CASE: BILABONNEMENT A/S

PS&E IT SOLUTIONS & CONSULTANCY

https://github.com/Cannabis2013/KEABilAbonnementDK

https://trello.com/w/firstyearproject7

15-12-2022

Team: Dat22v2

Members

- Martin Hansen (Cannabis2013)   - mart2d32@stud.kea.dk (03.6.1985)
- Nikki Deleuran (delemedia) - nikk0375@stud.kea.dk (04.12.1989)
- Stefan Andreas Jensen (stkea) - stef240p@stud.kea.dk (28.05.2001)
- M. Kaan Arici (mura0717) - mura0717@kea.stud.dk (28.01.1984)

Preface

This report is written for the main assignment for the 2nd semester of Computer Science (*Datamatiker*) degree at KEA (Københavns Erhvervsakademi). The report consists of our initial ideas and considerations regarding the given project proposal, our planning & scheduling of the 4-week timeline, our solution to the problem along with critical assessment of the entire process.

Despite *Bilabonnement.dk* does exist as a legal organization and that it is currently operating, the proposal given is a pretend one for the purposes of education. For that reason, for our solution, we have decided to take this aspect of pretending a step further and imagined ourselves, working for a company, named 'PS&E IT Solutions & Consultancy,' which *Bilabonnement.dk* contacts for the project.

As for the back story for this imaginary business: it was established about a year ago by four classmates from college. And for the past year, they have been working on similar IT projects, given to them by a variety of clients, including *Mario's Pizza Bar*, *Delfin Swimming Pool, Stop Madspild* and *Ønskeskyen*. And so, the analysis and the project itself are written from the perspective of PS&E.

We have come to agree that this approach would make the report more compelling in terms of the subjects in discussion. Plus, it is just more fun like this.

Members Agreement

All members are expected to take part in planning, structuring, coding, designing, and writing the project.

All members are expected to meet physically and online for the group meetings.

All members are expected to be happy, and they are not expected to complain.

All members for one, one partner for all.

Last but not least: No one can talk crap about *FCK*.

Problem Statement

Bilabonnement A/S was established in 2016 and is a subsidiary of K.W. Bruun & Co. K. W. Bruun & Co is a family-owned business with its roots dating back to 1914 when founder Karl Wilhelm Bruun started out selling automobiles. Currently, they are one of the largest automobile importers in the Nordics along with spare parts imports of the brands Peugeot, Citroën, DS, Opel and Mitsubishi in Denmark and Sweden and Fiat, Alfa Romeo, and Jeep in Denmark[1].

As a subsidiary of K.W. Bruun & Co., *Bilabonnement A/S* is an online car-rental company with a fleet of automobiles and a subscription-based model. Their contract options vary from 3 to 36 months with a fixed monthly payment[2].

The company currently employs an internally developed customer support system, in which potential customers can choose a car, a subscription model and a pick-up location, by entering data on the company website. After this, the remaining processes are handled in Microsoft Excel, which the company acknowledges is inflexible and prone to mistakes. Being a joint stock company via K.W. Bruun & Co. with many stakeholders involved, they aspire to improve this current remaining process-handling by incorporating a new internal system.

The new requested system should perform (as service) the following:

- Registration of new rental agreements.
- Registration of damages after the rental period.
- Reporting and monitoring of rentals.

PS&E IT Solutions & Consultancy was commissioned to deliver a solution with a business study on the feasibility, design, and the development of the mentioned internal system.

In the following, the entire process is assessed, and documented in detail.

Planning & Scrum

Planning

Author: M. Kaan Arici

Technologies Used

When starting any project, it is important to plan appropriately in advance to make the most of resources and time that are accessible. Specific tools can be used to aid with the process, just as the specific project management method Scrum, which we have decided to use for the process.

Accordingly, we have chosen several technologies as services, which would allow us to design and manage both the development and the documentation phases more transparently and efficiently. These were:

- Discord

  As a VoIP and instant messaging social platform, Discord allowed us to communicate with voice calls, video calls, text messaging, screen sharing, media, and files sharing in our online group work.

- Trello

  As a web-based, Kanban style[3] listing application, Trello allowed us to create a product backlog as well as sprint backlogs for the four scrum sprints we went through, and in which we assigned members to specific tasks.

- OneDrive & Microsoft Office

  As a file-hosting service, OneDrive & online Microsoft Office allowed us cloud-based file synchronization and back up for all the research and writings related to the project report.

- GitHub

  As an internet hosting service for software development, GitHub allowed us to host our code in the cloud with version control.

- Figma

  As a collaborative web application for interface design, Figma allowed us to create the front-end design of our application.

Scrum

Author: Stefan Andreas Jensen

What is Scrum?

Scrum is an agile software development technique used to structure development into smaller manageable iterations of as little as a few days to multiple weeks in length. These are what are called sprints. For each sprint, tasks are selected to be completed within that sprint. If a task is not finished or is decided uncomplete, it is moved to the next upcoming sprint. Tasks are selected from a product backlog, which is a collection of all tasks to be finished. The selected tasks are added to a sprint backlog for that specific sprint.

A team utilizes a scrum master. They make sure the scrum operation runs smoothly. See problems are solved, deadlines are met, the team improves as well as many other things. They are a key role for a scrum team to function well.

Daily, during sprints, the team will be having a “daily standup” meeting. Every team member explains what tasks they will be working on that day as well as if they have any problems which they need help with.

After each sprint, a sprint review and a sprint retrospective take place. It is during the sprint review the team decides if specific tasks should be moved to the next sprint as they might not have been completed or completed to a certain standard. The sprint retrospective dives into what went well and what went not so well for the team during the sprint. This is done so that the team may improve lesser good things and keep at it at the good things in the future. Multiple sprints can form a sprint iteration for which a certain goal is set to be achieved.

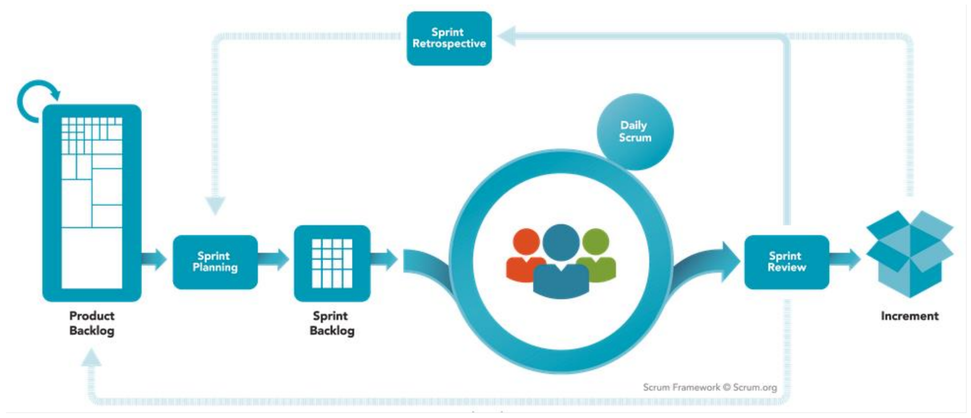


Figure 1:The cycle of scrum[4]

Our Work with Scrum

We chose to follow a sprint length of 4 days. This would total to 4 sprints at the end of the project period. The sprints would last from Mondays through Thursdays. Fridays were reserved for sprint reviews and sprint retrospectives. In addition, we also decided to theme each sprint according to what would be done during that sprint. The first sprint during the first week of the project would only focus on the preliminary tasks. Second sprint would focus on building the product and testing it. The third and fourth sprint would focus on writing and proofreading the report for the project.

During the sprints, we would have at least one daily meeting primarily focused on what everyone would be doing that day based on the tasks we would have split between us, as well as discuss potential problems where someone would need help. We chose one person to be the 'Scrum Master' throughout the project. Their responsibility was to organize the daily meetings during sprints and keep the Trello product and sprint backlogs organized with tasks.

Our four sprints would come to define one full scrum iteration in which the goal was to create a Minimal Viable Product (MVP) based on the following criteria's:

- Data registration – must be able to register new rental agreements.
- Damage and reparation – must be able to register any damage or defects upon return delivery of the vehicles at the end of their rental period.
- The business must be able to see
  - The number of vehicles that are in circulation (currently rented out).
  - What is the total worth of rentals in circulation?

Product Backlog and Sprint Backlog

We utilized five boards in total. One for product backlog as well as one for each of the sprints. We could assign team members to each task as we could easily move tasks between boards if needed be.

Analysis

Feasibility Study

Author: M. Kaan Arici

Focus & Introduction

A feasibility study aims to determine the probability of a project's success and whether it is profitable before investing resources in it by providing a rough assessment in certain criteria[5] By doing such an analysis, the company can predict which problems could arise in connection with the proposed project and evaluate it against others to determine if it should be proceeded with.

Accordingly, the following study intends to cover these areas:

- Project Overview
- Technical Feasibility
- Economic Feasibility
- Operational Feasibility
- Legal Feasibility
- Organizational Feasibility
- Behavioral Feasibility

Project Overview

The requested system needs to track information on subscriptions, rental agreements, damages, and total worth of currently rented out vehicles. The challenges in developing such a system also comprise the scope of the feasibility.

The system will be developed in Java programming language for back-end along with *Spring boot* framework and *thymeleaf* template engine for front-end. It should also follow GRASP principles for future additional implementations.

The following analysis is conducted based on these aspects.

Technical Feasibility

In brief, technical feasibility determines if the necessary tech, which consists of IT (Information Technology) infrastructure and resources, produces the viable solution for the commissioned system. Additionally, it also determines if the organization's current available technological infrastructure is adequate to achieve the project's objectives efficiently[6].

Even though there is always the possibility of unexpected technical drawbacks in any technological development, regarding the project objectives at hand, it can be confirmed that the commissioned system does not require any use or incorporation of novelty technology. Instead, what the project objectives require are a 'object-oriented programming language' for the back-end development in combination with a simple, user-friendly, front-end design, hence making it a typical full stack development project. Something that PS&E IT Solutions & Consultancy is already experienced in, as well as possessing the technological resources for it. In addition, an open-source distributed version control hosting service would be used for collaboratively developing the source code.

Therefore, the project does not bear any explicit technical difficulty that can hinder or prevent its completion.

Economic Feasibility

Financial aspects constitute a significant part of the feasibility study, as it attempts to determine whether the project is an acceptable financial risk before proceeding further. There are several factors to consider in this determination. Yet two key issues to address are:

- Do the expenses exceed the gains while the company completes the project?
- Can the project be developed and delivered on time[7]?

Expenses

Start-up Costs: There are no expected start-up expenses as the project does not require any office space, additional staff members, or purchase of additional equipment, server space or new development software. The existing members of the staff, equipped with their computers, will be using open-source platforms and software to carry out the project.

Ongoing Costs: The ongoing costs of the four staff members are constant regardless of whether the business proceeds with the project or not. If the 4-week schedule of completion of minimum requirements is met, there will be no additional expense. And for a project that size, currently there is no indicator of it being over schedule.

Sources of Funding: After initial and ongoing costs are assessed and established, there may be necessary to consider the sources of funding if the company does not have the required cash flow. If that is the case, then the allocation of additional funds is needed. However, when the company's current finances are considered, there seems no need to search for funding sources.

Revenues

Expected Earnings: The complementary side to the project's expenses are the revenues expected generated upon the completion of the project. In other words, the return of investment (ROI), based both on physical costs and the time value of money. The project proposal by *Bilabonnement A/S* is a one-time assignment, in which the delivery of the product and the payment following it will be the end of the contract for both parties. Therefore, it is financially necessary to make the markup minimum at 50% in setting the price to be paid by *Bilabonnement A/S*, considering the 25% profit tax in Denmark.

That said, one added benefit of this project may be the possibility of digital maintenance or further development of the final product, which would eventually generate more possibilities of income for PS&E.

Based on these assessments in expenses and revenues, the study shows that the project, if proceeded with, is estimated to be economically feasible because the net income (if the pricing is set as suggested) of the project will exceed the costs, thus resulting in profit.

Operational Feasibility

The ability to increase performance, in which costs are reduced without diminishing the end quality is an applicable approach in operational feasibility. In that regard, economic feasibility becomes a guiding part of the operational feasibility. Based on previously discussed economic assessment, it is established that the proposed project only requires the standard ongoing costs. For that, the operational feasibility may become a natural candidate for exploration in improvement for the matured systems of development. In other words, team members, who are well-versed in carrying out the tasks based on previous similar experiences, may explore new improvements. Improvements that can lead to cost-cutting opportunities, such as reducing the number of members working on the project and still delivering on time and without cutting down on quality because they are experienced enough.

Coincidentally, for this project, the 4-member team will already face a forced cutdown in the workforce due to one of them will be unavailable for some time for a planned medical operation. Since the developing team is aware of that, they already plan accordingly to maintain the operational feasibility as it currently is.

Legal Feasibility

As in many IT development projects dealing with personal data, GDPR is the principal concern in determining the legal feasibility[8]. With that regard, the delivered product should ensure that data handling, which in this case is personal data including driver's license information, follows the GDPR.

Clearly, *Bilabonnement A/S* is the primary agency responsible for ensuring the safe storage of their customers' data since PS&E will not be supervising how the final delivered product will be used.

However, from our organizational perspective, we need to communicate the need of safeguarding of customer personal data for future updates either by encryption while stored in database or by creating log-in requirements to access the database containing the data.

This has already become standard procedure in terms of legal aspects in similar projects, which PS&E has worked on, and hence is experienced with. Therefore, in terms of legal feasibility, it does not bear any hindrance to the project.

Organizational Feasibility

Organizational feasibility determines how well the proposed project is in line with the organization’s core objectives and its strategic planning. If it is the case for a newly forming company, then this would be how well the company’s ‘*proposed information system supposes the objective of the organizations and its strategic plan for an information system*’[9] Since PS&E IT is already established, the key factor in assessment here is to see whether the introduction of the new project would entail organizational changes due to it deviating from the founding objectives. The answer to that question in brief is, no. The project, if accepted to proceed with, will not cause any changes to the organizational structure. Firstly, it resembles the previous projects that were carried out by the company. Secondly, the product is intended for external use, meaning once it is delivered, it will only be used by the client. There may be a need for maintenance, but this is a minimal concern since it is beyond the scope of the current study.

Behavioral Feasibility

Behavioral feasibility concerns human issues. People usually resist change and, in every system development project, to a degree there is always an introduction of change[10]. Therefore, the level of resistance from the employees, which may come in different forms and actions, must be considered.

At PS&E IT, the development team works quite welcoming in terms of novel approaches, except one of the members: Martin Hansen. He constantly advocates for C# programming language and resists fully committing to Java programming language.

Yet aside from this single occasion, the project does not demonstrate any critical problems regarding behavioral feasibility.

Feasibility Study Conclusion

The purpose of this in-depth study was to find any reasons if the project for *Bilabonnement A/S* is an achievable undertaking for PS&E to move forward with. The study has tried to touch on every area of interest regarding the project's feasibility and give a good insight into the company structure. In addition, the continuous communication with the development team and other parties within the company has given us a solid understanding of how the project can be handled without any major unexpected problem.

Therefore, the study concludes that *Bilabonnement A/S*'s project proposal is a feasible one for PS&E IT Solutions and Consultancy.

Stakeholder Analysis[11]

Author: Nikki Deleuran

The purpose of the analysis is to focus on the key stakeholders within the project development and to avoid losing time and effort on others that have no effect on the project's development.

A stakeholder analysis can be continuously updated, since changes may occur midway in the company's internal affairs, and hence can impact the course of the project.

The challenge is to get various stakeholders to contribute positively to the project's development rather than experiencing interruptions in the process. Certain stakeholders have more status than others. Therefore, a stakeholder can have several roles in a project's development.
Stakeholders are divided into 4 positional categories depending on where their impact of influence and implementation lie in the project. In the following, they are listed from the highest priority to the lowest in each section.

Internal Stakeholders

Resource Staff Influence on the project development team. Project Implementation. Biggest influence.	
Project Group & Leader: PS&E IT SOLUTIONS & CONSULTANCY	Develops the project for the company.
Control Group: Technical / IT department	The project of decisive power.
Project Admin: Technical Chief	Assist the project group with administrative routines.
Controllers: Financial team	Focuses on the economy.

Table 1: Resource staff

The Hostages Employees in the client company: Bilabonnement. And customers to keep the business running. Necessary for the result – Little influence on the project development.	
End users	Bilabonnement employees. Uses the product after development.
Follow-up groups	Employees in the other sectors can be informed about the project on an ongoing basis. *Customer service* + *Service teams* + *Mobility service.*
Customers	Unimportant influence for the development. But relevant for the company business turnover. *Renters* + *Advance buyers* + *Business customers* + *Purchaser* + *Auction house*

Table 2: The hostages

External Stakeholders

External Stakeholders Cooperation Partners & Competitors. No focus on development – Unless sudden market changes.	
Suppliers	Deliver sub-components for the project. Peugeot, Citroën, Opel, *etc.* - *see Appendix.*
Business partners	Independent stakeholders who contribute to the project. TopDanmark Forsikring A/S, Autoproff A/S, FDM (Rochs.dk), etc. - *see appendix.*

Table 3: External stakeholders

Gray eminence Not required for the project. Major influence on the project's further course. Entrepreneurs. Top concern businesspeople (Change / Prevent the progress unexpected) Will not be considered in the progress project group. If the businesspeople include themselves.	
Top Management	Allocates resources to Bilabonnement, then allocates to the project. Including primary companies – K.W Bruun & Co - *see appendix.*
Assigned Principal (Project-owner)	*Director - Who finances the project - Who owns the project.*
Reference group	*Management - Who finances the project - Who owns the project.*

Table 4: Gray eminence

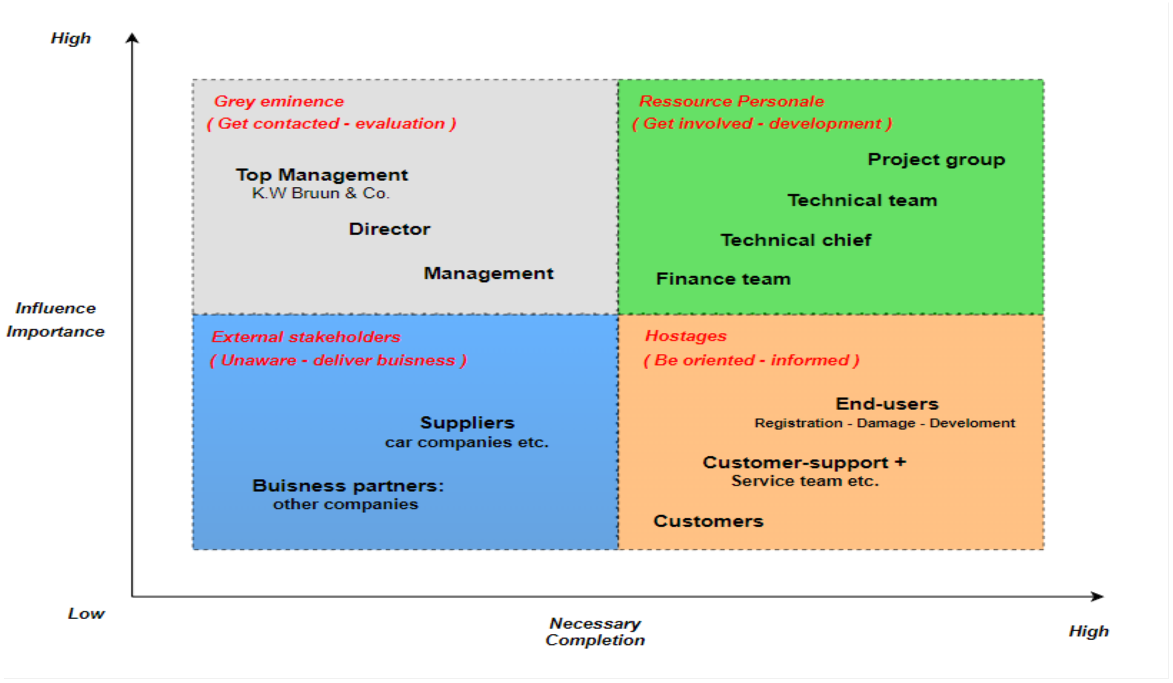


Figure 2: Stakeholder chart

Risk Management

Author: Martin Hansen

We, like any other developer team in the early stage of the process of developing a software solution for a given customer, allocate time and resources to find risks to counter eventual problems that can sideway our project. Choosing not to can have catastrophic consequences that can postpone delivery date or lead to a dissatisfied customer. We have several tools at our disposal, and below we outline the ones that we find cover most aspects:

- A SWOT analysis with identified strengths, weaknesses, opportunities, and threats.
- Simple risk analysis table with risk factors, score values and score product.
- An extended risk analysis table with new columns like precautions, responsibilities, and solutions.

Using a SWOT analysis as a starting point can be especially useful as risks for the company can reflect on the developer team in forms of sudden requirement changes, bankruptcy, or shift in direction that could lead to cancellation of the order. All these cases can compromise the project in several ways and therefore need to be found before going ahead to the development phase.

Next, we must find risks that pose a threat to the project and score them individually by their probability of occurrence and consequence to the project, and then assess the severeness of each risk by the product of these scores.

Finally, for each risk, we add details such as precaution steps, a solution, and assign responsibility for each of these added details. All done for impact reduction reasons.

We cover SWOT analysis in another section and skip to the risk analysis part. For space reasons we have placed a full table with risk factors, probabilities, and consequences and more, in the appendix “***Risk Analysis"***.

Conclusion

We feel that none of the risk factors pose a damage high enough to sideway our project. This is due to the project's size but having a plan for each factor makes the way to deliver as scheduled. We assess the likeliness of our project goes sideways as not likely.

User Stories

Author: Stefan Andreas Jensen

Utilizing user stories is a great way to break down complicated assignments into smaller assignments. When doing so, teams have an easier time understanding assignments as each assignment is only focused on a specific task and not cluttered with other task specifications that need to be solved. This in turn also allows the team to set more precise deadlines for assignments which suits the whole aspect of agile development very well.

User stories should follow the **I**ndependent **N**egotiable **V**aluable **E**stimable **S**mall **T**estable principle shorten to INVEST. In other words, they should be independent from each other and their value to the project be judged independently. They should be negotiable as they are not final. Rather they should be a leaping stone for further discussion and negotiation between the development team and the business. The value they bring to the project should be clear and well-phrased. The user story should be estimable and easy to put a deadline on. The user story should not be too big and should be precise and only focused on one task. Lastly, they should be testable to prove their value[12].

A user story should have acceptance criteria. Acceptance criteria are not functional criteria to the user story, but they are the condition of satisfaction being placed on the system[13].

With this in mind, we broke the criteria of the minimal viable product (MVP) down into the following user stories. These served as an outline for what we wanted the final product to consist of. As with user stories, the final stories are not contracts[14], which necessarily need to be fulfilled but served as a tool for us to better understand and structure our work with the MVP criteria. As stated, we used the stories as an outline for our project, and we therefore chose to create stories closer to epics than standard user stories. Epics are the same as user stories but much less focused as they can consist of multiple tasks, which in turn could be broken down into normal user stories. We found this decision justiciable as the user stories were never meant to be our primary task creation method.

*For space reasons, see appendix **"User stories"** to view our user stories.*

SWOT-Analysis

Author: Stefan Andreas Jensen

A SWOT-analysis is an excellent tool for determining a business' current strategic position in the market. It provides a clear overview of the business's internal as well as external values and can therefore be a great tool for further analysis and decision making. We have chosen to make a SWOT-analysis to further understand Bilabonnement.dk A/S as a business, where they succeed as well as where they lack as a company. The completed analysis is based on the material provided for the examination case and our own research done mainly on the Bilabonnement.dk A/S' website. A further explanation of each category and its findings will be presented after the analysis diagram.

Strengths	Weaknesses
- Part of established old company (K.W Bruun) - Existing platform with data - A lot of experience - A proven working business model - Big car inventory of multiple brands	- Not very modernized in terms of tech - Small team (8 people) - Complex daily routine (new cars coming in, old ones leaving, new deals etc.)
Opportunities	**Threats**
- Buy-outs of competition - Changing customer interests - A blooming marked	- The car inventory becoming outdated - Competition - Regulations – data security (- Excel) - Regulations – Combustion engine cars

Table 5: SWOT

Strengths

The strengths section is one of the two internal values part of a SWOT-analysis. It provides an overview of aspects the company is doing well at and succeeding in.

We find that Bilabonnement.dk A/S is operating a successful business with a great base of support and knowledge-sharing by them being part of the K.W Bruun group. They have access to many car brands, allowing them to target a broader section of customers. They have proven their business-model by being operational since 2016. Furthermore, they grant the customer a great experience through their website which allows the customer to lease a car from home. Great for the customer but also great for the business in terms of automation. This is an important strength considering the relatively small team.

Weaknesses

The weaknesses section is the second of the two internal values part of a SWOT-analysis. It provides an overview of aspects where the company lacks and where there is room for optimization.

We find that Bilabonnement.dk A/S is lacking in the modernization aspect. While they have a website for customers to use, they still depend on using Excel internally for data registration.

This is a lot of manual labor which could be automated. Their team is relatively small at only 8 people. With this number of people, one can assume every person on the team is specialized in their own field and therefore not able to help in other fields. In other words, they are not very set for scalability, which might turn into a bottleneck down the road.

Opportunities

The opportunities section is the first of the two external values part of a SWOT-analysis. It provides an overview of external aspects that might favor the company.

We find that Bilabonnement.dk A/S is a young company at growth. They are part of a big and knowledgeable company group (K. W Bruun), and this might create opportunities like the possibility of buying out competitors to further grow as a business. With a growing interest in a greener daily commute, customers are becoming increasingly interested in electric vehicles (EV). This could be a great opportunity for Bilabonnement.dk A/S to establish themselves as the go-to spot for EV’s as well as future-proofing their car inventory.

Threats

The threats section is the second of the two external values part of a SWOT-analysis. It provides an overview of external aspects that might cause a problem for the company.

The talk of a greener future has been established as a daily topic, and more initiatives are being put forward to achieve this. One of them being electrification of cars while combustion engine cars will be banned in the foreseeable future. Bilabonnement.dk A/S as a company is at huge undeniable risk because of these regulations and if they do not manage to update and future-proof their car inventory in time, they might end up with a worthless car fleet, which could cause them to lose out to competition. The ever-growing focus on data and privacy handling (GDPR) might also cause a problem for them down the road as they are still using Excel for handling registration of customer-related information.

Domain Model

Author: Martin Hansen

We feel that to create a good domain solution, it is important to get an overview of the problem domain on a more conceptual level. We need to identify entities, their values, and their relation to the company itself and to each other. In our case, we find terms like car, customer, agreement, damage report, and damage, as relevant artifacts to include in our domain model. It is not the entity itself that is interesting, but the value they hold in terms of attributes. We will not constrain ourselves to just entities as we aim to create a model that not only deals with entities and their attributes, but also how they relate to the company and to other entities. The level of details is kept on a low basis but might serve well as the foundation for further steps on our way to create a domain solution. Below is our domain model represented as a class diagram made in Visual Paradigm:

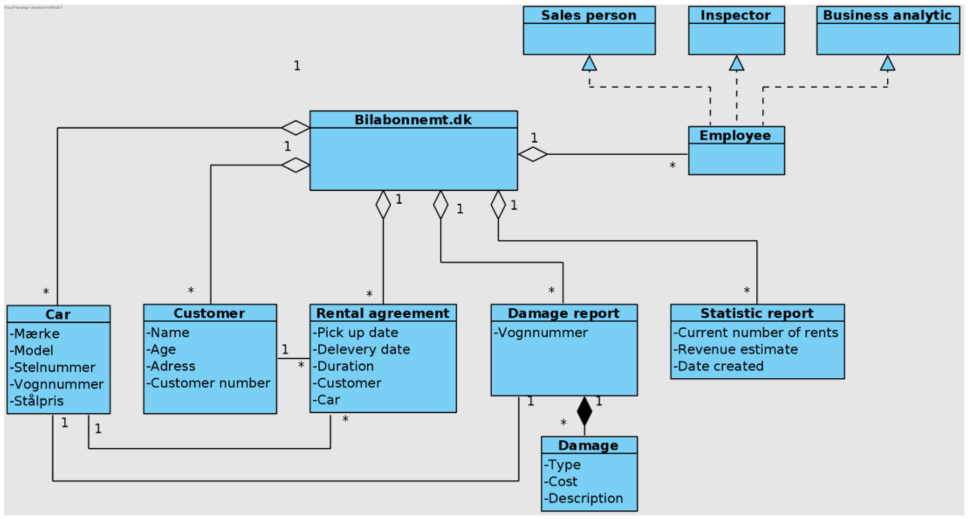


Figure 3: Domain model

The Core Domain

Then, as a next step, we decide that some areas are more important than others, and we need to focus on what some writers call *the core domain*[15]. The core domain is, in short, a term that describes the part of our domain model in which we must invest most of our time and resources in. That area we feel answers the question of why it is being built and holds most of the value important to our customer. This is the foundation for our MVP. Below we provide a domain model with the core domain highlighted in a dark rectangle:

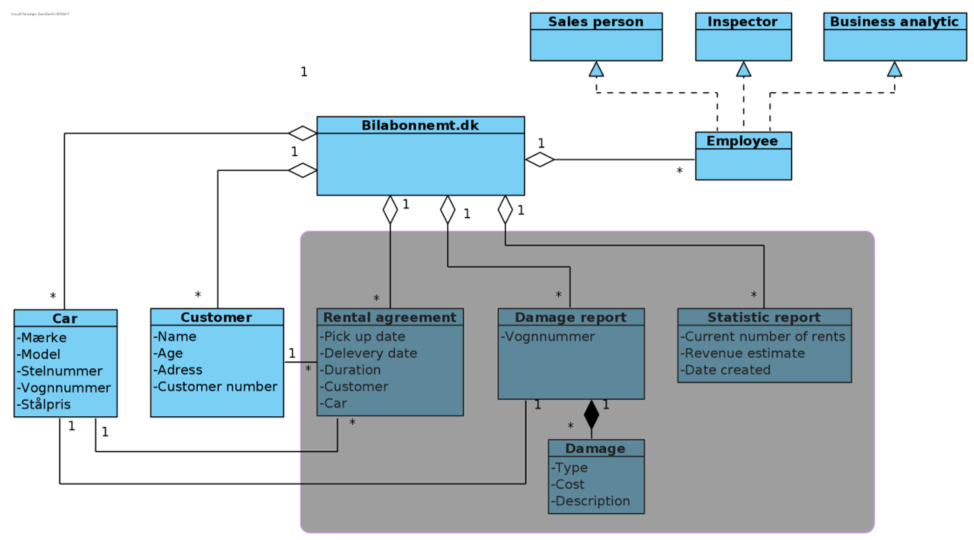


Figure 4: Core domain

We think entities like a rental agreement, a damage report and corresponding damages, and a statistical report, contain a lot of customer value, and therefore we think it should be the foundation of our MVP. Other areas like the employee section that involves salespersons, damage inspectors, and business analytics, are not a priority as this part can be imported from existing libraries. So, at this stage, our assessment is that this part of our domain model should take up most of our time and resources.

Design

Database

Author: Martin Hansen

In this section we will give a brief introduction to DBMS (Distributed Database Management System), give an account of some of the issues we encountered during the process and finally cover the overall database design in terms of entities and normalization.

MySQL

As covered in the domain model section, details like agreement and damages hold business value important to our customer, and as such, this value needs to be persisted someplace. There exist many ways to do this and a popular and modern choice is using a database. Database is a way to store data in such a way that we do not need to worry about the way it is stored. We do not have to format and manually write data to a CSV file as an example. We just want to send a database a command and it will do the rest. A system that accommodates these features is MySQL, which is an open-source DBMS maintained and owned by Oracle[16] and is a software solution that is designed to handle multiple databases. In fact, it is of the type Relational Database Management System (RDBMS) that stores data in tables that are related to each other and typically uses a query language called SQL (Structured Query Language) to handle database manipulations.

Azure SQL Database

The server could either be local or remote on cloud services like Azure or AWS (Amazon Web Services) to just name a few. Azure is widely used in Denmark and was also our choice as hosting in the cloud. It helped us overcome a series of obstacles such as platform and inconsistency -related issues. Reducing issues like “it works on my computer but not on yours" and avoiding refactoring on multiple local databases every time we alter a table was a good thing in our belief; and we still think it is.

But hosting is not free, so we decided to take advantage of educational accounts that provide 100$ of credits. This leaves us with plenty of room to work with. This service's cost is based on hourly usage and the server size in terms of processing power but most importantly the storage size. Azure charges 0.12 USD per gigabyte and the minimum system available is 20 GB. Further, the first 32 GB is free so we could grow it a little[17], but we will not go near that limit, so we choose the minimum size. As for the hourly usage billing, the first 750 hours (about 1 month) are free, then Azure charges 12.71 USD per month. With our 100 USD of credit in mind, that should leave us with about 8 months of usage with no payment charged on our credit cards. Therefore, we have hosted our database on Azure to reap all the above-mentioned benefits a hosted solution provides.

Entities and Normalization

The rows and columns in our database conform with the business value shown in our domain model. We persist entities like car, customer, rental agreement details, damage report and related damages, and as such we need to create tables to hold this information. We can store all this information in one single row, but this will add a substantial number of redundancies, which is something we want to avoid. Removing one car will also force us to remove multiple rows that functionally depend on this car. Therefore, we have normalized so every entity only contains data relevant for the given entity. Entities that are related to other tables are related by keys, to reduce coupling.

Our database consists of tables for cars, customers, rental agreements, damage reports and damages. They are normalized into the 3nf normalization mode[18]. That is, they comply with the following criteria's:

- Each cell contains one value (1nf)
- Each cell contains a unique primary key (2nf)
- Partitioned into tables (2nf)
- No transitive functional dependencies across columns (3nf)

Below ER diagram illustrates our tables and their relations:

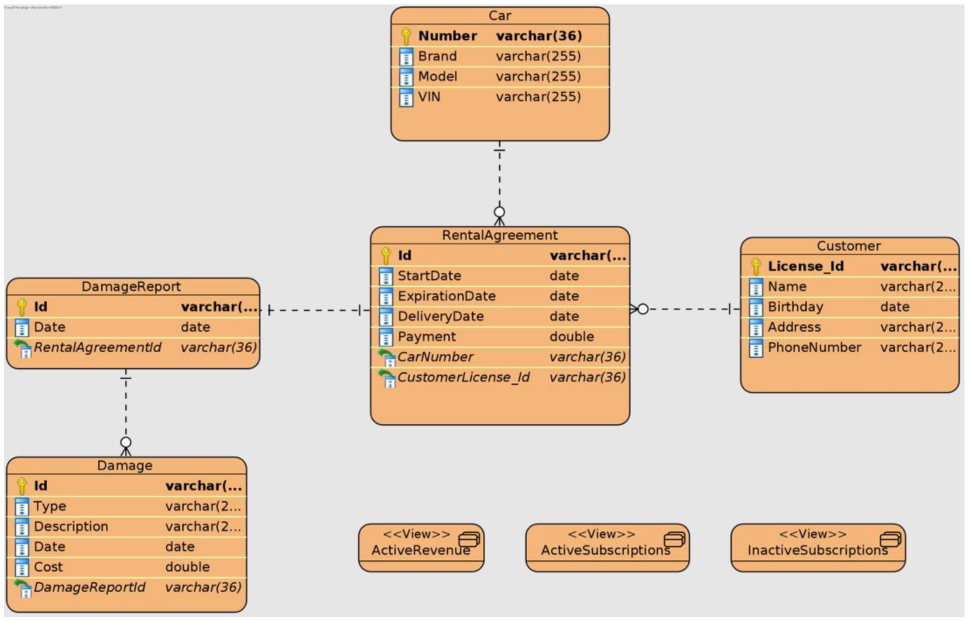


Figure 5: ER diagram

We decided to use varchar with fixed byte sizes to minimize database size and apply indexes on primary keys to increase query performance. Further, we favored UUID as id, so varchar with fixed byte length of 36 bytes was the ideal solution. Finally, we have constraints on our foreign keys to avoid rows without parents.

Database Initialization

As mentioned earlier we developed up against a local MySQL server and that made us open for inconsistency issues in terms of table differences across computers. To counter this we created some SQL script files that create and initialize the database with some random models and with version control we may have reduced a significant amount of time to correct inconsistencies.

Evaluation

We think our database meets our needs for now regarding our MVP, but we feel there is room for improvement. For example, much of the logic still resides in the java codebase and we could with some benefit take advantage of the more advanced features that MySQL provides like *procedures* and *triggers*.

To reduce code redundancy, we experimented a little with *views*. Queries can sometimes involve multiple tables that can result in large and deeply nested SQL code, so we thought now we could reduce the amount of duplicate code by using Views and reduce the amount of code for readability reasons. As mentioned earlier, we experimented with it, that means we have not fully utilized it throughout the whole project. This was partly because we began implementing at a late stage, and after implementation platform specific issues showed up that took up a significant amount of time to fix. So, we decided to limit the use to some statistical operations like calculating the annual revenue and some key numbers.

GRASP

Author: M. Kaan Arici

GRASP is an abbreviation of ‘General Responsibility Assignment Software Patterns (or Principles),’ and it is a set of nine fundamental principles first published by Craig Larman in his book, "Applying UML and Patterns"[19]. These principles help to organize the conceptual architecture of software development projects, and they are:

- Information expert
- Creator
- Controller
- Indirection
- Low coupling
- High cohesion
- Polymorphism
- Protected variations
- Pure fabrication

In terms of Object-Oriented Programming (OOP), responsibility assignment is a critical concept that does not only concern classes but also entire systems (Larman, p.6, 2002). Thinking in terms of assigning responsibilities is a way to think about the software's design. And GRASP proposes these principles as guidance to skillfully assigning these and achieving a well-designed system architecture.

In the following, we analyze five of the nine principles in detail and how we implemented them in our project.

Information Expert

As a basic principle, the information expert determines where to assign responsibilities to optimize efficiency. Because during object design, we make choices about the assignment of responsibilities to software classes to create systems that are easier to understand, maintain, and extend. As Larman notes, responsibility should be assigned to the class that has the relevant information regarding that responsibility[20].In other words, a class with the most knowledge about something should also be responsible for it, allowing easy reusability in other components. Accordingly, in our application, we have divided and assigned responsibilities in terms of 'Rental Agreement,' 'Damage Report' & 'Statistics Report.'

Creator

An object-oriented system typically has classes that specialize in creating objects. Therefore, a general principle for the assignment of creation responsibilities can be useful for the general design. This can also support low coupling, encapsulation, and reusability[21].

Accordingly, in our application, we have decided to use 'factories' in creating certain objects. A factory is an object class with a method that can be used to return other objects of varying prototype or class[22] . The factory classes in our application are set up to create empty forms of 'Rental Agreements' and 'Damage Reports,' which then would be filled out by the end-user/staff and saved in the database.

Controller

As Larman describes it, '*a controller is a non-user interface object responsible for receiving or handling a system event*'[23].

And regarding MVC, the controller pattern becomes responsible for accepting inputs and converting them into commands for the model or the view. It is an architectural pattern, which our application utilizes as well by the 'Registration' and 'Inspection' controllers that are responsible for accepting requests from the user. These requests are carried out via model classes and repositories and then returned to controllers, which then send requests to update the view that can dynamically render the final view to be delivered back to the user.

Low Coupling

'*Coupling is a measure of how strongly one element is connected to, has knowledge of or relies on other elements'*[24]. If a class, for instance, relies on many other classes, changes in related classes force local changes due to that class having too many dependencies to the others. Therefore, low coupling is a desirable aspect to make changes to the system easier.

Following this principle, in our application, the interface classes, which the repositories implement, provide low coupling for queries, coming from the controller to perform actions regarding the database.

High Cohesion

In simple terms, cohesion refers to how closely related and focused the responsibilities of an element are[25]. Therefore, elements with highly related responsibilities do not have to do a lot of work to keep themselves together, which is indicative of high cohesion.

Accordingly, classes and subsystems that are broken down into smaller yet related groups, more manageable pieces to achieve high cohesion. In that respect, high cohesion supports low coupling.

This principle's application in our project can be seen in the implementation of CRUD operations under repositories, in which CRUD operations are divided into subclasses. This allowed each element's responsibilities in the group to be closely related and focused on a specific function.

3-tier Architecture

Author: Stefan Andreas Jensen

It can often be a good idea to split the functionality of a backend application into different tiers. Each tier oversees its own type of operations. This limit coupling between classes which does not necessarily need to be connected directly but can be connected through a middleman. The Java framework - Spring - comes built-in with these features out of the box[26]. Spring annotations like

Controller, Repository and Service make it easy to assign java classes to different tiers of the application.

A controller receives the user request through an endpoint. It can forward this request to a respective repository in charge of the requested data type. The repository is in turn connected to a bunch of services all in charge of each their own specific job. The repository can hand the assignment out to these services and return the result to the controller. The data can then be optionally further processed or parsed to the front end finally. Using a tier-based architecture like this, we limit the number of couplings to the controller. The controller is only ever connected to the repository. The controller should not need to know how to handle the user input. This job should be of a service in charge of only that type of assignment. The controller finds this service through a repository. This way we create a data workflow with as low coupling as possible.

For the project, we decided to base our entire backend infrastructure on this pattern. Every database operation - Create, Read, Update and Delete - would be split into a java class annotated as a service. These services could be dependency injected into a respective java class annotated as a repository. We had a repository for each of the three main requested MVP features as well as an extra repository in charge of handling dummy car- and customer data. The number of repositories therefore totaled four. Depending on need, these repositories could be dependency injected into java classes annotated with controller, which would rule our 3-tier architecture complete.

UI Principles

Gestalt principles

Author: Nikki Deleuran

Is a set of graphical design principles/laws as a psychological tool to (UI) User-interface.
The layout, form, and order for the visual communication to the user perception.

The Design Laws

The focal point principle:

That whatever stands out visually will capture and hold the viewer's attention first.
Think about what is important to emphasize which elements get the most attention[27].

The Law of Proximity:

Elements who closer to each other, seems to affect more importance to those elements than elements with further distance. Object like text / picture etc. which are closer to each other will be perceived as a context. Our brain prefers to place elements in order. This law affects our brain more than the law of similarity between color, forms, and other factors.

The Law of Similarity:

The law is used in all forms of visual communication.
The law of proximity has priority. But the law of similarity is important to create transparency and continuity. If you line up a path of figures in the same sizes and distances between them. The figures would not be similar.

The Law of Closure:

The tendency to group elements that has been closed away from other elements.
With a border or a different background, it is an alternative to the law of proximity.
It is used to clamp a lot of different information into a relatively small area. For example, for a news website, it can be better to divide and separate for additional sections into a single page for an easier overview.

The Law of Continuity

Clarify which elements are connected and separate from other elements.

Always other ways than closing them into a box.

Our view sight is guided by a path / line / curve by coherent colors.

Combine elements which do not necessarily have the same properties.

But it appears as if they have something to do with each other through continuity.

The Law of Figure Ground:

Used to create a good balance between foreground and background.

If a foreground and background fill out the same, it can easily arise confusion.

Important to make a text easy to read so the background does not take the focus or disturb[28].

Visual example

The laws of proximity and similarity is expressed here. Since elements as text and headers have the similarly color and font and are separated or conceal as collected context based on their proximity. The similarity is transparent, and it creates a visual consistent continuity.

The laws of closure and figure ground is expressed here. Since elements is layout with borders to boxify certain closure content. It gives clear balance of back and foreground.
Highlight each car sections, better overview for the user. The focal point stands out with every car image. The details button gives symmetry with the headline.
The law of continuity has a clear right path from the headline to the car image to the details button. When the main process is done the user arrived back to the home overview page.

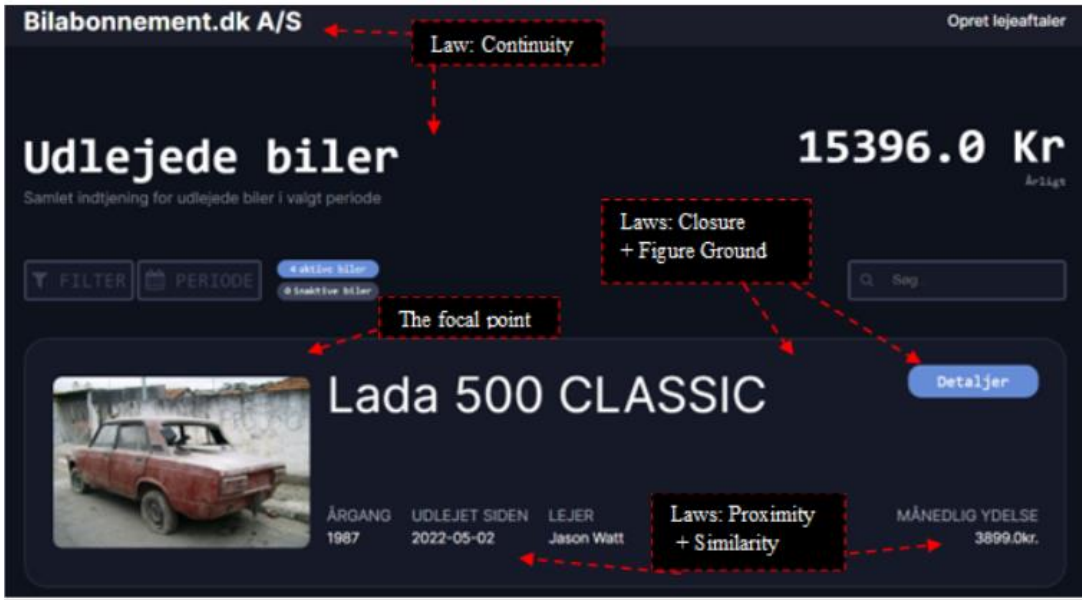


Figure 6: Overall design

8 Golden Rules of Interface Design (UX):

Author: Nikki Deleuran

The underlined rules must be translated, refined, and expanded for each environment.

They have their limits but give a good point of view for mobile, desktop and web design.

The rules focus on increasing user productivity by providing simplified data entry procedures, understandable displays to increase the sense of competence, mastery and control over the design system.

Strive for consistency:

Consistency arrives of a set of actions. That should require similar situations.

An identical terminology of consistent colors, layouts, letters sizes and types, etc.

A similar or identical pattern in menus, prompts, screens, troubleshooting etc.

Except for actions with a serious tone like confirmations, deleting, or echoes of passwords.

Seek universal usability:

Acknowledge the distinctive design of content for variation of user's needs of behavior.

From the beginners to the experts, from different age groups, some disability, from cultural differences, from technological diversity. Each considering spectrum that drives of the requirements to create usability for a pleasing user flow.

Beginners need adding of explanations and functionalities. Experts need shortcuts and a quicker tempo. Improvements that perceived the quality for better interface design.

Offer informative feedback:

For every user action, there should be interface feedback.

With frequent and minor actions, the response can be a message. While the response to rare and larger actions should be more comprehensive. The visual presentation of the interesting elements provides a more practical environment to explicitly show the changes.

Design dialogs to yield closure:

Sequences of actions should be organized with a beginning, a middle and an end.

Trying to avoid a user's mind and behavior of a contingency plans. Informative feedback, at the end of a set of actions gives the user satisfaction of achievement. Easier feeling of relief from their mind and indication to prepare for next set of action.

Like an ecommerce transaction website where the user picks a product to the checkouts.

Ending with feedback of a clear confirmation page to complete the transaction.

Prevent error:

The design should guarantee that users is not possible for users to approach their own severe mistakes. If the user approaches an unfortunate error, then the interface should be simpler, more constructive, and more specific instructions for the after thoughts of the redesign.

Users should not re-enter the whole text formula again, but guides to the defective part.
Faulty design actions should instantly leave the interface state or redesign the interface state.

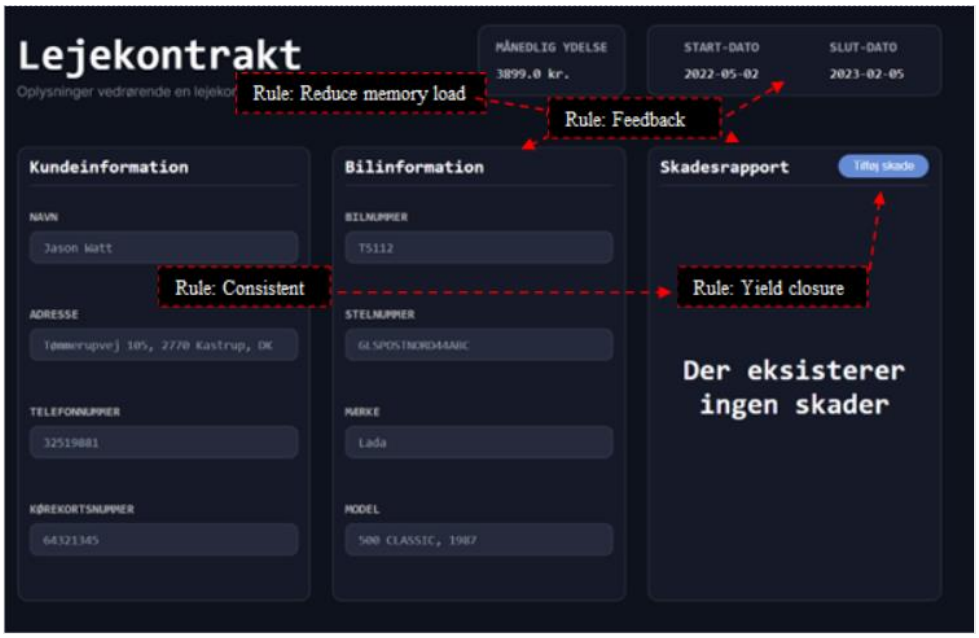
Permit easy reversal of actions:
Actions should be reversible, as much as possible. For avoiding deadens in the UX.
Users are used to their own mistakes being undone to get a sense of control and relieving anxiety for the interface trouble. Reversibility should be a simple action for the user.
Even for the data-input or a complete set of form actions.

Keep users in control:
Experience users have a strong feeling, that they are responsible for the interface,
and it should react properly to their actions. No surprises or changes in familiar behavior.
Annoyed of boring data-input sequences or a difficulty with necessary info and an inability to produce the desired result.

Reduce short-term memory load:
Users have a limit of memory capacity. The rule of thumb:
“That people can remember maximum 7 plus bites, or to minimum 2 bites of information”
Avoid design that requires the user to remember information from a screen to another screen that needs this information on the second screen. Re-entries become visible and long forms should be compromised to fit a single page[29].

Visual example

The golden rules is expressed. Since it have a consistent identical terminology of left to right layouts, sizing and borders type. Is an identical on-screens pattern that gives a usability of a quicker understanding with minor informative feedbacks or a response based on the add damage button section. Organized sequences start = Customer information, middle = Car information and end = Damage report. It yields to a closure of preparing to the next section.
Create relief by approaching to a new section guided with data forms not creating severe errors.
Possible to feel in control and reduce short-term memory load it fits on a single page.
Only minor consideration is there no layouted reversal button. Maybe in the next iteration?



Lejekontrakt

Oplysninger vedrørende en lejekor

MÅNEDLIG YDELSE	START-DATO	SLUT-DATO
3899.0 kr.	2022-05-02	2023-02-05

Kundeinformation

NAVN	Jason Watt
ADRESSE	Tømmerupvej 105, 2770 Kastrup, DK
TELEFONNUMMER	32519881
KØREKORTSNUMMER	64321345

Bilinformation

BILNUMMER	T5112
STELNUMMER	GLSPOSTNORD44ABC
MÆRKE	Lada
MODEL	500 CLASSIC, 1987

Skadesrapport

Der eksisterer ingen skader

Figure 7: UX

Implementation

Class Diagram

Author: Martin Hansen

Before we dive into some user story specific implementations, we need to cover the overall design of our solution. First, we must point out that it is not a fully 1:1 copy and some details, and even classes, are left out for image quality and limited document space reasons; but nothing of high importance. Again, for limited space reasons, we have partitioned the class diagram into two parts to improve readability and to provide as many details as possible.

Interfaces are denoted with methods, so it is implicitly given that any concrete realizations implement these methods. Furthermore, illustrations of RentalCar and RentalAgreement only provide attributes, not getters and setters. We justify this by the fact that no entity contains methods other than getters and setters.

Below figure illustrates the agreement part:

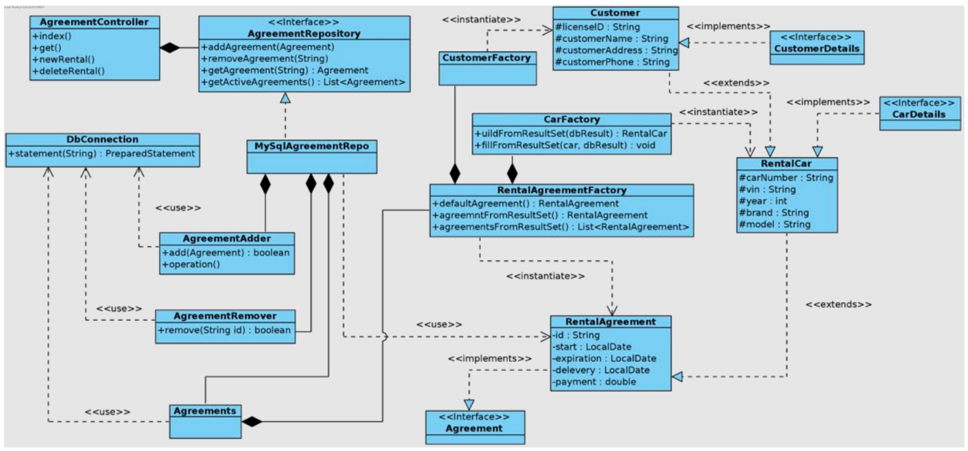


Figure 8: Class diagram of Agreement context

This part is the heaviest and for a good reason. It is the part that fills up most of our core domain and therefore, this part makes up a good share of our codebase. Of course, the part that contains damage inspections is also a vital part of the core domain, but it is not as complex as the agreement part as this part gets most of its details from *RentalAgreement*; so, no factories and other heavy class dependencies here.

Next figure illustrates the inspection part:

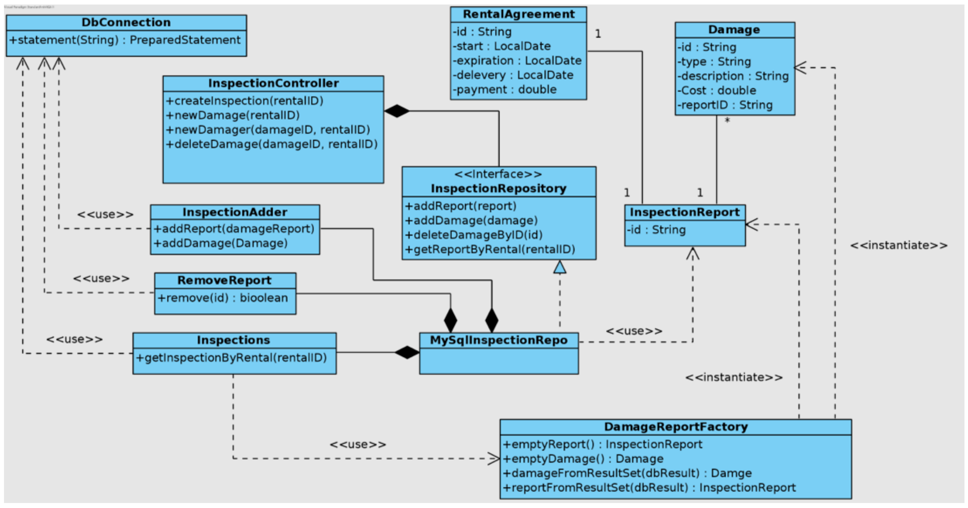


Figure 9: Class diagram of inspection context

State diagram

Author: Nikki Deleuran

We find a state diagram useful to describe the behavior and flow of our system. Figure 10 describes the overall flow of our solution:

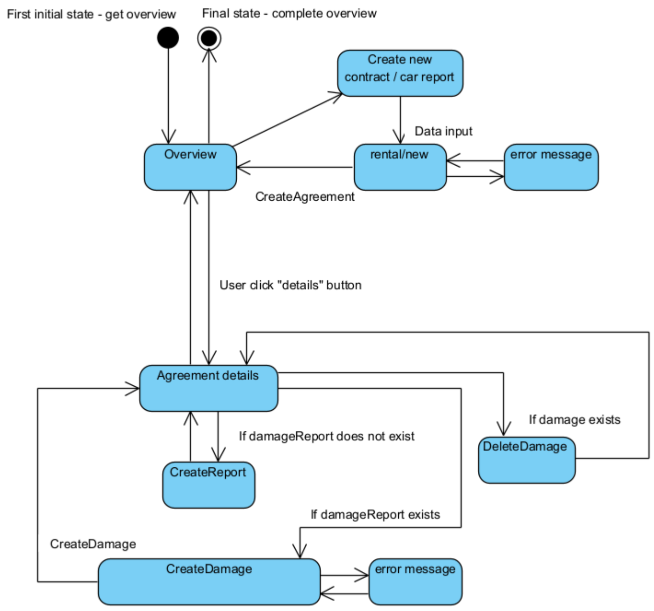


Figure 10: State diagram

The first state is the overview as shown in Figure 10. In this state the user might decide wether to register an agreement or get a more detailed view of a single agreement, both of which we describes as states. The transition to one of these states is triggered by corresponding events, which means the user clicks a link or button.

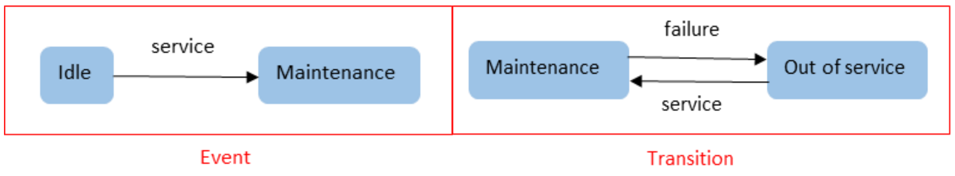


Figure 11: Event and transition

Backend

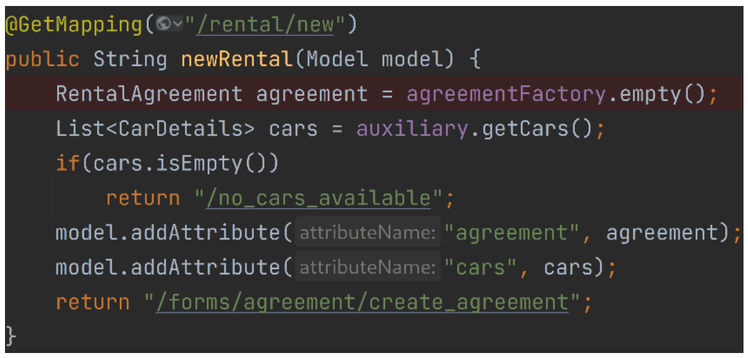
Code snippet: Add Agreement

Author: Martin Hansen

In this section we cover the process of persisting a rental agreement from the point where the user clicks the button and gets redirected to the overview page or to an error page. We will make use of snippets and describe the various layers we will encounter in this process.

The Process

Our solution is designed such that, the UI layer invokes "*/rental/new*" endpoint as the first endpoint. This endpoint is responsible for parsing back data to be filled in by the end-user. This is done with the help of RentalAgreementFactory, which instantiates an object of type *RentalAgreement* with default values, and *CarCustomerRepository* to retrieve a list of Car objects, as shown in the snippet below:

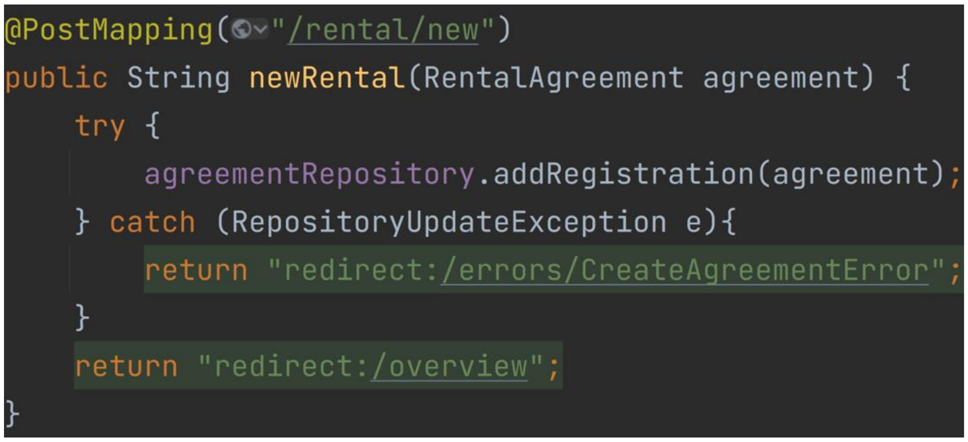


```
@GetMapping("/rental/new")
public String newRental(Model model) {
    RentalAgreement agreement = agreementFactory.empty();
    List<CarDetails> cars = auxiliary.getCars();
    if(cars.isEmpty())
        return "/no_cars_available";
    model.addAttribute( attributeName: "agreement", agreement);
    model.addAttribute( attributeName: "cars", cars);
    return "/forms/agreement/create_agreement";
}
```

Figure 12: Controller layer GET

It is worth mentioning the condition that checks if the list is empty. This is done to ensure that an end-user us unable to enter the form if no available car exists, In that case, the end-user is routed to a page that informs the end-user that no cars are available. Otherwise, routed to a form page.

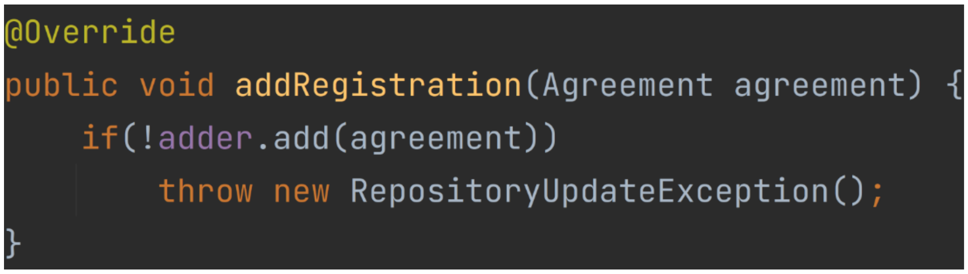
The form page parses back an updated instance of *RentalAgreement*:



```java
@PostMapping("/rental/new")
public String newRental(RentalAgreement agreement) {
    try {
        agreementRepository.addRegistration(agreement);
    } catch (RepositoryUpdateException e){
        return "redirect:/errors/CreateAgreementError";
    }
    return "redirect:/overview";
}
```

Figure 13: Controller layer POST

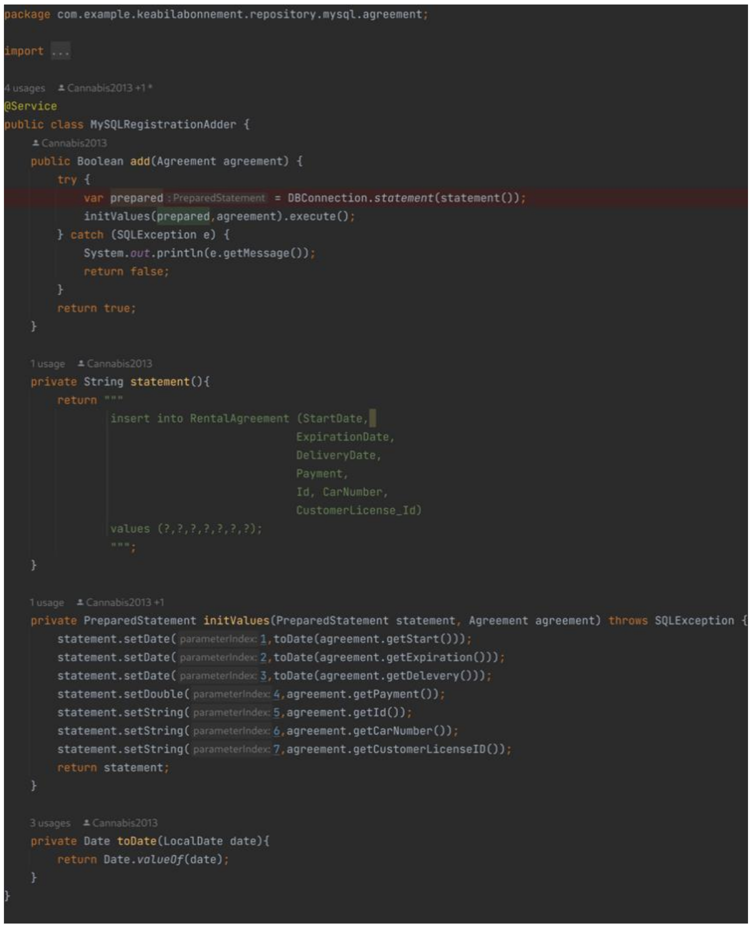
As seen in the snippet above, our controller implementation parses the instance of RentalAgreement to the MySQLAgreementRepository. The repository throws an instance of *RepositoryUpdateException* if anything gets wrong. If this is the case, the implementation redirects to an error page that indicates the operation did not go as expected; otherwise, it will redirect to the overview page. Let us dive into the *addRegistration* method of *MySQLAgreementRepository*:



```java
@Override
public void addRegistration(Agreement agreement) {
    if(!adder.add(agreement))
        throw new RepositoryUpdateException();
}
```

Figure 14: Service layer - Add method

MySQLAgreementRepository calls one of its service classes *MySQLRegistrationAdder* to persist the Agreement object to the persistence layer and throws an exception if fails. Let us look at the *AgreeementPersistence*[i] class where the domain logic exists:



```
package com.example.keabilabonnement.repository.mysql.agreement;

import ...

4 usages  Cannabis2013 +1 *
@Service
public class MySQLRegistrationAdder {
    Cannabis2013
    public Boolean add(Agreement agreement) {
        try {
            var prepared :PreparedStatement = DBConnection.statement(statement());
            initValues(prepared,agreement).execute();
        } catch (SQLException e) {
            System.out.println(e.getMessage());
            return false;
        }
        return true;
    }

    1 usage  Cannabis2013
    private String statement(){
        return """
                insert into RentalAgreement (StartDate,
                                            ExpirationDate,
                                            DeliveryDate,
                                            Payment,
                                            Id, CarNumber,
                                            CustomerLicense_Id)
                values (?,?,?,?,?,?,?);
                """;
    }

    1 usage  Cannabis2013 +1
    private PreparedStatement initValues(PreparedStatement statement, Agreement agreement) throws SQLException {
        statement.setDate( parameterIndex: 1,toDate(agreement.getStart()));
        statement.setDate( parameterIndex: 2,toDate(agreement.getExpiration()));
        statement.setDate( parameterIndex: 3,toDate(agreement.getDelevery()));
        statement.setDouble( parameterIndex: 4,agreement.getPayment());
        statement.setString( parameterIndex: 5,agreement.getId());
        statement.setString( parameterIndex: 6,agreement.getCarNumber());
        statement.setString( parameterIndex: 7,agreement.getCustomerLicenseID());
        return statement;
    }

    3 usages  Cannabis2013
    private Date toDate(LocalDate date){
        return Date.valueOf(date);
    }
}
```

Figure 15: MySQLRegistrationsAdder (now AgreementPersistence)

This part involves four steps:

1. The creation of an insert SQL statement
2. The creation of a *PreparedStatement* to be executed
3. A call to a method that inserts the user entered values to the relevant places in the SQL Statement

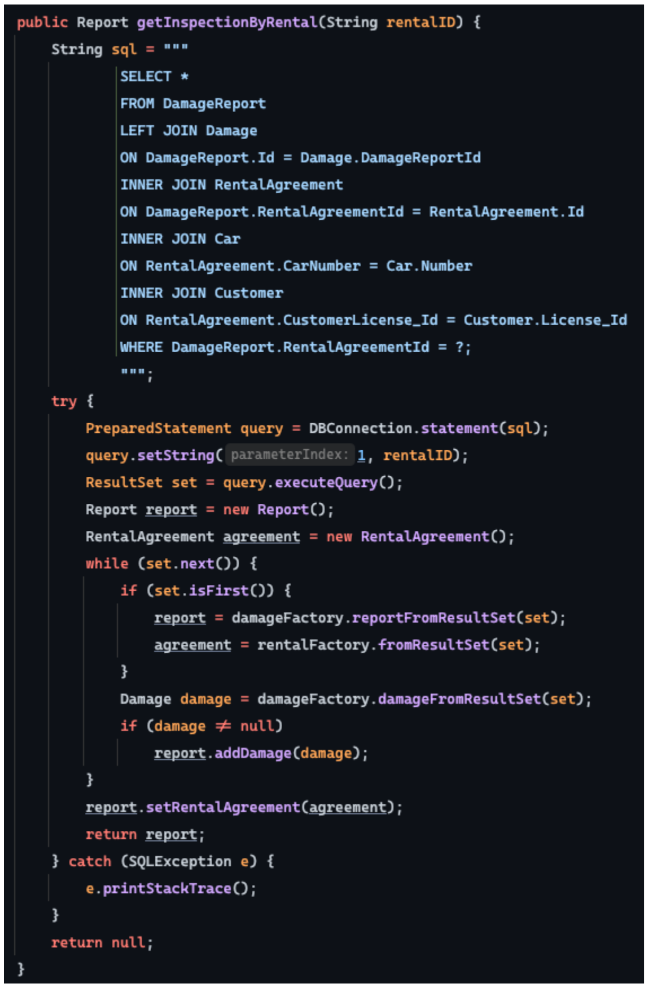
[i] The one presented in figure 14 is from an earlier commit as noted in the caption.

4. The execution of the statement

Returns true if successful otherwise, prints a message from *SQLException* and returns false.

Code snippet: getInspectionByRental

Author: Stefan Andreas Jensen



```java
public Report getInspectionByRental(String rentalID) {
    String sql = """
            SELECT *
            FROM DamageReport
            LEFT JOIN Damage
            ON DamageReport.Id = Damage.DamageReportId
            INNER JOIN RentalAgreement
            ON DamageReport.RentalAgreementId = RentalAgreement.Id
            INNER JOIN Car
            ON RentalAgreement.CarNumber = Car.Number
            INNER JOIN Customer
            ON RentalAgreement.CustomerLicense_Id = Customer.License_Id
            WHERE DamageReport.RentalAgreementId = ?;
            """;
    try {
        PreparedStatement query = DBConnection.statement(sql);
        query.setString( parameterIndex: 1, rentalID);
        ResultSet set = query.executeQuery();
        Report report = new Report();
        RentalAgreement agreement = new RentalAgreement();
        while (set.next()) {
            if (set.isFirst()) {
                report = damageFactory.reportFromResultSet(set);
                agreement = rentalFactory.fromResultSet(set);
            }
            Damage damage = damageFactory.damageFromResultSet(set);
            if (damage ≠ null)
                report.addDamage(damage);
        }
        report.setRentalAgreement(agreement);
        return report;
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return null;
}
```

Figure 16: MySQLInspections

Our method getInspectionByRental returns a report object and it takes a rentalId of datatype string as parameter. We first create our SQL string which consists of a lot of joins on tables. Damage contains a reference to a DamageReport through the id of the report. The relationship between these two tables is a 1-to-many relationship as one DamageReport can have many Damages associated but a Damage can only have one DamageReport associated. DamageReport contains a reference to RentalAgreement through its id. This is a 1-to-1 relationship. The same is the case for RentalAgreement as it contains a reference to both Customer and Car through their respective ids.

To join the above tables, we primarily use INNER JOIN, except when we join Damage to DamageReport as a report does not necessarily have damages associated. LEFT JOINS leave empty columns in place whereas an INNER JOIN would leave the missing Damage columns out of the resultset. A LEFT JOIN is therefore the desired solution here.

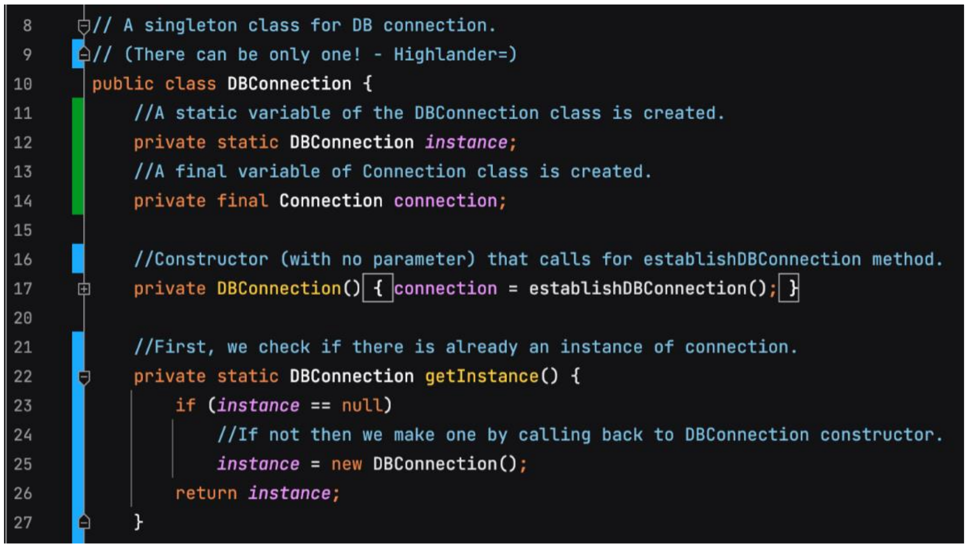
Lastly, we specify that we only want rows where the RentalAgreement reference stored in the DamageReport is equal to an input value, which in this case would be our rentalId parameter.

The 1-to-many relationship between DamageReport and Damage dictates the number of rows we will be returned. As a row will always contain information about the same DamageReport, RentalAgreement, Car and Customer, we extract the DamageReport and RentalAgreement - Car and Customer through RentalAgreement - information using a factory, which returns an object of the respective datatype, only once at the first value in the ResultSet. These objects can then be stored for later use. For a row, we check if a damage object returned from our factory, which oversees building a damage from a ResultSet, is null. If so, we can confirm the DamageReport did not indeed have any Damages associated, if not we add the damage to the report, which we extracted earlier in the process. Lastly, as we have finished reading the ResultSet, we can store the RentalAgreement object in the Report object and return Report. This entire process, except for the SQL String, is encapsulated in a try catch. If an SQLException is caught from the database querying, we print the error, and a null value will at that point be returned for the method.

Code Snippet: DBConnection – Singleton Pattern

Author: M. Kaan Arici

Our Database Connection method is created as a singleton pattern[30]. A singleton is a design pattern that only allows the instantiation of a class to a singular instance. This pattern is often used in situations where it is necessary to share a single instance of a class across multiple objects. In that respect, it made sense to establish DB connection as a singleton since all the repositories throughout the application would be able to access it easily without having to recreate it or losing the information inside it.

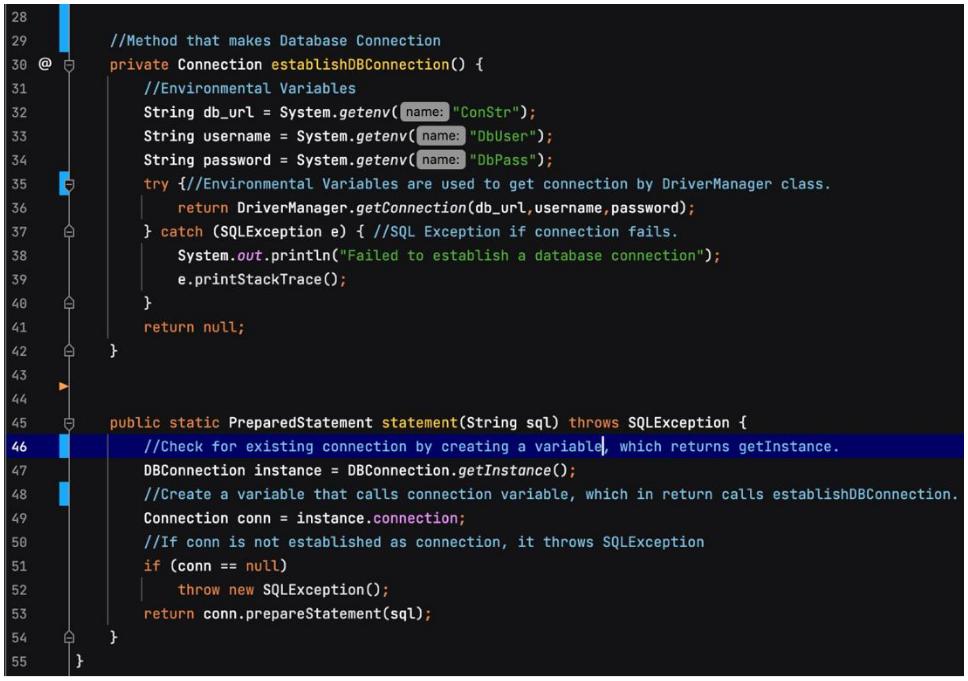


```java
// A singleton class for DB connection.
// (There can be only one! - Highlander=)
public class DBConnection {
    //A static variable of the DBConnection class is created.
    private static DBConnection instance;
    //A final variable of Connection class is created.
    private final Connection connection;

    //Constructor (with no parameter) that calls for establishDBConnection method.
    private DBConnection() { connection = establishDBConnection(); }

    //First, we check if there is already an instance of connection.
    private static DBConnection getInstance() {
        if (instance == null)
            //If not then we make one by calling back to DBConnection constructor.
            instance = new DBConnection();
        return instance;
    }
```

Figure 17: DbConnection

In DBConnection singleton class, the two private variables are created. *DBConnection instance* is a static variable and *Connection connection* is final. The constructor *DBConnection()* that takes no parameters is also private and it sets up the final *connection* variable to return the private e*stablishDBConnection()* method.

Following the constructor, the *getInstance()* method first makes sure if there is not already an instance of connection running. If the instance is null then it calls back to the constructor, which would again return the private method of e*stablishDBConnection()*.



```java

    //Method that makes Database Connection
    private Connection establishDBConnection() {
        //Environmental Variables
        String db_url = System.getenv( name: "ConStr");
        String username = System.getenv( name: "DbUser");
        String password = System.getenv( name: "DbPass");
        try {//Environmental Variables are used to get connection by DriverManager class.
            return DriverManager.getConnection(db_url,username,password);
        } catch (SQLException e) { //SQL Exception if connection fails.
            System.out.println("Failed to establish a database connection");
            e.printStackTrace();
        }
        return null;
    }


    public static PreparedStatement statement(String sql) throws SQLException {
        //Check for existing connection by creating a variable, which returns getInstance.
        DBConnection instance = DBConnection.getInstance();
        //Create a variable that calls connection variable, which in return calls establishDBConnection.
        Connection conn = instance.connection;
        //If conn is not established as connection, it throws SQLException
        if (conn == null)
            throw new SQLException();
        return conn.prepareStatement(sql);
    }
}
```

Figure 18: establishDbConnection() method

e*stablishDBConnection()* method is the private function that sets up the connection by first defining the environmental variables and then passing them into the *DriverManager.getConnection()* method's parameters within a *try catch statement* with *SQLException* if the connection fails to establish.

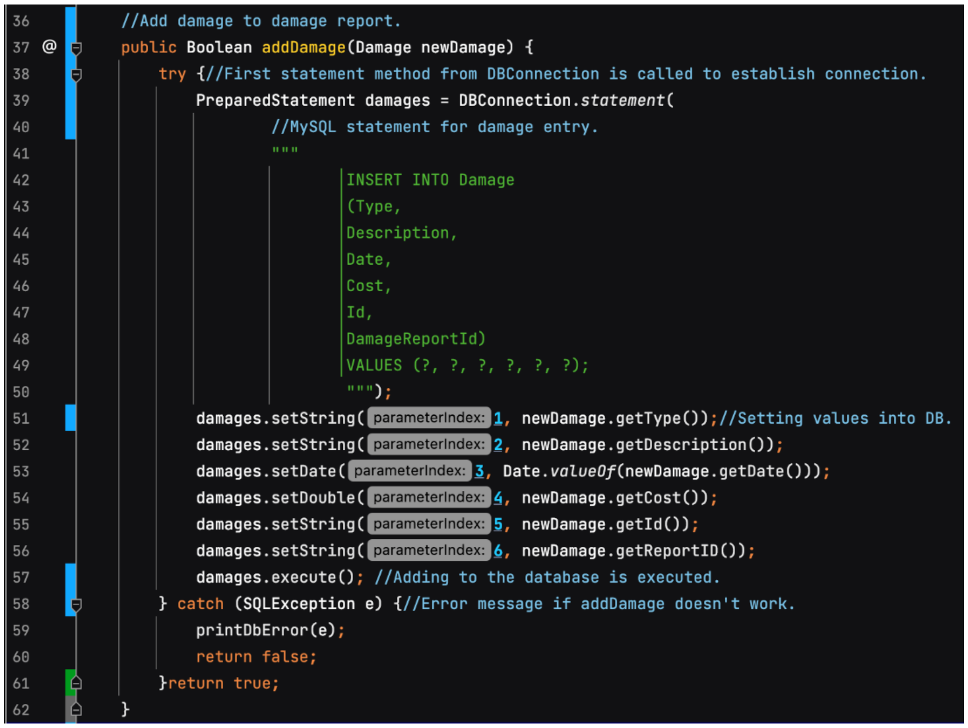
Once the private and static connection is set up within the class, a public static *PreparedStatement* method is created to be used by the repositories.

This method, which is called *statement*, first checks if there is a connection by calling *getInstance()* and if there is none then calls the connection in the constructor, which again calls *establishDBConnection* to finalize connection. Yet if the connection fails, it is again set up to throw an *SQLException*.

Code Snippet: Inspection – Add Damage

Author: M. Kaan Arici

MySQLInspectionAdder class under *MySQLInspectionsRepo* is responsible to add damages to an existing damage report with and Id. Wrapped in a *try catch statement*, the Boolean method, which requires the *Damage* class from models as parameter, first establishes DB connection via a new *PreparedStatement* variable that calls the public *statement* method from *DBConnection* class. The *statement* method requires a String parameter, in which a MySQL code statement for damage entry is passed in with all the required values.



```
//Add damage to damage report.
public Boolean addDamage(Damage newDamage) {
    try {//First statement method from DBConnection is called to establish connection.
        PreparedStatement damages = DBConnection.statement(
                //MySQL statement for damage entry.
                """
                        INSERT INTO Damage
                        (Type,
                        Description,
                        Date,
                        Cost,
                        Id,
                        DamageReportId)
                        VALUES (?, ?, ?, ?, ?, ?);
                        """);
        damages.setString( parameterIndex: 1, newDamage.getType());//Setting values into DB.
        damages.setString( parameterIndex: 2, newDamage.getDescription());
        damages.setDate( parameterIndex: 3, Date.valueOf(newDamage.getDate()));
        damages.setDouble( parameterIndex: 4, newDamage.getCost());
        damages.setString( parameterIndex: 5, newDamage.getId());
        damages.setString( parameterIndex: 6, newDamage.getReportID());
        damages.execute(); //Adding to the database is executed.
    } catch (SQLException e) {//Error message if addDamage doesn't work.
        printDbError(e);
        return false;
    }return true;
}
```

Figure 19: Add damage

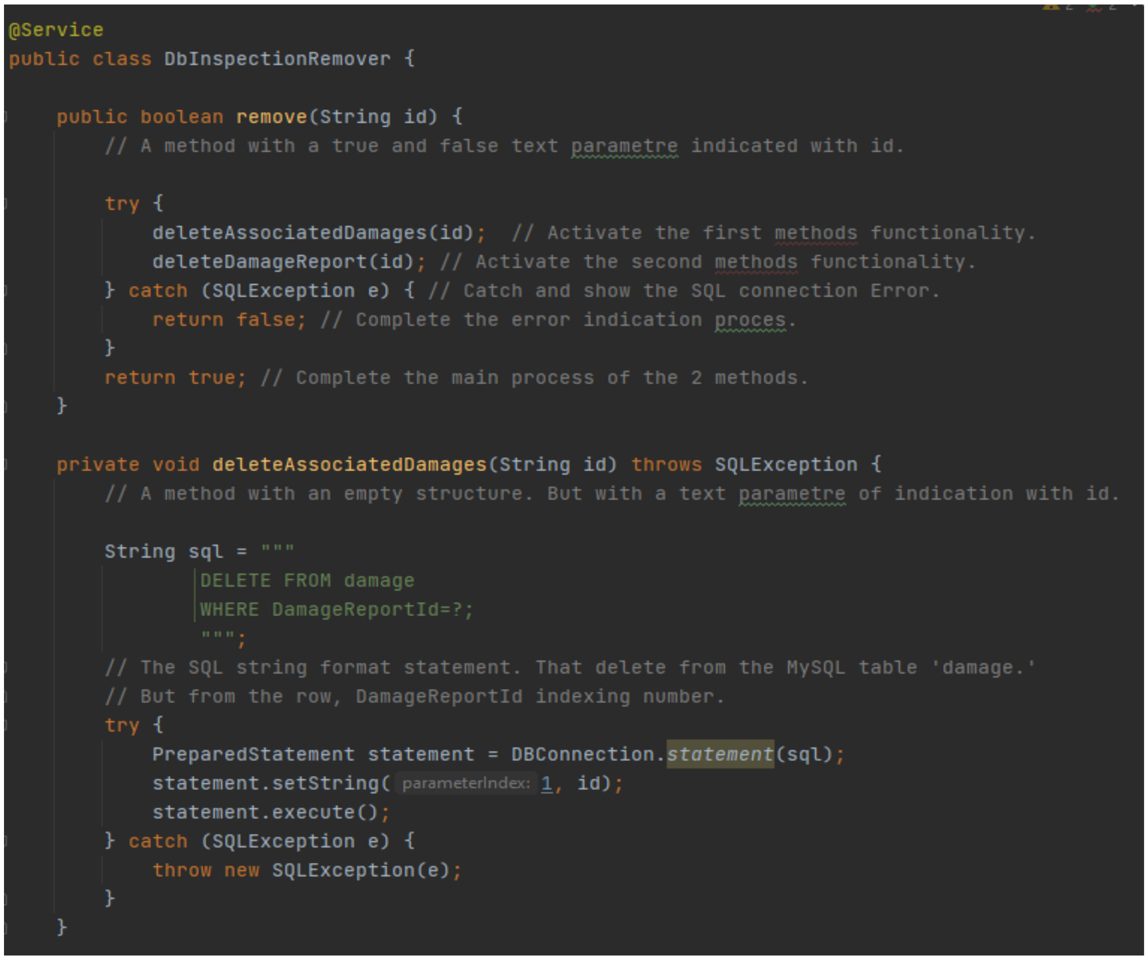
Once the statement passed in, the required values are set into the *damages* variable that was previously initialized by calling the relevant *setters* & *getters* from the model Damage class. Then the *execute* method is called to finalize the addition of the damage to the DB. Finally, *SQLException* is added under catch in the case the addition fails.

Code Snippet: DbInspectionRemover

Author: Nikki Deleuran

This java class removes every damage data with SQL statements.

It is the backend system, for creating a remove button in the dashboard.



```java
@Service
public class DbInspectionRemover {

    public boolean remove(String id) {
        // A method with a true and false text parametre indicated with id.

        try {
            deleteAssociatedDamages(id);  // Activate the first methods functionality.
            deleteDamageReport(id); // Activate the second methods functionality.
        } catch (SQLException e) { // Catch and show the SQL connection Error.
            return false; // Complete the error indication proces.
        }
        return true; // Complete the main process of the 2 methods.
    }

    private void deleteAssociatedDamages(String id) throws SQLException {
        // A method with an empty structure. But with a text parametre of indication with id.

        String sql = """
                DELETE FROM damage
                WHERE DamageReportId=?;
                """;
        // The SQL string format statement. That delete from the MySQL table 'damage.'
        // But from the row, DamageReportId indexing number.
        try {
            PreparedStatement statement = DBConnection.statement(sql);
            statement.setString( parameterIndex: 1, id);
            statement.execute();
        } catch (SQLException e) {
            throw new SQLException(e);
        }
    }
```

Figure 20: Remove damage

public Boolean remove (String id):

It is the method that wraps and activates the 2 other methods. It is a Boolean method. This lets it return with a true and false. With a parameter of a text (String) called id. Try/Catch block is used for SQL exception handling[31] in Java by catching the sql errors[32]. The deleteAssociatedDamages(id) and deleteDamageReport(id) methods are called with the String parameter id as input. If no SQLException's are caught, then the code proceeds and true is returned. If an exception is caught, false is instead returned.

public void deleteAssociatedDamages(String id):
It is a method that has a String parameter of **id.**

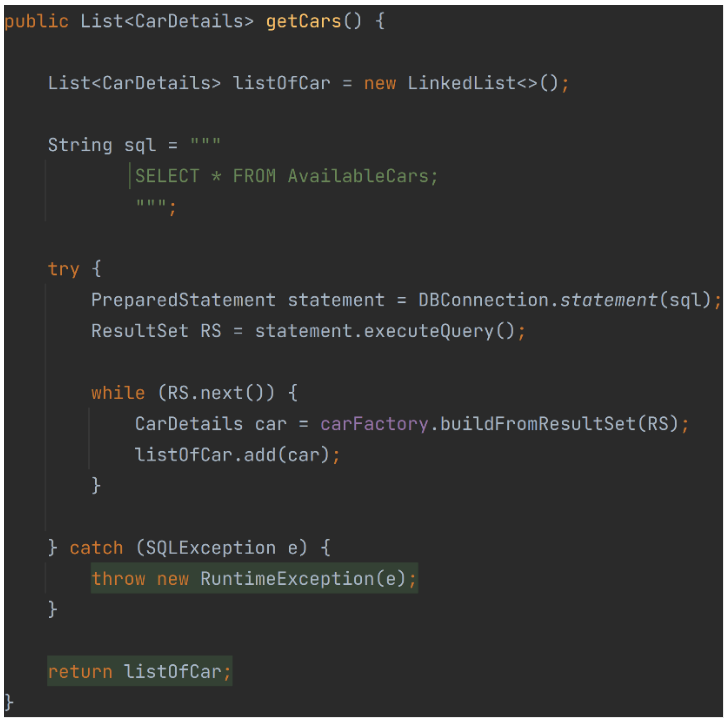
First the method defines a String variable called sql. This variable defines the sql code which will be parsed to the database. We only want to delete a specific row in the MySQL table named "damage" where a "DamageReportID" is equal to the id we specify further down in the code. We create a PreparedStatement as well as parse our sql variable to this. The PreparedStatement is a pre-compiled SQL query of our sql variable[33]. During compilation, the PreparedStatement inserts the method parameter id into a specified slot in the SQL code, which is annotated with a question mark. Finally, our statement is executed.

public void deleteDamageReport(String id):
This method is the same, except the sql variable specifies that a damage report with a specific id should be deleted instead.

Code snippet: getCars

Author: Nikki Deleuran

The goal is to retrieve all data associated with the tables Car in the MySQL database (DBMS) as shown in the following snippet:



```java
public List<CarDetails> getCars() {

    List<CarDetails> listOfCar = new LinkedList<>();

    String sql = """
            SELECT * FROM AvailableCars;
            """;

    try {
        PreparedStatement statement = DBConnection.statement(sql);
        ResultSet RS = statement.executeQuery();

        while (RS.next()) {
            CarDetails car = carFactory.buildFromResultSet(RS);
            listOfCar.add(car);
        }

    } catch (SQLException e) {
        throw new RuntimeException(e);
    }

    return listOfCar;
}
```

Figure 21: Get cars from database

public *List<CarDetails> getCars*():

The method returns a List of CarDetails, which contains every detail about cars pulled from the MySQL database. The goal is to return a list of CarDetails, and we therefore create a new LinkedList, in which we can store the data pulled from the database[34]. This list, which we call listOfCar, is created as the first thing in the method.

This time in the String SQL format we want to retrieve everything from our MySQL DBMS. Everything (* asterisk) from the table, called ‘Car’. Statement: SELECT * FROM Car.

The ResultSet (RS) retrieves a table of data generated by executing database queries[35]. Performs an **executeQuery()** on any object implementing by the PreparedStatement.

We loop through the RS whilst there are still values left to read. For each iteration we construct a CarDetails object using our factory carFactory, which takes in the result set. This object is then added to the LinkedList listOfCar. If no SQLException is encountered during this process, we then at last return the listOfCar, otherwise a runtime exception is thrown.

Frontend

Fragments

Author: Stefan Andreas Jensen

Fragments are a great way to componentize code. They allow for reuse of code as well as high modularity. A perfect example for the effectiveness of fragments could be a navbar on a website. The navbar is going to be the same on every page of the site. Traditionally we would have to paste the same code on each of the pages utilizing the navbar. Instead, we create one single fragment component containing the navbar code. We can insert this fragment on each of the pages with a navbar. Now, if we want to make a change to the navbar, we simply only need to change the fragment component containing the actual navbar code. Traditionally we would have had to change the value on every single page with a navbar[ii].

Fragments also allow for parsing of data. When creating a fragment, you can specify what parameters you want parsed to it when it is inserted on a page[36]. This allowed us to parse say an object from a Thymeleaf for-each-loop into the fragment, which would then oversee displaying the data of the object.

By using fragments, we can split a page up into modules. We can have a module for say a navbar, a table, a footer etc. Each module can therefore focus on their own task without having to rely on other parts of the frontend.

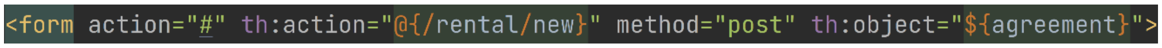
We made heavy use of fragments throughout our product. Not only did they easily allow us to split front-end tasks between us without being bound by one another to be finished with tasks, but they also made for cleaner code[iii]. A collection of cards could be broken down into card fragments each also consisting of a card-title fragment. Forms could be broken down into form input fragments and form button fragments. The frontend of our product therefore took shape of a layered architecture in some way. Once someone was done with their assigned module, they simply had to insert it onto the specific page for it to be part of the design.

[ii] See appendix under “fragmens : navbar”
[iii] See appendix under ”fragments : modularity and clean code”

Forms

Author: Martin Hansen

Thymeleaf primarily parses data through the *form* tag which takes some parameters like *th:action*, *method* and *th:object* as shown the below figure:



```
<form action="#" th:action="@{/rental/new}" method="post" th:object="${agreement}">
```

Figure 11: Thymeleaf form element

A form captures a submit event fired by an input element of type *submit* and parses the data to the given endpoint specified in *th:action*. Thymeleaf usually takes an object it receives from Spring that acts as a container, and this object is set by *th:object*, which in our example is an object of type *RentalAgreement*. Each input elements have a tag attribute *name* assigned to it which helps Thymeleaf identify which fields of the object to update. Finally, when a user pushes the submit button a submit event is fired hand handled by Thymeleaf, which will parse the updated model back to the control of Spring.

Conclusion

We think that our solution complies with the minimal end-product requirements as provided by Bilabonnement.dk. We found that the project was feasible as shown in our feasibility study and the project was likely to be delivered as scheduled due to the size of the project. Yes, it is an MVP, and as such it still lacks features to provide a fully-fledged user experience, but with the utilization of GRASP we made our solution refactor friendly and prepared for future iterations. That means our software complies with principles like Single Responsibility, low-coupling, high-cohesion, and other high valued GRASP/SOLID principles. It is important to consider that we as a team invest a lot of time and resources in 80% of the core domain to assure a certain level of quality, and then deal with the rest in future iterations. Although we did not utilize the full potential of MySQL features, we feel that the database works as expected and utilization of the features is not hard to implement as we overcame the issues we encountered.

As a customer you always want a system in place as quickly as possible and it should contain all the features you ordered. As much as we as a team feel sympathy for this view, we also want to provide a system that is stable and provides as much customer value as possible without too much unforeseen behavior due to a focus on too many features. Features vary in size, and as such it can be a time-consuming process, and we must at this stage keep our focus solely on the domain that holds value: the core domain.

User Manual

Prerequisites

- IntelliJ
- Java Version 17 (Only tested version)
- Basic Git knowledge
- An installed MySQL driver

Start by cloning the project from GitHub using Git from the following link https://github.com/Cannabis2013/KEABilAbonnementDK. Open and load the project in IntelliJ.

Connecting IntelliJ to the Azure database

This guide is based on use within IntelliJ and thus the same guide might not work for other programs.

To connect to the Azure hosted database: In IntelliJ spot the "Database" tab on the right side of your screen next to the scrollbar. Click the tab. Click the add button "new", select "Data Source" and finally select "MySQL". This option might not be visible for everyone at first but should be otherwise found under "other" at the bottom of the list, if not visible at first. Insert the following information in the respective inputs:

User	mh
Password	PS&E-IT-Solutions
URL	jdbc:mysql://mhserver.mysql.database.azure.com:3306

Table 6: DBMS credentials

Locate the SSH/SSL tab at the middle top, click the tab and enable "Use SSL". Head back to the General tab. Make sure all drivers are installed, otherwise IntelliJ should let you know if you have any missing ones. Test the connection by clicking "Test Connection". If everything works as intended, you should not receive an error. You should now be able to view and see the table content of "bilabonnement" in the database within IntelliJ.

Connecting the application to the database

For the application to know how to connect to the database, we need to create a few environment variables in IntelliJ from which the application will pull the database credentials from. Click “KeaBilAbonnementApplication” on the top bar next to the run button. Select “Edit Configurations”. Click the add button and select “Spring Boot”. Click “Edit environment variables – a button inside of the “Environment variables” input field. Add the following as environment variables:

Name	Value
ConStr	jdbc:mysql://mhserver.mysql.database.azure.com:3306/bilabonnement?useSSL=true
DbUser	mh
DbPass	PS&E-IT-Solutions

Table 7: Azure DBMS - Environmental variables

You should now be able to run the application by starting it using the green “run” button in the top right corner of IntelliJ.

Connecting the Application to a Local Database Instance

Start by creating a local database instance with a set username and password. From here you should be able to follow the “Connecting the application to the database” guide. However, use the following environment variable instead and replace **<username>** and **<password>** with your own local database values:

Name	Value
ConStr	**For Windows and Linux users:** jdbc:mysql://localhost:3306/bilabonnement?user=**<username>**&password=**<password>** **For Mac users:** jdbc:mysql://localhost:3306/bilabonnement?user=**<username>**&password=**<password>**&

	&sessionVariables=sql_mode='STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION,PIPES_AS_CONCAT'

Table 8: Local DBMS - Environmental variables

Connecting your local database to IntelliJ is optional as it is assumed you already have a way to view the content of the database through your database program.

Lastly, follow the “SQL scripts” guide to set up the database for use and populate it with dummy data.

You should now be able to run the application by starting it using the green “run” button in the top right corner of IntelliJ.

SQL Scripts

If for some reason the azure database is empty or you have decided to host a local database, you can use our SQL scripts to reset the database and instantiate it with dummy data. Localize the “sql” folder at the root of the project. Start a new query console in your selected database program. Run the init.sql script first followed by the views.sql script and lastly the populate.sql script.

Appendix

User stories

Data Registration

As an employee, I can input personal details about the customer, so that it is labelled who the company is entering an agreement with.

- Clearly labelled which data is required.
- Employee is helped throughout the process (system auto-completes)
- The form is up to date with the latest leasing rules.

As an employee, I can input details about the cars, so that it is labelled which car the customer is leasing.

- Clearly labelled which data is required.
- Employee is helped throughout the process (system auto-completes)
- The form is up to date with the latest leasing rules.

As an employee, I can input contract specific details like start date and ending date, so that every party knows when the contract is valid.

- Clearly labelled which data is required.
- Employee is helped throughout the process (system auto-completes)
- The form is up to date with the latest leasing rules.

Damage Assessment

As an employee, I can register the type of damage, so that the type of damage can be displayed on the damage report.

- Clearly labelled which data is required.
- Employee is helped throughout the process (system auto-completes)

As an employee, I can register the cost associated with a type of damage, so that the cost can be displayed on the damage report.

- Clearly labelled which data is required.
- Employee is helped throughout the process (system auto-completes)

Business Developers

As an employee, I can view a list of currently leased cars, so that I can gain an overview of the current car assortment.

- Updates every 24 hours.
- Easy to navigate.

As an employee, I can sort the list of currently leased cars by brand, so that I can better understand the popularity of the available brands.

- Updates every 24 hours.
- Easy to navigate.

As an employee, I can sort the list of currently leased cars by leaser type, so that I can better understand the type of customer.

- Updates every 24 hours.
- Easy to navigate.

As an employee, I can view the current total price of all currently leased cars in different intervals, so that I can better understand the exposure of the company according to the intervals.

- Updates every 24 hours.
- Easy to navigate.

Risk analysis

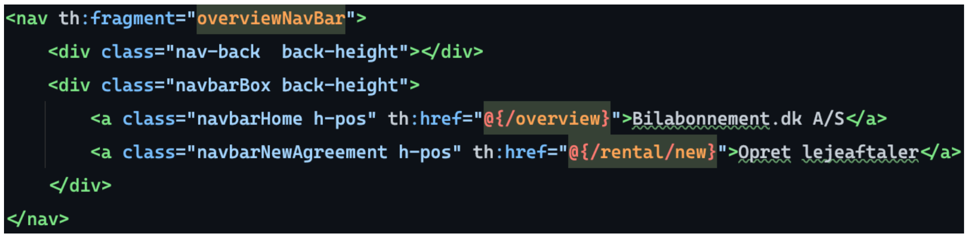
Risks							
Risks	Probability	Consequence	Product	Precautions	Responsibility	Proposed solution	Responsibility
Developers/team							
Planned leave of absence	5	2	10	Walk through everybody's schedule before the project starts.	Project leader	Discuss and plan a split of the absent member's tasks between the remaining members.	Project leader and project members
Illness	3	3	9	Discuss an eventual plan based on the severity of the illness	Project leader	Discuss and plan a split of the ill member's tasks between the remaining members.	Project leader and project members
School drop outs	1	3	3	Be transparent with each other with regard to mental state and supportive.	Project leader	Take and redistribute the work of the student that left	Project leader and project members
Individual(s) misbehavior	1	4	4	Setup group agreement that involves rules of behavior	Project leader and project members	Try and solve it within the group at first. If no success, then involve the boss	Project leader
Organization							
Epidemic outbreak (Corona)	1	1	1	Get vaccinated	Project members	Stay home	Project members
Estimation							
Too short sprint intervals	2	2	4	Make emphasis on Scrum retrospective regarding sprint intervals	Project leader	Adjust sprints accordingly and perform continuously evaluations	Project leader
End product							
Bad customer satisfaction	2	4	8	Make a thorough review of the end-product requirements. Perhaps work out a fallback agreement to execute in case of bad customer satisfaction.	Project leader and customer	Execute fall-back agreement	Project leader and customer
Product specification							
End product requirements change	2	3	6	Use of GRASP/SOLID principles under the development phase. Conform strictly to agile methods.	Customer and project leader	A new meeting to resolve eventual conflicts	Customer and project leader

Bad requirements understanding	2	4	8	Customer: Formalize a clear and comprehensible product requirement document. Team: Have a mutual understanding of the end goal based upon discussion as a team Team: Have a mutual understanding of the end goal based upon discussion as a team	Customer, project leader and project members	Categorize and split requirements into enumerated terms	Project leader
Technology							
Network failure	2	4	8	Short git commit intervals. Make emergency plans like prewritten messages that we can send with ravens.	Project members	Relocate to a different location during the working hours and commit changes often	Project leader and project member
Computer failure	2	2	4	Use online tools and services which saves to the cloud during the project	Project members	Try troubleshooting. If fails, then try various data recovery options. In worst case scenario, this needs urgent priority, and group members must try to recreate the lost data.	Project leader and project member

Table 9: Risk table

Fragments

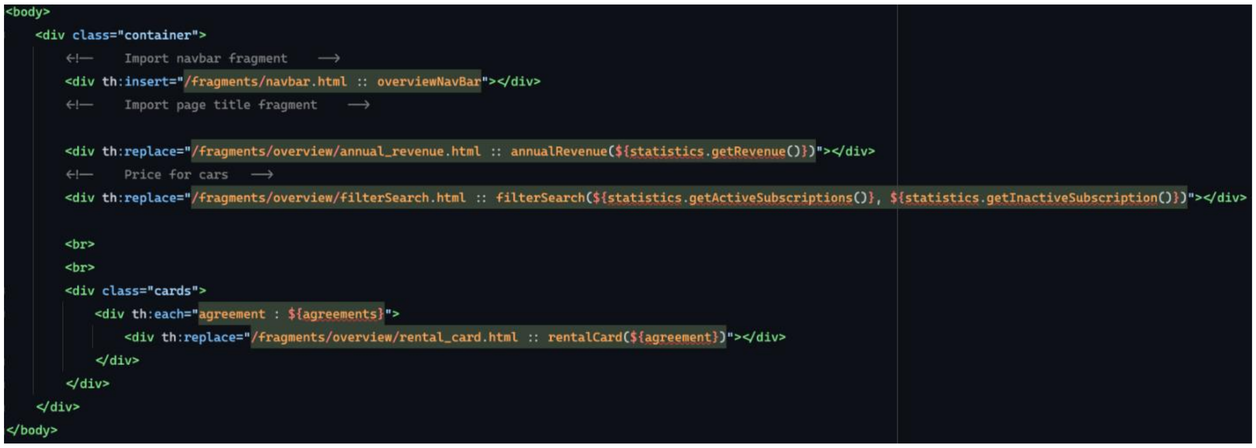
Navbar fragment



```html
<nav th:fragment="overviewNavBar">
    <div class="nav-back  back-height"></div>
    <div class="navbarBox back-height">
        <a class="navbarHome h-pos" th:href="@{/overview}">Bilabonnement.dk A/S</a>
        <a class="navbarNewAgreement h-pos" th:href="@{/rental/new}">Opret lejeaftaler</a>
    </div>
</nav>
```

Figure 12: Fragment example

Modularity and clean code



```html
<body>
    <div class="container">
        <!--    Import navbar fragment    -->
        <div th:insert="/fragments/navbar.html :: overviewNavBar"></div>
        <!--    Import page title fragment    -->

        <div th:replace="/fragments/overview/annual_revenue.html :: annualRevenue(${statistics.getRevenue()})"></div>
        <!--    Price for cars    -->
        <div th:replace="/fragments/overview/filterSearch.html :: filterSearch(${statistics.getActiveSubscriptions()}, ${statistics.getInactiveSubscription()})"></div>

        <br>
        <br>
        <div class="cards">
            <div th:each="agreement : ${agreements}">
                <div th:replace="/fragments/overview/rental_card.html :: rentalCard(${agreement})"></div>
            </div>
        </div>
    </div>
</body>
```

Figure 13: Modularity and clean code

Figures

Tables

Abbreviations

1NF	First Normal Form
2NF	Second Normal Form
3NF	Third Normal Form
CRUD	Create, Read, Update, Delete
DB	Database
DBMS	Distributed Database Management System
DM	Domain Model
ERD	Entity Relationship Diagram
GDPR	General Data Protection Regulation
GRASP	General Responsibility Assignment Software Patterns
IDE	Integrated Development Environment
INVEST	Independent Negotiable Valuable Estimable Small Testable
MVC	Model-View-Controller
MVP	Minimum Viable Product
RDBMS	Relation Distributed Database Management System
ROI	Return of Investment
SOLID	Single Responsibly principle, Open/Closed principle, Liskov substitution principle, Interface Segregation principle, Dependency Inversion principle
SQL	Structured Query Language
SSD	System Sequence Diagram
SWOT	Strengths, Weaknesses, Opportunities, Threats
UI	User Interface
UML	Unified Modeling Language
USD	US Dollar
UX	User Experience

Bibliography

Bilabonnement. *"Godt Udvalg Af Biler."* . 13. December 2022. Web site. 13. December 2022. <http://www.bilabonnement.dk/>.

EU. *"GDPR Archives." GDPR.eu,*. 12. December 2022. HTML document. 13. December 2022. <https://gdpr.eu/tag/gdpr/>.

Kirkegaard, Marcus. *Gestaltlovene: Vigtige Principper I Design Multimediedesigneren*. 6. December 2022. 13. December 2022. <https://multimediedesigneren.dk/gestaltlovene/>.

KWBruun, K.W. Bruun & Co.". *"History*. 12. December 2022. 13. December 2022. <http://www.kwbruun.com/en/history>.

Larman, Craig. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis & Design and the UNIFIELD Process*. Prentice Hall PTR, 2002.

Leffingwell, Dean and Pete Behrens. *A User Story Primer*. Leffingwell, LLC, 2009.

Millett, Scott og Nick Tune. *Patterns, Principles, Practices of domain-driven design*. John Wiley & Sons, Incorporated, 2015.

Peterson, Richard. *What Is Normalization in DBMS (SQL)? 1NF, 2NF, 3NF Example*. 10. December 2022. 13. December 2022. <https://www.guru99.com/database-normalization.html.>.

Shneiderman, Ben. *The Eight Golden Rules of Interface Design*. 13. December 2022. University of Maryland. 13. December 2022. <https://www.cs.umd.edu/users/ben/goldenrules.html>.

Study, QS. *Organizational Feasibility*. 23. November 2022. 13. December 2022. <qsstudy.com/organizational-feasibility>.

Trustpilot. *Bilabonnement Er Bedømt 'Fremragende' Med 4,4 / 5 På Trustpilot*. 13. December 2022. 13. December 2022. <https://dk.trustpilot.com/review/bilabonnement.dk. >.

Turban, Efraim. *Information Technology for Management.* . John Wiley & Sons, Incorporated, 2018.

Course material

Zwisler, Projekt og analyseredskaber. Forlaget Globe A/S 2005

Skriver, Hans Jørgen, et al. *Organisation*. Trojka / Gads Forlag, 2019.

Endnotes

[1] ("History — K.W. Bruun and Co")
[2] ("Bilabonnement Er Bedømt 'Fremragende' Med 4,3 / 5 På Trustpilot")
[3] https://en.wikipedia.org/wiki/Kanban_board, accessed 13/12/2022
[4] https://www.scrum.org/resources/scrum-framework-poster, accessed 14/12/2022
[5] (Turban et al., 2018, p.389)
[6] (Turban et al., 2018, p.389)
[7] (Turban et al., 2018, p.389)
[8] ("GDPR Archives - GDPR.eu")
[9] ("Organizational Feasibility").
[10] (Turban et al., 2018, p.390)
[11] (Skriver, 2019 & Zwisler, 2002)
[12] (Leffingwell, Dean et al., 2009,p. 7)
[13] (Leffingwell, Dean et al., 2009,p. 6)
[14] (Leffingwell, Dean et al., 2009,p. 4)
[15] (Millett et al., 2015, p. 35)
[16] https://en.wikipedia.org/wiki/MySQL, accessed 13/12/2022
[17] https://azure.microsoft.com/en-us/pricing/details/mysql/flexible-server/, price is for northern Europe region and for student accounts, accessed 13/12/2022
[18] https://www.guru99.com/database-normalization.html, accessed 13/12/2022
[19] (Larman, 2002)
[20] (Larman, p.221, 2002)
[21] (Larman, p.226, 2002)
[22] https://en.wikipedia.org/wiki/Factory_(object-oriented_programming , accessed 13/12/2022
[23] (Larman, 2022, p.237)
[24] (Larman, 2002, p.229)
[25] (Larman, 2022, p.232)
[26] https://www.digitalocean.com/community/tutorials/spring-annotations, accessed 13/12/2022
[27] https://www.usertesting.com/resources/topics/gestalt-principles#proximity, accessed: 7/12-2022
[28] https://multimediedesigneren.dk/gestaltlovene/, accessed: 7/12-2022
[29] https://www.cs.umd.edu/users/ben/goldenrules.html, accessed: 8/12-2022
[30] https://en.wikipedia.org/wiki/Singleton_pattern, accessed 13/12/2022
[31] https://developer.android.com/reference/java/sql/SQLException, 6/12.2022
[32] https://beginnersbook.com/2013/04/try-catch-in-java/, 6/12.2022
[33] https://www.geeksforgeeks.org/how-to-use-preparedstatement-in-java/, 6/12.2022
[34] https://www.w3schools.com/java/java_linkedlist.asp, date accessed 7/12.2022
[35] https://www.baeldung.com/jdbc-resultset, accessed 7/12/2022
[36] https://www.baeldung.com/spring-thymeleaf-fragments, accessed 13/12/2022