P3 project

Car mechanical website including a management system solution

Project Rapport

Group 2

Aalborg University

Computer science and software 2020

AALBORG UNIVERSITY
STUDENT REPORT

Title: Web development with a management system

Semester: 3. semester

Semester theme: A Well-structured Application

Project period: Fall 2021

ECTS: 15

Project group: SW3 Gruppe 2

Supervisor:
Daniel Russo

Participants:
Abdallah Ziad Al Naif
Beniamin Lobodziec
Lojain Ajek
Hijab Nazir
Oskar Kibsgaard Schöbel

Publisher: AAU
Pages: 63
Appendix: 62
Date of completion:
December 22th 2021

Abstract:

This project aims to develop a web-application for the Danish company ZN Autocenter. The web-application will consist of a redesign of their old website and an inventory management system, that will be developed and accessed through a login system that is integrated on the web-application. The system requirements are attained through conversation with ZN Autocenter, and are based on the needs they have expressed. Following, the results of this project will be highlighted with some extensive feedback from the company and a detailed discussion on further improvements will also be discussed.

0.1 Preface

This project aims to develop a program based on an object-oriented approach in cooperation with the customer. The project goal is to develop a website with a database for a ZN Autocenter. To fulfill these requirements of the project the front-end was made using web programming, and the back-end was made with the help of Java, where there is access from the website to the database.

The project process took place in the period between 10th September 2021 to the 22nd December, 2021, and is made by group 2 from 3rd semester of the bachelor Software on Aalborg University in Copenhagen. We want to thank Daniel Russo for his role as supervisor on this project.

Source references are indicated by the Vancouver method. Figures and pictures are numbered, in accordance with the chapters the appear in. Sources are presented in square brackets in following fashion; [1].
Link to GitHub:
`https://github.com/Delicatetheone2247/Autoshop-Website-with-an-integrated-management-inventory-sy`

Contents

Introduction

Many automobile repair branches in Denmark lack advanced knowledge about the software used by the company. This issue is not surprising since auto mechanics are not familiar with the concept of software development, and neither are they expected to be. Therefore, it is easy for "software developers" to overcharge the businesses for the initial development and ongoing maintenance. [2] This results is them being an easy target for such companies that are providing low effort, low-quality solutions while often not satisfying the needs of the particular company. [3] One of these automobile repair branches is ZN Autocenter ApS. They are a company founded in 2019 by the entrepreneur Ziad Al Naif. The company is located in the capital region of Denmark and currently employs five people, consisting of four mechanics and a secretary. The auto shop is capable of maintaining and repairing multiple vehicles at the same time. The workshop also offers the ability for customers to store their tires at the workshop; nonetheless, they also sell their own. Besides that, the company provides a loaner car to their customers while their vehicle is getting repaired.

As mentioned above, ZN Autocenter ApS is one of the many Automobile repair branches that many companies have approached regarding website development. However, these offers have been declined due to their previous similar experience with a company called "Seek4Cars A/S", which have developed their current website that does not fulfill the actual needs of the company and remains an unfinished product.

ZN Autocenter has declined these offers based on their previous experiences with a company called "Seek4Cars A/S", which is responsible for developing their current website. A website that lacks essential features, such as customer tire storage management, and does not fulfill the company's actual needs.
They wished for a solution that would decrease the number of daily tasks by digitalizing them partially. For example, as mentioned above, the company has a few shelves for storage of tires and rims and a couple of literal tire piles, which are not being tracked. This results in some issues regarding a general overview and status of their inventory.

Therefore there is still a growing desire for a functional inventory management system so that the staff would have better software resources for managing their daily tasks. The company currently handles inventory management by employing third-party software to manage loaner cars and the company website, which was initially intended to handle rental storage for customers, though that feature was never developed.

We intend to develop a new website for ZN Autocenter and a database for managing the inventory of their tires and rims. The database will be accessible by employees through an integrated login system on the new company website. How could a software-based application system be developed and implemented so that it could effectively help solve the daily tasks in compliance with the eventual guidelines of the Automobile repair branch?

Problem analysis

2.1 Pre-analysis

To develop an efficient inventory solution for ZN Autocenter, it is essential to examine what constitutes a sound inventory system. Therefore, this section 2.1, will introduce the status of the current website, regulations regarding auto-part storage, and state-of-the-art analysis.

2.1.1 Status of the current website

As mentioned in section 1, regarding the current company website for ZN Autocenter, there is a lack of desired functionality such as a website feature handling wholesale of vehicles, and another website feature for managing customer storage inventory. These features are so far incomplete, and development has ceased.

2.1.2 Tire regulations

Some regulations in Denmark dictate the guidelines and criteria for the storage of tires. For example, one such code addresses the maximum life span of a tire, which depends on storage conditions, such as environmental humidity and temperature. The legality of the tire depends on the state of the tire tread, which must exceed the legal minimum for the tire to remain legal. It is said that tires can get too old within 5-6 years. For this reason, it is good practice the check the condition of tires being brought out of storage, before putting them to use.[2] [3]

2.1.3 State of the Art

To have a general idea of the direction we need to take, we have a successful solution to our customers' problems. It is a good idea to look into the already existing solutions on the market. It is also important to remember that every company has its own needs. Therefore it is hard to make a general solution that will work everywhere. We will look into three different websites that offer the same "solution" we do:

- The first system we will look into is called **NetSuite** whose parent company is **Oracle**. The solution from NetSuite offers several different "Enterprise Resource Planning," which helps manage the resources. One of their solutions used in warehouse management offers a quicker way to handle the management and minimize the handling costs. It uses industry-leading practices such as bar-code scanning. The solution also offers mobile processing, which gives the customer a great and effective way to check on the current status of the warehouse and its inventory. The solution also offers flexibility which allows to send and request data from other systems and allows for the data to be synced.[**Oracle**]

- The second system is called Sortly. The idea behind this system is that the user types in every resource found in the warehouse to generate an overview and see the current status. In addition, Sortly offers a high customization field for its users, allowing them to personalize it and making it easier to track and see the current status. [4]
- The final solution that we will look into is not a specific website. Generally, any database that is sufficient in handling a high amount of information is going through. A database does not have the same functionality as the prior solutions had. However, it is a "good enough solution" for a small company. Nonetheless, this offers the best flexibility for the customer.

2.2 In-praxis

2.2.1 Interview

This section will describe the methodology behind the interview with ZN Autocenter secretary Sara, which was conducted by group members Abdallah and Oskar, and the results obtained from the interview.

2.2.2 Methodology

The purpose of interviews is to collect data and information to create an understanding of the partner's problem area, as well as requirements for what the system must be able to do. Three interview methods exist according to [5]

- **Structured interview:** The questions in a structured interview have been prepared in advance, and there will usually be no supplementary questions. Structured interview reminiscent of a questionnaire in which the questions are asked in a specific order and in which the persons answering the questions will give only limited answers.
- **Semi-structured interview:** In a semi-structured interview, there are some questions and themes made in advance, and there is room to go in other directions.
- **Unstructured interview:** There are no prepared questions in an unstructured interview, but the interviewer has a topic they know they want to talk about. This method is reminiscent of an ordinary conversation, where one question leads to the next.

Throughout this project, the semi-structured interview method was primarily used because it was able to supplement our prepared questions, which occurred during the interview itself. Thus, ZN Autocenter ApS could elaborate on their answers and provide an opportunity to ask their wanted follow-up-questions.

Brainstorm regarding the pre-made questions

Based on our brainstorm the following questions were made before the interview:

- Explanation of how the stock of tires and rims is currently managed.
- What actions should the database be able to perform? (Add, remove, edit? criteria search?)
- What data is relevant for rims and tires in your current system, and are there any other data you would like to see in the new system? (Eg Dimensions / summer tires etc.)
- What improvements would you like to see in the current storage system?

Based on the brainstorm we hereby conclude that it was an effective strategy and several goals were achieved during the conducted interview:

- How does the business currently work, manage inventory and what software does it employ?
- What are the logistics, constraints, issues and regulations regarding inventory storage?
- What improvements could be made to the current inventory storage system, and what information would the system need to properly manage inventory?

2.2.3 Interview with Sara, secretary at ZN Autocenter

During the interview with Sara, she explained how they currently manage their inventory of rims and tires, the storage layout of the auto shop, what data is relevant for such inventory, and what kind of improvements and functionality she would like to see in a future inventory system. Sara emphasized the necessity of solid search functionality, demand for the ability to attach comments to specific inventory items, and the importance of quickly assessing the physical location of a particular inventory. In this context, we were also informed of the necessity of FDM-rule compliance regarding tires (reference here to FDM regulations). Sara also went on to explain the logistics of managing loaner cars. However, it did not appear fitting for a software solution. There was not much company interest in such a solution either, so we decided not to pursue this idea further.

2.2.4 Results / Evaluation

Following the interview with Sara, we were able to develop an adequate understanding of how the business approaches inventory management and what selection of the shop inventory is relevant for a transition to a digital system. We were able to attain a comprehensive list of relevant inventory data, essential and desired system functionality, physical storage layout, and applicable regulations through the prepared questions. We also learned that there was barely any interest in a software solution for managing loaner cars, as the administrative burden of managing six cars was negligible. Consequently, we changed the focus of our system to handle rim and tires.

2.3 Our idea for a good inventory management system

The essential functionalities got brought upon after our initial meeting with the customer. The impression that the company laid out was a general wish for a better overview of their inventory. With the help of the interview, we found out several functionalities that the system should have to solve our customers' problems. 2.2

With the help of our State-of-the-art research 2.1.3, we would like to make software that has all of the advantages that they offer and use them in our system to make it "good." Our system will naturally be user-friendly and provide all the user's needs from it. That means the system would be intuitive and easy to use, which would create a good experience for our users. Considering that the user will use our system daily to ensure the storage is fine, we need to make the system dependable. To ensure that the experience of using the database is as friendly as possible, we would like to implement a decent amount of quality-of-life functionalities that make the system friendly and dependable. Some of the functionalities of our system could be; roll-back to reverse changes, Sorting ability, Recently Searched Items. Those are some of the functionalities we would like to have in our system to make it "good."

Problem Statement

Based on the problem analysis and the requirements of ZN Autocenter ApS, the scope of this project has been narrowed down to...

3.1 Problem statement

How can we utilize Java and Object-Oriented Programming principles to develop and implement a new and more efficient company website with the ability to keep track of available rim and tire inventory accessed through a log-in system on a website??

With the related sub questions:

- What is expected of the software?
- How can we use Java to implement the solution?
- How can we develop a fast-efficient website?
- How can we make a good intuitive design that is easy to use?
- What will a further development for the future be?

3.2 Problem delimitation

The focus of this project is limited to the development of an inventory management system and a new company website allowing access to the inventory system for company employees. The software is being developed for ZN Autocenter, and the scope and functionality of the system were determined through continuous dialogue with the company. The system will be limited to managing ZN Autocenters' own inventory of rims and tires 2.1.1, along with their in-shop inventory of customer-owned rims and tires stored on a rental basis on behalf of customers. The inventory management software will not manage ZN Autocenters' parts inventory, their fleet of loaner vehicles, or vehicle wholesale 2.2.4, as these areas do not impose a considerable administrative burden. As such, these features were not a priority from the company's side and were consequently omitted from the final software.

System definition

In this section, we will describe the system definition by using factor analysis and rich picture to consider our program idea in a different category. Therefore we will talk about our idea for a good management system and make sure what we want to make in our program.

4.1 System definition

A web-application for a car mechanic which on one hand act as website for the company and on the other hand as an administrative tool for the employees to catalogue and manage the inventory. Beside the company's own inventory, it must be able to keep track of the location of the customers' tires. It must also keep track of the age of the tires and hereby notify when they are too old. The administrative system must be intuitive and easy to use due to fact that the users are non it-experts.

4.2 Factor-Analysis

FACTOR analysis will be based upon the system-definition development.
Factor stands for functionality, application domain, conditions, technology, objects, and responsibility [6]

- F: The functionality of the system will be providing an overview and managing storage inventory. Users will be able to search for, access, and edit specific inventory details, such as quantity, location, size, etc., of products currently in storage.
- A: A system is provided in the form of a website. The secretary will mainly use this website. To a lesser extent, the owner and employees of the car workshop have a general overview of the workshop's physical tire and rim inventory. Users will be able to see the products which they currently have in storage.
- C: The system will be developed by the project authors, Group 2, SW3, at Aalborg University in Copenhagen. The program will be developed in cooperation with the owner of the ZN Autocenter car workshop.
- T: Front-end will include React, CSS, HTML, JavaScript. Back-end will include a server-side, which will be programmed in Java connected to a database (SQL-Lite) that will run on Windows/Chrome as a website application.
- O: The main objects in the problem domain are tires, wheels, shelves, and users. These will be our focus for system development.

- R: The inventory management system has the responsibility of dynamical inventory tracking by notifying users if inventory is running low or unavailable), adding and removing tires to tire hotel, consisting of new, and used tires in various locations around the auto-shop, as well as temporary storage of customer-owner tires, search feature, and click to call - call number feature. The website has the responsibility of letting users log in to access, browse, and edit the content of the inventory management system.

4.3 Rich Picture

To get better understanding of what is the system is about, we need to explain it with a rich picture. The rich picture illustrates the relationships between the main elements in the system. In this rich picture below, we can see the main actors and their task works starting with the client who asks for a specific set of tires to the secretary who starts with the search if they have the wanted tires and therefore locates the item and finally removes the item from the warehouse or add the item if the client wants to preserve the tires. We are describing just one possible scenario from many other possible scenarios.

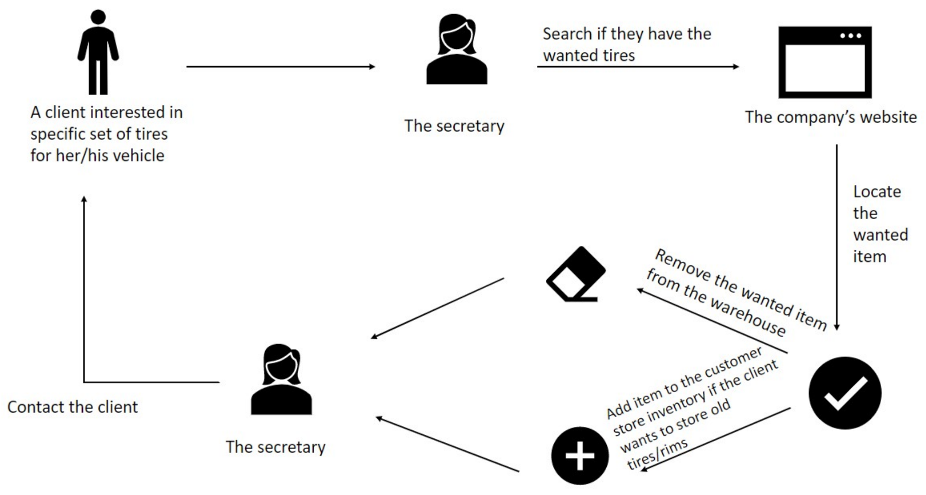

A client interested in specific set of tires for her/his vehicle
The secretary
Search if they have the wanted tires
The company's website
Locate the wanted item
Remove the wanted item from the warehouse
Add item to the customer store inventory if the client wants to store old tires/rims
The secretary
Contact the client

Figure 4.1: Rich picture of the system

After getting an overview of the system we will be working on, it is essential to know the critical things to keep in mind regarding inventory systems according to our customer.

Problem Domain Analysis

The analysis of the problem domain, i.e., the part of the system that needs to be administered, monitored, or controlled, is the main focus of the following chapter. This chapter will initially provide an overview of classes and events in the system 5.2 before detailing the behavior of classes through state-chart diagrams5.3. Relations between classes and events are then examined, revealing the underlying structure5.4, and class relations will be further illustrated by a class diagram.

5.1 Requirement Specification

In this section, the requirements of the software program are described by using the MoSCoW method.

5.1.1 MoSCoW method

Instead of the usual soft and hard demands, this report will be structuring the program's priority according to the MoSCoW model. This is the priority in which the program's functionality will be created so that the project's core ideas will be carried out in the most critical order.

5.1.1.1 Requirements

The requirements for this program are categorised into three MoSCo(W) categories, excluding the "Won't have" section.

Must have

- An intuitive, minimalistic website.
- A secure login-system that redirect the user to the inventory management system.
- A database that persists the needed data for the website.
- A search function that will allow us to find the data field for specific inventory item by their attributes.
- An add and delete function that allows the user to select a particular data field or a whole data-set and then provide the option to delete the selected fields or data-set in case the inventory has been either sold or no longer is in storage.
- An editing function that can edit every single element inside of any given inventory by referencing to their attribute.

Should have

- Should have a security layer added to the login system, such as hashing and salting.
- Should have an undo-function in case of any mistakes happening.
- Should have different account privileges such as an administrator account for creating and managing other users as well as the right to edit and delete from the inventory.

Could have

- Could have a reminder when the tire is getting too old (5-6 years).
- Could have a change log for edit history.
- Could have cross-device functionality.
- Could have a function to inform the user whenever storage space is running low or if the storage is fully occupied.
- Could have other integrated features such as a bill and rental management system.
- Could have a shelf class that persists the data from the other classes and allows users to see the exact inventory stored at a particular shelf and the current and total capacity of that shelf.
- Could have a move item functionality, as a shortcut to editing item location data.

5.2 Classes & Events

In this section, there will be a description of the different classes and events that will be in the system. And there will be also an event table to help with understanding the relation between classes and events.

Classes

Classes are a set of items used as a problem-domain abstraction. Three classes were found in the problem domain, and they are as follows:

- Tire
- Rim
- User

Events

Events are atomic occurrences that one or more objects participate in,and they are also used for problem-domain abstraction. Eight events have been found in the problem domain.

- Add
- Remove
- Edit
- Move
- Search
- Log in
- Log out
- Retrieve Password

Event table

An event table can be used to describe the classes and events that interact with each other. In the event table, a "+" indicates that an event can only occur once on objects within the given class, whereas a "*" indicates that an event can occur multiple times on objects within the class.

Events & Classes	Tire	Rim	User
Added	+	+	+
Removed	+	+	+
Edited	*	*	*
Moved	*	*	
Searched	*	*	*
Log In			*
Log Out			*
Retrieve Password			*
Store Customer Item	+	+	*
Retrieve Customer Item	+	+	*

Table 5.1: Class & Event Table

The analysis class and events activity has been completed after identifying the problem domain's classes and events. The structure activity is the next step in the analysis, and this is where the relationships between classes are defined.

Class Candidates

- **Tire – Collection of tire objects**

  Relevant attributes includes:

  Unique ID/ItemNumber

  Size/dimensions

  Type/Summer/Winter/AllYear

  Date

  Condition/expiration

  Brand

  Product Name

  Location

  Quantity

  Customer Info

  Comments

- **Rim – Collection of rim objects**

  Relevant attributes includes:

  Unique ID/ItemNumber

  Size/dimensions

  Brand

  Product Name

  Color

  Material

  Date

  Condition

  Location

  Quantity

  Customer Info

  Comments

- **User - Collection of user objects**

  Relevant attributes includes:

  User ID

  User Name

Password

Date

Privileges

Comments

Event Candidates

- **Added**

  Integrating a class object, along with all related data, into the database.

- **Removed**

  Deleting a class object, along with all related data, from the database.

- **Moved**

  Move a class object, by editing location data.

- **Edited**

  Edit a class object, by changing one or more data variables, in the database.

- **Searched**

  Search for specific item or category of items in the database, by searching for unique or common data values, respectively.

- **Log In**

  The process for a user to log in and gain access to the database.

- **Log Out**

  The process for a user to log out and sever access to the database.

- **Retrieve Password**

  The process for a user to retrieve a forgotten password.

5.3 Behaviour

In the Behaviour section we will look into each event and examine their so called behaviour. Their behavioral patterns of each class will be modelled into a state-chart diagram.
To understand the behavioral pattern we will need to first understand how to read the diagrams. We read it from left to right where we begin from the black dot. We follow the arrows that lead us to the black dot that has a circle around it, which marks the end. In the pattern, we will also find Squares, which explain the current state of the class. The state has also arrows that point towards it, this arrows explain the transitions that can happen to the class in that state and that these events can be done multiple times.

5.3.1 State-charts

Tire class

The Class **Tire** is one of the few classes that we have in our system. The class contains multiple information's about the tire, its owner and its current state. There are multiple events that can influence the class, the state-chart of the behaviour can be seen in the chart below.

On figure 5.1 it can be seen, that a tire-object is initialized by the 'tire added' event. When a tire is first added, it enters the Active state, where the events 'tire moved', 'tire edited' and 'tire searched' can happen. At the end there is a 'tire removed' event which will delete the tire object.

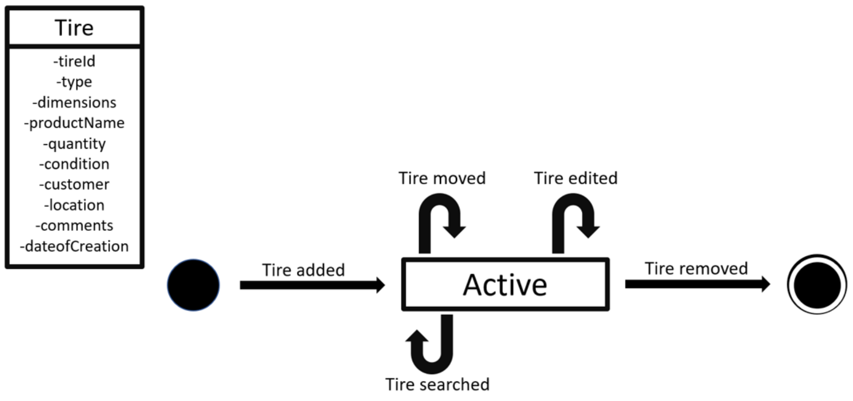


Figure 5.1: State-chart describing behavioral patterns of the tire class.

Rim class

The next class which is called **Rim**, will be used for the different rims that can be found in the workshop. Like in the previous class the same events can be seen influencing this class. This can be seen in the state-chart below.
On figure 5.2 it can be seen, that a rim is added by the ’rim added’ event. When a rim is first added, it enters the Active state, where the events ’rim moved’, ’rim edited’ and ’rim searched’ can happen. At the end there is a ’rim removed’ event which will delete the rim object.

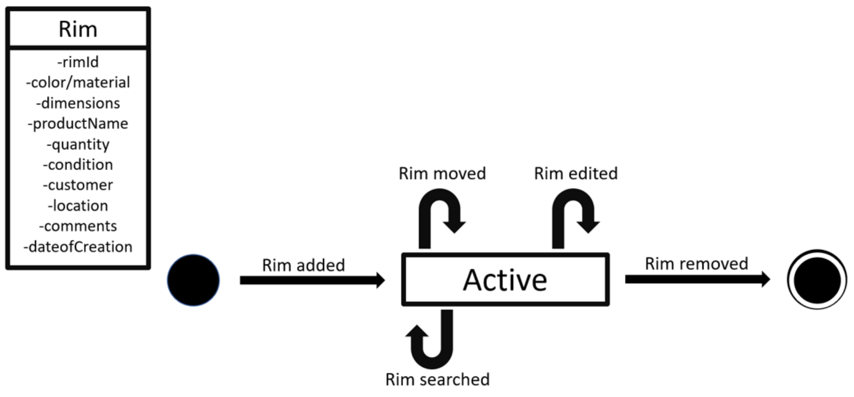


Figure 5.2: State-chart describing behavioral patterns of the rim class.

User class

On figure 5.3 it can be seen, that a user is added by the user added event. When a user is first initialized, it enters the Active state, at which point the user class can be edited, or the class can be deleted from the system through the ’user removed’ event.

On figure 5.3 it can be seen, that a user is added by the user added event. When a user is first initialized, it enters the Active state, from where the user can be deleted from the system through the ’user removed’ event, or a user can request to log into the database, at which point the class-state changes to ’awaiting responds’. From this state, the request can be denied whereby the state will change to active, the request can be confirmed and the state will change to ’logged in’, or the user will exceed their allowed log-in attempts and the state will change to ’inactive’. From the inactive state, a user with administrative priviledges will need to manually change the inactive users state to ’active’ to re-activate the user. And lastly, from the logged-in state, the user can ’log out’ of the database which will change the change back to ’active’.

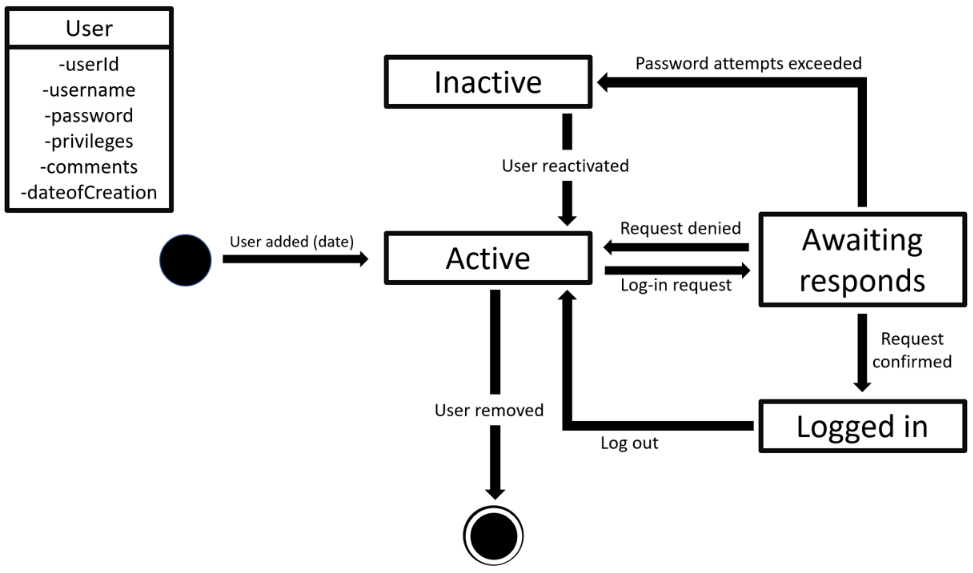


Figure 5.3: State-chart describing behavioral patterns of the user class.

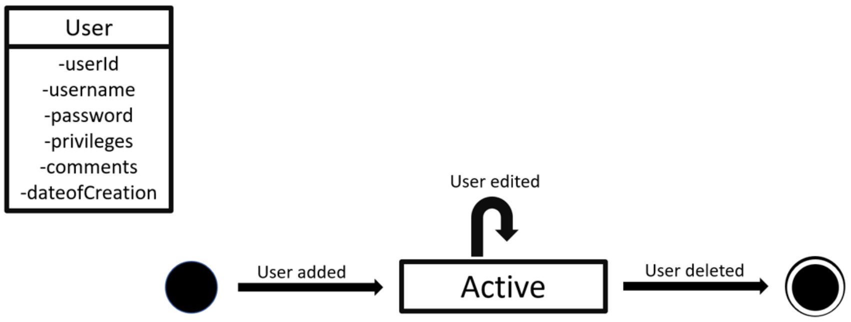


Figure 5.4: Ben, Adb - State-chart describing behavioral patterns of the user class.

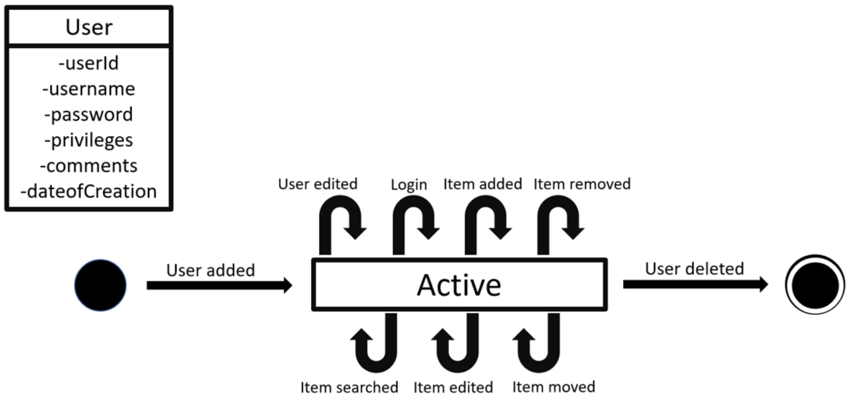


Figure 5.5: Osk, Loj - State-chart describing behavioral patterns of the user class

5.4 Structure

In this section, we will take a closer look at classes in the problem domain, to describe the structural relations between these classes, through modeling a class diagram of the system. The resulting class diagram can be seen in figure 5.6 below.

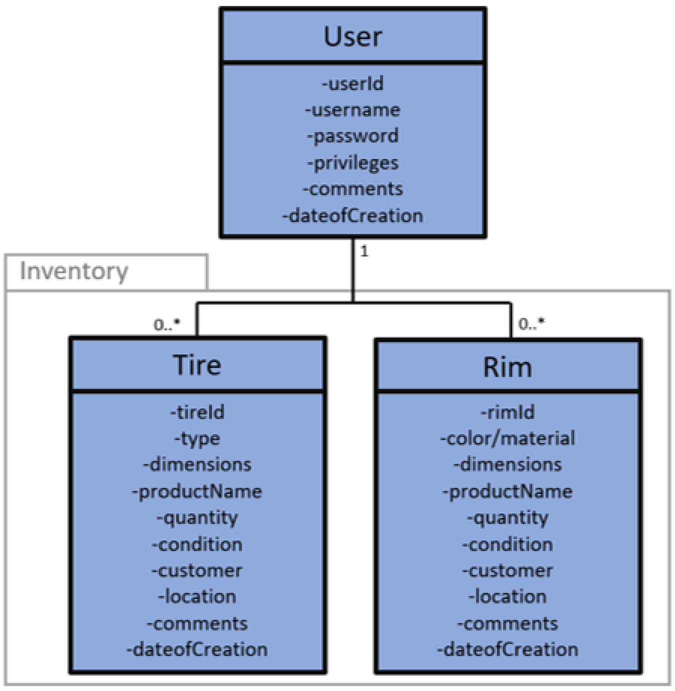


Figure 5.6: Class diagram of the problem domain

Descriptions of class relations

- **Tire/Rim - User**
  These relations illustrate that the user class can access and modify both the other classes. These relations also show that the system could exist in a state without rims, and tires, but not without users.

Application Domain Analysis

6.1 Solution Proposal

6.2 PACT Analysis

System on workshop Website will monitor tires movement and rent-a-car. 2-4 Mechanics 1 Owner Age group: 25-38 (change)

Pact analyse: We will do a PACT analysis with different aspects about centred design about the system we will make for the users.

- People: Our product will be made for the employees in the ZN-auto-center company. The workforce includes 2-4 mechanics (and two assistants) and a single owner. The website can be used by everyone, which would be implemented on tablets in the workshop. At any given time, the employee can check the availability of the tires in the shop/warehouse.
- Activities: Since the people are mechanics and might not be fond of machines, the website created should be simple to use. These are defined in the steps below: Temporal activities 1. This website will be used daily and frequently throughout the day. Customers will keep coming into the company, so employees will have to check for the size of tires. Therefore, the website design should be easy to use.
  2. The website should be straightforward and easy to use, so if there is more than one customer at the counter, it is easy to know what is in stock.
  3. While using the website, the employee could be distracted by some other query from another employee or customer. Therefore, it should have a lower refresh rate so the website could start from where they left it.
  4.The response time for the website should be quicker as more customers would come in. Customers would not want to wait a lot for their turn.
  5. The website could be used alone by any employee. 6. The activities should be well-designed using step by step approach. There should not be many as some steps can be clustered together, e.g., tire size, brand, feature (summer/winter), and quality.
  7. Mistakes in the website would not cause any injury. It is safe to use, and the user can restart the process again.
  8. If the website has an error, the process can be restarted.
  9. A tablet would have built in touch and keyboard, so it should be enough to use the

website.
10. The display keyboard should have alphanumeric keys; for example, tire sizes come with numbers and alphabets.

- Context: Physical environment – the website will be used in the company, so the tablets would be needed to connect to the leading network. They should be placed at different places, so they can be easily used.
  Social Context – the website would be straightforward and would not need much help from other employees. There would be no need for sound for the website to work.
  Organizational context – the organization would not change much over time, as the business operations would remain the same.

- Technologies: Input – the data would be entered using QWERTY Keyboard on the screen of the tablet. The tablet can be touched using the fingers of the employees or a touch pen attached to the tablet. This will allow the user to use the website quickly and would not require a lot of space to check for the inventory. Output – the screen of the tablet would be the output device. The user can easily see the data search results like the screen size of the tablet is good, and the keyboard would be minimized when the results are displayed. This would allow the employee to see the screen clearly.
  Communication – the tablets would need to be connected to the Internet at all times. This would allow the users to use the device fluently without interruption. Modern tablets are all equipped with built in Wi-Fi that can be connected to wireless routers.
  Content – the result of the search should be displayed as easy to read as possible. It should always be up to date and should reflect live data changes whenever anything changes. For example, if a set of tires is sold 5 minutes earlier, this should clearly be reflected in the report in the next run. This way employee is clearly informed of what is in stock, and the owner can reorder the tires which are in demand.

6.3 Actors

The first step in making the application for domain analysis is to know the people who will use that system.
This task is completed by doing actor analysis. In this analysis, different actors are identified. The actors can be categorized as a human or another system that connects to this system. The analysis of each actor is split into two major sections: Purpose and characteristic.

- Owner:
  Purpose: a person who owns the company. The owner needs to be able to reach everything in the system.
  Characteristic: the system's users include just one owner

- Employee:

  Purpose: a person who works in the company. The employee's primary need is to be able to add, remove, edit, search, and move items.

  Characteristic: the system's users include 2-4 employees.

Given the actor analysis is now complete, the work tasks of the actors can be identified.

Actor Table	Employee	Customer
Add Object	X	X
Edit Object	X	X
Remove Object	X	X
Move Object		
Search for Object	X	X

Table 6.1: Actor Table

6.4 Work Tasks

Following identifying the actors who will utilize the system, the following step is to determine the work tasks they will do in the system. The actors and users will perform work tasks.

- Login: is a work task, which requires the actor for username and password to allow access to the system.
- Create booking: is a work task, where the user will be able to make a booking appointment at ZN auto-center.
- Cancel booking: is a work task, where users will get an option to cancel the booking appointment from the system.
- Search: is a work task where users can search for any item in the system.
- Remove item: is a work task where users can remove an item used or sold.
- Add item: is a work task when the user wants to add a new item in the system.
- Edit item: is a work task, which gets used, when the user wants to edit tires and rims information in the system
- Move item: is a work task, where a user will get an option to move any item from the system.

After the work tasks have indeed been checked and described, the system can be further analyzed by establishing a class diagram with the actors and work tasks, along with functions.

This diagram shows the relation between main elements, where the main two actors of the system, use cases and work tasks. In the diagram below, we can see the main actors, their work tasks connected to the use cases. We're only discussing one conceivable scenario among many others. The first step in such methods was to analyze the existing physical system. We are describing just some of possible scenario of work tasks from many other possible scenarios.

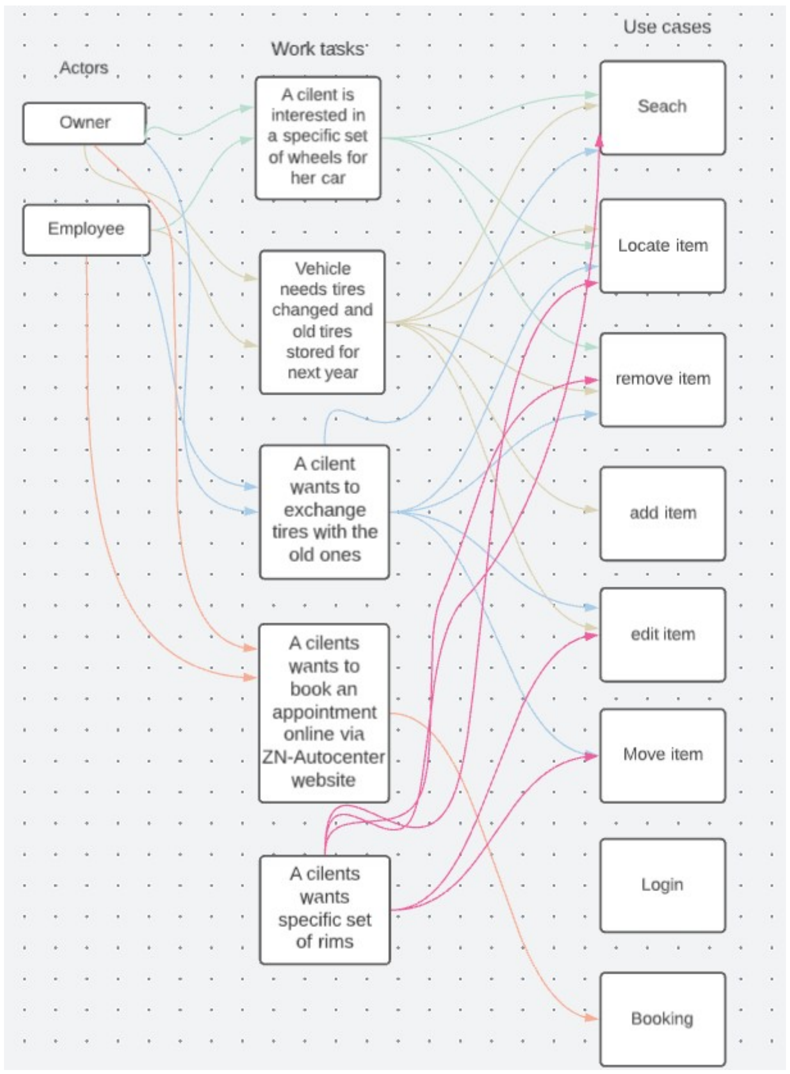

Actors
Owner
Employee
Work tasks
A cilent is interested in a specific set of wheels for her car
Vehicle needs tires changed and old tires stored for next year
A cilent wants to exchange tires with the old ones
A cilents wants to book an appointment online via ZN-Autocenter website
A cilents wants specific set of rims
Use cases
Seach
Locate item
remove item
add item
edit item
Move item
Login
Booking

Figure 6.1: Work Task diagram

6.5 Use case

In this section, the use-cases will be described specifically to give a general understanding of the use-case.

6.5.1 Use Cases for the website

- **Login**
  A user applies his personal info in the login landing page in order to enter the system panel.
- **Booking**
  in case a user applies his personal info in the login landing page in order to enter the system panel.
- **Retrieve forgotten login-info**
  A user applies his personal info on the login landing page to enter the system panel through a hidden page.
- **Email contact**
  A user applies
- **Telephone contact**
  A user applies

6.5.2 Use Cases for the Database

- **Adding Item**
  A user applies the system to add an object to the database.
- **Removing Item**
  A user applies the system to remove an object from the database.
- **Moving Item**
  A user applies the system to “relocate” an object by editing the location of the object.
- **Editing Item**
  A user applies the system to edit one or more attributes of an object in the database.
- **Searching For Item**
  A user applies the system to locate objects in the database that matches specified attribute values.

6.5.2.1 Use Case Descriptions

A system user should be able to log in, which is done through a login. When a user inputs their username and password, the information is validated, and the user can log in if it is correct. If the information is incorrect, the user must re-enter their login credentials.
"Login the system" is the initiating transition in the Login use case diagram. It enters the state of Waiting. There is just one transition in this state called "enter information", which leads o the state "info checked" The state "Info checked" has two transitions; one is approved, where approving all information is correct. And another transition is "invalid", leading back to the state of Waiting.

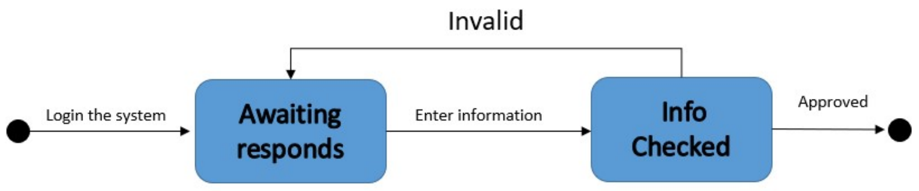


Figure 6.2: login

- **Searching For Item**

  The use case diagram for searching for an item starts with searching for the wanted item and therefore checking the list in the database and if the wanted item is there, so we wait until the item is being located. If the item has not been found, the process ends, and the user gets a message with rejecting.

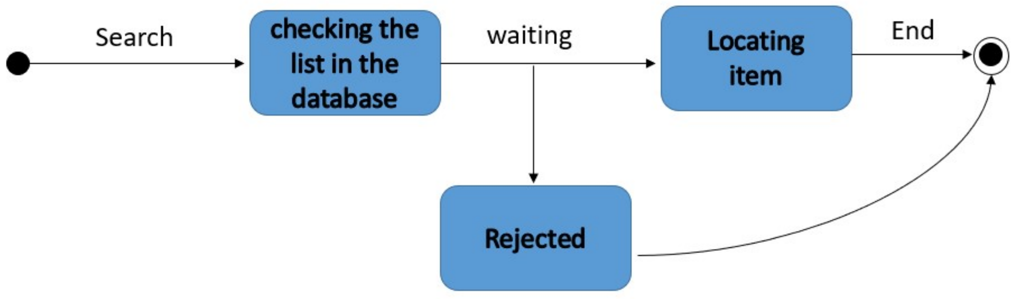


Figure 6.3: searching the item

6.6 Functions

In this section, we will take a closer look at each function that is found in the system. This will help us to determine the system's information processing capabilities. Each function will be described in addition to its complexity. We will have an overview of all the different functions, their complexity and type them in a table. Therefore we will describe every function itself.

List of Functions	Complexity	Type
Add Object	Simple	Update
Edit Object	Simple	Update
Remove Object	Simple	Update
Search	Medium	Read
Move Object	Simple	Update
Login	Complex	Update
Retrieve Forgotten Password	Complex	Update
Change Password	Medium	Update
View Inventory	Simple	Read

Table 6.2: Functions Table

related to. The categories are object, login, and inventory.
We will explain more about functions of higher complexity.

6.6.1 Object functions

- **Add the object:**
  A simple update function that allows the employee to add a new object.

- **Edit the object:**
  A simple update function that allows the employee to edit the object's information.

- **Remove the object:**
  A simple update function that allows the employee to delete the object.

- **Search for the object:**
  A medium read function that allows the employee to search for the object.

- **Move the object:**
  A simple update function that allows the employee to move the object.

6.6.2 Login functions

- **Login function:**

  A complex update function for handling the login process.

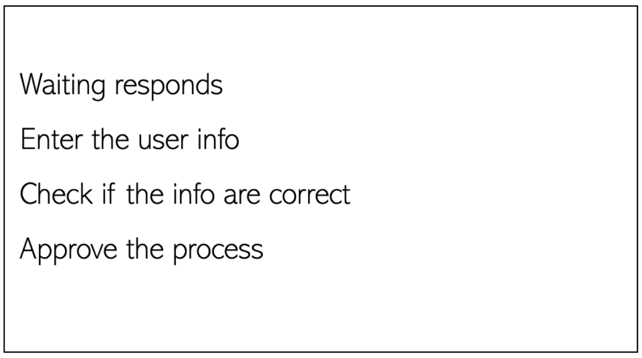


Figure 6.4: Login function

- **Forgotten login-info:**

  A complex update function for recovering forgotten passwords associated with user accounts.

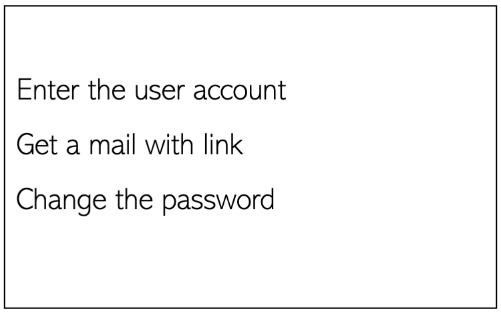


Figure 6.5: forgotten login

- **Change Password:**

  A medium update function that allows the employee to change the password.

6.6.3 Inventory function:

- **View Inventory:**

  A simple read function that allows the employee to view what the inventory has.

Criteria

Prioritization of criteria plays an essential part in a functional system. Therefore it is important to prioritize different design elements because it is impossible to focus on every design aspect.

Criteria Table	Very Important	Important	Less Important	Irrelevant	Easily Done
Usable		X			
Secure			X		
Efficient		X			
Correct	X				
Reliable		X			
Maintainable		X			
Testable		X			
Flexible	X				
Comprehensible		X			
Reusable			X		
Portable				X	
Interoperable				X	

Table 7.1: Criteria Table

- **Usable**

  We decided to classify "usable" as an important criterion because employees and users must be able to use it easily without having any problems.

- **Secure**

  We decided to classify "secure" as a less important criterion because we do not deal with important or sensitive data. Our data is just information related to tires and rims, etc.

- **Efficient**

  We decided to classify "efficient" as an important criterion because the system will contain a large amount of data, and inefficiency may quickly impact the system's responsiveness.

- **Correct**

  We decided to classify "correct" as a fundamental criterion because if the system does not fulfill the specified requirements, then the system will not be helpful in the intended use case scenario.

- **Reliable**
  We decided to classify "reliable" as an essential criterion. The system must be reliable because it belongs to a specific company and is supposed to work well.
- **Maintainable**
  We decided to classify "maintainable" as an important criterion because the system belongs to an existing and continuous company. Changes may occur, so it is important to be maintainable over time.
- **Testable**
  We decided to classify "testable" as an important criterion because one of the requirements of this project in this semester is to be able to test the program.
- **Flexible**
  We decided to classify "flexible" as a very important criterion because we are developing the system for a company, which intends to use the system to manage a constantly changing inventory, so it is important that they have the ability to change and adapt the contents of the system, whenever they like.
- **Comprehensible**
  We decided to classify "comprehensible" as an important criterion because the system exists to be used by people, so it must be understandable and they can easily access the information they want.
- **Reusable**
  We decided to classify "reusable" as a less important because the purpose of this system is to manage a system for a car mechanic company, so the system is usable for companies with same vision
- **Portable**
  We decided to classify "portable" as an irrelevant criterion because the system will run on a computer or a tablet and there is no need to be portable.
- **Interoperable**
  We decided to classify "interoperable" as irrelevant, since the system is not required to work with, or couple to other systems.

Component Specification

8.1 Model Component

The model component is described in this section. The model component's goal is to give information to the function component. The information might be both current and prior data, depending on the project's requirements and specifications. In this context, previous data could include things like a capacity that has been changed to something else. Because the client isn't interested in past data, it was decided that it wouldn't be relevant for this project. As a result, any change in the value of a property will only be recorded on that attribute.
A model component will be described. To this goal will be used class diagrams, event tables, and state charts.

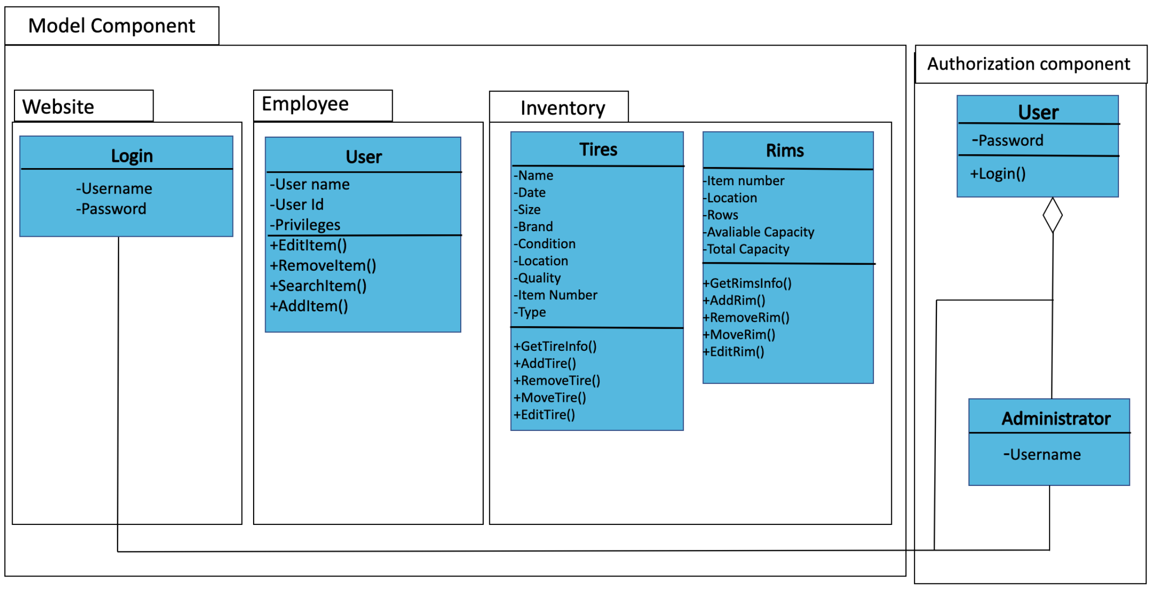


Figure 8.1: Model component

The upgraded version of the class diagram shown in 8.1 is the result of the model component design. The updated model component will also include new attributes and classes representing the previously established classes' behavioral events. In addition, the model component's design process is simplified because the system does not need to be able to provide past data. During the issue domain study, it was also determined that meta-information about the events, such as the time an event occurred, was not required for the majority of the occurrences. The model component's design has also been streamlined as a result of this.

8.2 Function Component

In this section we will be describing the Function Component. The purpose of a Function Component is to determine the implementation of them and provide an access to the model component from the user interface component. With this we can create a model that will describe the functions that will be used to provide that access.

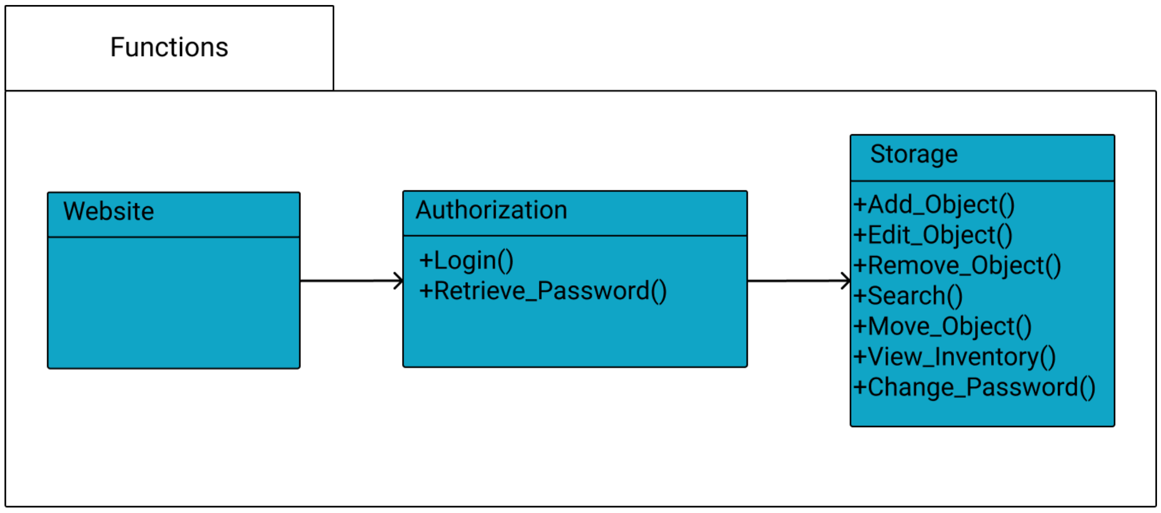


Figure 8.2: Function component

As seen on the model we have grouped up functions with 3 different classes. The three newly added classes are **Website**, **authorization** and **storage**.

- **Authorization** In this class we have all functions that are related to our customer, who will be the primary user of the storage system. In this class we have function that are used to authenticate the user, when he/she is trying to access the storage. We also have the function that helps them in retrieving their lost password.
- **Storage** In the Storage class, we can find all of the related functions that are relating to our other classes.5.1 Other than that we also have a function that helps our user in changing their password.

8.2.1 Connecting component

The system components that were designed in the previous sections are connected in this section. Furthermore, how these components should be integrated with the Interface and Technical Platform is described.

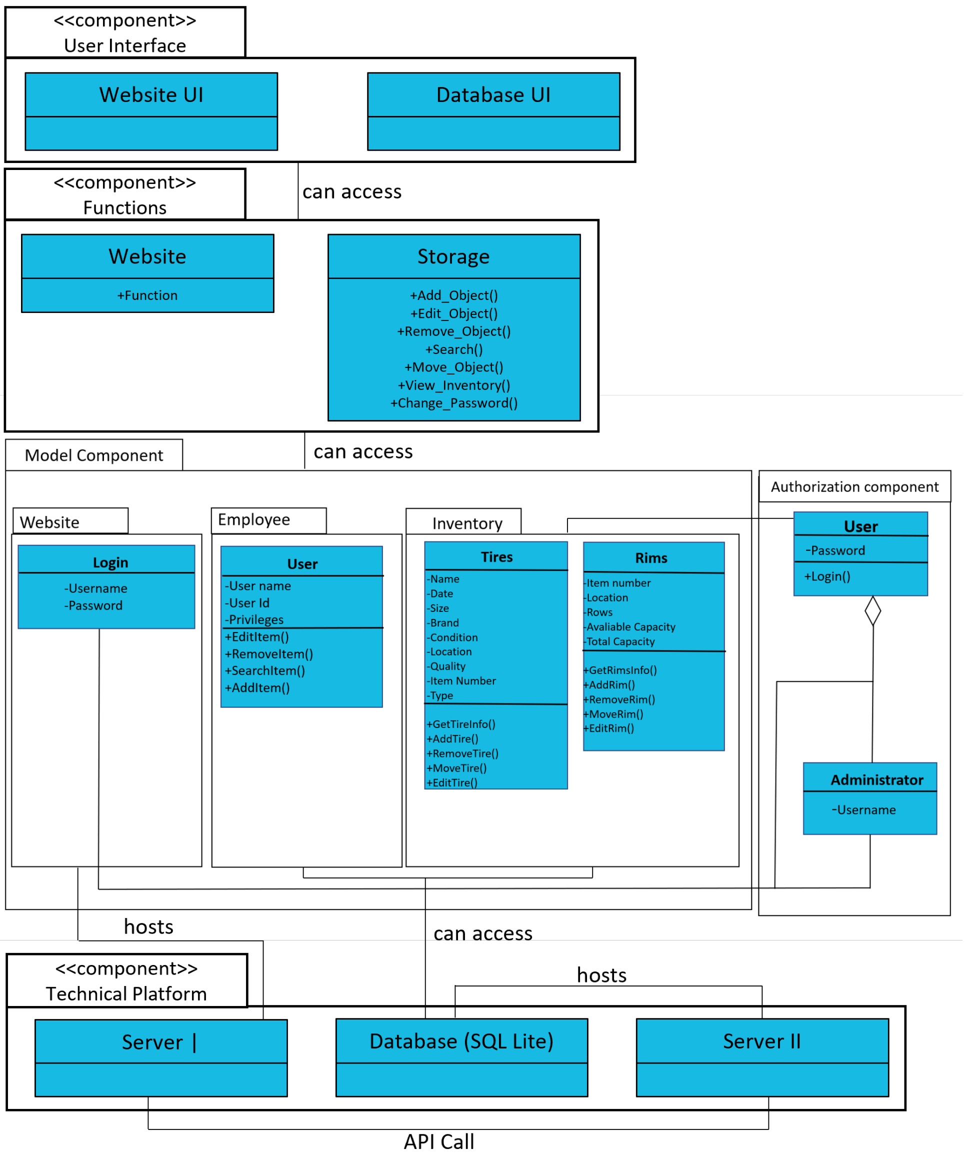


Figure 8.3: Connecting component

The Interface-, Model-, Authorization-, and Function component and the Technical Platform are the five main parts of the component structure. The detailed definitions of the classes in the model-, authorization-, and function components have not been included in the picture because this section is mainly about the organization of the components. See the respective sections in this chapter for further information on the detailed design of these classes.

Designing of user interface

This chapter will go over the application's design and the decisions taken along the process. To better understand the system and make it straightforward for all levels of competency, the conceptual design will be discussed. The principles that underpin the design will be presented first.

9.1 Design principles

It's important to have some ground principles in mind when creating an application. This section will explain the principles that will set the foundation for this application and an explanation of why certain design elements were chosen

9.2 Memory and attention

One of the key aspects a company values in the systems they are implementing is simplicity. In order to keep things simple, the principles of memory and attention would be taken into consideration. Handling different types of rims and tires involves different processes as there are different sizes and quality of the rims and tires. Moreover, the users of this software would be mostly semi-skilled labour so the software should be as simple as possible. Simple processes would allow the employees to move their attention to various tasks at hand. The system would be designed keeping divided attention principle. This way the employees can multitask and divert their attention on other tasks at hand rather than just be fully invested in the system. This would be achieved by simplifying complex processes or removing them from the system. Keeping a simple system would decrease a lot of mental pressure on the employees and improve the usability of the system.

9.3 Human Errors

Human error is something that often happens, so we will take it into account when designing a company's web page. In our case, no human errors will occur in Front End. But human errors can occur in the back-end, for example, wrong information can be written about a tire or rims or something else. But it is not a huge mistake that can lead to big losses. In any case to avoid these errors, prompts can be added to alert the user about whether he/she wants to continue the procedure or not. In this way, we avoid even small mistakes.

9.4 Wireframes of the design

This section will look at how the user interface was created and what considerations went into the design decisions.
Wireframes of the different parts of the system can be built based on the hand-drawn designs. Balsamiq was used to produce these wireframes, as it allows for quick wireframe creation.
In the figure 9.1 below, the first wireframe for the participant application are shown. Wireframes gives a general layout of the website, and this wireframe is for the home page of the website. In figure 9.1 below, it also shows some other important action, which are 4 main elements of the car workshop, and their experience.

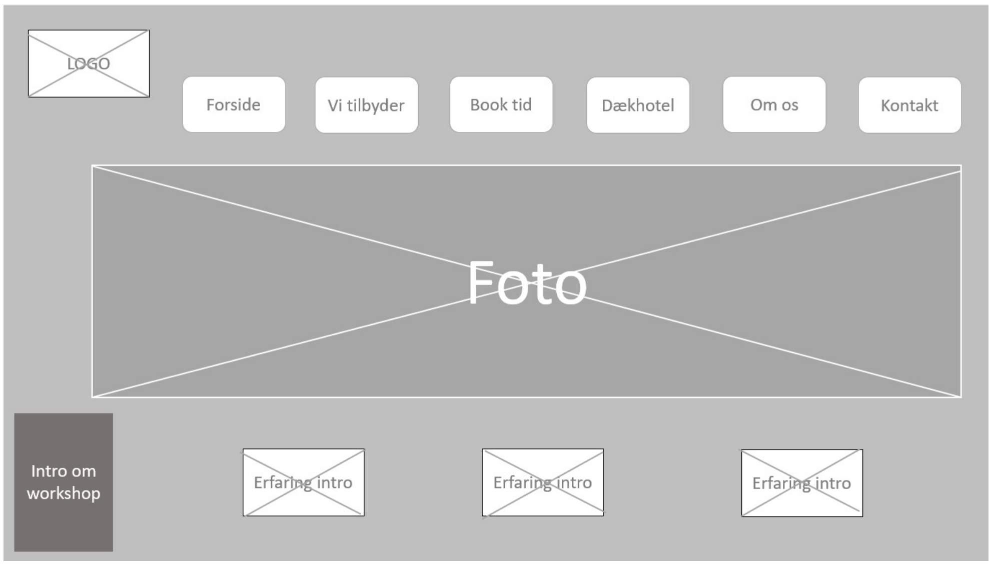
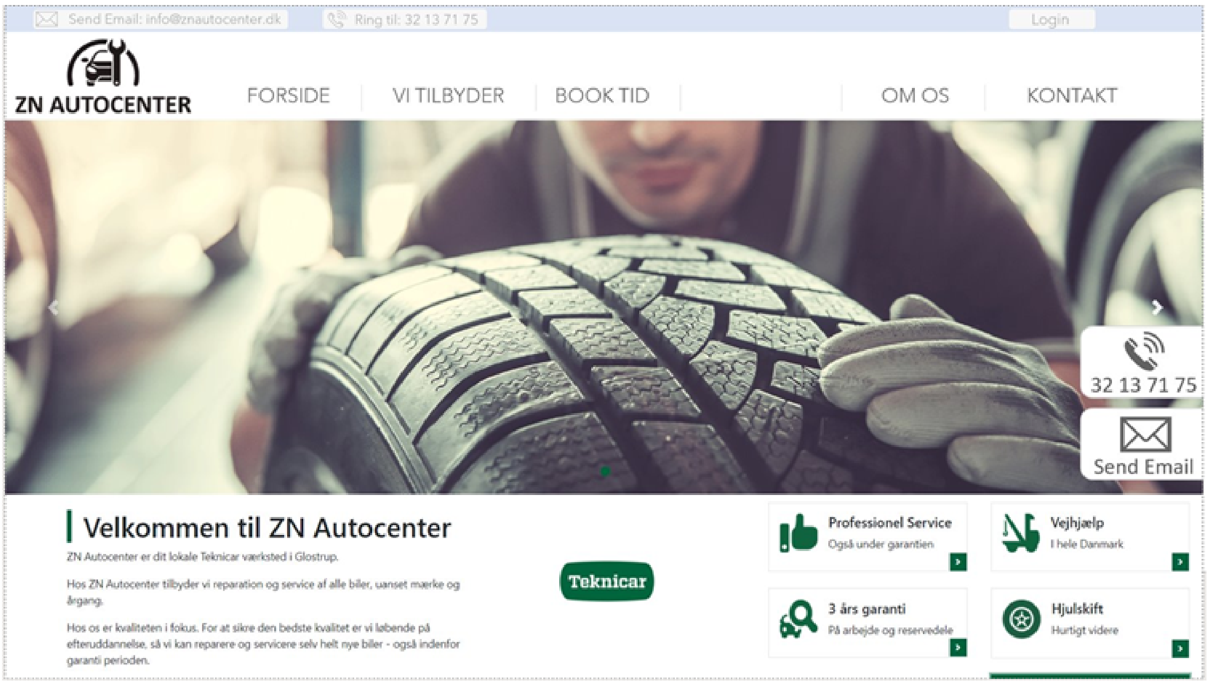


Figure 9.1: Home Page wireframe

In the figure 9.2 below, it's the second wireframe of the website home page with more details. we have added pictures and colors to it, with the icons and logos. information text about ZN auto-shop is also added in this wireframe.



Velkommen til ZN Autocenter

ZN Autocenter er dit lokale Teknicar værksted i Glostrup.

Hos ZN Autocenter tilbyder vi reparation og service af alle biler, uanset mærke og årgang.

Hos os er kvaliteten i fokus. For at sikre den bedste kvalitet er vi løbende på efteruddannelse, så vi kan reparere og servicere selv helt nye biler - også indenfor garanti perioden.

Figure 9.2: Home Page wireframe

The wireframe for the administration site can be seen in the image below. According to the wireframe for the administration site, the administrator can navigate to four pages: we offer, booking, about us, and contact. The wireframes show no unnecessary extra features because the page is designed to be simple to understand. The home page can update or do other actions in each of the parts. Only a portion of the sites displayed in figure are available to the venue management. where they can learn more about the auto workshop

9.4.1 wireframes for the database

We have different parts in the storage section. We will take a look at most important parts.

9.4.1.1 The home page

Here we can see the home page in the storage. On the left side of the figure, we can see the inventory which contains the tires, the rims, and the shelves. Slightly down we can see user administration and log-out buttons.

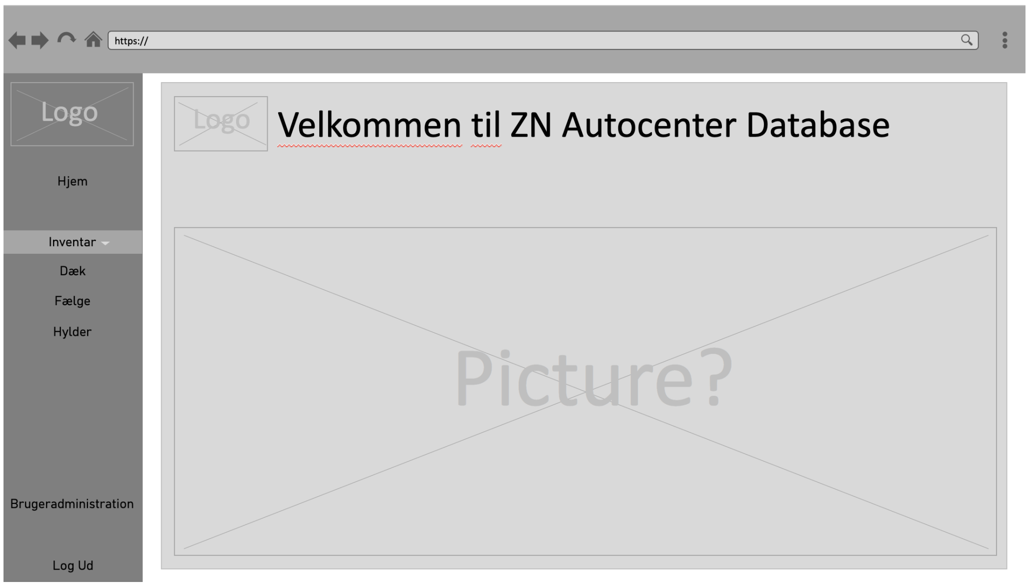
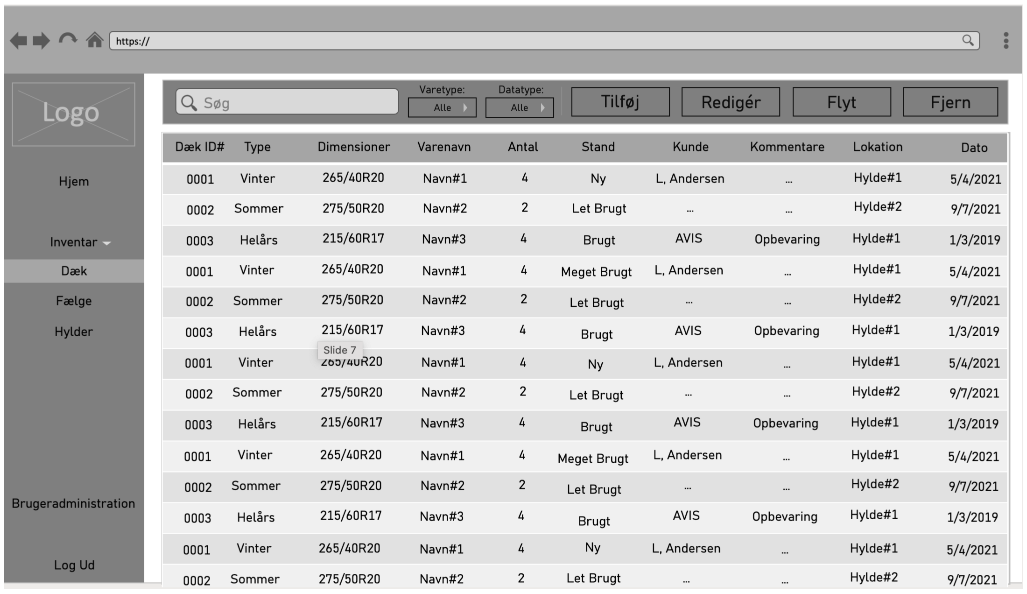


Figure 9.3: Wireframe for the database home page

9.4.1.2 The Tire Page

The tire page will be divided into two components. The first is the part on the top of the page. There will be a place to search, add, edit, move and remove the tire. The second part will be the rest of the page. There will be a list of the different tires which finds in the inventory. Every tire will have a list of information as to type, name, quantity, etc.



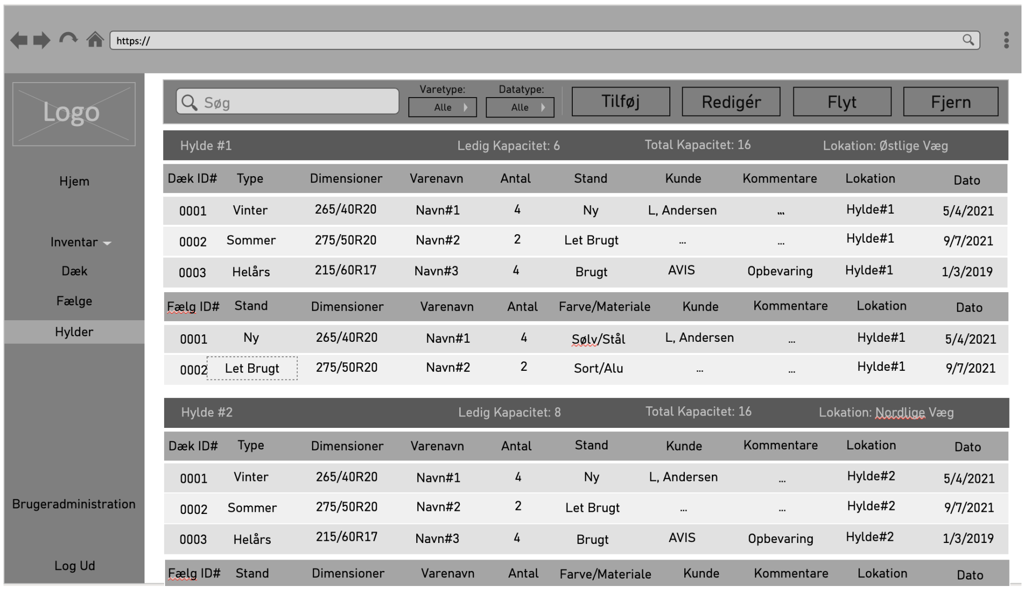
Dæk ID#	Type	Dimensioner	Varenavn	Antal	Stand	Kunde	Kommentare	Lokation	Dato
0001	Vinter	265/40R20	Navn#1	4	Ny	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#1	1/3/2019
0001	Vinter	265/40R20	Navn#1	4	Meget Brugt	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#1	1/3/2019
0001	Vinter	265/40R20	Navn#1	4	Ny	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#1	1/3/2019
0001	Vinter	265/40R20	Navn#1	4	Meget Brugt	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#1	1/3/2019
0001	Vinter	265/40R20	Navn#1	4	Ny	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021

Figure 9.4: Wireframe for the database tire page

We should mention that the rim page is very similar to the tire page, therefore there will be not necessary to add it to the report.

9.4.1.3 The Shelf Page

The shelf page is also very similar to the tire page, but the only difference is that here will be added two shelves and inside every shelf will be a list of what the shelf contains, the available capacity, the total capacity, and the location.



Hylde #1 Ledig Kapacitet: 6 Total Kapacitet: 16 Lokation: Østlige Væg

Dæk ID#	Type	Dimensioner	Varenavn	Antal	Stand	Kunde	Kommentare	Lokation	Dato
0001	Vinter	265/40R20	Navn#1	4	Ny	L, Andersen	...	Hylde#1	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#1	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#1	1/3/2019

Fælg ID#	Stand	Dimensioner	Varenavn	Antal	Farve/Materiale	Kunde	Kommentare	Lokation	Dato
0001	Ny	265/40R20	Navn#1	4	Sølv/Stål	L, Andersen	...	Hylde#1	5/4/2021
0002	Let Brugt	275/50R20	Navn#2	2	Sort/Alu	...	...	Hylde#1	9/7/2021

Hylde #2 Ledig Kapacitet: 8 Total Kapacitet: 16 Lokation: Nordlige Væg

Dæk ID#	Type	Dimensioner	Varenavn	Antal	Stand	Kunde	Kommentare	Lokation	Dato
0001	Vinter	265/40R20	Navn#1	4	Ny	L, Andersen	...	Hylde#2	5/4/2021
0002	Sommer	275/50R20	Navn#2	2	Let Brugt	...	...	Hylde#2	9/7/2021
0003	Helårs	215/60R17	Navn#3	4	Brugt	AVIS	Opbevaring	Hylde#2	1/3/2019

Fælg ID#	Stand	Dimensioner	Varenavn	Antal	Farve/Materiale	Kunde	Kommentare	Lokation	Dato

Figure 9.5: Wireframe for the database shelf page

9.5 Colors

One of the basic things that must be taken into account when designing any interface is the colors used. Therefore, we will talk in this section about the colors that we will use for the website. The colors that will be used on the site will be black and light blue. Where the black color will be used for the texts that will be present on the site. The blue color will be used for the buttons and icons. Compared to the web page before modification, we note that the most used color is dark green. The reason behind changing the color is and after doing some research we found that blue is the right color for the site for several reasons, we will mention them now. The first reason is that blue is a calm color and preferred by most people, especially men, in addition to the blue color is often seen as a sign of reliability. Therefore, it is good to use it for a site for a car mechanic company, where the blue color will give customers safety. [7]

9.6 Navigation map

Based on the wireframes from the previous section, a navigation map has been made to showcase the navigational structure of the system. The navigation map is shown in 9.6. A navigation diagram is a simple visual representation of how different parts of the program are connected.

We had two parts of the navigation map, on one website and another one is database. The website is going ot be the front-end, and everyone would have access to the website.
Website will have following pages: forside (main page), vi tilbyder (we offer), book tid (booking), kontakt (contact) and login. Where from main page, there will be access to the other under pages, as also shown in the navigation map figure 9.6

From login page, there will be access to the database pages. And database access is limited to the employee use only. On database page, there will be a home page (hjem), and under that there will be made 4 under pages with information stored about tires and rims, as it also shown on the navigation map at there will be 4 under pages, which are dæk (tires), fælge (rims), brugeradministration (user administration) and log ud (log out).

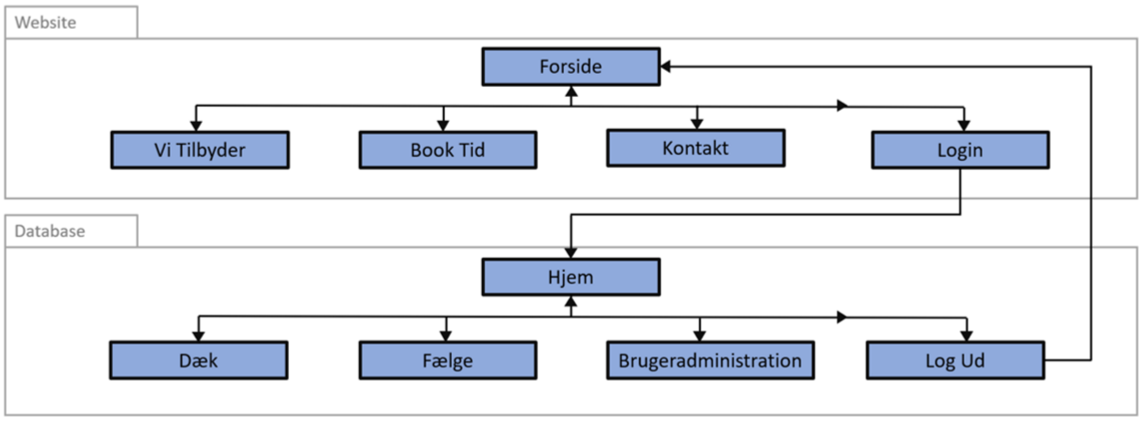


Figure 9.6: Navigation map

Implementation

In this chapter, several different technologies will be described since they have been used through the development and implementation of the system. Also, various program components and functions will be highlighted along with the database structure and variable names.

10.1 Programming Stack

10.1.1 React

React has been used for the purpose of developing our system due to it being fast, efficient, scalable, and simple. React is a User interface (UI) library that helps developers create web applications by using JavaScript and HTML. The reason behind the choice of React is the following:

- React is Flexible
- React Has a Great Developer Experience
- React Has Facebook's Support/Resources
- React Also Has Broader Community Support
- React Has Great Performance
- React is Easy to Test

Also there is a ton of features that helps developing websites easily using React. For example Routing is supported by react router library. Another example would be the concept of class and functions which gives an objected oriented and an imperative perspective as well as reusability of components. Nevertheless, react hooks which was introduced recently in one of the new React versions helps to manage state by writing minimal code.

10.1.2 API

API is short for Application Programming Interface. An API is simply the point of contact between a particular development environment and the developers. It allows these developers to take advantage of the services of this environment without having to build everything from scratch. In general, the goal of the API is to hide the details "Encapsulation" and to highlight the way to take advantage of the code.
So when we use a class, function, or plugin in our project, what matters to us is how we use it to serve our project and not how it works. API can be used to request and get data from another application by using API calls.

We've used the API in many parts of our programs. For example, we used it when asking to log in to see what's in the inventory.
Since we used React in the website in our project, we should wonder how to communicate with the back end server, which is an essential task. For this aim, we used Axios, which is a promise-based library used with Node.js and a browser to make requests to an API, return data from the API, and then do things with that data on our React website.

10.1.3 Maven

Maven is a build tool that helps in project management. The tool helps in building and documenting the project. Maven is based on the Project Object Model (POM) [8][9]
Maven is one of the popular open-source build solution for java applications. It is designed in such a way that automates whole build process. Unlike task-based applications such as Ant or other traditional build files, Maven uses a systematic approach. In this approach the project structure and contents are defined from the beginning. [8][9]

Build tool: It is necessary for the construction process. It's required for the following operations:

- Generating source code
- Generating documentation from the source code.
- Compiling of source code
- Packaging of the complied codes into JAR files
- Installing the packaged code in local repository, server, or central repository [8][9]

The Project Object Model (POM) is an XML file that contains all of the project and configuration information. Maven looks for the POM in the current directory when we try to run a task. Maven is mostly used to in Java Projects. It helps in the download of JAR file libraries. Maven is used in development of software. It helps in the creation and documenting of the project. It makes thing easier during the time of development of software. By using Maven, JAR files and requirements are not required to be downloaded manually. One can easily access all the information quickly. With project management, Apache Maven aids in the administration of all processes such as building, documenting, releasing, and distribution. [8][9]

10.1.4 Java Servlet

A servlet is a server-side web technology based on Java. When the web server receives a request from a client, this request is sent to the servlet, which act as the controller on the application server. As the controller, it is the responsibility of the servlet to get the request and evaluate what must be accomplished to satisfy the request, and then the servlet will utilize components from the model to process data before handing implementation over to the view. The view is then responsible for building a response by putting all the data together using HTML, CSS, and JS before sending this response back to the client through the web server.[10][11]

10.1.5 SQLite Query

One of the most important methods that we use from SQL is query. A query is an inquiry for a set of data, it is used to retrieve important and relevant information from the database. In our case we use Query in the logging in process when the user is trying to access the database. The way we do this is by retrieving the already existing usernames and passwords that are registered in the database, and match them with the ones that are trying to access the database. That is done by scanning the database, making usernames and passwords into a set, which makes it easier for us to run trough all of them and check if the user exists.

10.1.6 API Development

During the implementation of our database onto the front-end that will be accessed by our customer, we have developed multiple APIs that help us communicate across our back-end and front-end. One of such APIs can be found in the customer retrieving part in our code. The API uses the query method from mSQL and by communicating between front and back-end, we retrieve the information, which are then used to login onto the database.

10.2 Deep diving into the different components and the retrieval of the stored data

As mentioned earlier React is based on the idea of classes and functions which are also known as components. A component is a reusable piece of code, serving a specific element on the user interface (UI). There are multiple components in the project that powers this web-application.

10.2.1 The Login component

The login component serves as an entry point to the web-application for the users. Users are prompted to enter their username and password, and If the entered information successful, then users are redirected to the dashboard page and authentication details of the particular user is stored in local storage on the browser. These details in the local storage helps to identify the authenticated user based on which a decision is made to show the dashboard to the user or not.

The login functionality is powered by back-end API that is connected with the SQL-lite database. The back-end API end-point is "/Dashboard/getUser" that queries the "Users" table in the database. The schema of the "user" table is composed of fields such as: " userId, Username, Password, Admin" where "userId" is an auto-incremented field that also acts as our primary key. For efficient searching we have a secondary index on the username field. The back-end API returns the user information in case of success.

The DDL of the user table is as follows:

Listing 10.1: User Table DDL

```sql

CREATE TABLE IF NOT EXISTS users (userId integer primary key AUTOINCREMENT,
    username text NOT NULL, password text NOT NULL)
```

The database query used for fetching the data as follows:

Listing 10.2: Fetch User Query

```sql
SELECT * FROM users where username=? and password =?
```

10.2.2 The Table component

The table component is an important aspect of the web-application since, most of the data is rendered in tabular format. Following the react principle, we have reused the table component to render different kind of data. This is made possible by using dynamical rendering approach, which is based upon passing the data that is required for rendering. The table component is used to show the data related to Tires, Rims and Users.

The table component looks as follows:

Listing 10.3: Table Component in React

```javascript
import "./Table.css";
import ReadOnlyRow from "./ReadOnlyRow";
import EditableRow from "./EditableRow";
import React, { Fragment, useState } from "react";

class Table extends React.Component {
  constructor() {
    super();
    this.state = {
      editRowNumber: null
    }


  }

  setEditRowNumber = (index) => {
    this.setState({ editRowNumber: index });
  }

  resetEditRowNumber = () => {
    this.setState({ editRowNumber: null });
  }

  render() {
    return (
      <form>
        <table>
          <thead>
            <tr>
              {this.props.headers.map((header) => (
                <th>{header}</th>
              ))}
              <th>Actions</th>
            </tr>
          </thead>

          <tbody>
            {this.props.data && this.props.data.map((item, index) => {
              console.log("item ", item);
              return (
                <Fragment>
                  {this.state.editRowNumber === index ? (
                    <EditableRow headers={this.props.editHeaders} data={item}
                        editFun={this.props.editFun} onCancelClick={
                        this.resetEditRowNumber}/>
                  ) : (
                    <ReadOnlyRow data={item} headers={this.props.headers}
                        number={index} onEditClick={(index) => {
                        this.setEditRowNumber(index); }}
                        onDeleteClick={this.props.deleteFun}
                        actionsEnabled={localStorage.getItem("isAdmin")===
                        "true"}/>
                  )}

                </Fragment>);
            })
            }
          </tbody>
        </table>
      </form>
    );
  }
}

export default Table;
```

The data in the tables are also powered by the back-end API's which is also connected to the SQL-Lite database. There 3 different tables in the database such as Users, Rims, and Tires.The DDL of all these tables are as follows:

Listing 10.4: User Table DDL

```
    CREATE TABLE IF NOT EXISTS users (userId integer primary key AUTOINCREMENT,
    username text NOT NULL, password text NOT NULL)
```

Listing 10.5: Rim Table DDL

```
  CREATE TABLE IF NOT EXISTS rims ( rimId integer NOT NULL PRIMARY KEY
      AUTOINCREMENT, condition varchar(30) NULL, dimensions varchar(30)
      NULL,brand varchar(30) NULL,color varchar(10) NULL,material varchar(30)
      NULL,owner varchar(50) NULL,comments varchar(1024) NULL,location
      varchar(50) NULL,date varchar(20) NULL)
```

Listing 10.6: Tire Table DDL

```
  CREATE TABLE IF NOT EXISTS tires(tireId integer NOT NULL, PRIMARY KEY
      AUTOINCREMENT type varchar(20) NOT NULL, dimensions varchar(30) NULL,
      brand varchar(30) NULL, condition varchar(30) NULL, owner varchar(50)
      NULL, comments varchar(1024) NULL, location varchar(50) NULL, date
      varchar(20) NULL
```

The mentioned respective tables also support functionality to add, edit, delete a any particular records. All these futures are supported by the back end API with HTTP methods such as: " GET, POST, PUT, and DELETE". Nevertheless, the table also support Search based on different fields for the above mentioned respective tables.

The Search functionality is a feature rich functionality as it support prefix, suffix and any other part of the string search. Also it handles the case sensitive issues by smartly comparing only the uppercase versions of the strings.

10.3 Dynamical rendering of different types of tables

Dynamical rendering allows us to rendering different data from the same component, having similar layout. In our case we have used this to render the data of Tires, Rims, and Users using the table component which is highlighted above in section 8. The data used for dynamical rendering is passed to the table component from the parent components using props in React. An iteration over the props using the map function in JavaScript helps in dynamical rendering which is showed below in the code snippet:

Listing 10.7: Dynamical rendering

```

              {this.props.headers.map((header) => (
                <th>{header}</th>
              ))}

            {this.props.data && this.props.data.map((item, index) => {
              console.log("item ", item);
              return (
                <Fragment>
                  {this.state.editRowNumber === index ? (
                    <EditableRow headers={this.props.editHeaders} data={item}
                        editFun={this.props.editFun} onCancelClick={
                        this.resetEditRowNumber}/>
                  ) : (
                    <ReadOnlyRow data={item} headers={this.props.headers}
                        number={index} onEditClick={(index) => {
                        this.setEditRowNumber(index); }}
                        onDeleteClick={this.props.deleteFun}
                        actionsEnabled={localStorage.getItem("isAdmin")===
                        "true"}/>
                  )}
                </Fragment>);
            })
            }
```

Next is an example of where we are passing the data from the Tiretable component to the table component for the dynamical rendering.

Listing 10.8: Tire Table Using Dynamical rendering

```

render() {
   return (
     <div id="Tier-table">
       <AddButtonForm searchOptions={this.tierHeaders}
           formFields={this.tierHeadersEdit} onFormSubmit={this.addTire}
           handleSearch={this.handleSearch} />
       <Table tableName="Daek ID#" headers={this.tierHeaders}
           editHeaders={this.tierHeadersEdit} data={this.state.data}
           editFun={this.editTire} deleteFun={this.deleteTire} />
     </div>
   )
 }
```

10.4 Axios

Axios is as mentioned above a librabry used for making HTTP API request. Axios is being used in all our HTTP calls to the server. Here is an example of two API Calls made for the TireTable component which is the GET and Post request:

Listing 10.9: Get Request

```


componentDidMount() {
   axios.get("http://localhost:9000/Dashboard/GetTires").then(result => {
     const data = result.data;
     this.setState({ data: data.Tires });
   }).catch(error => {
     console.log(error);
   }
   )
 }
```

Listing 10.10: Post Request

```
   addTire = (tireInfo) => {
    axios
      .post("http://localhost:9000/Dashboard/addTire", tireInfo
      )
      .then((result) => {
        console.log(result.data);
        const tireId = result.data.tireId;
        if (!tireId) {
          return;
        }
        tireInfo = { ...tireInfo, tireId };
        console.log(tireInfo);
        var data = [...this.state.data];
        data.push(tireInfo);
        this.setState({ data: data });
      })
      .catch((error) => {
        console.log(error);
      });
  }
```

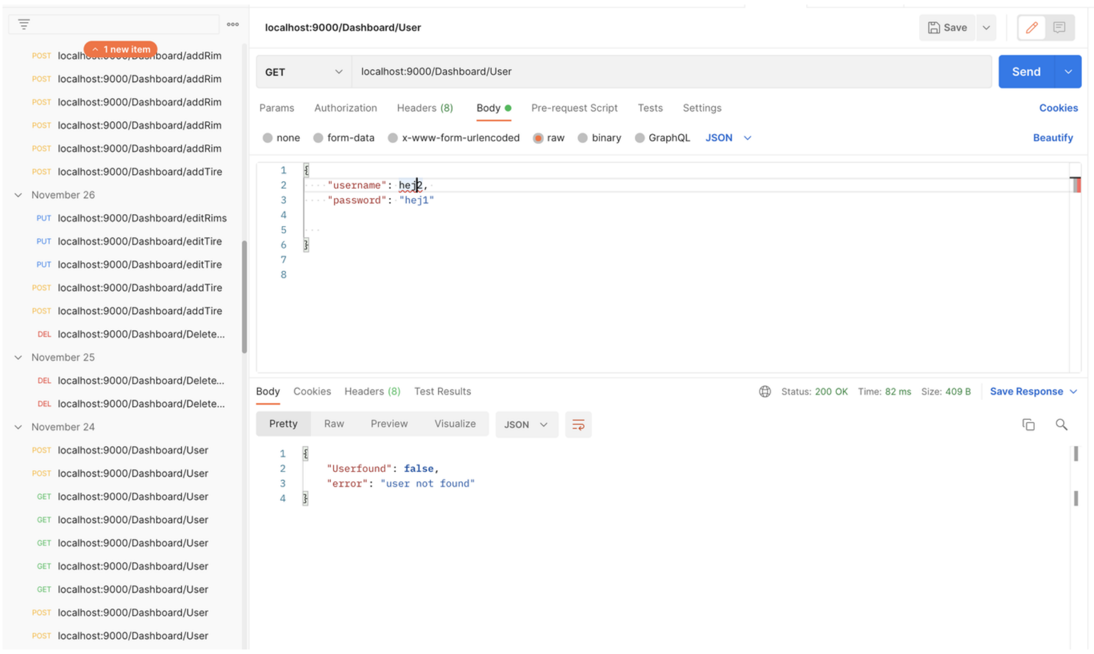
Testing

In this section, we will explain which tests were used and how they were performed on the code. To evaluate the functionality of the program and know whether the developed program meets the specified requirements or not, we need to do software testing. To get a product of good quality and have a high standard.

11.1 API testing

Throughout the development of the back end, we made multiple API tests that made sure everything was working within our database and the front end of the system. To make sure everything was working together, we used Postman, which is a platform used in testing the APIs. It offered us a lot of advantages that made it possible in testing all sorts of HTTP requests, such as GET, POST, PUT and PATCH. The test is a Integration test which shows us if there two components; UI and database, work with each other and there are no defects between them. [12]

To test API with Postman, we begin by opening a request on the Postman website and giving it a name. Once that is done, we select the HTTP request we want to test, and we finish it off by entering the request URL address.
An example of such a test can be found below, where we show how we tested the GET request when we wanted to create a new Tire in our database. We do it by following the steps that were presented before, and we try changing the original data and see if the API works and the database receives the changes.
The test below that is shown in the figure, is a test that was done on the GET request.



1 new item
POST localhost:9000/Dashboard/addRim
POST localhost:9000/Dashboard/addRim
POST localhost:9000/Dashboard/addRim
POST localhost:9000/Dashboard/addRim
POST localhost:9000/Dashboard/addRim
POST localhost:9000/Dashboard/addTire
November 26
PUT localhost:9000/Dashboard/editRims
PUT localhost:9000/Dashboard/editTire
PUT localhost:9000/Dashboard/editTire
POST localhost:9000/Dashboard/addTire
POST localhost:9000/Dashboard/addTire
DEL localhost:9000/Dashboard/Delete...
November 25
DEL localhost:9000/Dashboard/Delete...
DEL localhost:9000/Dashboard/Delete...
November 24
POST localhost:9000/Dashboard/User
POST localhost:9000/Dashboard/User
GET localhost:9000/Dashboard/User
GET localhost:9000/Dashboard/User
GET localhost:9000/Dashboard/User
GET localhost:9000/Dashboard/User
GET localhost:9000/Dashboard/User
POST localhost:9000/Dashboard/User
POST localhost:9000/Dashboard/User
localhost:9000/Dashboard/User
Save
Params
Authorization
Headers (8)
Body
Pre-request Script
Tests
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
Beautify
Send
Body
Cookies
Headers (8)
Test Results
Save Response
Pretty
Raw
Preview
Visualize
JSON

GET localhost:9000/Dashboard/User

```
{
    "username": hej2,
    "password": "hej1"

    
}
```

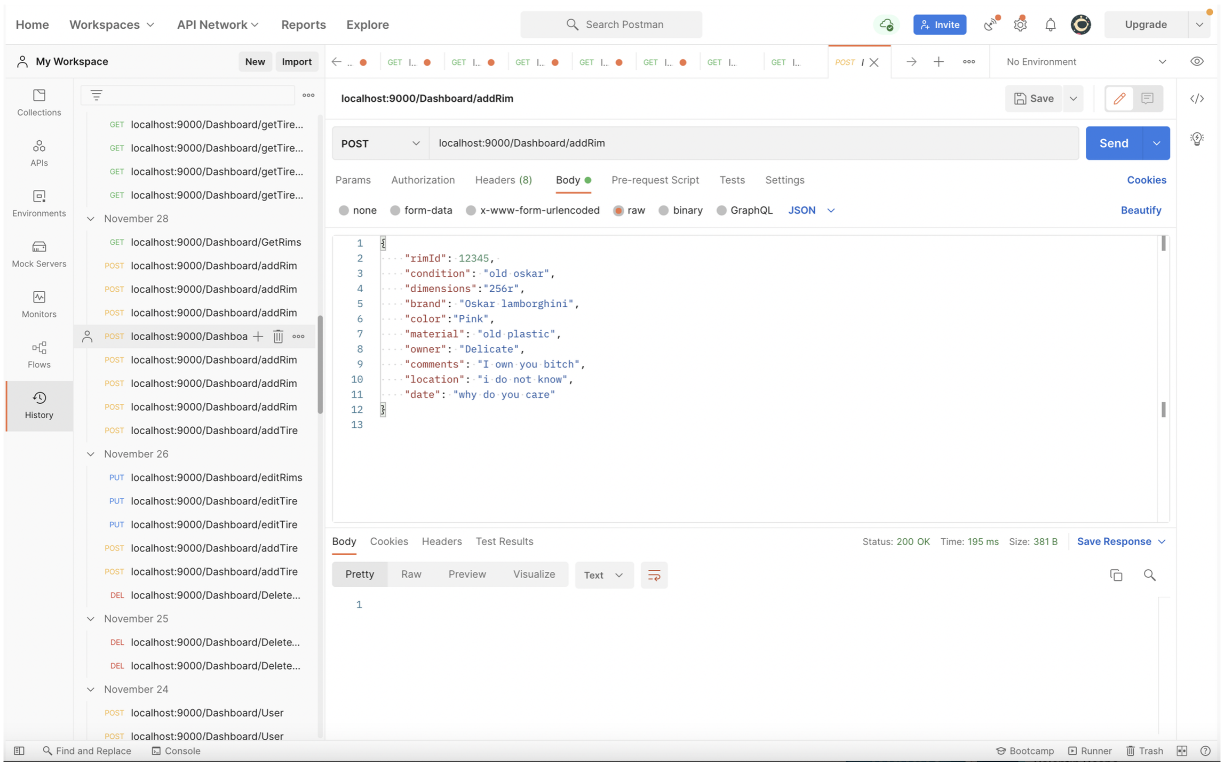
Status: 200 OK Time: 82 ms Size: 409 B

```
{
    "Userfound": false,
    "error": "user not found"
}
```

Figure 11.1: GET Test

As we can see from the figure, the test was a failure and that is because there is no data with the parameters we were looking for.

Another test can be done to make sure that the data is being send to the database. This can be easily done by simply changing the data on the Postman website and pressing the send button. To see if the API is working we simply check if the data changed on the website. The test can be seen below.



POST localhost:9000/Dashboard/addRim

```
{
    "rimId": 12345,
    "condition": "old oskar",
    "dimensions":"256r",
    "brand": "Oskar lamborghini",
    "color":"Pink",
    "material": "old plastic",
    "owner": "Delicate",
    "comments": "I own you bitch",
    "location": "i do not know",
    "date": "why do you care"
}
```

Status: 200 OK Time: 195 ms Size: 381 B

Figure 11.2: POST Test

When the data has been sent, we can check if the changes have been received by our database and can be seen on the UI.

11.2 Usability

The program's usability tests will be clarified and evaluated in the following section. The reason for doing so is to diagnose and fix any system or usability concerns that the current program could very well encounter, as well as to identify missing elements that will be added to the final result.

11.2.1 Usability Plans

Users: The users for the software are the employees of the company, mainly the secretary. Tasks: The tasks required to do is to update the inventory and keep track of the items in the warehouse.

11.2.2 Usability test

When we talk about usability testing, so we talk about the study of users' interaction with a product. A pilot test would be designed to use the software in order to know the errors. All the designed functions of the software would be used for a test and some movements in the inventory would be recorded. The movement of the inventory would then be checked by some reports designed to extract the data. Any errors that would be tested during this trial would be fixed. The data would be input using one employee while other employee would test run the reports and live movements in the inventory. Moreover, the UI of the software needs to be checked. The design should be clear for the employees to use the software. This way if new employees join the business, it should be easier for them to learn and use it.

For this purpose, we presented the final program to the owner of the company to test it and give us his feedback about the website and database. In this way, we have ensured that the customer is satisfied with the final result and if he has any other additions or developments that can be added to the program.

11.2.2.1 Testing the program

The feedback we got from the owner was overwhelmingly positive since he felt that the appearance of the website was simple for him to understand and navigate. He was likewise pleased with the fact that all of the functions on the dashboard worked. He easily managed by himself to log in and then add, edit, and delete a tire without any issues at all. After he did that, he mentioned that the whole web-application was what he expected and that we did a good job. However, we still have some flaws regarding the design part which will be enhanced and improved in the future since the company would actually like to host the web-application.
It is good that the customer was satisfied with the result, as his comment was very positive and he believes that the program is easy to use and serves the desired goal.

Discussion

This chapter will introduce a discussion of the project results and the final software product, which will be evaluated in relation to the goals formulated earlier in the project and in relation to similar existing software. This will be followed by a section where the possible improvements will be discussed regarding the software product.

12.1 Programming

At the start of the project, an idea of a web application was thought out along with many attractive solutions. However, we came to a realization that our initial solution needed to be modified. This occurred shortly after our first interview with the company, which resulted in a thorough understanding of the company's needs which is mentioned in section 2.2.4 and 2.3. Therefore, an internal group brainstorming occurred regarding how to approach possibly upcoming issues. Due to the project's complexity, we decided that a reassessment and re-evaluation of the general requirement have to be reiterated in order to avoid huge complications.

12.1.1 The implemented technology Stack and its complications

The stack that has been chosen for developing our web application which is mentioned in 10 was a good combination. However, many of us were inexperienced regarding the usage and the concepts of these new technologies since we were not familiar with them at the beginning of the project. Therefore, a lot of study time has been required so that our main requirements would be fulfilled within the scope of our MoscoW model which is mentioned in section 5.1.1. The following achievement of our new knowledge was about:

- Routing
- Dynamical Rendering
- API Development
- Inheritance
- Class component vs functional components
- Hooks in React
- Object-oriented approaches in Java
- Maven
- Design patterns and Minimalism

- Local storage (Cookies)

However, we constantly had to revise and change our requirements in the Moscow model. For example, we had to prioritize the main functionalities, which consisted of three main components:

- Rim-table
- Tire-table
- User-table

These above-mentioned components delayed the project due to the complexity of reusing components in React such as the table component which also needed to be dynamically rendered with its own data. These sudden issues became more challenging to solve proportional to the complexity of the web-application. These issues will be explained further in section 12.2.

12.2 Program limitations

This section will explain the current sources of program limitation and error and minor flaws.

12.2.1 Table components

Due to the fact that all variants of the table component are iterations of the same table component, issues arose regarding dynamically updating attributes. One of these issues was regarding the development of a shelf component, which should have consisted of both the tire and rim component with a header containing attributes such as for that class of objects.

Nevertheless, it was meant to be implemented before it was given a minor priority. This was because of the fact that it was prohibitively complicated to update, automating, and maintain certain table attributes, such as current capacity. 12.2

12.2.2 Design limitation

Our design of the web-application isn't completely done and fledged due to our prioritization of the functionality. For example, the rendered tables in the dashboard page isn't centered and the sidebar design can be optimized into a better version. Also the front-end design of the website can be optimized a bit more. These design flaws will be corrected after project turn-in. However, the objective goal regarding the whole design of the web-application isn't far away from the design vision.

12.3 Program improvements

The section focuses on future modification and optimization of the software product. This examination reviews potential improvements to the code and design that never made it to the final product, but could have been developed and implemented, had the project had more time and resources.

12.3.1 Shelves

A minor improvement to the software had been the inclusion of shelf objects that would allow users to view specific shelves containing information about the exact inventory stored, total storage capacity, and occupied storage capacity. This improvement was considered a non-essential, quality of life improvement, and consequently, this feature was given a low priority and never made it to the final product.

12.3.2 History

A history feature was discussed, essentially presenting users with a log containing all database changes in a specified time period.

12.3.3 Hashing & Salting

Hashing involves using complex mathematical algorithms for the encryption of password strings into random, non-reversible character strings. A specific input string will always produce identical encrypted output strings. These encrypted strings are stored in the database, where they will be compared to the newly hashed "password string" when trying to verify a login attempt.
If someone were to hack into the database, any attempt to steal any user accounts information would yield hashed strings, which cannot be easily translated into the original password string and would require some sort of rainbow- or brute force attack. Salting adds a short, random string of characters to the password string before it is hashed. The salt is usually stored in plain text, along with the hashed password, so the website knows which salt to use for a particular password. This will not prevent brute force attacks but will likely prevent any rainbow attacks trying to decipher hashed passwords. [13][14]

The addition of these cyber-security measures has been discussed, but as the database does not store any sensitive information, such features were considered non-essential.

12.3.4 Undo function

The software could be further improved by the implementation of an undo feature to the database UI, allowing users to reverse accidental or otherwise undesired changes.

12.3.5 Email

Usability of the website could be further improved be the inclusion of an email feature on the website, allowing users quick access to email correspondence with the ZN Autocenter.

12.4 Unit Test vs Integration Test

The section will explain the reasoning behind our choice of testing techniques for the software product.

We have used an integration test instead of a unit test regarding our web-application. The reasons for this choice of test was mainly because our application consist of several API's. Therefore, it much more important to test the API-endpoints in parallel with the development of the software to ensure that the persisted data is inserted and retrieved correctly in the database for a later purpose of implementing it on the interface of the web-application by an API-call request. This choice of method is ensuring time efficiency in the testing phase and avoiding any loss and mismanagement of data.
Postman which is an HTTP client with a graphical interface were used to test our each of our HTTP request manually to ensure that we would get a status code of 200. 11.1

We used a Postman manual integrated test for testing the database to make sure everything are working as we want. Where we used it to check about UI and database are communicating properly and there are no flaws between them, as also mentioned in 11.1.
The second reason why we did not choose to use the unit test is that it is very complex and needs more research and reading to find out how we can perform this type of testing on the database. In addition, it is outside the curriculum for this semester.

The unit test method tests pieces of source code and integration test analyze the integration between software modules. The unit test method tests pieces of source code. It divides the program into units and then tests them if the program is working correctly. On the other hand, Integration test combines modules in the application and see if they all work fine together. The unit test is a type of White Box testing; integration test is type of Black Box Testing.[15]

Unit testing can be carried out at any time. However, integration test is done after Unit testing and before System Testing has been carried out. Unit testing might not catch the integration errors or the system wide issues, while integration test is focused on detecting errors when systems get integrated.[15]

Unit test's starting point is module specification while integration test's starts with interface specification. Unit testing is conducted by the developer while integration testing is executed by the test team. Unit testing does not test relationships with other modules. Testing and finding errors in integration testing is difficult. Unit testing is cheaper to implement. Integration testing is expensive to test [15] The integration test is also called black-box testing: software testing that focuses on the input and output of software applications and is entirely dependant on software requirements and specifications.

Reiteration over the report and the code itself

It was well known that we had to develop a large application for a particular company before entering and starting on this particular project journey. And during so, we decided upon choosing an iteration approach regarding both the project paper and the code itself. The iteration approach is based on a recycle process where everything have to be revised more than twice at least. This approach was chosen because we wanted to avoid any big complication and was technically used as a fail-safe so the further development would reach the highest and potential quality that is needed for both parties- our own expectations and for the ZN Autocenter.

12.4.1 What have been revised

There have been many advantages for taking an iterative approach. one of these advantages was to identify the risks and resolve them during the iteration. Another advantage was the ability to have a parallel development plan. This provided us with the opportunity to break down the process of developing a large application into smaller parts, where each part would be tested and debugged too. In this way, we avoid any bugs in the program, and the result is a program that works very well without any bugs or errors. During the process of writing the code, we made some decisions that would make the program looks perfect. These decisions are related to appearance, and here we mean the website and how we can make it look better. For example, the use of React in the design of the website, the colors used, etc. Other decisions are related to the code itself and how to reach a perfect program free of any errors. Therefore, we used the iteration method, where we were able to divide the program into small parts to make it easier for us to develop and test its work and finally get a satisfactory result.

Conclusion

13.1 Summary

The aim of this project was to develop a new website and a database for ZN Autocenter to manage the inventory of their tires and rims. To solve this problem, we started by studying the status of the current website and arranged an interview with the company to gain a thorough understanding of the tasks and assets we were to digitalize, as well as to discuss the company's requirements and wishes for the program. Following an analysis of the information gathered during the interview, the problem statement was formulated.

The system definition was used as a starting point to our factor analysis to give us an overall overview of what the system should consist of, the scope of the system's functionality, who will be using the program, who will be developing it, what kind of technology stack will be used, the main objects and which responsibility has the system.

Based on our PACT analysis, work began on designing a user interface for the website and the database. Various wireframes were produced to experiment and demonstrate different design ideas, internally for project members and later externally for ZN Autocenter employees before a final design direction was selected. The front-end UI designs for both website and database evolved slightly from the initial wireframes. The front-end of the website was made using the standard web development methods such as HTML, CSS, and JavaScript, with the help of React.js. Whereas the back-end was made using Java with the help of APIs, Maven, Java Servlet, and SQLite.
The program code has been evaluated through Integration testing, where we have used Postman to test APIs and the system's overall functionality, as well as to highlight potential issues. Both front-end and back-end, appear to be working as intended.

13.2 Conclusion

So the final result is that we have a website that includes a login system that allows users to manage the inventory and database and a search function that helps find any item in the inventory. Moreover, some functions make it easier for users to control and see the different items in the inventory. Therefore, the program fulfills the requirements that were established at the beginning and the product was approved by the costumer who liked it.

Bibliography

[1] *Source.*

[2] Forenede Danske Motorejere. "FDM Daek". In: (2951). URL: `%7Bhttps://fdm.dk/alt-om-biler/test-udstyr/daek/hvor-gamle-maa-daek-vaere%7D`.

[3] Autopartner. "Gamle Daek". In: (3). URL: `%7Bhttps://www.autopartner.dk/tips-tricks/daek/kan-daek-blive-for-gamle%7D`.

[4] Sortly. "Sortly management". In: (unknown). URL: `%7Bhttps://www.sortly.com/%7D`.

[5] Benyon. *Designing user experience*, pp. 151–152.

[6] Lars Mathiassen. $factor_analysis$.

[7] Kendra Cherry. *The Color Psychology of Blue.* URL: `https://www.verywellmind.com/the-color-psychology-of-blue-2795815`.

[8] Simplilearn. *What Is Maven? | What Is Maven And How It Works?* URL: `https://www.youtube.com/watch?v=bSaBmXFym30`.

[9] maven.apach. "Welcome to Apache Maven". In: (2021). URL: `https://maven.apache.org/`.

[10] CraigPiercy. *Introduction to Java Servlets.* URL: `https://www.youtube.com/watch?v=uQzedmaebvk&ab_channel=CraigPiercy`.

[11] GeeksforGeeks. *Introduction to Java Servlets.* URL: `https://www.geeksforgeeks.org/introduction-java-servlets/`.

[12] "Introduction to Postman for API Development". In: (2021). URL: `https://www.geeksforgeeks.org/introduction-postman-api-development/`.

[13] Seytonic. *Password Hashing, Salts, Peppers | Explained!* URL: `https://www.youtube.com/watch?v=--tnZMuoK3E&ab_channel=Seytonic`.

[14] BetterProgramming. *What is Hashing and Salting.* URL: `https://betterprogramming.pub/salting-and-hashing-explained-b76f5af83554`.

[15] Thomas Hamilton. "Unit Test vs Integration Test: What's the Difference?" In: (2021). URL: `https://www.guru99.com/unit-test-vs-integration-test.html`.