all the necessary information for initialising an appointment first, and then assign the customer's credentials alongside with the relevant employees for the job to it before finishing the appointment and adding it to the database. In the end, the customer is notified via phone number that their appointment has been registered successfully.



Communication diagram - Schedule appointment

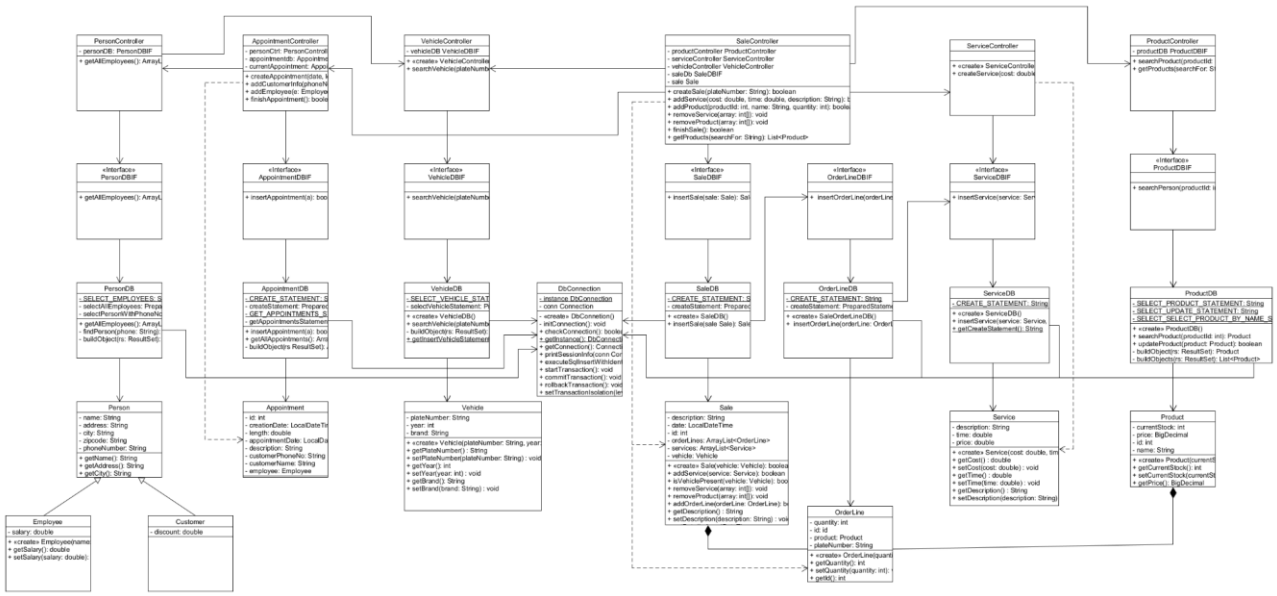
Design class diagram

The design class diagram from this iteration shows the progress done with the first iteration (all the classes and methods included in it) and in addition, it holds the classes, methods and attributes taken from the communication diagram and domain model that are specific for the second one.



Design class diagram 1 - Schedule appointment

This is the finished second iteration



Design class diagram 2 - Schedule appointment

Implementation

Code

The code in the second iteration uses the same patterns and is written following the same coding rules although there are minor changes:

To retrieve all employees from the database we used the Join clause to combine different rows from Person and Employee tables. It allows us to retrieve information necessary to build Employee objects, even when they are not in one table in the database.

```
public class PersonDB implements PersonDBIF {

    private static final String SELECT_EMPLOYEES = "SELECT * FROM Person "
            + " join City on Person.Zipcode = City.Zipcode "
            + " join Employee on Person.phoneNumber = Employee.phoneNo "
            + "and Person.countryCode = Employee.countryCode;";
```

Join

```
/**
 * Get all employees from the database
 * @return List of all employees
 */
@Override
public List<Employee> getAllEmployees() throws DatabaseAccessException {
    List<Employee> employees = new ArrayList<>();

    try {
        selectAllEmployees = DbConnection.getInstance().getConnection()
                .prepareStatement(SELECT_EMPLOYEES);
        ResultSet rs = selectAllEmployees.executeQuery();


        employees = buildObjects(rs);

    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

    return employees;
}
```

Return all employees method

In order to allow the user to be able to schedule an appointment we firstly needed him to choose a day the appointment was supposed to be held. To do this we used JDatePanel from JDatePicker which is a ready calendar we have found on GitHub . It allows us to choose a day and then retrieve the date whenever we continue with the process. ‘

After that we wanted to generate a list of hours that will state which hour of the specific day has an appointment already scheduled and which not. To do that we needed to use a custom CellListRenderer that will analyse all the appointments for the day, and set a different colour for the hours that are already “occupied”. getAppointmentsOnDate() is a method from the Appointment controller that returns a list of all appointments for the inserted day. Later the new CellListRenderer uses this array to generate the List that is later displayed in the UI.

```
@Override
public List<Appointment> getAppointmentsOnDate(LocalDateTime date) throws DatabaseAccessException {
    ArrayList<Appointment> appointments = new ArrayList<>();

    Appointment currentAppointment = null;

    try {
        getAppointmentsStatement = DbConnection.getInstance().getConnection().prepareStatement(GET_APPOINTMENTS_STATEMENT);

        getAppointmentsStatement.setTimestamp(1, Timestamp.valueOf(date));

        ResultSet rs = getAppointmentsStatement.executeQuery();

        while(rs.next()) {
            currentAppointment = buildObject(rs);
            appointments.add(currentAppointment);
        }

    } catch (SQLException e) {
        throw new DatabaseAccessException(DatabaseAccessException.CONNECTION_MESSAGE);
    }

    return appointments;
}
```

Get appointments method

```
public class HourListCellRenderer implements ListCellRenderer<Integer> {

    private List<Appointment> a;

    public HourListCellRenderer(List<Appointment> a) {
        this.a = a;
    }

    @Override
    public Component getListCellRendererComponent(JList<? extends Integer> list, Integer value, int index,
            boolean isSelected, boolean cellHasFocus) {
        DefaultListCellRenderer dlcr = new DefaultListCellRenderer();
        dlcr.getListCellRendererComponent(list, value, index, isSelected, cellHasFocus);
        for (Appointment element: a) {
            if (Integer.compare(element.getAppointmentDate().getHour(), value) == 0) {
                dlcr.setBackground(Color.RED);
            }
        }

        return dlcr;
    }
```

Hour list cell renderer

Method below is called in the UI class to fill the list with availability taken into consideration.

```
/**
 * Fill list of available hours
 * @param List of appointment
 */
public void fillList(List<Appointment> a){
    list.setCellRenderer(new HourListCellRenderer(a));

    DefaultListModel<Integer> dlm = new DefaultListModel<>();

    for (int i = 0; i < 24; i++) {
        dlm.addElement(i);
    }

    list.setModel(dlm);
}
```

Fill list method

Regular expression

A regular expression is a search pattern that can be specified by a list of characters. Using this we can make sure that the string we want to test matches the pattern. Regexes have different components that specify different characters, for example [] means exactly one character, ^ means negating, \d means that it is a number and we can specify how many characters we want by putting the number in {}. To make it we can use Java's Pattern class. We used regular expressions in case of the phone number, to make sure that it can have a + in the beginning and it can only have numbers or spaces afterwards. But since we did not want to limit our system to only Danish phone numbers we did not specify the number of characters.

Tests

Use case scenarios

We analysed the main flow of events and the alternative flows in order to create some use case scenarios. They look as it follows:

Scenario 1	Appointment scheduled	Basic flow	
Scenario 2	Incorrect length value	Alternate flow	A1
Scenario 3	Overlapping appointments	Alternate flow	A2
Scenario 4	Database not connected	Alternate flow	A3
Scenario 5	Incorrect employee	Alternate flow	A4
Scenario 6	Employee cancels appointment	Alternate flow	A5

Use case scenarios - Schedule appointment

The basic flow of events:

1. Input appointment details
2. Input customer's credentials
3. Select the employee that will attend the appointment
4. Confirm the appointment

The alternate flows:

1. Incorrect length value
2. Overlapping appointments
3. Database not connected
4. Incorrect employee
5. Employee cancels appointment

Test cases

Test cases containing valid/invalid values

After we made the use case scenarios we decided to create test cases for all the scenarios. As we explained in the previous iteration, we will first create the cases containing valid/invalid values and after that we will replace them with real values.

Test case ID	Scenario	Length	Employee	Appointment	Expected results
RC 1	Scenario 1 - Appointment scheduled	V	V	V	The appointment is scheduled
RC 2	Scenario 2 - Incorrect length value	I	n/a	n/a	System is asking for a different length
RC 3	Scenario 3 - Overlapping appointments	n/a	n/a	n/a	System is asking for a different length or day
RC 4	Scenario 4 - Database not connected	V	n/a	n/a	System is trying to reconnect to the database

RC 5	Scenario 5 - Incorrect employee	V	I	n/a	System is asking for a different employee
RC 6	Scenario 6 - Employee cancels appointment	V	V	n/a	The appointment is cancelled

Invalid/Valid test cases - Schedule appointment

Test cases with real values

Then, we wrote some data values to test the scenarios.

Test case ID	Scenario	Length	Employee	Appointment	Expected results
RC 1	Scenario 1 - Appointment scheduled	60	Mihai Mihut	registered	The appointment is scheduled
RC 2	Scenario 2 - Incorrect length value	-1	n/a	n/a	System is trying to reconnect to the database
RC 3	Scenario 3 - Overlapping appointments	n/a	n/a	n/a	System is asking for a different employee
RC 4	Scenario 4 - Database not connected	60	n/a	n/a	System is trying to reconnect to the database
RC 5	Scenario 5 - Incorrect employee	60	Joe Banana	n/a	System is asking for a different employee
RC 6	Scenario 6 - Employee cancels appointment	60	Mihai Mihut	n/a	The appointment is cancelled

Real values test cases - Schedule appointment

createAppointment()

This method in the AppointmentController checks if the appointment is not overlapping with an existing appointment and if not, then creates an appointment.

Equivalence Class Partitioning

The length cannot be lower or equal to zero.

Boundary Value Testing

For tests we insert an existing appointment to the database that starts at 13:00 and is 1 hour long. The invalid scenarios happen if the new appointment has at least one minute overlapping with an existing one.

Test ID	Appointment time	Length	Expected result
1	12:00	59	true
2	12:00	60	true
3	12:00	61	false
4	12:00	119	false
5	12:00	120	false
6	12:00	121	false
7	13:00	59	false
8	13:00	60	false
9	13:00	61	false
10	13:59	1	false
11	13:59	2	false
12	14:00	1	true
13	14:00	0	LengthUnderrunException
14	14:00	-1	LengthUnderrunException

Boundary value testing - Schedule appointment

Unit tests

1. testInsertAppointment()

   In this method we are testing the AppointmentDB class. This class has a method that should insert the provided appointment to the database and the testing method is checking the insertion by retrieving the values from the database.

2. testGetAllAppointments()

   This method is testing the AppointmentDB class. This class has a method to get all appointments and this testing method is checking if all appointments were retrieved correctly.

3. testGetAllEmployees()

   This method is testing the PersonDB class. This class has a method to get all employees and this testing method is checking if all of the employees were retrieved correctly.

Integration tests

1. testCreateAppointment()

   This testing method should test the connection between logic and database. It is called on the AppointmentController class.

Conclusion

In our second elaboration iteration we covered the scheduling of an appointment, which resembles an important part in the business's well-being, mainly because it is capable of generating revenue from structuring meetings with customers. Before designing and implementing this use case, we considered the same artefacts and structure as we did on our first iteration. We also had to construct classes like Appointment, Person and Employee from scratch in order to make this iteration fully functional. This iteration concludes our elaboration phase, as we are moving towards the less important use cases that will be covered in the construction phase.

Construction

In construction we are trying to complete the system so in the end of it we can have a system which can be operated in a beta customer environment and then be able to test the system in this way later. We take the foundation built in Elaboration and expand the system on it and finish coding the rest of the use cases.

First iteration - Register check-up

Brief use case description

Register checkup - The office employee will be reminded when a customer's checkup period is going to come close. Then, the employee can notify the customer and ask for permission to make an appointment for them.

Fully dressed use case description

<table>
<tr><td>Use case name</td><td colspan="2">Register check-up</td></tr>
<tr><td>Actors</td><td colspan="2">Office employee</td></tr>
<tr><td>Pre-conditions</td><td colspan="2">Vehicle and Customer exists</td></tr>
<tr><td>Post-conditions</td><td colspan="2">Appointment is registered and notification is sent</td></tr>
<tr><td>Frequency</td><td colspan="2">Every time the check-up period expires</td></tr>
<tr><td rowspan="4">Main Success Scenario</td><td>Actor (Action)</td><td>System (Response)</td></tr>
<tr><td>1. Office employee gets notified by the system and opens the check-up menu</td><td>2. Lists all the vehicles and their remaining period until a check-up is needed</td></tr>
<tr><td>3. Employee notices a car’s check-up period will expire in one week or less and calls the customer for permission to make an appointment for their vehicle</td><td></td></tr>
<tr><td>4. Selects the vehicle for the appointment and chooses a specific date and length</td><td>5. Asks for the employees that will attend for it</td></tr>
</table>

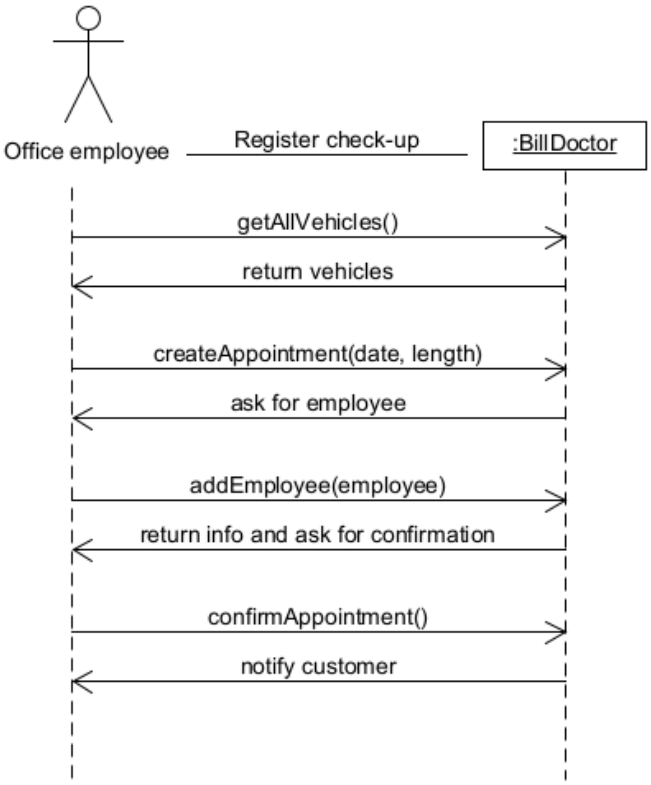
	6. Selects the employee that will attend the appointment	**7. Returns all the information of the appointment**
	8. Confirms the appointment	**9. Finishes appointment and inputs it into database, later sends a notification**
Alternative flows	***2a. Database is down and vehicles cannot be listed*** ***Check the connection and try again*** ***4a. Length has an incorrect value*** ***System asks for a different length*** ***4b. Appointment overlaps with another*** ***System asks for a different day or length*** ***5a. Database is down and employees cannot be assigned*** ***Check the connection and try again*** ***5b. Incorrect employee assigned to the appointment*** ***System asks for another employee*** ***7a. Employee cancels the appointment*** ***System cancels the appointment***	

Fully dressed use case - Register check-up

Use case analysis

System sequence diagram

In this system sequence diagram we are highlighting the system operations and the system responses of the Register check-up use case.



SSD - Register check-up

Operation contracts

The operation contracts listed below are created in order to analyse the system operations that are changing the state of the domain model.

Operation	createAppointment(date, length)
Use case	Register Check-up
Precondition	Date is available and length is correct
Postcondition	Appointment is scheduled with date and length

Operation contract 1 - Register check-up

Operation	addEmployee(employee)
Use case	Register Check-up
Precondition	Object appointment exists and the employee is

	present in the database
Postcondition	Employee is assigned to the appointment

Operation contract 2 - Register check-up

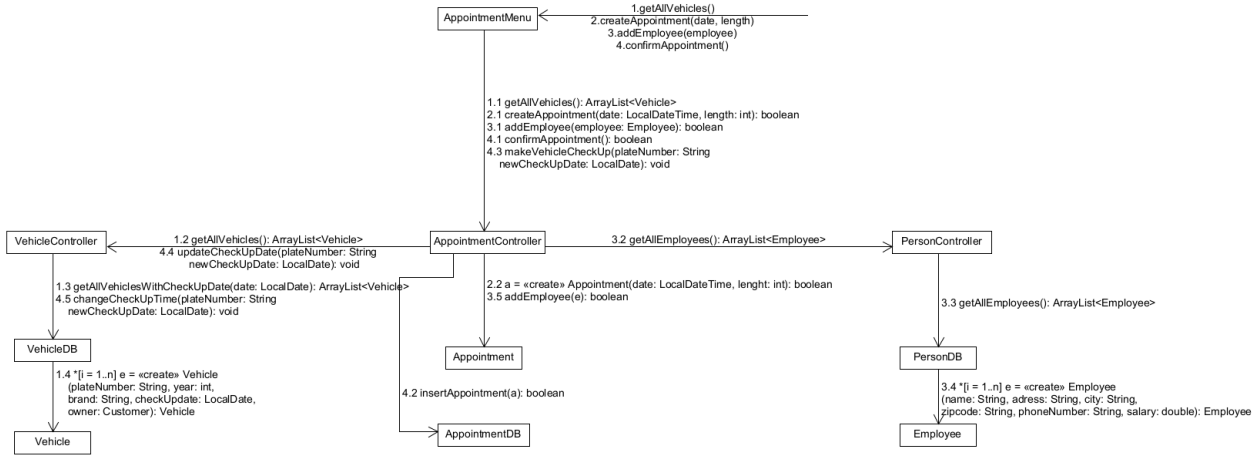
Operation	confirmAppointment()
Use case	Register Check-up
Precondition	Object appointment exists with at least one employee
Postcondition	Appointment is registered in the database and the notification is sent

Operation contract 3 - Register check-up

Design

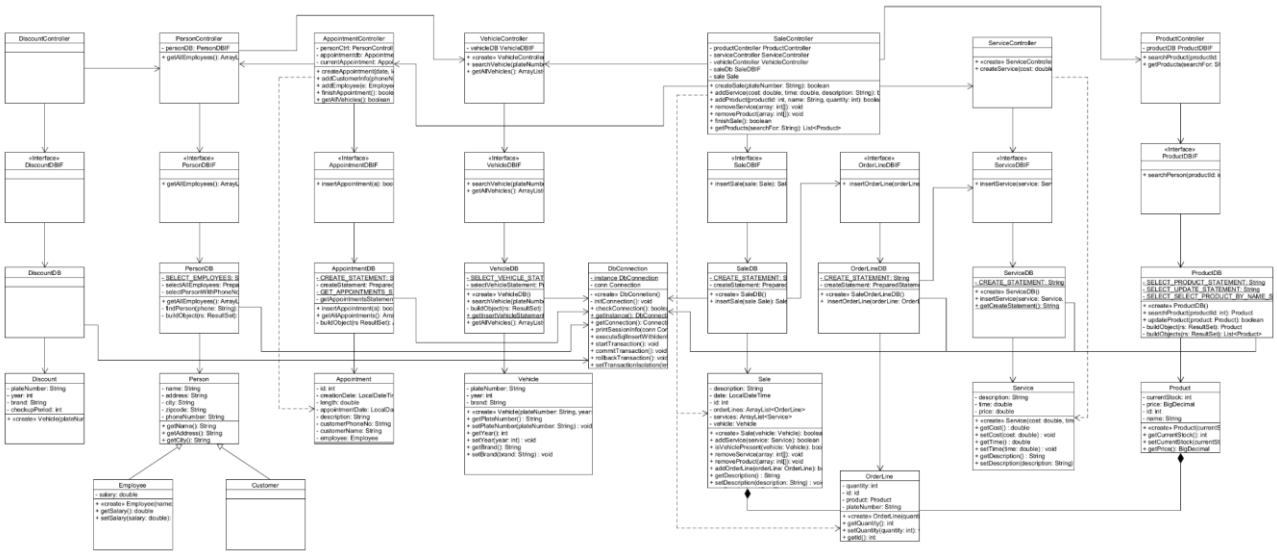
Communication diagram

The communication diagram of the third iteration is similar to the one from the second iteration with the exception that we are getting the car objects before we are creating the appointment and we got rid of passing the customer's information since the vehicle is already linked with a customer.



Communication diagram - Register check-up

Design class diagram



Design class diagram - Register check-up

Implementation

Code

The implementation of the third iteration hasn't changed too much from that of the second one, due to the main methods remaining the same. One of the additions of this use case was searching for all vehicles and displaying them, so you can check which vehicles need their check-up done. We have showcased below the methods that are supporting this feature.

```
private static final String GET_ALL_VEHICLES_STATEMENT = "SELECT * "
        + "FROM dbo.Vehicle "
        + "WHERE Vehicle.checkUpDate ⩽ ?";
```

Get all vehicles statement

```
public ArrayList<Vehicle> getAllVehiclesWithCheckUpDate(LocalDate date) {
    ArrayList<Vehicle> vehicles = new ArrayList<>();

    Vehicle currentVehicle = null;

    try {
        getAllVehiclesStatement = DbConnection.getInstance().getConnection()
                .prepareStatement(GET_ALL_VEHICLES_STATEMENT);

        getAllVehiclesStatement.setDate(1, Date.valueOf(date));

        ResultSet rs = getAllVehiclesStatement.executeQuery();
        while(rs.next()) {
            currentVehicle = buildObject(rs);
            vehicles.add(currentVehicle);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }

    return vehicles;
}
```

Return all vehicles method

In the GUI there is another tab in the main tabbed pane for the check ups. There is a number showing the number of available check ups next to the name. In the check up screen there is a list of vehicles that have a check up date within one week. The user can select a vehicle and register it for check up. When the user clicks the “Register” button, then the screen will be removed and appointment scheduling will be shown, but with values already inserted according to the vehicle. After creating the appointment, the next check up date will be moved by one month and the vehicle will be removed from the list of required check up vehicles.

Tests

Due to the third use case being similar with the second one, the test cases and methods will prove to be really similar. The whole testing analysis from the second iteration of the elaboration phase satisfies the functionalities of the Register check-up except for updating the check up column in the database and getting the vehicles with the check up date.

For that there are two unit tests:

1. updateCheckUpTest()

   Test VehicleDB’s changeCheckUpTime method. This method is supposed to change one column in the database which is checkUpDate.

2. getAllVehiclesTest()

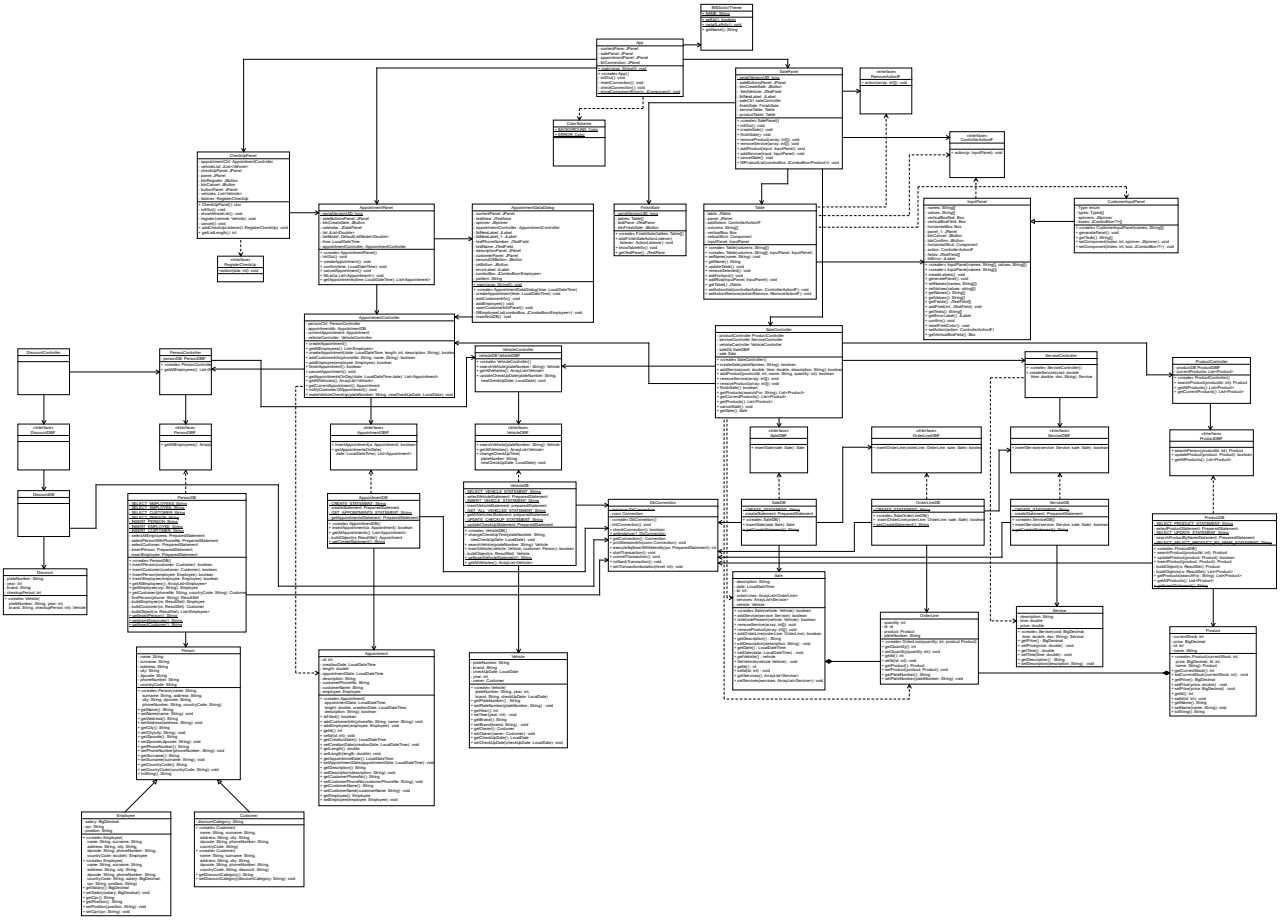
   There we are testing the method getAllVehiclesWithCheckUpDate in VehicleDB class. This method is supposed to return a list of vehicles that have a check up date sooner than the entered date.

Conclusion

The third use case is a part of the construction phase, resembling the first use case in this phase. Being a part of the construction phase means that the priority of the use case is not too high for it to be analysed in a complex manner. Our advantage was that the Register check-up use case has proven to be similar with the previous one, giving us an easier time in designing and implementing it successfully. The reason why we chose this use case to be our third one was mainly because the business importance of it is quite high, due to the potential revenue that it can generate in the long term, especially taking into consideration loyal customers.

Final design class diagram

The design class diagram is the most complex design diagram that is made in order to show all the details about the system by mapping out all the classes with their respective types, attributes, methods and links between them. The design class diagram also showcases the system's architecture, for us being a three layered architecture with an open design. Like in the domain model, in the design class diagram we are also showing generalisation patterns, aggregation relations and connections between the system's elements. As we said before, each class has its own attributes and methods described, by including the name of them, their types or return values, the private or public access and in case of the methods their attributes with the respective types. By providing so many details about them we can ensure a better traceability and an overall easier time for the programmes that are implementing the system.



Final Design Class Diagram

Reflect on the development

Coding standards

Our Code Standards are based mainly on the general conventions established by Oracle for writing Java Code. Working in a group can prove difficult when it comes to synchronising work and understanding what the other teammates have done. This is why we are using code conventions while implementing our system. These conventions mainly refer to aspects such as writing the names of the methods or variables by starting with a small letter and following with a capital letter for each other individual word in the name, positioning the curly brackets after the method header or simply using spaces between chunks of code for better readability. Other important elements are keeping the code clean and readable, giving meaningful and short names to methods or variables and writing Javadocs for each method, for an easier understanding of the functionality that it provides. All in all, coding standards are important to use while participating in a team-working project and we are making good use of them in order to be able to contribute properly to the coding activities and tasks.[8]

Coding and design principles

As we mentioned in the previous paragraph about the coding standards, we are trying our best to stick to some rules that are allowing us to form a working discipline and ultimately achieve our project goals. Other than the simple standards of naming, coding aspect and understandability we are coming across other issues that are considerably more important and that have to be respected for the sake of having a well designed and well thought project solution.

Here, we are referring to the principles of designing that will nonetheless end up affecting our code as well. When designing and implementing our system we are trying to keep in check the following principles: keeping a low coupling and high cohesion, getting rid of the code duplication and redundancies and ensuring a high code quality, all whilst fully covering the scope of the project. To explain in a bit more detail we will explain what these concepts mean and how we can fulfil them properly.

Coupling refers to the links that exist between different elements of the project. We can ensure a low coupling by reducing the details that are shared between two classes within the system and by making sure that our fields remain private. Cohesion refers to the number of logical tasks that a single entity is responsible for. A high cohesion can be ensured by assigning each unit with a single logical task. Getting rid of the code duplication and redundancies can prove to be difficult, but an efficient method to do this is refactoring. By refactoring, we are getting rid of duplicate code and keeping the same functionalities like before. A high quality when it comes to code can be achieved by frequent testing and a good design that covers useful patterns.

Concurrency issues

While working on our system we needed to take into consideration some concurrency problems which we solved differently. Concurrency is when we are executing code simultaneously, which might communicate with each other. We can use concurrency to solve many things including unresponsive UIs, long waiting times when running the program or simply data source (database) accessing issues. In our system we are solving concurrency issues by checking the availability of the database. We are doing this by using a method that checks the connection to the database every 5 seconds. The method starts when we are running the program and from that point on it is checking the connection regularly, eventually letting the user know in case the connection is lost. In order to make it possible for this method to run in the background, we are creating a new thread and calling the method in this thread.

While using concurrency we also need to be aware of some problems as well. The local variables in the method are unique to each thread, but the instance fields are the same on each thread. These parts are called shared resources and can be accessed by different threads at the same time. The part of the shared resources which can be modified is called the critical section. In case of the code in the critical section we need to make sure that we synchronise in which order the threads can access the data. Otherwise it can lead to conflicts and deadlocks. The synchronisation basically

blocks the access from the other thread while one is working on it, but when it finishes it notifies other threads that they can use it.

```
/**
 * Create the frame.
 */
public App() {
    // Start the connection checking on a new thread
    Thread connectionCheck = new Thread(() -> {
        resetConnection();
    });

    connectionCheck.start();

    // Initialise the GUI
    initGui();
}
```

Concurrency app

```java
/**
 * Checks the connection every 5 seconds
 *
 * If the connection is lost it will try to reset connection
 */
private void checkConnection() {
    boolean uninterupted = true;

    boolean con = true;

    // Check connection every 5 seconds
    while (uninterupted) {

        try {
            con = DbConnection.getInstance().checkConnection();
        } catch (SQLException e) {
            con = false;
        }

        if (!con) {
            uninterupted = false;
            resetConnection();
        }

        try {
            // Sleep for 5 seconds
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            uninterupted = false;
        }

    }
}
```

Concurrency check connection

```java
// After the connection is back
remove(lblConnection);
revalidate();
checkConnection();
```

Concurrency follow-up

Transactions

As we mentioned previously in the report we are using transactions in order to have a system that is more secure and that can keep integrity of the data that it holds. Whenever we are working with multiple tables into the database we are starting a transaction by setting the auto commit to false. After finishing the steps in the transaction we are committing it and then setting the auto commit to true.

```
/**
 * Sets auto commit to false and starts the transaction
 */
public void startTransaction() {
    try {
        getConnection().setAutoCommit(false);
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

/**
 * Commits transaction and sets the auto commit to true
 */
public void commitTransaction() {
    try {
        try {
            getConnection().commit();
        } catch (SQLException e) {
            throw e;
        } finally {
            getConnection().setAutoCommit(true);
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```

Transaction 1

In case the transaction is not completed we are aborting it by rolling back and setting the auto commit back to true, preparing for the next transaction to come. With the method setTransactionIsolation() we can also set the isolation level of a transaction.

```
    /**
     * Rollbacks transaction and sets the auto commit to true
     */
    public void rollbackTransaction() {
        try {
            try {
                getConnection().rollback();
            } catch (SQLException e) {
                throw e;
            } finally {
                getConnection().setAutoCommit(true);
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }

    }

    /**
     * Sets the isolation level of the transaction
     * @param level the level to set to
     */
    public void setTransactionIsolation(int level) {
        try {
            getConnection().setTransactionIsolation(level);
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Transaction 2

System architecture

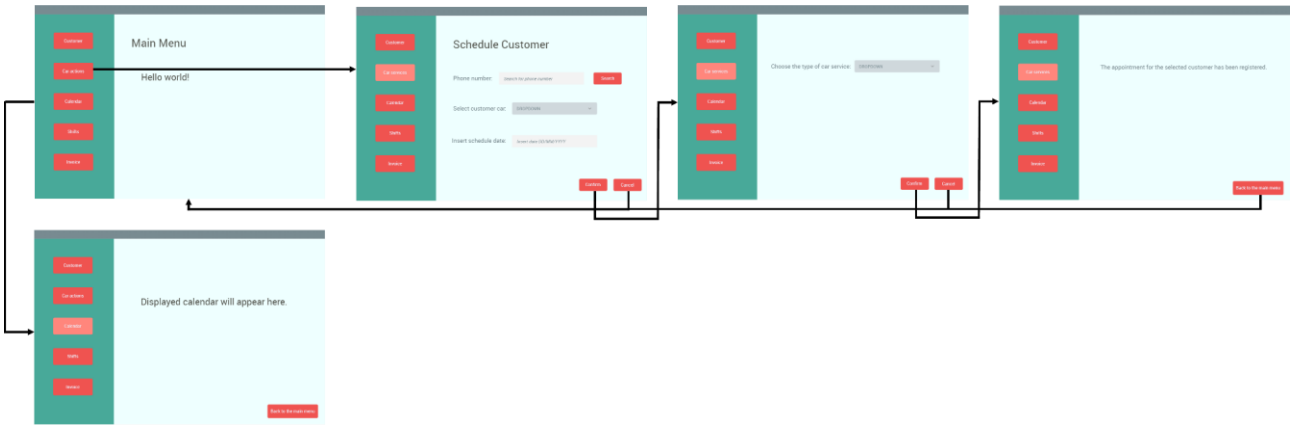
Our system is based on a three-layered architecture and is built on a database, as its data source. For integrating the usability of the database within our system we are splitting the model layer into two separate layers (the original model layer and the database layer), both being interconnected by the Data Access Object pattern. The retrieval of data from the database is assured by the database connection class that allows for a connection to be made and by the methods and interfaces provided by the IDE. The two other main layers of this architecture are the controller layer and the GUI (graphical user interface). All those three layers play a big role in the functionality of the system with the model layer being the datasource or the “base”, the controller which is passing methods and attributes from the UI to the model layer, acting as a bridge and the GUI which provides the interface necessary for the user to interact with the system. We also chose to work with an open

architecture instead of a closed one, the reason being the increased coding flexibility and an easier design process.

UI Design

Navigation diagram

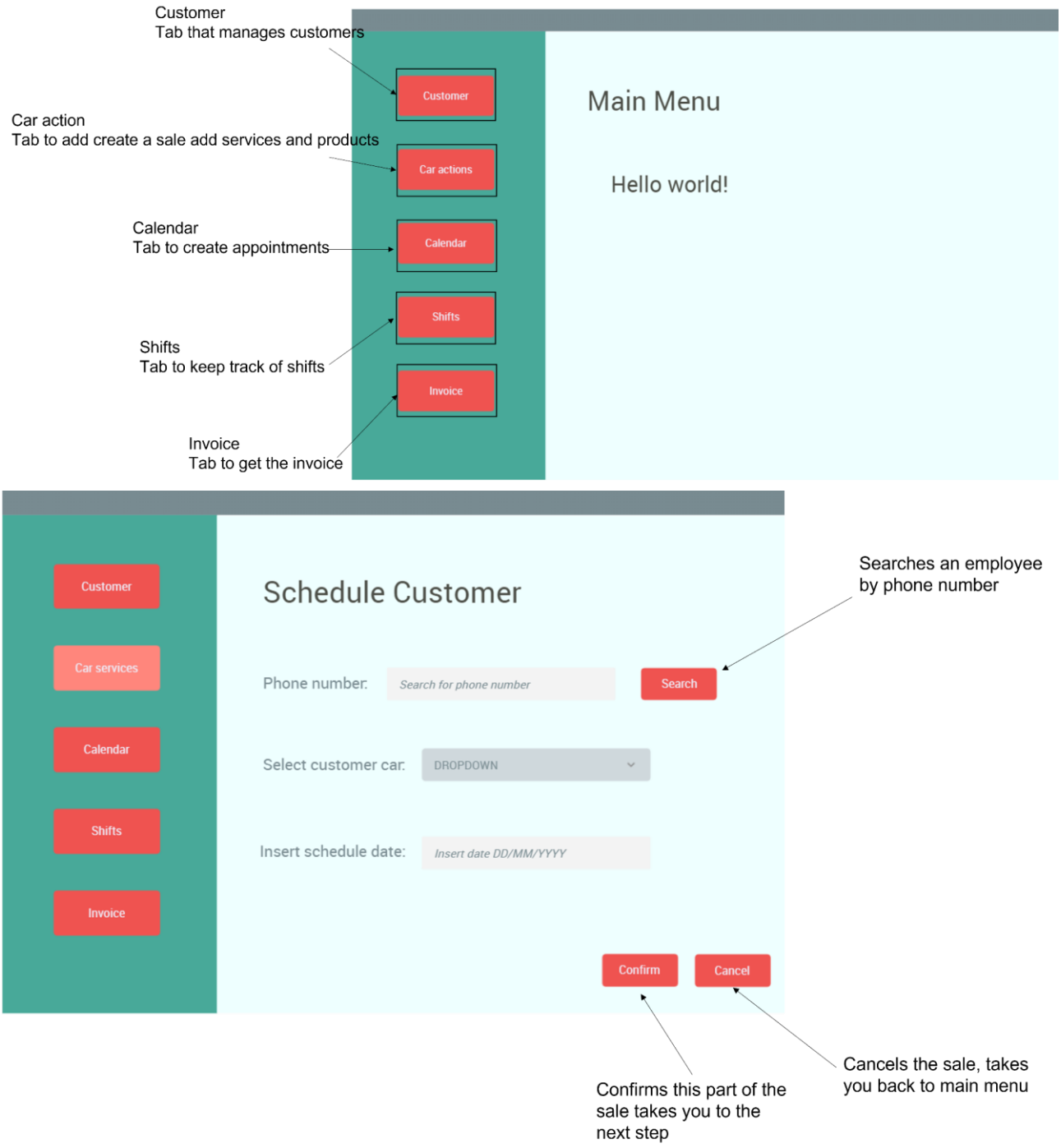
Before creating the GUI we made a navigation diagram, which shows us an overview of how we can navigate from one window to the other. In order to create this, we used our previously created mock-ups and the buttons there. We also created a prototype which can be found in appendix 7.



UI Design 1

Window Diagram

We made some window diagrams as well, these are used to describe what everything does that we can see in a window. This just gives us a brief description and can be useful when coding the UI to keep in mind what everything does.



UI Design 2

In the end we ended up refactoring how the UI looks, but the bases were still the same so we could use these diagrams.

IT security

In developing our system IT security doesn't represent a priority, but security elements exist in a few places. The fact that the database can only be accessed with the user's credentials or that the repository of work is private already constitutes a few security measures, but they are really general

in this sense. Even though we are not taking security into consideration we can expand a bit on a few ways that we could integrate this concept into our project.

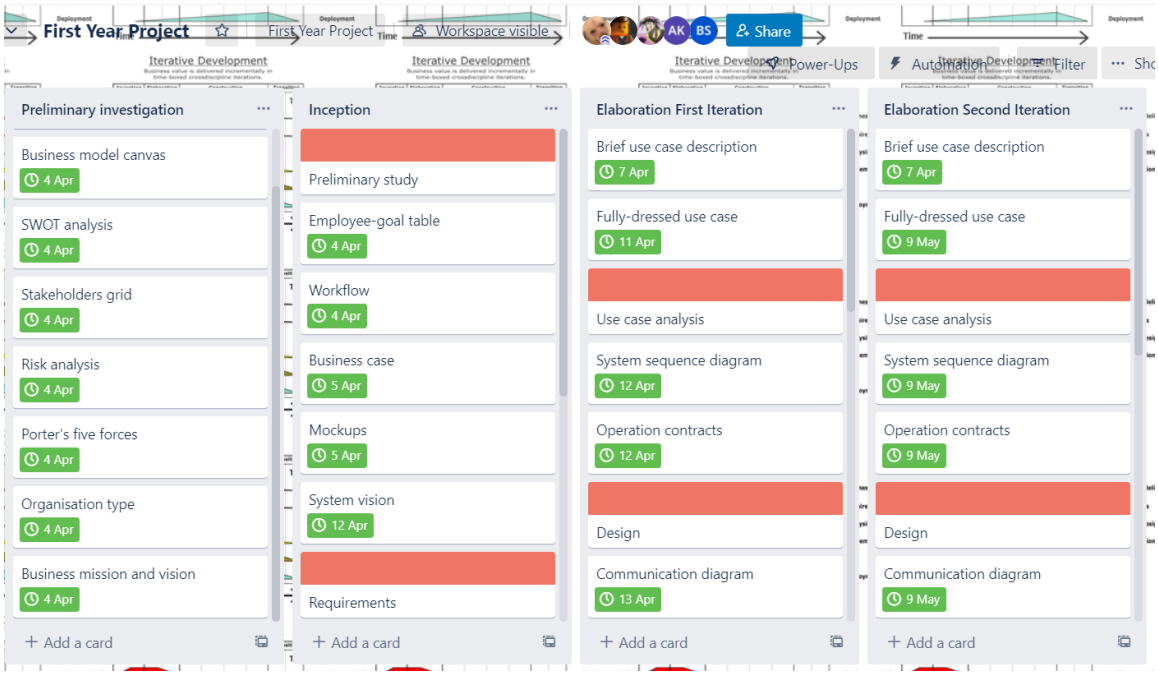
First of all, in order to settle on some security solutions, we have to conduct an IT risk assessment that will give us all the variables in finding the right solution for securing our software product. After that we can think about how to enhance our IT infrastructure in order to save the company from risks that could cripple the business' economy. These risks can be of multiple types, such as informational leaks, vulnerable systems that can be hacked easily, susceptibility to data loss and a slow and buggy software product due to insufficient hardware resources and so on. All the issues can be tackled, but it is essential to first conduct the previously mentioned assessment in order to see if the efforts put into the issue are worth it for the outcomes and if the whole idea is feasible overall. After everything was done right, it is the programmers' responsibility to implement the system in a way that can solve the issues that were found in the study phase.

Tools used

In order to achieve our goals faster and to be able to work in a group project environment, we had to use some tools that made our lives easier.

Trello

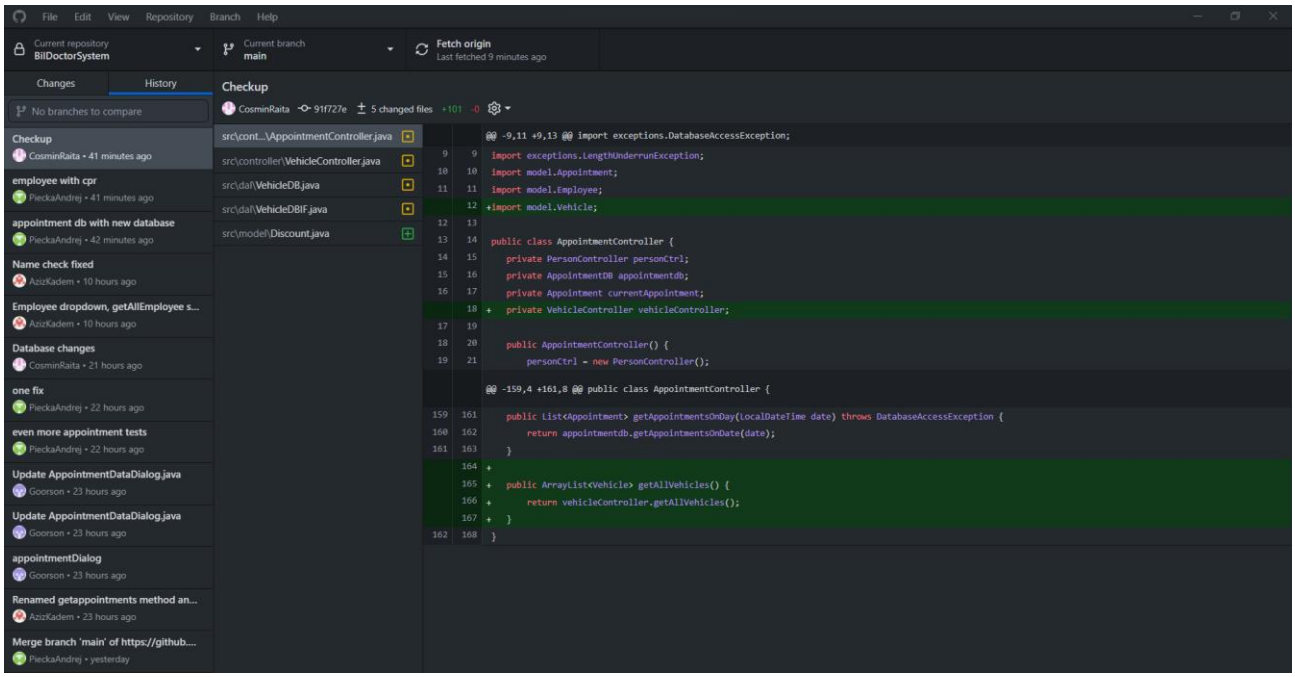
We used trello as a project management tool in order to keep track of all the steps and the tasks of the process. Within this tool we can also assign certain tasks to members of the group or timestamp certain tasks with the time that they have to be finished until or the time that they were finished at.



Trello

GitHub

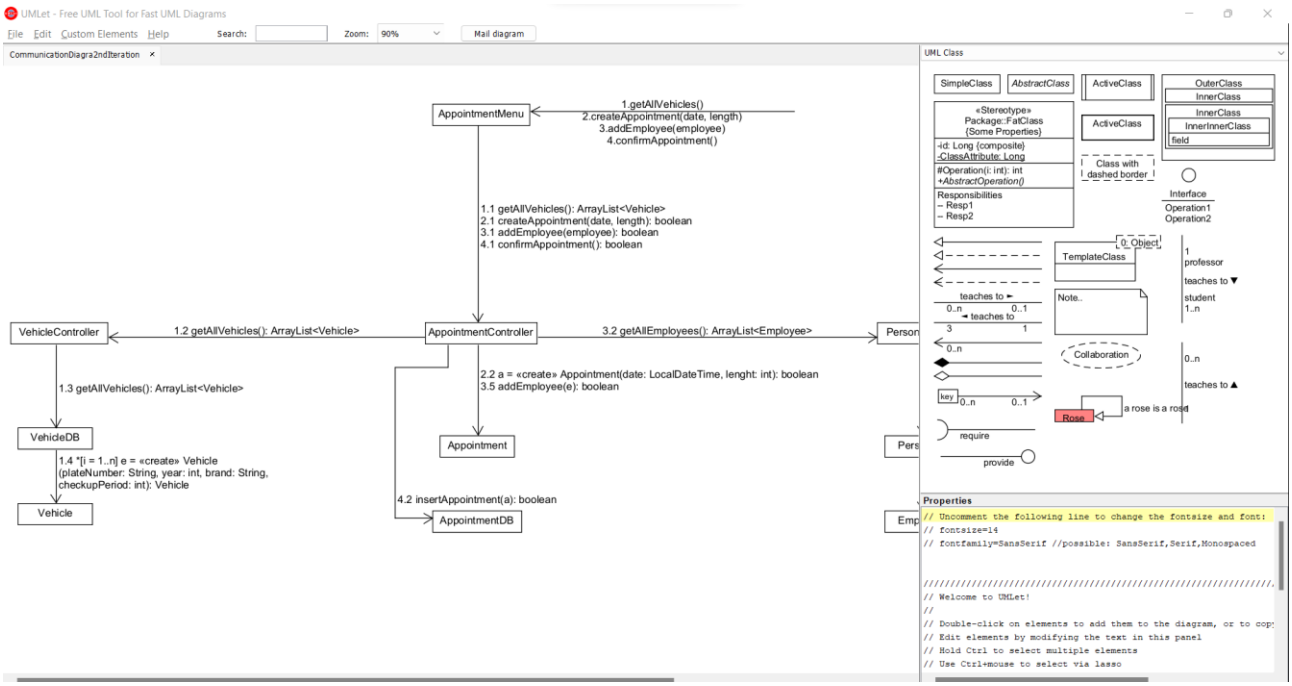
For sharing the same Java document and SQL scripts we are using GitHub. It is an efficient tool because it allows the users to have a shared repository where they can push and pull changes that they are making in the code. By this we are ensuring that all the members of the team get to code a part of the system. The platform also provides us with alerts in case changes are conflicting and it is giving us a useful history of changes, with each push containing information about the classes that were modified and the dates when the pushes were made.



GitHub

UMLet

For creating our diagrams and for designing the project in general we are using the UMLet tool. This tool provides us with all the necessary elements that we need in order to be able to showcase our design logic and to map out the connections between multiple elements within our system.



UMLet

Conclusion

Assessment

We declare that with good conscience that as a group we achieved what we wanted. If this project would have been a real life scenario, then we can state that we reached the pinned milestones in time and we were able to deliver what we promised to the client. The overall goal was to finish everything until the first use case of the construction phase until the hand-in deadline. The software has a functional user-friendly GUI. The client can resolve the most important tasks from his perspective in the software. He can register sales, make appointments and check if he should call the customer for a regular check-up. The delivered product is reliable, has eligible quality and generates business value for the company.

Perspective

Although the team progressed at a healthy pace with the project, there are still a pair of use cases left. To finish them it takes approximately 2 weeks of working days. Other than that we could implement additional features, which are out of scope right now, like connect the system to the network of wholesalers, or create a connection between the website and the system, making the time reservation more automative.
We also want to include the next step of reaching out to more customers, which is to devise a plan for the primary research. We already did the secondary research and named our sources, but the

next step would involve deep analysis of the existing customers of the company. We would ask their agreement for a more in-depth cooperation to increase customer retention.

Reflecting on the process

This project summarised everything that we learned in the last two semesters. Learned how to approach a problem and evaluate the situation, learned how to plan a solution from scratch, learned how to execute the plan and nonetheless stick to the plan during the whole time, how to adjust the plan if something doesn't seem right, learned about software architecture, and built a database. We believe we learned how to utilise the knowledge and lived up to the expectations of a client.

Working with UP

We built up the whole process based on the UP model. We found it as a good guide for the processes, because each iteration works like a milestone and it seemed logical to follow it and respect the iterations. It demands from the user to be thorough and if you differ from the diagrams you go back to them and modify them accordingly, but it is a good way of being transparent in the work.

Group evaluation

We believe that from the report the reader can conclude that we spent a serious amount of time on the project. We never skipped one day and when the class was cancelled we worked on the project instead. We discussed the difficult processes together and tried to figure out how to proceed further. We did not want to go with the easy way, but always examine the problem from a business perspective. We helped each other and let everybody work on the tasks, where he wanted to improve. If we should have to name our group culture, we could call it adhocracy, because we adjust fast to the situation and switch to a problem solving mode.

References

[1] A. Górecki *et al.*, “Mini-Project Persistence.” [Online]. Available: Mini_Project_Persistence/Mini_Project_Persistence.pdf at main · PieckaAndrej/Mini_Project_Persistence · GitHub

[2] “Porter’s Five Forces Analysis Tutorial.” https://www.visual-paradigm.com/tutorials/five-forces-analysis-tutorial/ (accessed May 25, 2022).

[3] *Barnabás, S. (2022) 'Vestbjerg'. University College of Northern Denmark. Unpublished essay.*

[4] “The Boyce-Codd Normal Form (BCNF) | Vertabelo Database Modeler.” https://vertabelo.com/blog/boyce-codd-normal-form-bcnf/ (accessed May 25, 2022).

[5] C. Larman, *Applying UML and patterns : an introduction to object-oriented analysis and design and iterative development*, 3. ed. Upper Saddle River, N.J.: Prentice Hall Professional Technical Reference, 2005.

[6] “Design Pattern - Singleton Pattern.” https://www.tutorialspoint.com/design_pattern/singleton_pattern.htm (accessed May 25, 2022).

[7] “The DAO Pattern in Java | Baeldung.” https://www.baeldung.com/java-dao-pattern (accessed May 25, 2022).

[8] “Java Code Conventions,” 1997.

[9] "• Denmark: Passenger car stock 1990-2018 | Statista." https://www.statista.com/statistics/452288/denmark-number-of-registered-passenger-cars/ (accessed May 25, 2022).

[10] "Danes and Cars: Why Real Men Drive Bicycles." https://www.howtoliveindenmark.com/stories-about-life-in-denmark/danes-and-cars/ (accessed May 25, 2022).

[11] "Means of transport - Statistics Denmark." https://www.dst.dk/en/Statistik/emner/transport/transportmidler (accessed May 25, 2022).

[12] "The average Dane - Statistics Denmark." https://www.dst.dk/en/Statistik/nyheder-analyser-publ/Publikationer/gennemsnitsdanskeren (accessed May 25, 2022).

[13] "Social Media Stats Denmark | Statcounter Global Stats." https://gs.statcounter.com/social-media-stats/all/denmark (accessed May 25, 2022).

Appendices

Appendix 1

First interview questions

Questions for the interview

Background/general questions

- What is the company that we should make a system for?
- What is the current management team?
- Can you describe the company's stakeholders (managers*, employees, customers, collaborators)?
- What are the services that you provide? Are there any products that you are selling?
- What do you consider as being a strength to your company?
- What do you consider as being a weakness to your company?
- Do you have in mind any possible opportunities?
- What are the imminent threats in the long term/short term?

General interview questions

Workflow questions

- Can you describe what kind of systems you use currently? What about the one you have use previously?
- Do you currently use any governmental system? (optional)
- What are the tasks that an employee has to do? And that we can automatize?
- Are there any tasks that are specific only to managers or employees?
- What would you describe as being the hardest task right now? (optional)
- Are the customers satisfied? What do they expect from a system/ from your company in general?
- Extra: What is the hardware that the company's systems are running on?
- What do you think would be the most important use of the system and could you describe it, how do you use it(workflow)?

Can they choose those categories in the current system as well?

Workflow interview questions

Requirements questions

Present in a few lines how we see this project:

Idk it will turn out ok

- What are your expectations about the system that we are about to make?
- What features would you want us to include?
- Any open projects that you got?

When we have the general idea we'll reach back to him

Does he know that we will probably not do anything usable?

Ideas for later

- Customer feedback system
- Periodic car check-ups (changes of oil etc.)
- User interface with login potential and reservations for car repairs

Requirements interview questions

- Prices of extra spare parts or components of the car can be added to the total cost of the reparation
- Customer can see his reparation progress and track potential delays
- Subscription-based repairs and services (maybe discounts)

Interview ideas

Appendix 2

UP phase plan

UP phase	Planned activities	Dedicated time
Preliminary investigation	Business analysis: ● Business model canvas ● Organisation type ● Stakeholder's grid ● SWOT analysis ● Risk analysis	01.03.2022 - inception
Inception	Preliminary study: ● Business case ● Mockups ● System vision ● Use case diagram ● Brief use case descriptions ● Domain model	5 days
Elaboration (first iteration)	Fully-dressed use case	2 days
	Use case analysis: ● System sequence diagram ● Operation contracts	
	Design: ● Communication diagram ● Design class diagram	
	Implementation: ● Code implementation ● Coding patterns ● Tests ● Use case scenarios ● Test cases ● Equivalence class partitioning ● Boundary value testing ● Unit tests ● Integration testing	3 days
Elaboration (second	Fully-dressed use case	2 day

iteration)	Use case analysis: ● System sequence diagram ● Operation contracts	
	Design: ● Communication diagram ● Design class diagram extension	
	Implementation: ● Code implementation ● Coding patterns ● Tests ● Use case scenarios ● Test cases ● Equivalence class partitioning ● Boundary value testing ● Unit tests ● Integration testing	3 days
Construction	Fully-dressed use case	2 day
	Use case analysis: ● System sequence diagram ● Operation contracts	
	Design: ● Communication diagram ● Design class diagram extension	
	Implementation: ● Code implementation ● Coding patterns ● Tests ● Use case scenarios ● Test cases ● Equivalence class partitioning ● Boundary value testing ● Unit tests ● Integration testing	3 days

UP phase plan

Appendix 3

Team contract

Team Contract

Team Name: CSC-CSD-S211 Group 1 Date: 08-02-2022

GOALS: What are our team goals for this project? What do we want to accomplish? What skills do we want to develop or refine?
Get familiar with the design, implementation and testing of a system. Get comfortable with real life working scenarios, groupwork and new tools (f.e.: git, trello). Be involved in a collaboration with another company.
EXPECTATIONS: What do we expect of one another in regard to attendance at meetings, participation, frequency of communication, the quality of work, etc.?
Actively participate in scheduling of the project work and be present at the team meetings. Follow the schedule and work with the part each member is tasked with. In case of special events members should announce through means of remote communication such as Teams.
POLICIES & PROCEDURES: What rules can we agree on to help us meet our goals and expectations?
Help each other when we encounter difficulties. Notice in case a task is too hard to handle. Value the group work and the other members. Work steady, don't panic.
CONSEQUENCES: How will we address non-performance in regard to these goals, expectations, policies and procedures?
A warning is first given in case a member under-performs. If the mistakes are repeated the member will be replaced or kicked out the group.

We share these goals and expectations, and agree to these policies, procedures, and consequences.

Aziz Kadem - 10407533@ucn.dk

Cosmin Raita - 10407643@ucn.dk

Barnabás Selymes - 10407536@ucn.dk

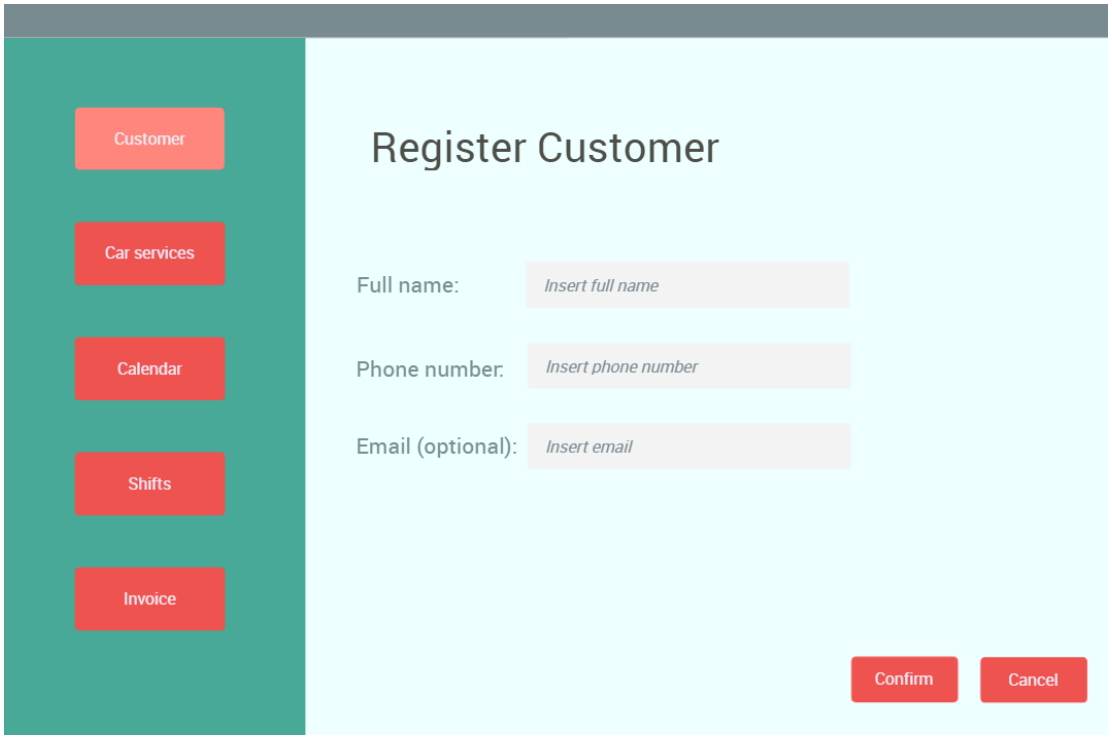
Andrej Piecka - 10407548@ucn.dk

Adam Górecki - 10407543@ucn.dk

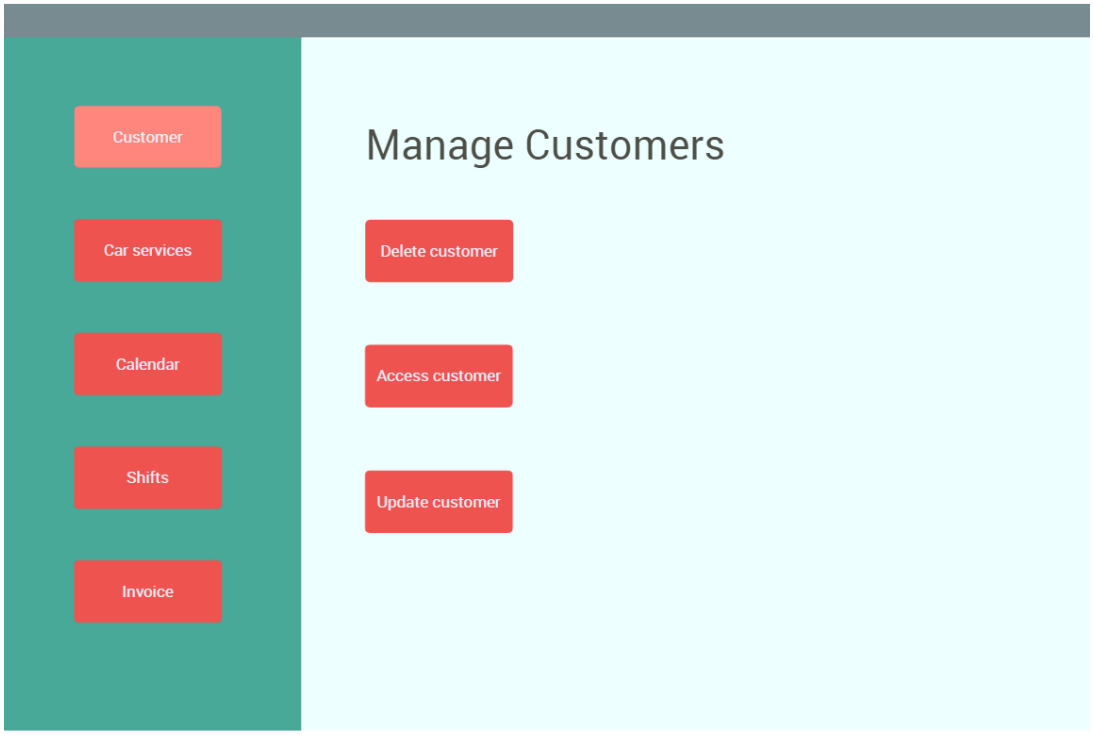
Team contract

Appendix 4

Mockups



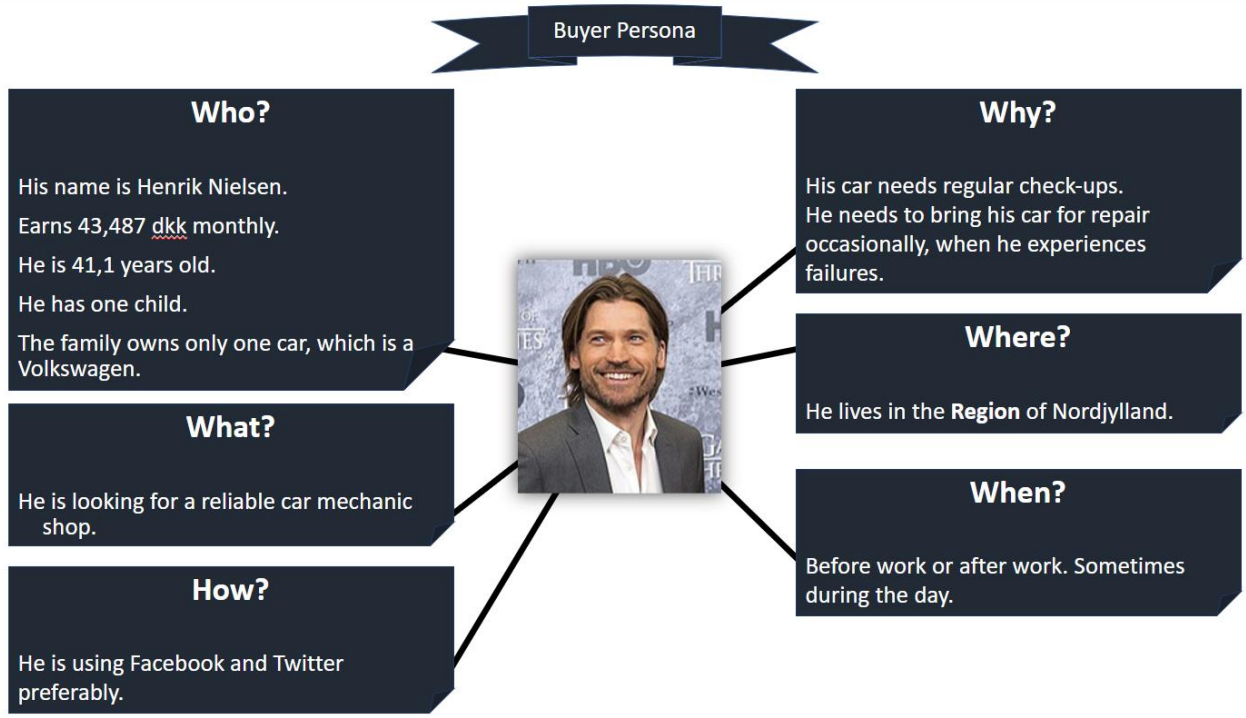
Mockup - Register customer



Mockup - Manage customers

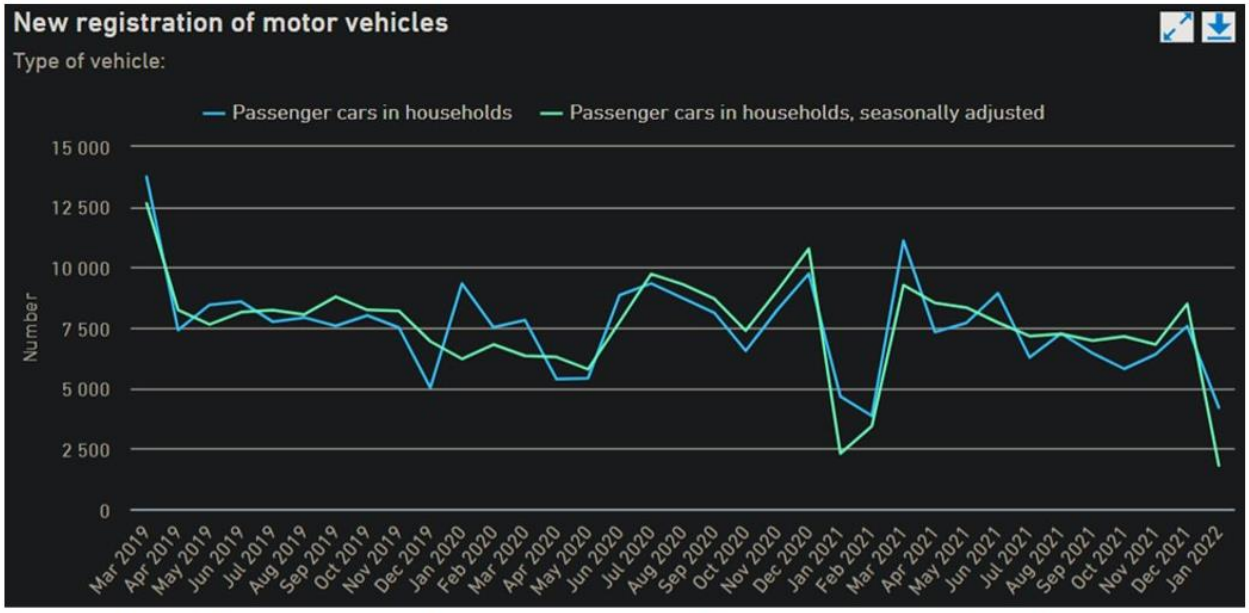
Appendix 5

Target Audience research



[9][10]

Buyer persona



The number of new registrations decreased in the recent years, therefore we expect that the demand for repairs will increase, because the average age of a car is older, which increase the possibility of failures and a need for car mechanic.

[11]

New registrations diagram

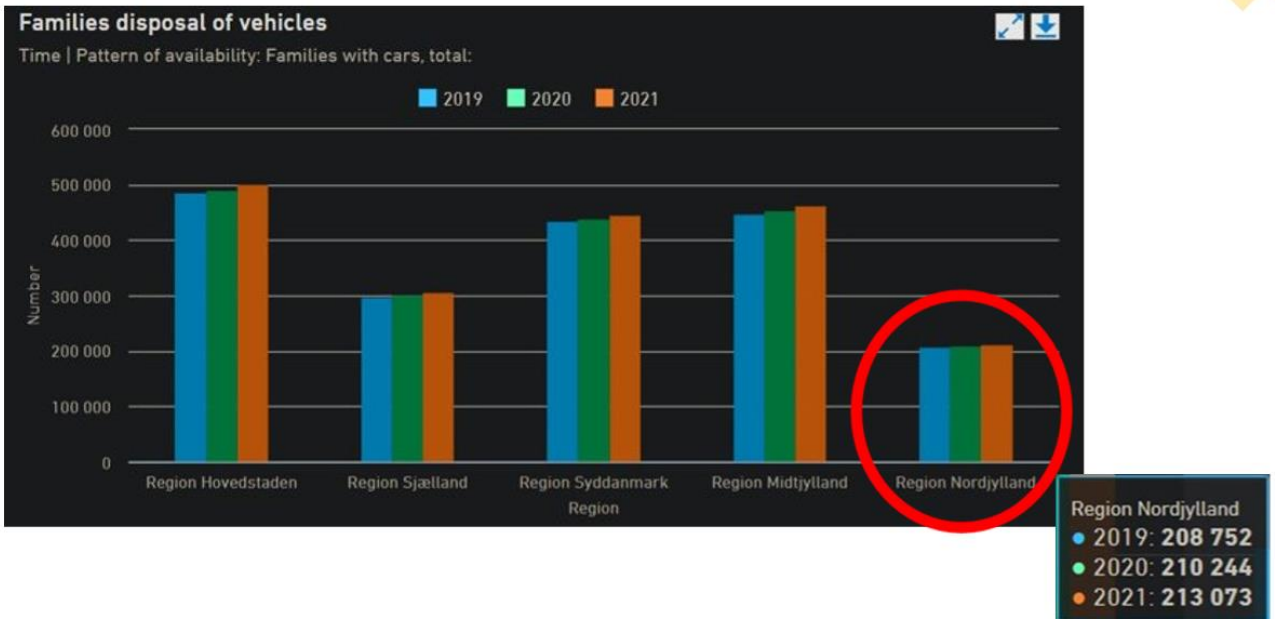
Families disposal of vehicles by region, pattern of availability and time	2021
Region Nordjylland	
Families, total	312 499
Families without cars, total	99 426
Families with cars, total	213 073
Families with 1 car, total	149 348
Families with 1 private car	144 701
Families with 1 company car	1 836
Families with 1 van	2 811
Families with 2 cars, total	54 526
Families with 2 private cars	47 605
Families with 2 company cars	104
Families with 2 vans	57
Families with 1 private car and 1 company car	3 590
Families with 1 private car and 1 van	3 093
Families with 1 company car and 1 van	77
Families with 3 cars, total	7 557
Families with more than 3 cars	1 642

Family possession of cars

A family is in possession of a car, when one or more family members own one or more passenger cars or vans for private goods transport or has a company car at their disposal.

However, in several municipalities surrounding the major towns, almost eight out ten families own a car.

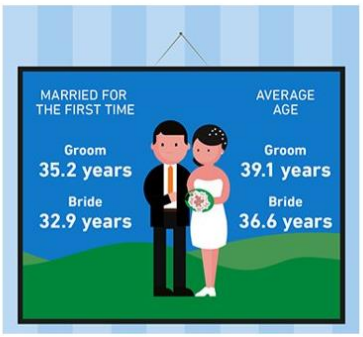
Vehicles per family



[11]

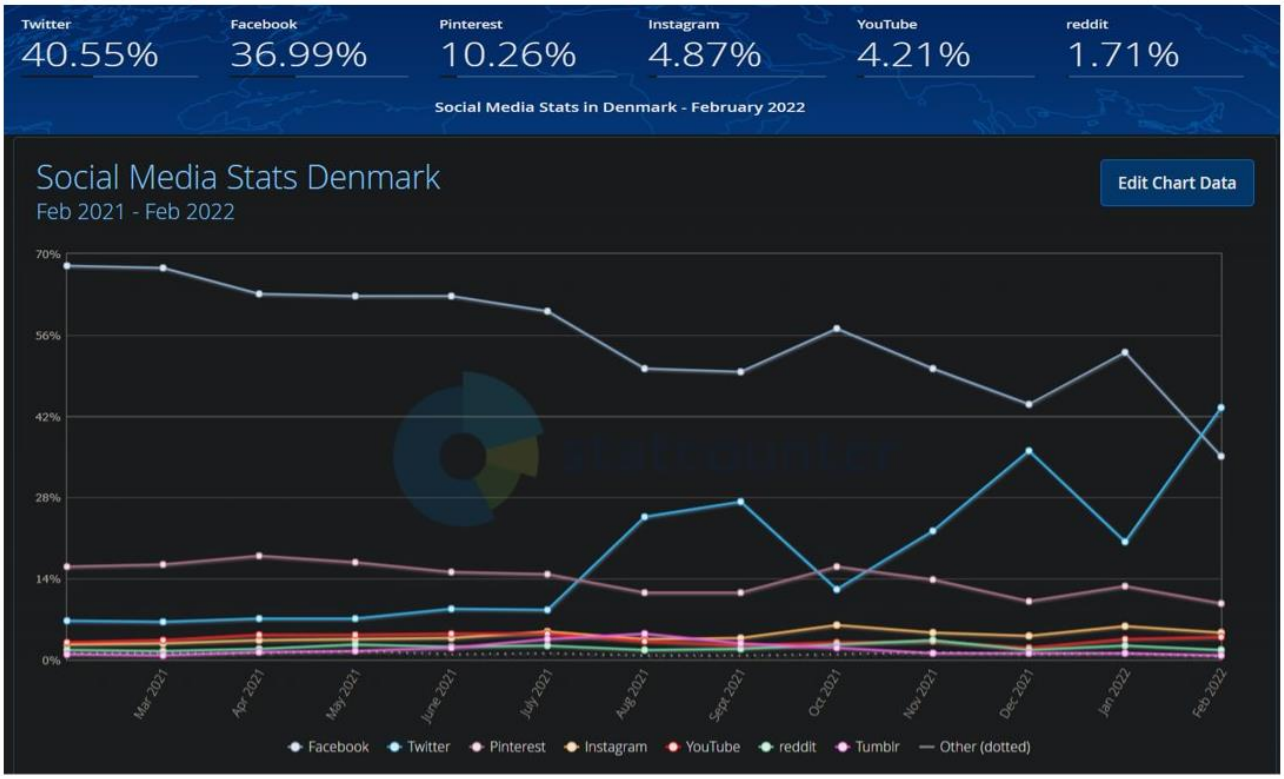
Family disposal of vehicles

- From the statistics we conclude that our target audience is married, because more married citizen has car than single citizen. They are above 30 years old, because most citizen gets married after 30. They own a Volkswagen; Danish cars don't have automatic transmissions.
- (This is part of the same Viking ethic that dissuades them from taking an aspirin for a headache: an automatic transmission, like a painkiller, is considered the "easy way out.")



[12]

Average citizen



[13]

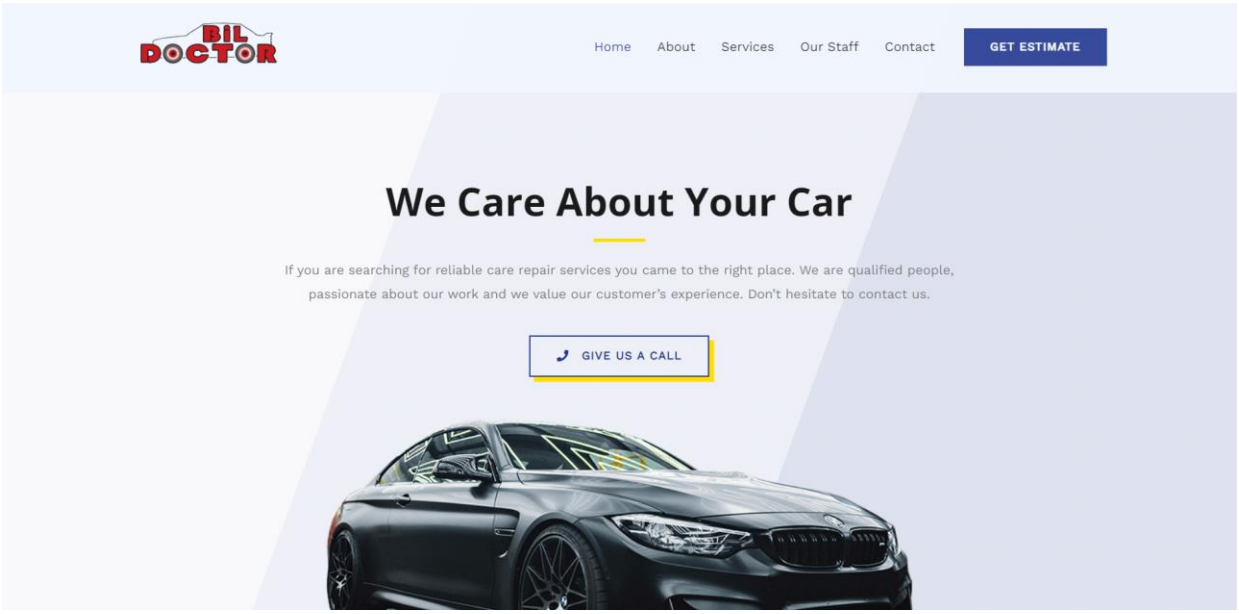
Social media stats in Denmark

Marketing strategy

Appendix 6

Company website

While working for a business assignment we decided to also create a website in wordpress that fits the business. The website is live and you can find it here.



Website picture 1

Appendix 7

The prototype of the UI can be found here.

Appendix 8

List Of Images

List Of Tables